**Ловушка «лёгкого входа»: как работодатели заманивают новичков красивыми вакансиями, скрывая реальность**

Бывает так: вы видите вакансию мечты для старта карьеры. Никаких жёстких фильтров, дружелюбный тон, обещания работы с современными технологиями. Вы смело откликаетесь, но дальше происходит одно из двух: либо вы получаете мгновенный отказ, столкнувшись с завышенными требованиями на собеседовании, либо выясняется, что сама работа не имеет ничего общего с тем, что было написано в объявлении.

На рынке труда всё чаще используют маркетинговые уловки, чтобы искусственно раздуть поток кандидатов. Разбираемся, зачем работодателям такая «массовка» и как именно они создают иллюзию лёгкого старта.

**Зачем компании устраивают искусственный «кастинг»?**

Главная цель намеренно упрощённой вакансии — снять психологический барьер у соискателя и создать для компании максимальный выбор.

Если написать в объявлении реальные требования, откликнутся 10 человек — выбирать будет не из кого. Если написать «опыт не нужен, всему научим», откликнутся 300 человек. Из этой толпы рекрутер спокойно выберет 2–3 самых сильных и при этом недорогих «самородков». Остальные 297 человек нужны лишь для статистики и создания большого выбора, хотя их шансы пройти собеседование с нуля изначально близки к нулю.

**Как работодатели создают иллюзию «лёгкого входа»**

Упрощение требований для расширения воронки кандидатов на практике реализуется через несколько конкретных уловок в тексте вакансии:

- **Размытые требования вместо конкретных навыков.** Формулировки вроде «стрессоустойчивость», «желание учиться», «быстрая обучаемость» без указания технологий, инструментов или уровня задач. Это позволяет приглашать на собеседования практически всех, но на техническом интервью выясняется, что нужен готовый специалист с опытом.
- **Завышенные обещания по задачам.** В описании может быть указано «работа с современным стеком» или «участие в интересных проектах», тогда как реальная работа сводится к рутинной поддержке легаси-кода или монотонному вводу данных. Новичок обнаруживает это только после выхода на работу или на финальном этапе отбора.
- **Использование «дружелюбного тона» для снижения критичности.** Неформальный стиль, эмодзи, обращения на «ты» и обещания «уютного офиса» создают ощущение безопасности и снижают бдительность. Кандидат меньше внимания уделяет изучению реальных требований и больше — эмоциональной привлекательности объявления.

**Крайняя степень уловки: «айтишник» с гаечным ключом**

Отдельный приём — подмена понятий, когда тяжёлую, рутинную или физическую работу заворачивают в модную IT-обёртку.

В заголовке пишут: *«Младший IT-специалист»*, *«Инженер внедрения»* или *«Специалист по настройке систем»*. В тексте обещают работу с компьютерами и современным оборудованием. А на собеседовании выясняется, что вам придётся с гаечными ключами и изолентой копаться в проводке автомобилей как автоэлектрик, тянуть кабели на складах или работать в жёстком сменном графике 4/4.

Расчёт простой: на вакансию «автоэлектрик-установщик» откликается мало людей, особенно среди молодёжи. Стоит добавить слова «IT» и «цифровые системы» — и поток резюме вырастает в разы. Работодатель надеется, что из сотни пришедших хотя бы пара человек махнёт рукой и согласится попробовать.

**Как распознать вакансию-приманку до отклика**

Чтобы не тратить время на пустые собеседования и бесплатные тестовые задания, проверяйте объявления по четырём признакам:

1. **Ищите конкретику про рабочий день и стек.** Если вместо списка программ и реальных задач в вакансии только общие слова про «дружную команду» и «интересные вызовы» — это повод насторожиться.
2. **Уточняйте формат работы.** Если написано «работа с оборудованием и ПО», но не сказано, где именно вы находитесь (в офисе за столом или на выездах/в боксе), спросите об этом в чате до собеседования.
3. **Обращайте внимание на размытый график.** Фразы вроде «гибкий график» или «сменный режим, обсуждается на встрече» часто скрывают тяжелые смены 4/4, ночные дежурства или постоянные разъезды.
4. **Проверяйте, кто будет вас учить.** Если обещают «всё показать с нуля», но в описании нет ни слова про наставника, программу стажировки или конкретные задачи на первый месяц — скорее всего, учить вас никто не собирается.

**Резюме:** Простое и дружелюбное описание вакансии сегодня — это прежде всего инструмент воронки продаж. Работодатели научились привлекать внимание яркими заголовками и обещаниями лёгкого старта, чтобы собирать максимум откликов. Относитесь к таким объявлениям критически: всегда уточняйте реальные задачи и условия труда до того, как потратите время на тестовое задание или поездку в офис.

**Юридические и цифровые подводные камни: когда компании «нет на бумаге»**

Отдельная категория ловушек для новичков — красивые IT-стартапы и «экосистемы» с привлекательными стажировками без опыта (часто с обещанием удалёнки и наставника). Если начать проверять такую компанию перед откликом, нередко всплывают тревожные сигналы. Разберём на типичном примере (похожая картина встречается у многих молодых проектов вроде Pulsea):

- **Компании нет в ЕГРЮЛ и открытых реестрах.** На сайте проекта нет ни ИНН, ни ОГРН, ни реквизитов — только красивая презентация продукта, форма отклика и свежий домен со скрытыми данными владельца.
- **Оформление через чужое или непрофильное ИП.** Если начать задавать вопросы о договоре, рекрутер может прислать реквизиты стороннего индивидуального предпринимателя. При проверке по базе ФНС внезапно выясняется, что основной вид деятельности (ОКВЭД) этого ИП — ресторанный бизнес, общепит или торговля, а среди дополнительных кодов кое-как вписаны размытые формулировки вроде «оказания прочих услуг» или «деятельности по обработке данных».
- **Чем это грозит новичку.** Работа через непрофильное ИП (часто по договору ГПХ, самозанятости или вообще без официального оформления) снимает с реальных организаторов проекта почти всю ответственность. Если стажёр отработает месяц или два и не получит обещанную оплату, доказать связь между «модным IT-брендом» и владельцем шашлычной или кафе будет практически невозможно.

**Чек-лист: как распознать вакансию-приманку до траты времени**

Чтобы не работать бесплатно и не ездить на пустые собеседования, проверяйте каждую «простую» вакансию по пяти пунктам:

1. **Ищите конкретику про рабочий день и стек.** Если вместо списка программ и реальных задач в вакансии только общие слова про «дружную команду» и «интересные вызовы» — это красный флаг.
2. **Уточняйте формат работы и график заранее.** Сразу спросите рекрутера в переписке: *«Это работа исключительно за компьютером в офисе или предполагает выезды и физический монтаж?»* и *«Какой точный график смен?»*.
3. **Проверяйте, кто будет вас учить.** Если обещают «всё показать с нуля», но в описании нет ни слова про наставника, программу стажировки или конкретные задачи на первый месяц — скорее всего, учить вас никто не собирается.
4. **Пробивайте ИНН до тестового задания.** Спросите, от какого юрлица идёт наём. Проверьте ИНН через бесплатные сервисы проверки контрагентов: совпадает ли название бренда с названием компании и соответствуют ли коды ОКВЭД заявленной IT-сфере.

5. **Оценивайте возраст сайта и реальность продукта.** Если домену пару месяцев, в соцсетях тишина, а у сервиса нет живых клиентов и отзывов — не соглашайтесь на бесплатную работу «за опыт и строчку в резюме».

**Резюме:** Простое и дружелюбное описание вакансии сегодня — это прежде всего инструмент маркетинга и воронки отбора. Зная, как устроены эти уловки изнутри, вы сможете вовремя отсеивать пустые обещания, защищать своё время и находить компании, где новичков действительно ждут и готовы оформлять по-белому.

**Ловушка с «зарплатным проектом через открытие ИП»**

Отдельный «красный флаг», который всё чаще встречается на собеседованиях в подобных проектах — предложение оформить сотрудничество через открытие собственного ИП. HR подаёт это как стандартную и удобную процедуру: *«Мы работаем через открытие ИП, потому что так идёт подключение к нашему зарплатному проекту, вам откроют расчётный счёт, куда будет приходить зарплата»*.

На языке бизнеса это называется **подменой трудовых отношений** и экономией на налогах за счёт сотрудника. Для новичка здесь скрыто сразу четыре серьёзные ловушки:

- **Вы сами платите налоги и взносы.** При обычном трудоустройстве налоги (НДФЛ) и страховые взносы за вас платит работодатель. Как только вы открываете ИП, вы становитесь самостоятельным предпринимателем: вам придётся платить налог с поступлений (обычно 6% по УСН или НПД) и обязательные фиксированные страховые взносы в Социальный фонд, даже если вам задержали выплату или вы вообще ничего не заработали.
- **Никаких трудовых гарантий.** У ИП нет оплачиваемого отпуска, больничных, декретных и защиты Трудового кодекса. Расстаться с вами могут одним днём без выходного пособия — просто расторгнув договор оказания услуг.
- **Штрафы от налоговой.** Налоговая инспекция активно борется с такими схемами. Если вы получаете фиксированную «зарплату» два раза в месяц от одного и того же контрагента, ФНС может признать договор трудовым и доначислить налоги и штрафы.
- **Головная боль с закрытием и отчётностью.** Если через месяц вы поймёте, что проект вам не подходит или вас обманули, просто уволиться не получится — придётся подавать декларации, платить взносы за отработанные дни и официально закрывать ИП через налоговую.

**Важное правило:** «Зарплатный проект» по закону бывает только у штатных сотрудников по трудовому договору (когда банк выпускает

карту для физлица). Расчётный счёт ИП предназначен для бизнеса, а не для получения зарплаты. Если на старте карьеры от вас требуют открыть ИП ради «удобных выплат» — это повод сразу отказаться от предложения.

Есть куда более опасный мотив — **использование расчётного счёта новичка в качестве «транзитного» (дропперство или обналичивание средств)**.
В этой схеме предприниматель отвечает всем своим личным имуществом, а реальные организаторы просто исчезают.
Ниже — дополнение к блоку про ИП, которое объясняет именно **скрытую выгоду организаторов** и риски стать «крайним» перед банками и правоохранительными органами.

**Зачем им это на самом деле: риск стать «транзитным звеном»**

Помимо банальной экономии на налогах за штатного сотрудника, у недобросовестных работодателей есть ещё один, гораздо более опасный мотив. Когда новичок ОТКРЫВАЕТ ИП и расчётный счёт по указке «работодателя» (часто ещё и по их реферальной ссылке в конкретном банке, за что рекрутер получает бонус от банка), этот счёт могут использовать в тёмную:

- **Прогон чужих денег и обналичивание.** Через ваш расчётный счёт могут проводить сомнительные платежи от сторонних фирм под видом «оплаты за IT-услуги», а затем просить вас перевести эти деньги дальше или снять. На языке банков и закона это называется транзитом или участием в схемах по отмыванию средств.
- **Блокировка по 115-ФЗ и «чёрный список» банков.** Как только банк увидит подозрительные операции по вашему **ИП**, счёт заблокируют по антиотмывочному закону (115-ФЗ). Вы попадёте в межбанковский «чёрный список» (так называемый «светофор» ЦБ), после чего вам годами не откроют счёт и не дадут карту ни в одном нормальном банке.
- **Вся ответственность — на вас.** По закону индивидуальный предприниматель отвечает по всем долгам, налогам и сомнительным сделкам **ВСЕМ СВОИМ ЛИЧНЫМ ИМУЩЕСТВОМ** (квартирой, машиной, счетами). Если к вашему ИП придут с вопросами налоговая или полиция, реальные организаторы «IT-стартапа» просто удалят чаты, а крайним по документам останетесь вы.
**Главный вывод:** Никогда не открывайте ИП и расчётный счёт ради того, чтобы просто устроиться на работу стажёром или джуном.

Бесплатный «зарплатный проект» для ИП — это маркетинговая уловка, за которой скрывается либо перекладывание налогов на ваши плечи, либо попытка сделать вас подставным лицом в чужих финансовых схемах.

Каким бы «молодым» или «суперсовременным» ни называл себя проект, **любая реальная компания должна биться по ИНН в официальной выписке ЕГРЮЛ (или ЕГРИП)**.
При регистрации бизнеса в России запись в государственном реестре и присвоение ИНН появляются **уже спустя сутки** (на следующий рабочий день). Поэтому оправдания в духе *«мы новый стартап, ещё не успели всё оформить»* — это всегда сигнал тревоги.
Запомните три коротких правила безопасности на старте карьеры:

1. **Нет ИНН в ЕГРЮЛ — нет компании.** Если проекта нет в базе налоговой, а на сайте спрятаны реквизиты, вы рискуете потратить время впустую или остаться без оплаты.
2. **Сверяйте профиль бизнеса.** Если вам присылают реквизиты стороннего ИП с кодами ОКВЭД из общепита или торговли вместо IT, а в вакансии обещают «работу с современным стеком» — перед вами ширма.
3. **Никакого открытия ИП ради «зарплатного проекта».** Легальный работодатель оформляет новичков по трудовому договору (или официально по ГПХ/самозанятости без принуждения к открытию расчётного счёта). Требование открыть ИП для получения зарплаты — это попытка переложить на вас налоги или втянуть в сомнительные финансовые схемы.

**Психологический спектакль: «оффер без проверки» и пустой договор**

Чтобы новичок потерял бдительность и быстрее согласился на сомнительные условия (например, на открытие ИП), недобросовестные HR-менеджеры часто разыгрывают целый спектакль с игрой на эмоциях:

- **Имитация бурной занятости и «качели».** Рекрутер обещает вернуться с ответом через сутки, но специально пропадает на два-три дня. Затем пишет с подчеркнуто виноватым и дружелюбным видом: *«Ой, извини, пожалуйста, что задержалась, у нас тут такой аврал!»*. Это создаёт у кандидата ощущение, что компания реально кипит работой, а HR к нему тепло относится.
- **Внезапное принятие на работу вопреки правилам.** Сразу после извинений следует радостная новость: *«Мы посоветовались с техническим директором и решили принять тебя в команду!»*. При

этом грубо нарушается их же собственный регламент. Например, в условиях тестового задания было чётко написано, что после сдачи обязателен технический созвон с разбором кода или решения, но никто ничего не проверяет и не созванивается. Реальному бизнесу важен навык сотрудника, а мошенникам или сборщикам ИП важно как можно быстрее дожать человека до оформления документов.

- **«Красивый» договор-пустышка без реквизитов.**
На финальном этапе кандидату присылают внешне солидный бланк договора — с грамотными юридическими формулировками, пунктами про конфиденциальность и обязанности. Но если вчитаться в шапку и реквизиты сторон, оказывается, что это пустой шаблон: там нет ни **ИНН**, ни **ОГРН**, ни юридического адреса «молодой организации». По сути, это просто скачанный из интернета текст, который не имеет никакой юридической силы и не связывает реальных организаторов никакими обязательствами.
**Как это работает вместе:** Эмоциональная радость от «победы» («меня взяли без опыта, сам техдир одобрил!») отключает критическое мышление. На этой волне новичок не замечает ни отсутствия технического интервью, ни пустых граф в договоре и бежит открывать ИП по присланной ссылке.

**простая вакансия → эмоциональный крючок → скрытая выгода работодателя.**
Ниже — разбор четырёх самых частых сценариев, которые вы можете добавить в статью в виде отдельных кейсов.

**1. Схема «Фейковая стажировка за плату или покупку курса»**

**Как выглядит вакансия:** «Набираем джунов без опыта на оплачиваемую стажировку. Обучим с нуля, после успешного прохождения — гарантированное трудоустройство в штат».
**Как развивается схема:**

1. Вы откликаетесь и проходите очень лёгкое «собеседование», где вас хвалят и говорят, что у вас отличный потенциал.
2. За пару дней до старта HR сообщает: *«Вы нам подходите, но у нас есть обязательное внутреннее обучение / сертификация по стандартам компании. Оно платное, но с первой зарплаты мы всё компенсируем»*.
3. Либо второй вариант: вам говорят, что уровень чуть-чуть не дотягивает, и скидывают ссылку на «партнёрскую онлайн-школу», после окончания которой вас якобы точно возьмут.

**В чём выгода организаторов:** Никакой работы и стажировки не существует. Это скрытая воронка продаж курсов или сбор денег за

«методички и доступ к платформе». После оплаты связи обрываются или кандидат «не сдаёт финальный экзамен».

## 2. Схема «Краудсорсинг под видом тестового задания»

**Как выглядит вакансия:** Простая вакансия на удалёнку (копирайтер, дизайнер, фронтенд-разработчик, аналитик, переводчик). Требования минимальные, обещают быстрый старт.

**Как развивается схема:**

1. Вместо беседы всем откликнувшимся рассылают объёмное тестовое задание: написать 3 полноценные статьи, сверстать реальный лендинг, нарисовать баннеры для рекламной кампании или спарсить базу контактов.
2. Вы тратите 10–15 часов, сдаёте идеальную работу.
3. Через пару дней приходит стандартный отказ: *«Спасибо, вы молодец, но мы выбрали другого кандидата»*.

**В чём выгода организаторов:** Компания вообще никого не нанимает. Сотни новичков бесплатно закрывают реальные задачи бизнеса по кусочкам. Из 200 присланных тестовых заданий менеджер собирает готовый сайт, контент-план на месяц или базу клиентов, не заплатив ни рубля.

## 3. Схема «Теневой аутстафф (продажа новичка под видом сеньора)»

**Как выглядит вакансия:** «Ищем начинающего разработчика / тестировщика в дружную команду. Главное — базовые знания и готовность много общаться».

**Как развивается схема:**

1. Вас быстро берут на скромную зарплату (или вообще на «испытательный срок за копейки»).
2. Вместо внутренних задач вам дают чужое резюме с приписанными 4–5 годами опыта и отправляют проходить собеседования в крупные компании или банки от имени «опытного специалиста».
3. На созвонах вам в наушник или в соседний чат подсказывают ответы реальные мидлы/сеньоры.

**В чём выгода организаторов:** Недобросовестная аутстафф-студия продаёт заказчику новичка по ставке опытного специалиста (например, за 300–400 тысяч рублей в месяц), а самому новичку платит 50 тысяч. Когда заказчик через пару месяцев понимает, что

специалист не тянет уровень, новичка просто выкидывают и ставят на его место следующего.

**4. Схема «Сбор персональных данных и привязка к микрозаймам / картам»**

**Как выглядит вакансия:** Максимально простая удалёнка: «Оператор ПК», «Модератор чатов», «Тестировщик банковских приложений». Обещают ежедневные выплаты и свободный график.
**Как развивается схема:**

1. HR пишет в мессенджере и сразу радостно сообщает, что вы приняты.
2. Для «оформления пропуска, проверки службой безопасности или подключения выплат» вас просят прислать фото паспорта (главная страница + прописка), СНИЛС, селфи с паспортом в руках, а иногда — оформить дебетовые карты нескольких банков по реферальным ссылкам и прислать их данные.
3. Как только документы отправлены, переписка удаляется.

**В чём выгода организаторов:** На ваши паспортные данные оформляют микрозаймы, регистрируют электронные кошельки для теневых переводов или продают связку документов на чёрном рынке. А за оформление карт по ссылкам рекрутер забирает партнёрские выплаты от банков.

**Общая формула всех этих схем**

Во всех случаях работает один и тот же сценарий:

1. **Занижение порога входа** (чтобы отключить страх отказа и привлечь побольше новичков).
2. **Ускоренный «успех»** (вас хвалят и берут без реальной проверки навыков, чтобы вызвать эйфорию).
3. **Перевод в закрытый мессенджер** (Telegram или WhatsApp вместо официальной почты и корпоративных каналов).
4. **Нестандартное требование на финале** (открыть ИП, заплатить за обучение, сделать бесплатный проект, прислать фото с паспортом или пройти собеседование под чужим именем).

ТЕПЕРЬ пойдут всякие полезности как исправить те или иные ошибки в быту в плане орг.техники:

## Неочевидные поломки принтеров, периферии и ПК (и как их чинить)

В работе системного администратора и специалиста техподдержки самые коварные заявки — те, где устройство «вроде работает, лампочки горят, но результата нет». Ниже собраны 9 типовых, но неочевидных сбоев оргтехники и компьютеров с пошаговым алгоритмом их решения.

### 1. Принтер: документ отправлен, но устройство «молчит»

Внешне принтер готов к работе, кабель подключён, ошибок нет, но задание исчезает из очереди без печати.

- **Зависшая служба диспетчера печати (Print Spooler).** Очередь забивается битым файлом, и все последующие документы блокируются.
  - *Решение.* Откройте командную строку от имени администратора и выполните по очереди: `net stop spooler`, затем очистите папку `C:\Windows\System32\spool\PRINTERS` и запустите службу обратно командой `net start spooler`.
- **Слетел порт или IP-адрес (для сетевых принтеров).** Если у принтера не закреплён статический IP, роутер после перезагрузки мог выдать ему новый адрес.
  - *Решение.* Распечатайте страницу конфигурации с самого принтера (обычно зажатием кнопки «Отмена» или «ОК» на 5–10 секунд), узнайте текущий IP и обновите порт в свойствах принтера на ПК.
- **Конфликт WSD-порта.** Windows любит подключать сетевые принтеры через протокол WSD, который регулярно «отваливается» после спящего режима.
  - *Решение.* Удалите WSD-порт и создайте стандартный порт TCP/IP с жёсткой привязкой к IP-адресу.

### 2. Принтер гудит, прогоняет бумагу, но выдаёт белые листы

Механика работает исправно, лист проходит весь тракт без замятий, но выходит чистым (для лазерных аппаратов).

- **Нет контакта на фотобарабан или магнитный вал.** После заправки или тряски отошла контактная пружина или её залили тонером. Без высокого напряжения тонер не переносится на бумагу.
  - *Решение.* Вытащите картридж, осмотрите контакты на торцах и внутри принтера, протрите их сухой безворсовой салфеткой.
- **Не открывается шторка блока лазера.** Обламывается пластиковый флажок на картридже, который нажимает на рычаг шторки — лазер светит в закрытую заслонку.
  - *Решение.* Сделайте «стоп-тест»: откройте крышку принтера прямо во время прохода листа и посмотрите на фотобарабан. Если барабан чистый — проблема в лазере или контактах; если изображение на барабане есть, но не перешло на бумагу — неисправен вал переноса (Transfer Roller).

- **Забыли выдернуть защитную чеку из нового картриджа.** Пломбу, перекрывающую отсек с тонером, забыли вытянуть перед установкой.

**3. Принтер: бумага на месте, ролик целый, но лист не захватывается**

Аппарат щёлкает, пытается забрать лист, но бумага остаётся в лотке.

- **«Залипание» муфты соленоида.** Внутри принтера резиновая демпферная прокладка соленоида со временем превращается в липкую массу. Якорь прилипает, и ролик не делает полный оборот.
  - *Решение.* Снять боковую крышку, найти соленоид, счистить липкий слой спиртом и наклеить тонкую полоску малярного скотча или изоленты.
- **Задубевшая резина ролика.** Верхний слой резины глянцевеет и перестаёт цеплять бумагу.
  - *Решение.* Протрите ролик жидкостью для восстановления резины (Platenclene) или временно спиртом, а стопку бумаги перед загрузкой «распушите» веером.

**4. Принтер пишет «Замятие бумаги», хотя внутри пусто**

Фантомная ошибка, возникающая при абсолютно чистом тракте прохождения листа.

- **Заклинивший флажок датчика (оптопары).** На пути движения бумаги стоят крошечные пластиковые рычажки. Если пружинка соскочила или намоталась бумажная пыль, датчик остаётся перекрытым.
  - *Решение.* С фонариком осмотрите тракт (особенно в зоне печки) и аккуратно пошевелите каждый пластиковый язычок — он должен свободно возвращаться назад.

**5. Сканер «видится» системой, но не сканирует**

Устройство подключено, драйвер стоит, но при запуске процесс зависает или выдаёт ошибку «Scanner not found».

- **Не снята транспортная блокировка.** В планшетных сканерах есть механический переключатель, фиксирующий каретку. В положении «Lock» сканер будет гудеть, но не поедет.
  - *Решение.* Найдите на корпусе рычажок с иконкой замка, переведите в положение «Unlock» и перезапустите питание.
- **Просадка питания по USB.** Сканеры чувствительны к длине кабеля и пассивным разветвителям.
  - *Решение.* Подключите кабель напрямую в разъём на задней панели системного блока (минуя переднюю панель и USB-хабы).
- **Сбой драйвера после обновления Windows.** Встречается после крупных апдейтов системы (например, Windows 11 24H2).
  - *Решение.* Удалите устройство вместе с драйвером через «Диспетчер устройств», перезагрузите ПК и поставьте официальный драйвер с сайта производителя.

**6. Монитор пишет «Нет сигнала», хотя компьютер включён**
Вентиляторы в системном блоке крутятся, но экран чёрный, а индикатор монитора горит оранжевым.

- **Кабель воткнут в материнскую плату вместо видеокарты.** Если в ПК установлена дискретная видеокарта, встроенные видеовыходы на материнской плате (рядом с USB) обычно отключены.
  - *Решение.* Переставьте кабель в разъёмы видеокарты (расположены ниже, горизонтально).
- **Сбился источник сигнала на мониторе.** Экран ждёт сигнал с HDMI, а кабель подключён в DisplayPort.
  - *Решение.* Кнопками на корпусе монитора откройте меню «Source / Input» и вручную выберите активный порт.
- **Диагностика методом исключения.** Оранжевый индикатор означает, что сам монитор исправен, но не видит сигнала. Подключите его к ноутбуку или другому ПК: если картинка есть — проблема в кабеле, видеокарте или оперативной памяти системного блока.

**7. Мышь или клавиатура «отваливаются»**
Курсор дёргается, клавиши пропускают нажатия или устройство периодически пропадает.

- **Энергосбережение USB-портов.** Windows может отключать питание портов для экономии энергии.
  - *Решение.* Откройте «Диспетчер устройств» → «Контроллеры USB» → свойства «Корневого USB-концентратора» → вкладка «Управление электропитанием» → снимите галочку «Разрешить отключение этого устройства для экономии энергии».
- **Подсевшие батарейки или помехи радиоканала.** У беспроводных моделей разряд часто проявляется не полным отключением, а рывками курсора и задержкой ввода.
  - *Решение.* Замените элементы питания и переставьте USB-приёмник из задней панели (где он экранируется металлическим корпусом) на переднюю панель или через удлинитель.

**8. Пропал звук, хотя колонки и наушники подключены**
Иконка громкости на месте, но звука нет.

- **Переключилось устройство вывода по умолчанию.** После подключения монитора по HDMI или гарнитуры Windows часто отправляет звук на монитор без встроенных динамиков.
  - *Решение.* Кликните на иконку динамика в трее и вручную переключите вывод на «Динамики (Realtek Audio)».
- **Остановилась служба Windows Audio.**
  - *Решение.* Нажмите `Win + R`, введите `services.msc`, найдите «Windows Audio», перезапустите службу и выставите тип запуска «Автоматически».

**9. Компьютер включается на секунду и гаснет (или не реагирует на кнопку)**

Вентиляторы дёргаются и останавливаются, либо система вообще не подаёт признаков жизни.

- **Не подключено питание процессора.** 4- или 8-контактный кабель CPU в верхней части материнской платы не вставлен до щелчка.
- **Залипла кнопка корпуса.** Проверить можно, отключив разъём `Power SW` от материнской платы и кратковременно замкнув два его контакта плоской отвёрткой. Если ПК стартовал — проблема в кнопке корпуса.
- **Проверка блока питания «методом скрепки».** Отключите БП от всех комплектующих и замкните зелёный провод с любым чёрным на широком 24-pin разъёме. Если вентилятор БП не закрутился — блок питания неисправен.

**Правило диагностики:**

При любой непонятной поломке двигайтесь от простого к сложному по правилу трёх шагов:

1. **Физика:** проверьте питание, кабели, переключатели, наличие бумаги/батареек.
2. **Перезапуск:** перезагрузите устройство, службу в Windows или переткните порт.
3. **Исключение:** подключите заведомо исправный кабель, другой порт или соседний ПК, чтобы точно понять, на чьей стороне проблема.

Перепрошивка роутера:

В зависимости от модели роутера, сделать это можно двумя способами: **на штатной прошивке** (если производитель уже встроил нужные протоколы) или **через перепрошивку на альтернативную систему (OpenWrt / Padavan)**.

**VPN прямо в роутере: когда достаточно штатной прошивки, а когда пора ставить OpenWrt**
Настраивать **VPN** на каждом устройстве отдельно неудобно, а на **Smart TV**, игровые консоли или умные колонки клиент зачастую вообще не поставить. Решение — поднять клиент прямо на домашнем роутере. Разбираемся, как это реализуется на практике, какие протоколы актуальны и как не превратить роутер в «кирпич» при перепрошивке.
**Проверяем, нужна ли вообще перепрошивка**

Прежде чем лезть в загрузчик роутера, проверьте возможности его заводской прошивки. Многие современные модели умеют работать с VPN «из коробки»:

- **Keenetic (Кинетик).** Лидер по удобству. Поддерживает **WireGuard**, **OpenVPN**, **IPsec** и **SSTP** прямо в веб-интерфейсе, а через установку пакетов Entware позволяет настроить современные протоколы (**VLESS**, **Trojan**, **Shadowsocks**) и выборочную маршрутизацию (когда через VPN идут только нужные сайты, а остальной трафик идёт напрямую).
- **ASUS (с прошивкой Asuswrt или Asuswrt-Merlin).** Отлично работает с **OpenVPN** и **WireGuard** на уровне системы, позволяет гибко делить трафик по устройствам.
- **Xiaomi, TP-Link, Mercusys, D-Link.** В стоковых прошивках бюджетных моделей обычно есть только устаревшие PPTP/L2TP (в лучшем случае OpenVPN), которые часто режутся провайдерами или сильно режут скорость. Вот здесь на помощь приходит перепрошивка.

**Этап 1. Подготовка (что скачать и как подключиться)**

1. **Подключитесь кабелем.** Возьмите сетевой кабель (патч-корд): один конец вставьте в сетевую карту ПК/ноутбука, второй — в жёлтый **LAN-порт** роутера. По Wi-Fi прошивать роутер категорически нельзя.
2. **Скачайте файл прошивки.** Зайдите в базу устройств **OpenWrt**, найдите свою модель и ревизию (написана на наклейке снизу корпуса, например `Ver: 2.0`). Скачайте файл, в названии которого есть слово `factory` (обычно с расширением `.bin` или `.img`).
3. **Откройте админку роутера.** Запустите любой браузер (Chrome, Edge, Firefox), в адресной строке сверху введите `192.168.0.1` или `192.168.1.1` и нажмите `Enter`. Введите логин и пароль от роутера (по умолчанию часто `admin` / `admin` или указан на наклейке снизу).

**Этап 2. Заливка прошивки через браузер**

1. В меню роутера найдите раздел **«Системные инструменты» (System Tools)** → **«Обновление встроенного ПО» (Firmware Upgrade)**.
2. Нажмите кнопку **«Обзор» (Browse / Choose File)** и выберите скачанный ранее файл `...factory.bin`.
3. Нажмите кнопку **«Обновить» (Upgrade)**.
4. **Ничего не трогайте 3–5 минут.** Появится полоса загрузки, затем роутер сам перезагрузится. Не выключайте его из розетки и не закрывайте вкладку, пока индикаторы на корпусе не стабилизируются.

**Этап 3. Первый вход в OpenWrt (интерфейс LuCI)**

После перезагрузки интерфейс роутера полностью изменится.

1. В адресной строке браузера введите `192.168.1.1` и нажмите `Enter`.

2. Откроется окно входа в **OpenWrt** (LuCI). Поле пароля при первом запуске пустое — просто нажмите синюю кнопку **«Login»**.
3. Сразу задайте пароль: в верхней панели нажмите **System** → **Administration**. Впишите свой пароль в поля *Password* и *Confirmation*, прокрутите вниз и нажмите **Save & Apply**.

**Этап 4. Подключение интернета на новой прошивке**

Чтобы роутер мог скачать пакеты для VPN, ему нужен выход в сеть:

1. Вставьте кабель провайдера в синий **WAN-порт** роутера.
2. Если провайдер выдаёт интернет автоматически (DHCP), интернет появится сразу. Если нужен логин/пароль (PPPoE):
   - Откройте в верхнем меню **Network** → **Interfaces**.
   - Напротив строки **WAN** нажмите синюю кнопку **Edit**.
   - В выпадающем списке *Protocol* выберите **PPPoE**, нажмите **Switch protocol**.
   - Введите логин и пароль от провайдера в поля *PAP/CHAP username* и *password*, нажмите **Save**, а затем **Save & Apply**.

**Этап 5. Установка и включение VPN (на примере WireGuard)**

Теперь превращаем роутер в VPN-клиент прямо через меню в браузере:

1. В верхнем меню откройте **System** → **Software**.
2. Нажмите зелёную кнопку **Update lists** (обновить список доступных программ) и дождитесь завершения.
3. В поле **Filter** (фильтр) введите `wireguard` и установите по очереди два пакета (нажимая кнопку **Install** напротив каждого):
   - `wireguard-tools`
   - `luci-proto-wireguard`
4. После установки перезагрузите роутер: **System** → **Reboot** → **Perform reboot**.
5. Снова зайдите на адрес через браузер `192.168.1.1`, перейдите в **Network** → **Interfaces** и нажмите кнопку **Add new interface...** (внизу страницы).
6. В окне создания интерфейса:
   - В поле *Name* напишите `wg0`.
   - В списке *Protocol* выберите **WireGuard VPN**.
   - Нажмите **Create interface**.
7. В открывшихся настройках нажмите кнопку **Load configuration...** (Загрузить конфигурацию), вставьте текст из вашего готового `.conf`-файла от VPN-провайдера и нажмите **Import settings**.
8. Перейдите на вкладку **Firewall Settings** (внутри этого же окна) и в пункте *Create / Assign firewall-zone* выберите красную зону **`wan`**. Нажмите **Save**, а затем **Save & Apply**.

Готово — после применения настроек весь трафик всех устройств, подключённых к роутеру по проводу или Wi-Fi, пойдёт через защищённый канал.

**1. Active Directory (AD) — «паспортный стол и охрана» офиса**

Представьте, что в компании работает 150 человек, и у каждого свой компьютер. Если не использовать **AD**, сисадмину придётся подходить к каждому из 150 столов, чтобы создать пользователю учётную запись, поменять забытый пароль или запретить сотрудникам устанавливать игры.

**Active Directory (AD)** — это единая база данных всех людей, компьютеров и принтеров в компании, которая крутится на главном сервере (он называется **контроллером домена**).

**Зачем туда лезть на практике (жизненные примеры):**

- **Единый вход (SSO).** Сотрудник может сесть за любой свободный компьютер в офисе, ввести свой логин и пароль — и сразу увидит свой рабочий стол, почту и сетевые папки.
- **Увольнение за 10 секунд.** Если увольняется менеджер по продажам, админ заходит в **AD**, нажимает правую кнопку на его фамилии → *«Отключить учётную запись»*. В ту же секунду у человека блокируется доступ к компьютеру, 1С, корпоративной почте и общим папкам.
- **Управление правами (Группы безопасности).** Вместо того чтобы настраивать доступ к папке «Бухгалтерия» для каждого бухгалтера по отдельности, в **AD** создают группу `Buhgalteria`, добавляют туда сотрудников и дают права всей группе сразу.

**2. Групповые политики (GPO) — «пульт управления» всеми компьютерами**

Если **AD** — это список сотрудников, то **GPO (Group Policy Objects)** — это правила поведения, которые сервер принудительно рассылает на все компьютеры в сети.

**Что админ делает через GPO в реальной жизни:**

- **Подключает сетевые диски и принтеры.** Вы настроили правило один раз — и при входе в систему у всех сотрудников бухгалтерии автоматически появляется сетевой диск `Z:` и нужный принтер на этаже.
- **Запрещает лишнее.** Можно одной галочкой отключить флешки на компьютерах (чтобы не слили базу клиентов), запретить менять обои рабочего стола или заблокировать установку программ без пароля администратора.
- **Настраивает безопасность.** Заставить всех сотрудников менять пароль раз в 90 дней и блокировать экран, если человек отошёл от компьютера на 5 минут.

**3. Принцип работы веб-серверов (`IIS`, `Apache`, `Nginx`)**

В вакансиях часто пугают этими названиями, но суть у них одна. **Веб-сервер** — это программа, которая принимает запросы от браузера (или программы на компьютере пользователя) и отдаёт нужную страницу или данные.

- **Жизненный пример с 1С.** Раньше, чтобы бухгалтер зашёл в 1С, ему нужно было ставить тяжёлую программу на его личный компьютер. Сейчас админ ставит 1С на мощный сервер, включает там веб-сервер (**IIS** от Microsoft или **Apache** для Linux) и «публикует» базу. Теперь бухгалтер просто открывает Chrome, вводит адрес `http://1c-server/buh` и работает в 1С прямо в браузере.
- **В чём разница между ними:**
  - **IIS (Internet Information Services).** Встроенный веб-сервер внутри Windows Server. Включается парой галочек в диспетчере серверов. Чаще всего используется в связке с Windows-серверами и 1С.
  - **Apache.** Классический, очень гибкий веб-сервер (чаще всего работает на Linux). Отлично переваривает сложные настройки и публикации 1С на Linux.
  - **Nginx.** Сверхбыстрый веб-сервер и «регулировщик» (обратный прокси). Его часто ставят перед Apache или серверами приложений: он принимает на себя первый удар от тысяч пользователей, раздаёт лёгкие картинки и равномерно распределяет сложные запросы между серверами.

- **FS (File Server / Файловый сервер) — «общий шкаф с документами».** Сервер с большими жёсткими дисками, где лежат общие папки отделов.
  - *В жизни.* У бухгалтерии своя папка, у конструкторов — своя с чертежами. Сотрудники сохраняют файлы туда, а не на рабочий стол, чтобы ничего не пропало, если сгорит компьютер, и чтобы можно было удобно пересылать ссылки коллегам.
- **GPO (Групповые политики) — «пульт управления офисом».** Правила, которые сервер автоматически применяет ко всем компьютерам.
  - *В жизни.* Чтобы не бегать к 100 компьютерам с флешкой, вы один раз настраиваете политику: «Всем сотрудникам склада подключить сетевой диск Z: и запретить менять заставку на рабочем столе».
- **Print Server (Сервер печати) — «диспетчер принтеров».** Служба, которая управляет всеми офисными принтерами и МФУ из одного окна.
  - *В жизни.* В офисе купили новый сетевой принтер. Вы добавляете его на сервер печати один раз, и он сам появляется в списке принтеров у нужного отдела без ручной установки драйверов на каждом ПК.
- **TS (Terminal Server / Терминальный сервер) — «мощный компьютер для всех».** Сервер, к которому сотрудники подключаются по удалённому рабочему столу (RDP) и работают внутри него.

- *В жизни.* У менеджера в цеху стоит слабый, старый компьютер, который еле тянет браузер. Но он нажимает значок «Удалённый рабочий стол», заходит на мощный сервер TS — и спокойно работает в тяжёлой 1С, потому что все вычисления идут на сервере, а на экран менеджера просто передаётся картинка.

**Виртуализация (`Microsoft Hyper-V`)**
В вакансии: *«Развертывание и настройка Microsoft Hyper-V»*.

- **Что это простыми словами.** Это программа, которая позволяет внутри одного очень мощного физического сервера создать 5 или 10 независимых виртуальных компьютеров (виртуальных машин).
- **Жизненный пример.** Раньше, если нужно было запустить 1С, почту и сайт, приходилось покупать три отдельных железных сервера. С Hyper-V вы покупаете один мощный сервер и «нарезаете» его на виртуальные машины: на одной крутится 1С, на второй — контроллер домена AD, на третьей — тестовый Linux. Если одна виртуальная машина «сломалась» после кривого обновления, остальные продолжают работать, а сломанную можно за 2 минуты откатить назад из снимка (чекпоинта).

**Сети на оборудовании `MikroTik` (`EoIP`, `VPN`, маршрутизация)**
В вакансии: *«Обязателен опыт построения сетей на оборудовании Mikrotik (Eoip, VPN, маршрутизация сетей)»*.
**MikroTik** — это профессиональные роутеры с огромным количеством настроек, которые обожают в малом и среднем бизнесе за надёжность и цену.

- **Маршрутизация сетей.** Умение объяснить роутеру, куда отправлять пакеты данных.
  - *В жизни.* У вас на заводе есть офисная сеть (например, `192.168.1.x`) и сеть станков в цеху (`192.168.2.x`), а ещё два интернет-провайдера (основной и резервный). Вы настраиваете MikroTik так, чтобы бухгалтерия не видела станки, а если у основного провайдера порвётся кабель — весь офис за 3 секунды без обрыва связи переключился на резервный интернет.
- **VPN и EoIP (Ethernet over IP).** «Невидимый кабель» через обычный интернет.
  - *В жизни.* У компании есть главный офис в Санкт-Петербурге и склад во Всеволожском районе. Чтобы сотрудники склада могли печатать на офисном принтере и заходить в общую 1С так, будто они сидят в соседнем кабинете, админ настраивает между двумя роутерами MikroTik защищённый туннель (**VPN** или фирменный протокол **EoIP**). Для компьютеров это выглядит так, будто между офисом и складом протянули обычный сетевой провод.

**IP-телефония**

В вакансии: *«IP телефония»*.

- **Что это простыми словами.** Офисные телефоны, которые работают не по старой телефонной лапше, а через обычный интернет-кабель (или программу на компьютере/смартфоне).
- **Жизненный пример.** Клиент звонит на единый городской номер компании. Виртуальная или офисная **АТС** (например, **Asterisk** или облачная **АТС**) проигрывает приветствие *«Нажмите 1 для отдела продаж»*, переводит звонок на свободного менеджера, записывает разговор в **CRM** и присылает уведомление, если звонок пропустили. Задача админа — подключить новый телефонный аппарат на стол сотруднику, настроить голосовое меню и следить, чтобы в разговоре не «квакал» голос (настроить приоритет голосового трафика на роутере).

**Сервер 1С с нуля и базовые доработки**

В вакансии: *«Настройка сервера 1С с нуля (установка платформы, ключей, настройка SQL, публикация в Web) и базовый опыт разработки (обработки, печатные формы)»*.

Здесь от админа ждут не глубокого программирования, а умения «поднять и обслуживать» 1С в небольшой компании, где нет отдельного штата 1С-ников:

- **Установка платформы, ключей и SQL.**
Поставить серверную программу **1С**, активировать лицензии (программные пин-коды или USB-ключ защиты) и связать её с базой данных **Microsoft SQL Server**, где физически хранятся все цифры.
- **Публикация баз в Web.** Настроить веб-сервер (IIS или Apache), чтобы менеджеры могли открывать **1С** просто в браузере по ссылке без установки программы на компьютер.
- **Обновления, бэкапы и мелкие доработки.** Регулярно делать резервные копии (чтобы при сбое не потерять бухгалтерию за год), обновлять конфигурации при изменении законов и уметь поправить мелочь — например, добавить логотип компании или лишнюю колонку в печатную форму накладной.

**Связываем офис и склад через MikroTik (VPN + EoIP)**

**Жизненная ситуация**

У вашей компании есть **главный офис в центре города** (там стоит сервер 1С и сетевой принтер) и **склад за городом** (там сидят два кладовщика с ноутбуками и принтером этикеток).

Руководитель подходит и говорит: *«Сделай так, чтобы кладовщики со склада могли заходить в нашу офисную 1С, кидать накладные на*

*наш офисный принтер, и чтобы всё это было безопасно, а не торчало в открытый интернет».*

**Как размышляет сисадмин (логика решения)**

1. **В чём проблема?** Компьютеры в офисе живут в своей локальной сети (например, `192.168.1.0/24`), а компьютеры на складе — в своей (`192.168.2.0/24`). Между ними — обычный интернет. Прямой кабель через весь город не протянешь.
2. **Какое решение?** Нужно проложить через интернет **зашифрованный «невидимый кабель» (VPN-туннель)** между роутером MikroTik в офисе и роутером MikroTik на складе.
3. **Зачем здесь нужен `EoIP`, о котором писали в вакансии?** Обычный VPN связывает две разные сети через маршрутизацию (офис — одна сеть, склад — другая). Но иногда старые программы или сетевые принтеры «видят» друг друга только если находятся **в одной общей локальной сети** (как будто воткнуты в один свитч в соседних комнатах). Протокол **EoIP (Ethernet over IP)** в MikroTik как раз создаёт иллюзию такого прямого кабеля поверх интернета.
4. **Что нужно для настройки?** В офисе должен быть «белый» (статический) IP-адрес от провайдера, чтобы склад знал, куда стучаться.

# ПРОГРАММА «МЕРКУРИЙ»

**Урок 1. Зачем нужен «Меркурий» и как он устроен (Базовая логика)**

Представьте систему «Меркурий» (ФГИС «ВетИС») как **единый цифровой паспортный стол для еды животного происхождения**. Раньше к каждой фуре с мясом, молоком или рыбой прикладывали стопку бумажных справок с печатями ветеринара. Их легко было потерять или подделать. Теперь всё переехало в интернет, и государство видит путь каждого куска сыра или пачки масла: **от фермы → до завода → до оптового склада → до полки магазина или кухни кафе**.

**4 главных понятия, на которых держится вся система:**

- **Подконтрольная продукция.** Это всё, что имеет животное происхождение: мясо, рыба, молоко, творог, яйца, мёд, колбаса, сливочное масло, а также корма для животных. *(Овощи, хлеб или крупы в «Меркурии» не учитываются).*
- **ХС (Хозяйствующий субъект).** Это само юридическое лицо или ИП (например, ООО «Ромашка»).
- **Площадка.** Это конкретный физический адрес, куда едет товар (например: «Магазин №1 на ул. Ленина» или «Склад в промзоне»). У одного ХС может быть хоть 100 площадок.

- **эВСД (Электронный ветеринарный сопроводительный документ).** Главный документ в системе. Без него товар считается нелегальным. Когда поставщик отгружает вам коробку сосисок, он создает у себя «исходящий ВСД», а у вас в системе он появляется как **«входящий ВСД»**.

**Главное действие новичка: «Гашение ВСД»**
В жизни это выглядит так: к вам приехала машина с товаром, вы сверили коробки с бумажной накладной, убедились, что всё в порядке, и расписались в получении.
В «Меркурии» эта роспись называется **гашением**.

1. Поставщик отправил товар →у документа статус **«Оформлен»** (товар как бы едет в пути).
2. Машина приехала, вы проверили товар → вы нажимаете в системе кнопку **«Погасить»**.
3. Документ меняет статус на **«Погашен»**. В этот момент товар официально списывается со склада поставщика и попадает на ваш виртуальный склад («Журнал входной продукции»).

 **Важное правило (Ловушка новичков):** По закону погасить ВСД нужно в течение **24 часов** (одного рабочего дня) с момента доставки товара. Если просрочить — система зафиксирует нарушение, а за регулярные просрочки учетную запись могут заблокировать.

**Практика №1: Ваша первая приемка**
Представьте, что вы работаете приемщиком в небольшом кафе «Уют». Утром экспедитор привез вам **10 кг охлажденной куриной грудки** от поставщика «Птицефабрика Заря».
Вы открыли программу «Меркурий.ХС», зашли во вкладку **«Ветеринарные документы» → «Входящие»** и увидели новый документ в статусе *«Оформлен»*.
**Что вы должны проверить глазами (сверяя коробку и экран компьютера), прежде чем нажать кнопку «Погасить»?**

**Перед нажатием кнопки «Погасить» приемщик должен убедиться, что виртуальный паспорт в «Меркурии» на 100% совпадает с реальной коробкой на столе.** Если нажать кнопку вслепую, вы официально примете чужие ошибки на свой баланс и получите штраф при первой же проверке.
**Пошаговый алгоритм проверки и рассуждения приемщика**

1. **Сверка наименования и объема (веса)**
   - **Что делаем.** Смотрим на этикетку коробки и сравниваем название товара и его вес с тем, что написано в карточке ВСД.

- **Рассуждение.** Поставщик мог перепутать машины и привезти вам 10 кг бедра вместо грудки, либо недоложить 2 кг. Если погасить документ на 10 кг при фактических 8 кг, у вас на виртуальном складе зависнут несуществующие 2 кг («воздух»).

2. **Проверка сроков годности и даты выработки**
   - **Что делаем.** Находим на заводской упаковке дату производства и срок годности, а затем сверяем их с графами «Дата выработки» и «Годен до» на экране.
   - **Рассуждение.** Иногда на складе поставщика кладут свежую партию, а документ по ошибке выписывают на вчерашнюю, у которой срок заканчивается завтра. Система «Меркурий» автоматически следит за сроками, и как только виртуальный срок выйдет, она заблокирует работу с этой партией.
3. **Контроль целостности упаковки и условий хранения**
   - **Что делаем.** Осматриваем коробки на предмет вскрытия, помятостей и проверяем температуру в кузове (для охлажденной курицы это от 0 до +2 °C).
   - **Рассуждение.** Если курица приехала теплой или упаковка разорвана, такой товар нельзя принимать на кухню. Погасив **ВСД**, вы подтверждаете, что товар безопасен и соответствует всем ветеринарным нормам.
4. **Сверка номера транспортного средства**
   - **Что делаем.** Смотрим в разделе «Транспорт» номер автомобиля, который указан в **ВСД**, и сравниваем его с номером фуры у разгрузочной зоны.
   - **Рассуждение.** По правилам прослеживаемости система должна знать, в какой именно машине ехал продукт. Если номера не совпадают, это считается нарушением правил перевозки (хотя на практике многие приемщики закрывают на это глаза, формально это повод для претензии).

**Итог: как выглядит само гашение в программе**

Если все 4 пункта сошлись без нареканий, мы нажимаем кнопку **«Погасить»** в самом низу электронного документа. Система попросит указать **фактическую дату поступления** (обычно ставится сегодняшняя автоматически) и нажать **«Сохранить»**. После этого партия курицы официально встает на ваш виртуальный склад.

## (Урок 2): А что, если нашлось расхождение?

В реальной жизни идеальная приемка бывает не всегда. Представьте, что при проверке той же партии курицы вы обнаружили одну из двух ситуаций:

- **Ситуация А (Полный отказ).**
- Вся партия курицы испорчена (плохой запах, нарушена температура), и вы отправляете её обратно поставщику.

- **Ситуация Б (Частичная приемка).**
- По документам едет 10 кг, а по факту привезли только 8 кг (одну коробку забыли на складе), но эти 8 кг отличного качества и нужны кухне прямо сейчас.

В обоих случаях нажимать обычную кнопку «Погасить» нельзя, иначе вы примете на себя лишний или испорченный товар. Для этого в «Меркурии» при гашении есть специальная галочка — **«Разделить партию»** (оформление возвратного ВСД).

**При расхождениях в «Меркурии» НЕЛЬЗЯ просто нажать «Погасить» — нужно оформить ВОЗВРАТНЫЙ эВСД.** От того, отказываетесь ли вы от всей машины или забираете только часть коробок, зависит порядок действий в программе.

**Ситуация А. Полный возврат всей партии (брак или пересорт)**

Представьте, что курьер привез вам порванный пакет с протекшим молоком. Вы не забираете его на кухню, а сразу отдаете обратно водителю, чтобы он увез товар на склад.

- **Откройте входящий ВСД и нажмите «Погасить».** Даже при полном отказе от товара документ нужно обработать в системе, чтобы официально отправить груз назад.
- **Поставьте галочку «Разделить партию».** Эта функция открывает доступ к редактированию фактического веса, который вы принимаете.
- **Укажите фактический объем 0.** Система сама перенесет все 10 кг из графы приемки в графу возвратного объема.
- **Выберите причину возврата.** В списке укажите реальную причину отказа, например нарушение температурного режима или истекающий срок годности.
- **Нажмите «Сохранить».** На ваш виртуальный склад зачислится 0 кг, а поставщику автоматически придет возвратный ВСД на 10 кг.

**Ситуация Б. Частичная приемка (привезли меньше, чем в документах)**

Это похоже на ситуацию, когда по чеку должно быть 10 пачек масла, а в коробке лежит только 8. Вы забираете 8 пачек в работу, а за 2 недостающих оформляете возврат.

- **Нажмите «Погасить» и выберите «Разделить партию».** Так мы разделим один виртуальный документ на две независимые части.
- **Впишите реальный вес в поле «Фактический объем».** Укажите 8 кг, после чего программа сама рассчитает остаток в 2 кг для возврата поставщику.

- **Укажите причину расхождения.** Выберите пункт «Недостача» или «Расхождение по весу», чтобы проверяющие органы видели причину изменения объема.
- **Подтвердите гашение.** На ваш баланс встанут ровно 8 кг курицы, а виртуальные 2 кг вернутся на склад поставщика.

**Главная ловушка на практике:** «Меркурий» отвечает только за ветеринарный контроль, а не за деньги. После оформления возврата в программе обязательно вычеркните непринятый товар в бумажной накладной водителя и составьте акт расхождения (ТОРГ-2), иначе бухгалтерия выставит вам счет за все 10 кг.

## (Урок 3). Что делать с товаром после приемки?

После нажатия кнопки «**Погасить**» ваши 8 кг курицы попадают во вкладку **«Журнал продукции»** → **«Входная продукция»**.
Если вы работаете в магазине или кафе, списывать каждую проданную порцию в «Меркурии» не нужно. Однако со временем на виртуальном складе накапливаются «виртуальные остатки» (товар давно съели или продали, а в программе он всё ещё числится). Для наведения порядка используют **Инвентаризацию**.

Представьте ваш домашний холодильник. В течение недели вы покупали молоко, творог и курицу, складывали их на полки и записывали в блокнот. К выходным вы всё съели, холодильник пуст, но в блокноте всё ещё числится 5 литров молока и 3 килограмма курицы.
В «**Меркурии**» происходит то же самое. Когда вы гасите входящие ВСД, товар попадает в ваш виртуальный холодильник — **«Журнал входной продукции»**. Если вы работаете в магазине или кафе, вы не обязаны списывать каждую проданную пачку масла на кассе. Из-за этого в журнале скапливаются «мертвые» остатки — товар, которого давно нет в реальности, особенно с истекшим сроком годности.
Для наведения порядка в этом журнале раз в неделю или раз в месяц проводят **инвентаризацию**.

### Пошаговый разбор: как списать проданный или просроченный товар

Допустим, прошёл месяц, и в вашем журнале висят 8 кг курицы и 20 пачек творога, которые вы уже давно продали посетителям.

1. **Заходим в раздел «Инвентаризация»**

- **Что делаем.** В левом меню «**Меркурий.ХС**» открываем вкладку **«Инвентаризация»** и нажимаем кнопку **«Добавить»** (создать новую заявку).
- **Рассуждение.** Мы открываем АКТ СПИСАНИЯ, в который сейчас сложим все позиции, требующие корректировки.

2. **Выбираем тип операции и причину**
   - **Что делаем.** В выпадающем списке причин выбираем **«Реализация в розницу»** (если товар продали покупателям или приготовили из него блюда) либо **«Истечение срока годности»** (если товар реально испортился и вы его выкинули).
   - **Рассуждение.** Очень важно НЕ путать причины. Если списать 100 кг мяса по причине «**Порча**», у инспектора возникнет вопрос: куда вы дели 100 кг тухлого мяса без договора на утилизацию. А причина «**Реализация в розницу**» объясняет, что товар спокойно ушел людям.
3. **Добавляем записи из журнала и обнуляем остатки**
   - **Что делаем.** Нажимаем «Добавить» внутри инвентаризации, находим в списке нашу курицу и творог, а в поле **«Фактический объем»** ставим **0** (или реальный остаток, если пара пачек ещё лежит на витрине).
   - **Рассуждение.** Система сравнивает графу «По документам (8 кг)» и графу «По факту (0 кг)». Разницу в 8 кг она автоматически помечает на списание.
4. **Оформляем (проводим) инвентаризацию**
   - **Что делаем.** Проверяем список и нажимаем кнопку **«Оформить»**.
   - **Рассуждение.** Как только документ проведен, виртуальные остатки в журнале обнуляются. Теперь ваш цифровой склад чист и соответствует реальному холодильнику.

**Две главные ловушки при инвентаризации**

- **Ловушка «Прибавить из воздуха».** Через инвентаризацию можно уменьшить остаток (списать), но **НЕЛЬЗЯ** добавить 50 кг мяса, на которое у вас нет входящего **ВСД**. Если увеличить вес в инвентаризации без документов, система выдаст предупреждение о «нелегальной продукции без происхождения», а вашу учетную запись заблокируют на срок до 6 месяцев.
- **Ловушка «Забытая просрочка».** Если в журнале висит товар с истекшим сроком годности и вы его НЕ списали, автоматические алгоритмы Россельхознадзора видят это как «торговлю просрочкой». Чистить просроченные позиции нужно до того, как выйдет дата «Годен до».

## (Урок 4): Перемещение товара и возврат поставщику после приемки

Мы уже научились принимать товар, отказывать водителю прямо у двери и чистить склад. Но бывают более сложные ситуации:

1. Вы приняли товар вчера, погасили ВСД, а сегодня открыли коробку — внутри брак, и поставщик согласен забрать его обратно. Здесь галочка «Разделить партию» уже не сработает, нужно оформлять **исходящий транспортный ВСД**.
2. У вас два магазина (две площадки), и нужно перевезти 5 кг сыра из первого магазина во второй.

Представьте, что вы купили в магазине упаковку сосисок, принесли домой, убрали в холодильник и выкинули чек. На следующий день вы открыли пачку, а сосиски оказались испорченными. Чтобы вернуть их в магазин, вы сами выступаете отправителем: берете пачку и везете её обратно продавцу.
В «**Меркурии**» работает точно такой же принцип. Как только вы нажали кнопку **«Погасить»**, товар официально стал вашим. Если вы хотите отправить его обратно поставщику на следующий день или перевезти со своего склада в свой же магазин, нужно оформить **исходящий транспортный ВСД** — то есть выписать цифровой паспорт на дорогу.

### Пошаговый алгоритм оформления исходящего ВСД

Разберем на примере: у вас есть склад «Центральный» и кафе «Уют» (или вы возвращаете поставщику 5 кг бракованного сыра).

1. **Заходим в «Транзакции» и создаем новую отправку**
   - **Что делаем.** В левом меню выбираем раздел **«Транзакции»** → **«Добавить»**.
   - **Рассуждение.** Любое движение товара с вашей площадки наружу (продажа, возврат поставщику, перевозка на другую вашу точку) оформляется через транзакцию.
2. **Заполняем «Шапку» документа (Куда и на чём едет груз)**
   - **Тип транзакции.** Выбираем **«Перевозка без смены владельца»** (если везем сыр со своего склада в свое кафе) или **«Перевозка со сменой владельца»** (если возвращаем товар поставщику или продаем другому юрлицу).
   - **Получатель.** Находим по ИНН или названию фирму-получателя и обязательно выбираем **конкретную площадку** (адрес, куда приедет машина).

- **Транспорт.** Указываем тип транспорта (например, «Автомобильный»), госномер машины и способ хранения при перевозке (например, «Охлаждаемый»).
- **Рассуждение.** Без точного адреса получателя и номера машины система не выпустит документ — она должна знать маршрут груза от двери до двери.

3. **Выбираем товар из своего «Журнала входной продукции»**
   - **Что делаем.** Нажимаем кнопку **«Добавить»** в блоке «Продукция», находим наш сыр на виртуальном складе и вписываем объем отправки — **5 кг**.
   - **Рассуждение.** Вы не придумываете товар заново, а отщипываете 5 кг от той самой партии, которую погасили ранее. Вся история происхождения (от какой коровы и с какого завода этот сыр) подтянется автоматически.
4. **Указываем цель отправки и оформляем документ**
   - **Что делаем.** В поле «**Цель**» выбираем **«Реализация в пищу людям»** (для перемещения в кафе) или **«Возврат поставщику»** (если возвращаем брак). Нажимаем **«Оформить»**.
   - **Рассуждение.** В эту секунду с вашего виртуального склада спишутся 5 кг сыра, а у получателя во вкладке «Входящие» появится новый ВСД в статусе «***Оформлен***».

**Две важные ловушки при оформлении исходящих ВСД**

- **Ловушка «Возврат просрочки».** Если срок годности сыра уже истек (сегодня 17 ноября, а срок вышел 16 ноября), система **заблокирует** отправку такого товара обратно поставщику для реализации. Просроченную продукцию можно отправить только на утилизацию на специальный завод, поэтому возврат брака нужно успеть оформить до конца срока годности.
- **Ловушка «Не та площадка».** У крупных поставщиков или сетей могут быть десятки складов по всей стране. Если выбрать в получателях юридический адрес офиса вместо реального склада возврата, водитель привезет товар, а кладовщик не сможет погасить документ на своей площадке. Всегда уточняйте у контрагента **GUID (уникальный код) площадки**.

Представьте такую ситуацию:
*В понедельник вы приняли и погасили 20 кг сливочного масла (срок годности до пятницы). В среду повара заметили, что 5 кг масла имеют горький привкус (заводской брак), и поставщик согласился забрать эти 5 кг обратно. Остальные 15 кг повара уже израсходовали на выпечку.*

Чтобы на виртуальном складе остался ноль без нарушений, в среду нужно оформить два документа в строгой последовательности:

сначала возврат брака поставщику, а затем инвентаризацию для списания остатков.

**Пошаговый разбор решения кейса**

**Шаг 1. Возврат 5 кг брака поставщику (Исходящий транспортный ВСД)**

- **Почему делаем это первым.** Срок годности масла истекает в пятницу, а сегодня среда — значит, товар ещё не просрочен и его можно легально вернуть поставщику. Если по ошибке списать все 20 кг через инвентаризацию, на складе не останется виртуального объема для возврата.
- **Что нажимаем в системе.** Заходим в **«Транзакции» \(\rightarrow \) «Добавить»**, выбираем тип **«Перевозка со сменой владельца»**, указываем площадку поставщика и госномер машины.
- **Что указываем в грузе.** Выбираем из «Журнала входной продукции» нашу партию масла, ставим объем **5 кг** и цель **«Возврат поставщику»**. После нажатия «Оформить» на нашем складе останется **15 кг**.

**Шаг 2. Списание 15 кг использованного масла (Инвентаризация)**

- **Почему делаем это вторым.** Эти 15 кг повара уже пустили в готовку (реализовали), поэтому физически в холодильнике их нет, но в программе они всё ещё висят.
- **Что нажимаем в системе.** Переходим во вкладку **«Инвентаризация» \(\rightarrow \) «Добавить»**, выбираем причину **«Реализация в розницу»** (она же используется для списания сырья, ушедшего в блюда в общепите).
- **Что указываем в остатках.** Находим нашу запись с маслом (где числится 15 кг) и в поле **«Фактический объем»** ставим **0**. Нажимаем **«Оформить»** — виртуальный склад полностью обнуляется.

**«Какой инструмент выбрать в Меркурии»**

| Ситуация в жизни | Какой инструмент берем | Что происходит со складом |
|---|---|---|
| **Приехала машина с товаром, всё верно** | Входящий ВСД \(\rightarrow \) **Погасить** | Товар зачисляется на ваш склад. |
| **Привезли меньше или брак у двери** | Входящий ВСД \(\rightarrow \) **Погасить** + галочка **«Разделить партию»** | Принимается только хороший остаток, лишнее уходит назад. |
| **Товар продали или съели на кухне** | **Инвентаризация** (причина «Реализация в розницу») | Виртуальный остаток обнуляется (списывается). |

| | | |
|---|---|---|
| **Нужно вернуть принятый товар или перевезти на другую точку** | **Транзакции** (Исходящий транспортный ВСД) | Объем списывается у вас и едет на другую площадку. |

## (Урок 5): Производство (Переработка сырья в готовый продукт)

До сих пор мы работали по правилам **магазина или кафе** (купили → продали или приготовили). Но если ваше предприятие делает полуфабрикаты на продажу (например, купили 10 кг мяса и 2 кг специй, а на выходе получили 12 кг пельменей для отправки в магазины), в «Меркурии» включается **производственная транзакция** («Незавершенное производство» / «Переработка»).

Представьте, что вы на кухне взяли **10 кг куриного филе** и **2 кг сливочного масла**, пропустили через мясорубку и налепили **12 кг котлет**, чтобы продать их в соседний магазин.
Для «Меркурия» курица, масло и котлеты — это три разных товара со своими сроками годности. Чтобы легально отправить котлеты в магазин, нужно объяснить системе: *«Мы не взяли эти котлеты из воздуха — мы сделали их вот из этой конкретной курицы и вот из этого масла»*.
Такая операция называется **ПРОИЗВОДСТВЕННОЙ ТРАНЗАКЦИЕЙ** («Переработка / производство»).

### Как это работает: 4 шага производства

1. **Заходим в «Транзакции» и выбираем тип «Переработка / производство»**
   - **Что делаем.** Нажимаем **«Добавить»**, в поле «Тип транзакции» выбираем **«Переработка / производство»**.
   - **Рассуждение.** В отличие от перевозки, здесь товар никуда не уезжает — он превращается из одного вида в другой прямо на вашей площадке.
2. **Закладываем «Сырьё» (То, что уходит в мясорубку)**
   - **Что делаем.** В блоке **«Сырьё»** нажимаем «Добавить», открываем наш «Журнал входной продукции» и выбираем погашенные ранее **10 кг филе** и **2 кг масла**.

- **Рассуждение.** В этот момент система проверяет сроки годности сырья. Если масло просрочено хотя бы на один день, сделать из него котлеты программа не даст.

3. **Указываем «Вырабатываемую продукцию» (То, что получилось на выходе)**
   - **Что делаем.** В нижнем блоке **«Вырабатываемая продукция»** нажимаем «Добавить», выбираем из справочника позицию **«Котлеты куриные охлажденные»**, вписываем вес **12 кг**, дату выработки и новый срок годности (например, 5 суток).
   - **Рассуждение.** Теперь эти котлеты получили свой собственный «цифровой паспорт» (производственный ВСД), в котором зашита вся родословная: на какой птицефабрике выросла курица и на каком заводе сделали масло.
4. **Оформляем транзакцию**
   - **Что делаем.** Нажимаем кнопку **«Оформить»**.
   - **Рассуждение.** Со склада «Входная продукция» спишутся 10 кг курицы и 2 кг масла, а во вкладке **«Вырабатываемая продукция»** появятся **12 кг готовых котлет**. Теперь их можно спокойно отгружать в магазины через обычный исходящий **ВСД** (как в Уроке 4).

**Две главные ловушки при производстве**

- **Ловушка «Волшебное умножение веса» (Авансовое / нелогичное производство).** Система строго следит за математикой. Если вы заложили 10 кг мяса, а на выходе написали 50 кг котлет, автоматический алгоритм Россельхознадзора (ФГИС «ВетИС» компонент «Аналитика») сразу пометит это как фальсификат (добавление неучтенного сырья) и заблокирует учетную запись. Вес готового продукта должен соответствовать рецептуре (обычно выход меньше сырья из-за ужарки/усушки или немного больше за счет добавления воды, муки или специй).
- **Ловушка «Неподконтрольное сырьё».** Мука, соль, перец, панировочные сухари и вода в «Меркурии» не учитываются. В блок «Сырьё» вы добавляете только животную продукцию (мясо, молоко, яйца, жир). Специи вписывать никуда не нужно, но их вес учитывается в итоговой массе котлет.

## Урок 6: Главные штрафы и блокировки в «Меркурии»

В системе «**Меркурий**» почти нет «живых» инспекторов, которые сидят и читают каждый ваш документ. За всеми следит **робот-**

**аналитик (робот Россельхознадзора)**. Он круглосуточно сканирует миллионы транзакций и автоматически находит нарушения.

**За что чаще всего наказывают сотрудников и компании:**

1. **Несвоевременное гашение (более 24 часов).** Если машина приехала во вторник в 10:00, а вы нажали «Погасить» в четверг — это нарушение. За единичный случай робот выпишет предупреждение, за систематические — приостановит регистрацию сотрудника на срок от 10 до 90 дней (вы просто не сможете войти в систему и работать).
2. **Продажа или производство из просрочки.** Оформление документов на товар, у которого уже истек срок годности, влечет штраф на должностное лицо (от 3 000 до 5 000 руб.) и на юрлицо (от 100 000 до 300 000 руб. по ст. 14.43 КоАП РФ).
3. **Ввод продукции «из воздуха» через инвентаризацию.** Попытка увеличить остатки без входящих документов считается легализацией неизвестного товара и ведет к блокировке учетной записи на 3–6 месяцев.
4. **Негашеные «хвосты».** Ситуация, когда поставщик выписал документ (например, ошибочно), а покупатель его не увидел и не погасил в течение нескольких недель. Раз в неделю обязательно нужно проверять вкладку «**Входящие** →**Оформленные**», чтобы там не висели забытые документы.

**Блок 1. Приемка и возвраты**

**1. Сколько времени дается на гашение ВСД и с какого момента идет отсчет?**

- **Ответ:** На гашение дается **24 часа (1 рабочий день)** с момента **фактического прибытия** товара на вашу площадку (а не с момента, когда поставщик выписал документ).
- **Логика:** Если фура выехала из Москвы в понедельник, а приехала к вам в Санкт-Петербург в среду в 10:00, отсчет 24 часов начинается в среду в 10:00. Погасить документ нужно до четверга 10:00.

**2. Поставщик выписал ВСД на 50 кг мяса, а по факту приехало 52 кг (перевес). Как принять лишние 2 кг?**

- **Ответ:** Через гашение принять 52 кг **нельзя**. Вы гасите документ ровно на **50 кг**, а на лишние 2 кг поставщик должен выписать **новый отдельный ВСД**.
- **Логика:** «Меркурий» позволяет уменьшить объем при приемке (через «Разделить партию»), но строго запрещает его увеличивать. Если принять 52 кг по документу на 50 кг, 2 кг появятся «из воздуха», что расценивается как легализация неучтенного товара.

**3. Поставщик ошибся и прислал вам чужой ВСД (вы этот товар не заказывали и не получали). Что делать?**

- **Ответ:** Нажать **«Погасить» →поставить галочку «Разделить партию» → указать фактический объем «0»** и причину «Ошибочно оформленный ВСД».
- **Логика:** Нельзя просто игнорировать чужой документ, иначе он повиснет в «негашеных» и испортит вашу статистику. Обнуление при гашении официально закрывает документ у вас и возвращает товар обратно отправителю.

**4. Вы приняли товар вчера, нажали «Погасить», а сегодня нашли внутри коробки брак. Поставщик согласен забрать его. Как оформить?**

- **Ответ:** Оформить **исходящий транспортный ВСД** («Транзакции» →«Перевозка со сменой владельца») с целью «Возврат поставщику».
- **Логика:** После нажатия кнопки «Погасить» сделка закрыта, и вернуть товар «кнопкой отмены» уже невозможно. Теперь вы сами выступаете отправителем, но сделать это нужно **строго до истечения срока годности** товара.

**Блок 2. Склад, остатки и инвентаризация**

**5. Обязательно ли в кафе или продуктовом магазине списывать каждую проданную пачку творога в «Меркурии»?**

- **Ответ: Нет, не обязательно.** В рознице и общепите списывать каждую покупку на кассе не требуется. Достаточно периодически чистить склад через **Инвентаризацию**.
- **Логика:** По закону прослеживаемость в «Меркурии» заканчивается в момент продажи товара конечному потребителю. Инвентаризация нужна, чтобы в программе не висели «мертвые» остатки и просрочка.

**6. Как правильно списать через инвентаризацию товар, который повара использовали на кухне, и товар, который реально протух?**

- **Ответ:** Это делается **двумя разными документами** с разными причинами:

1. Использованное на кухне — причина **«Реализация в розницу»**.
2. Протухшее — причина **«Истечение срока годности»** (или «Порча»).

- **Логика:** Если списать всё как «Порчу», при проверке у вас потребуют акты утилизации и договор со спецполигоном (выкидывать биологические отходы в обычный мусорный бак запрещено).

**7. В «Журнале входной продукции» висит 10 кг сыра, у которого срок годности вышел вчера. Что делать прямо сейчас?**

- **Ответ:** Немедленно списать эти 10 кг через **Инвентаризацию** (причина «Истечение срока годности», фактический объем — **0**).
- **Логика:** Робот-аналитик Россельхознадзора проверяет остатки на площадках. Если просроченный товар висит на балансе магазина или кафе дольше нескольких дней, система автоматически фиксирует нарушение «Хранение и реализация просроченной продукции».

**Блок 3. Производство, интеграции и сложные случаи**

**8. У ИП два магазина в разных районах города. Нужно перевезти 10 кг колбасы из первого во второй. Нужен ли ВСД?**

- **Ответ: Да, обязательно.** Оформляется исходящий транспортный ВСД с типом **«Перевозка без смены владельца»**.
- **Логика:** Владелец (ИНН) у магазинов один, но **площадки (физические адреса)** разные. Любое перемещение подконтрольного товара по дорогам общего пользования между разными адресами должно сопровождаться эВСД.

**9. Кафе само печет пирожки с мясом и продает их посетителям за столиками. Нужно ли оформлять «Производство» в «Меркурии»?**

- **Ответ: Нет, не нужно.** Если блюда готовятся и продаются напрямую гостям в самом кафе (или через собственную доставку кафе), вы просто списываете сырье через инвентаризацию.
- **Логика:** Производственная транзакция нужна только тогда, когда вы делаете товар для **передачи другому юрлицу** или **перевозки на другую свою площадку** (например, у вас фабрика-кухня, которая развозит готовые обеды по 5 точкам).

**10. В кулинарном цеху из 10 кг сырого мяса после жарки получилось 7 кг готовых котлет. Не посчитает ли робот пропавшие 3 кг нарушением?**

- **Ответ: Нет, это норма.** Потеря веса при термообработке (ужарка, уварка) заложена в алгоритмы системы.
- **Логика:** Робот-аналитик бьет тревогу, когда готового продукта получается **значительно больше**, чем сырья (например, из 10 кг мяса сделали 30 кг котлет), так как это признак добавления неучтенного сырья.

**11. Что такое «авансовое производство» и почему оно запрещено?**

- **Ответ:** Это ситуация, когда вы оформляете выпуск готовой продукции в «Меркурии» **раньше**, чем приняли и погасили сырье, из которого она сделана.
- **Логика:** Например, машина с мясом приехала утром, вы сразу нажарили котлет и отправили их в магазин в 11:00, а ВСД на мясо погасили только в 15:00. Для программы это выглядит так, будто вы

сделали котлеты «из воздуха». Сначала всегда гасим сырье, и только потом проводим производство.

**12. Нужно ли сидеть в браузере на сайте «Меркурия», если в магазине стоит 1С или Контур.Маркет?**

- **Ответ: Нет.** Почти все современные учетные программы (1С, Контур.Маркет, СБИС, iiko, r_keeper) умеют общаться с «Меркурием» напрямую через шлюз (**ВетИС.API**).
- **Логика:** Вы нажимаете кнопку «Принять накладную» в своей привычной 1С, а программа сама в фоновом режиме отправляет команду в «Меркурий» и гасит документ. Веб-интерфейс в браузере нужен только в сложных случаях (например, при сбоях, настройке прав доступа или ручной инвентаризации).

Самый быстрый способ перестать бояться системы — это получить доступ к **демо-стенду (песочнице) ФГИС «ВетИС»**. Россельхознадзор предоставляет специальную учебную версию сайта (**mercury.vetrf.ru/hs** в тестовом контуре), где можно нажимать любые кнопки, создавать виртуальные поставки, ошибаться и сжигать товар без каких-либо реальных последствий.

# ПРОГРАММА «ЧЕСТНЫЙ ЗНАК»

## Урок 1. Зачем нужен «Честный ЗНАК» и как он устроен (Базовая логика)

Представьте, что «**Меркурий**» — это **паспорт НА ЦЕЛЫЙ вагон** с коробками сметаны. Он говорит: *«В этом вагоне едет 1000 банок сметаны с такого-то завода, коровы были здоровы»*.
А **«Честный ЗНАК» (ГИС МТ)** — это **ЛИЧНЫЙ ИНН ДЛЯ КАЖДОЙ отдельной баночки сметаны, бутылки воды или пары кроссовок**.
Если «Меркурий» следит за весом и партией, то «Честный ЗНАК» следит за **каждой штукой** от станка на заводе до писка сканера на кассе.

### 4 главных понятия «Честного ЗНАКа»

1. **Код маркировки (DataMatrix)**
   - **Что это.** Черно-белый квадратик, похожий на QR-код, который нанесен прямо на упаковку (крышку бутылки, пачку творога, этикетку футболки).
   - **В чем отличие от обычного штрихкода.** Обычный штрихкод одинаковый на всех пачках «Простоквашино». А код

**DataMatrix уникален для каждой отдельной пачки** — в нем зашифрован криптографический хвост, который невозможно подделать.

2. **ЭДО (Электронный документооборот)**
   - **Что это.** «**Цифровая почта**» между поставщиком и магазином (например, **Контур.Диадок**, **СБИС**, **Такском**).
   - **Зачем нужен.** В отличие от «Меркурия», где документы ходят внутри самой системы, в «Честном ЗНАКе» коды марок передаются от поставщика к вам внутри **электронной накладной (УПД)** через ЭДО.
3. **УПД (Универсальный передаточный документ)**
   - **Что это.** Главный документ при приемке маркированного товара. Это привычная бухгалтерская накладная, в «начинку» которой вшиты все уникальные коды **DataMatrix** из приехавшей коробки.
4. **Разрешительный режим на кассе**
   - **Что это.** Встроенный «фейс-контроль» на кассе. Когда кассир пикает код **DataMatrix**, касса за доли секунды спрашивает у «Честного ЗНАКа»: *«Эта бутылка легальна? Она не просрочена? Не продана ли она уже в другом городе?»*. Если с кодом проблема — касса просто заблокирует продажу.

**Главное действие при приемке: «Подписание УПД»**

В «**Меркурии**» мы нажимали кнопку **«Погасить»**. В «**Честном ЗНАКе**» аналог этого действия — **подписание входящего УПД электронной подписью (УКЭП)**.

**Как это происходит в жизни по шагам:**

1. Поставщик собрал для вас коробку с 20 пачками масла, отсканировал коды с каждой пачки и отправил вам электронный **УПД** через **ЭДО**.
2. Машина приехала в магазин. Вы берете **2D-сканер** (обычный лазерный сканер для штрихкодов не подойдет, нужен именно 2D-сканер, как для QR-кодов) и пикаете коды на упаковках, сверяя их с тем, что прислал поставщик в **УПД**.
3. Если всё сошлось — вы нажимаете в системе **ЭДО** кнопку **«Подписать»** (подтвердить).
4. В эту секунду **ЭДО** автоматически сообщает в «Честный ЗНАК»: *«Эти 20 пачек масла теперь принадлежат этому магазину»*. Товар встает на ваш баланс.

 **Главная ловушка новичков (Пересорт кодов):** Представьте, что поставщик отгрузил вам 2 одинаковые пачки творога, но в документ вписал код от третьей пачки, которая осталась у него на складе. Если вы подпишете УПД вслепую, не отсканировав упаковки, на ваш баланс

встанет «виртуальный» код, а реальная пачка на полке окажется «чужой». На кассе её не смогут продать.

**Практика №1: Сравниваем две системы**
Представьте, что утром в ваш магазин привезли **коробку с 10 бутылками питьевого йогурта** (это молочный продукт). Поскольку йогурт попадает сразу под обе системы, водитель привез его с документами в «**Меркурии**» и в **ЭДО** («**Честный ЗНАК**»). При проверке вы заметили, что **в коробке лежит 10 бутылок, всё свежее, но у одной бутылки порвана этикетка с квадратиком DataMatrix (сканер её не читает)**.
**Можно ли принять такую бутылку в магазин и продать её на кассе, или её нужно вернуть поставщику?**

**Принимать такую бутылку в магазин категорически нельзя — её нужно вернуть поставщику, оформив расхождения одновременно в «Честном ЗНАКе» (ЭДО) и в «Меркурии».** Продать товар с нечитаемым кодом **DataMatrix** на кассе невозможно, а любая попытка обойти систему приведет к штрафу.
**Пошаговый разбор и рассуждения приемщика**

1. **Почему нельзя принять бутылку с порванным кодом**
   - **Рассуждение.** На кассе работает «разрешительный режим». Когда покупатель принесет этот йогурт, кассир не сможет отсканировать поврежденный **DataMatrix**, а пробить товар по обычному штрихкоду программа запретит. В итоге бутылка превратится в неликвид, который невозможно продать.
2. **Действие в «Честном ЗНАКе» (через ЭДО) — отказ от 1 бутылки**
   - **Что делаем.** Мы **не подписываем** входящий УПД полностью. В системе **ЭДО** нажимаем кнопку **«Отклонить»** или **«Подписать с расхождениями»** (формируем УКД — универсальный корректировочный документ), указывая, что принимаем только **9 бутылок** с читаемыми кодами, а 1 бутылку возвращаем.
   - **Рассуждение.** Если подписать **УПД** на все 10 штук, виртуальный код 10-й бутылки повиснет на балансе магазина навсегда, хотя физически вы отдадите её водителю.
3. **Действие в «Меркурии» — синхронизация остатков**
   - **Что делаем.** Открываем входящий эВСД на эту партию йогурта (на 10 штук), нажимаем **«Погасить»**, ставим галочку **«Разделить партию»** и меняем фактический объем с 10 на **9 штук** (оформляя возвратный ВСД на 1 бутылку).
   - **Рассуждение.** «**Меркурий**» и «**Честный ЗНАК**» должны зеркально совпадать. Если принять 9 бутылок в ЭДО и 10 бутылок в «Меркурии», при автоматической сверке систем у инспекторов возникнет вопрос, откуда взялась лишняя бутылка.

# (Урок 2): Как товар выбывает из «Честного ЗНАКа» (Касса, кафе и списание)

В «**Меркурии**» мы помним правило: продали на кассе или съели на кухне — просто списали раз в месяц через инвентаризацию. В **«Честном ЗНАКе»** всё строже: государство хочет знать судьбу **каждого** кода маркировки.

Существует **3 главных способа**, как код списывается с вашего баланса:

1. **Продажа на кассе (Розничный магазин).** Кассир сканирует DataMatrix → **онлайн-касса** (**ОФД**) сама передает чек в «**Честный ЗНАК**» → код автоматически меняет статус на **«Выбыл из оборота»**. Никаких ручных документов создавать не нужно.
2. **Использование на кухне / для собственных нужд (Кафе, рестораны, офисы).** Если кафе купило молоко для кофе или воду для кулера, оно не продает каждую бутылку по чеку. В этом случае сотрудник заходит в личный кабинет «**Честного ЗНАКа**» (или **1С**) и создает документ **«Вывод из оборота»** с причиной *«Использование для собственных нужд»* (или *«Производственные цели»*).
3. **Порча, бой или утеря.** Если грузчик уронил паллету с водой или у молока вышел срок годности, оформляется документ **«Вывод из оборота»** с причиной *«Уничтожение»* или *«Истечение срока годности»*.

 **Главная ловушка общепита и розницы:** Нельзя продавать на кассе товар, который вы ещё не приняли в ЭДО! Если машина приехала утром, вы выставили молоко на полку, а УПД в ЭДО подписали только вечером, все дневные продажи пройдут с нарушением («продажа товара, не принадлежащего продавцу»).

Чтобы закрепить тему, разберем 5 типовых рабочих ситуаций:

- **1. Что делать, если привезли коробку с 20 пачками масла, и на коробке есть один общий код (КИГУ — транспортная упаковка)?**
  - **Ответ:** Можно отсканировать только один общий код с коробки (агрегат), и в УПД автоматически подтянутся все 20 пачек внутри. Но если хотя бы одну пачку вы отбракуете, коробку придется «расформировать» и сканировать остальные 19 пачек поштучно.
- **2. Можно ли продать на кассе 10 одинаковых бутылок воды, отсканировав одну бутылку 10 раз?**
  - **Ответ: Категорически нет.** У каждой бутылки свой уникальный код. Если пикнуть одну бутылку 10 раз, в чеке пробьется

10 продаж одного и того же кода (это грубое нарушение), а 9 других бутылок так и останутся висеть на балансе магазина.

- **3. Что такое ОСУ (Объемно-сортовой учет) и где он применяется?**
  - **Ответ:** Это упрощенный режим приемки (сейчас действует для части молочки, воды, безалкогольных напитков). Поставщик не вписывает каждый код в УПД, а указывает только общий код товара (GTIN) и количество (например: «Молоко 3,2% — 20 шт.»). Но на кассе при продаже всё равно нужно пикать DataMatrix с каждой пачки.
- **4. Покупатель вернул в магазин купленную вчера футболку с целой биркой DataMatrix. Как вернуть её в продажу?**
  - **Ответ:** Кассир оформляет чек **«Возврат прихода»** и сканирует код маркировки с бирки. Система «Честный ЗНАК» увидит чек возврата и автоматически вернет код из статуса «Выбыл» обратно в статус **«В обороте»** на баланс магазина.
- **5. Покупатель вернул товар, но оторвал и выкинул бирку с кодом DataMatrix. Можно ли выставить такой товар обратно на полку?**
  - **Ответ:** Продавать обезличенный товар запрещено. Магазин должен зайти в личный кабинет «Честного ЗНАКа», заказать **перемаркировку** (выпустить новый код DataMatrix за 60 копеек), распечатать его на принтере этикеток, наклеить на товар и только после этого вернуть на витрину.

# ПРОГРАММА «ЕГАИС»

## Урок 1. Зачем нужен ЕГАИС и как он устроен (Базовая логика)

Представьте, что **ЕГАИС (Единая государственная автоматизированная информационная система)** — это **строгий сейф с охранником** для всего алкоголя в стране. Если «**Меркурий**» следит за здоровьем животных и свежестью еды, а «**Честный ЗНАК**» — за подлинностью вещей, то **ЕГАИС** контролирует **каждую каплю спирта**, чтобы в магазины и бары не попадал суррогат и не уходили мимо кассы акцизные налоги.

По логике работы ЕГАИС стоит ровно посередине между «**Меркурием**» и «**Честным ЗНАКом**»:

- **Крепкий алкоголь (водка, коньяк, виски, вино)** учитывается **поштучно** — на каждой бутылке наклеена федеральная специальная марка (**ФСМ**) со штрихкодом, как в «**Честном ЗНАКе**».
- **Пиво, сидр, медовуха (разливное и фасованное)** учитываются **партиями (в литрах или штуках)** по электронным

накладным, как в «Меркурии» *(при этом фасованное пиво и кеги дополнительно проходят через «**Честный ЗНАК**»)*.

## 4 главных понятия ЕГАИС

1. **УТМ (Универсальный транспортный модуль)**
   - **Что это.** Специальная бесплатная программа от государства (Росалкогольтабакконтроля), которая ставится на компьютер магазина или бара.
   - **Зачем нужна.** У ЕГАИС нет привычного «личного кабинета в браузере», где можно руками нажимать кнопки, как в «Меркурии». УТМ работает в фоновом режиме как **почтовый курьер**: он забирает электронные накладные из базы государства и передает их в вашу учетную программу (1С, Контур.Маркет, iiko), а ваши ответы отправляет обратно.
2. **Электронная ТТН (Товарно-транспортная накладная)**
   - **Что это.** Главный документ при приемке алкоголя (полный аналог входящего ВСД в «Меркурии»). Пока поставщик не отправит ТТН в ЕГАИС, машина с алкоголем не имеет права даже выехать со склада.
3. **ФСМ / Акцизная марка**
   - **Что это.** Бумажная марка с уникальным штрихкодом на каждой бутылке крепкого алкоголя и вина. В ней зашифрована вся информация о заводе, дате розлива и крепости.
4. **Регистры остатков (Ваш виртуальный бар/склад)**
   - **Что это.** Виртуальный баланс вашего магазина или кафе в системе ЕГАИС (сейчас весь крепкий алкоголь учитывается на **Регистре №3** поштучно, а пиво — на **Регистре №1** по партиям).

## Главное действие при приемке: «Подтверждение ТТН»

В «**Меркурии**» мы нажимали **«Погасить»**, в «**Честном ЗНАКе**» — **«Подписать УПД»**, а в **ЕГАИС** это называется **«Подтвердить ТТН»** (отправить акт приема).

**Как это выглядит на практике по шагам:**

1. Поставщик привез ящик водки (20 бутылок) и отправил электронную **ТТН** через **УТМ** в вашу программу.
2. Вы берете **2D-сканер** и сканируете **акцизную марку с каждой из 20 бутылок** в ящике.
3. Программа сравнивает отсканированные марки со списком, который прислал поставщик в **ТТН**.

4. Если все 20 бутылок на месте и марки читаются — вы нажимаете **«Подтвердить»**. В эту секунду 20 бутылок официально встают на ваш виртуальный баланс в **ЕГАИС**.

**Главная ловушка ЕГАИС (Сроки приемки):** Подтвердить **ТТН** нужно строго в установленный законом срок: в городах — в течение **1 рабочего дня** с момента фактической приемки товара, в сельской местности — в течение **3 рабочих дней**. За просрочку грозит крупный штраф (от 10 000 руб. на должностное лицо и от 150 000 руб. на компанию).

## Урок 2. Бой при приемке и списание в магазине / баре

В работе с алкоголем есть две самые частые ситуации, которые нужно знать наизусть:

**Ситуация 1. При разгрузке разбили 1 бутылку или перепутали марки**

Если в ящике из 20 бутылок одна разбилась в дороге или её марка не совпадает с документом:

- Вы **НЕ подтверждаете ТТН** полностью, а отправляете поставщику **«Акт расхождения»** (указываете, что 19 бутылок приняли, а 1 отклонили).
- Поставщик в своей программе подтверждает ваш акт расхождения — после этого 19 бутылок встают на ваш баланс, а 1 разбитая списывается у поставщика.

**Ситуация 2. Как алкоголь списывается с баланса (Магазин против Бара)**

Здесь логика кардинально отличается в зависимости от того, где вы работаете:

- **В розничном магазине (Навынос):** Кассир на кассе сканирует сначала обычный штрихкод бутылки, а затем **акцизную марку**. Касса отправляет чек в **ЕГАИС** через **УТМ**, и бутылка **автоматически списывается** с баланса. Никаких ручных списаний в конце дня делать не нужно.
- **В кафе, баре или ресторане (В розлив):** Бармен не продает целую закрытую бутылку — он открывает её, чтобы налить гостю 50 грамм виски или бокал вина. Поэтому в общепите работает правило: **«Вскрыл бутылку — спиши её в ЕГАИС»**. Не позднее **следующего рабочего дня** сотрудник оформляет в учетной программе документ **«Акт списания»** с причиной **«Реализация»**, куда сканирует марки всех бутылок, открытых за смену (плюс отдельно фиксирует

порционные продажи через кассу по новым правилам учета чистого алкоголя в миллилитрах).

**Закрепление: 4 главных вопроса по ЕГАИС**

1. **Можно ли принять и продать бутылку вина, если у неё в дороге оторвалась или сильно затерлась акцизная марка?**
   - **Ответ: Нет.** Без читаемой марки бутылка считается нелегальной. Её нужно вернуть поставщику через Акт расхождения при приемке. Если марку повредили уже в магазине, бутылку списывают актом с причиной «**Порча**» (продавать её запрещено).
2. **Что такое «пересорт марок» в ящике водки?**
   - **Ответ:** Ситуация, когда в ящике лежит 20 бутылок правильной водки, но поставщик по ошибке положил одну бутылку из другого ящика (с другой акцизной маркой). Если не просканировать все 20 бутылок при приемке и принять накладную «вслепую», при продаже этой бутылки на кассе **ЕГАИС** выдаст ошибку, так как эта конкретная марка числится на другом складе.
3. **Как списывается разливное пиво в кегах в ЕГАИС?**
   - **Ответ:** В **ЕГАИС** кега списывается целиком в момент её подключения к крану (вскрытия) через «**Акт списания**», а в «**Честном ЗНАКе**» дополнительно подается отчет о постановке кеги на кран с дальнейшим учетом каждого пролитого литра через кассу.
4. **Нужна ли электронная подпись (УКЭП) и специальная флешка (Рутокен / JaCarta) для работы с ЕГАИС?**
   - **Ответ: Да, обязательно.** Без вставленной в компьютер крипто-флешки с электронной подписью руководителя **УТМ** не запустится, и вы не сможете ни принять накладную, ни пробить алкоголь на кассе.

| Система | Что контролирует | В чем измеряет | Как принимаем | Как списываем |
|---|---|---|---|---|
| **Меркурий** | Мясо, рыба, молоко, яйца, мёд | **Партии** (кг, литры, штуки) | Кнопка **«Погасить»** (за 24 часа) | **Инвентаризация** (раз в неделю/месяц) |
| **Честный ЗНАК** | Молочка, вода, одежда, обувь, табак, пиво | **Каждую штуку** (код DataMatrix) | Подписание **УПД в ЭДО** | **Касса** (автоматически) или **Вывод из оборота** |
| **ЕГАИС** | Крепкий алкоголь, вино, пиво | **Каждую бутылку** (акцизная марка) / литры | **Подтверждение ТТН** через УТМ (за 1 день) | **Касса** (в магазине) или **Акт списания** (в баре) |

В конкретной работе товаровед или кладовщик не держит открытыми три разных окна в браузере. Вместо этого используется **товароучетная программа** — это как «единый пульт управления» для всего магазина или кафе.

## Как всё работает в одном окне (1С / Контур.Маркет / СБИС)

Представьте, что в ваш магазин приехала одна машина, которая привезла сразу три товара:

1. **10 кг замороженной курицы** (контролируется только в **«Меркурии»**).
2. **20 пачек сливочного масла** (контролируется одновременно в **«Меркурии»** и в **«Честном ЗНАКе»**).
3. **10 бутылок вина** (контролируется в **ЕГАИС**).

Вот как выглядит ежедневный рабочий процесс сотрудника в учетной программе по шагам:

**Шаг 1. Загрузка входящих документов (Кнопка «Обмен» / «Обновить»)**

- **Что происходит.** Утром вы открываете рабочую программу (например, 1С:Розница или Контур.Маркет) и нажимаете кнопку **«Загрузить входящие»**.

- **Как это работает под капотом.** Программа сама стучится в шлюзы всех трех госсистем (через ВетИС.API, оператор ЭДО и модуль УТМ) и выкладывает перед вами в одном списке все пришедшие накладные.

**Шаг 2. Сопоставление номенклатуры (Самый важный навык новичка!)**

- **В чем суть проблемы.** У поставщика в базе товар называется *«Масло сл. Крестьянское 72,5% 180г БЗМЖ»*, а у вас в кассе он заведен как *«Масло сливочное 180г»*. Компьютер не знает, что это один и тот же товар.
- **Что делает сотрудник.** При первой приемке нового товара вы вручную связываете строчку из накладной поставщика с карточкой товара в вашей базе (это называется **мэппинг** или сопоставление). Программа запоминает эту связку, и в следующий раз подставит товар автоматически.

**Шаг 3. Физическая приемка со сканером в руках**

- **Курица (10 кг).** Взвешиваете коробку, сверяете дату выработки на этикетке с экраном.
- **Масло (20 пачек).** Берете 2D-сканер и пикаете коды **DataMatrix** на пачках (или один общий код на коробке). Программа подсвечивает строки зеленым цветом — значит, коды совпали с УПД.
- **Вино (10 бутылок).** Тем же 2D-сканером сканируете акцизную марку на каждой бутылке.

**Шаг 4. Единое подтверждение и постановка на приход**

- **Что происходит.** Когда все строки проверены, вы нажимаете в программе **«Провести / Подтвердить»** и прикладываете электронную подпись (флешку с **УКЭП**).
- **Результат одной кнопки.** Программа сама:
  - Погасит **ВСД** на курицу и масло в **«Меркурии»**.
  - Подпишет **УПД** и примет коды масла на баланс в **«Честном ЗНАКе»**.
  - Отправит акт подтверждения **ТТН** на вино в **ЕГАИС**.
  - Оприходует все товары на ваш склад в **1С**, чтобы касса знала остатки и цены для продажи.

**Правила товароведа:**

1. **Сначала сканируем и проверяем глазами — потом нажимаем «Подтвердить».** Отменить подписанный документ в госсистемах почти невозможно, исправлять ошибки придется через сложные возвраты.
2. **Следим за таймером.** На приемку в «**Меркурии**» и **ЕГАИС** дается **1 рабочий день (24 часа)** с момента приезда машины. В ЭДО («**Честный ЗНАК**») обычно дается до 3 рабочих дней, но продавать товар на кассе нельзя, пока УПД не подписан.

3. **Не копим «хвосты» на складе.** Раз в неделю проверяем висящие непринятые документы и списываем проданную/испорченную молочку и мясо в «**Меркурии**» ЧЕРЕЗ ИНВЕНТАРИЗАЦИЮ, чтобы не копить виртуальную просрочку.

## ФГИС «Зерно» — «Меркурий» для муки, круп и зерна

Представьте, что **ФГИС «Зерно»** — это родной брат «Меркурия», только вместо коров и курицы он следит за **пшеницей, мукой, гречкой, рисом, овсянкой и отрубями**. Если вы работаете в пекарне, кондитерской, пиццерии, кафе или продуктовом магазине, вы обязательно с ним столкнетесь.

**3 главных понятия системы**

1. **СДИЗ (Сопроводительный документ по идентификации зерна).** Главный электронный документ системы (полный аналог эВСД из «Меркурия»). Без него муку или крупу нельзя перевозить и продавать.
2. **Партия продукции.** Как и в «**Меркурии**», учет идет не поштучно, а **по весу (в килограммах или тоннах)**. У каждой партии мешков муки с завода есть свой уникальный номер партии.
3. **Учет в 1С или личном кабинете.** Работать можно напрямую на сайте `zerno.mcx.gov.ru` (вход через Госуслуги) или через учетную программу (1С, Контур, СБИС).

**Пошаговый алгоритм работы (на примере пекарни)**

- **Шаг 1. Приемка (Гашение СДИЗ).** Поставщик привез вам 10 мешков пшеничной муки по 50 кг (всего 500 кг) и отправил СДИЗ. Вы проверяете вес, целостность мешков, заходите в раздел **«СДИЗ»** → **«Входящие»** и нажимаете кнопку **«Погасить»**. Мука встает на ваш виртуальный склад.
- **Шаг 2. Если привезли меньше (Частичное гашение).** Если по документам едет 500 кг, а один мешок порвался и вы приняли только 450 кг, при гашении вы вписываете фактический вес **450 кг**. Остальные 50 кг система автоматически вернет поставщику.
- **Шаг 3. Списание муки в работу (Выпечка).** Когда пекарь берет муку со склада и печет хлеб, мука перестает существовать. Чтобы она не висела на виртуальном складе, раз в неделю или месяц сотрудник оформляет операцию **«Списание»** с причиной **«Производство / переработка»** (если мука ушла в тесто) или **«Реализация в розницу»** (если магазин продал пачки гречки покупателям на кассе).

 **Главные ловушки ФГИС «Зерно»**

- **Ловушка «Перепутали СДИЗ на зерно и на продукты переработки».** В системе есть два разных раздела: просто «**Зерно**» (сырое зерно с поля) и «**Продукты переработки зерна**» (мука, крупы, хлопья). Новички часто ищут входящую накладную на муку не в той вкладке и думают, что поставщик её не прислал.
- **Ловушка «Забытые остатки».** Как и в «Меркурии», если пекарня год принимает муку, но ни разу её не списывает в производство, на виртуальном складе накапливаются десятки тонн «мертвой» муки, что привлекает внимание проверяющих.

## ГИИС ДМДК — «Честный ЗНАК» для золота и ювелирных изделий

Представьте, что **ГИИС ДМДК** (**Государственная система контроля за оборотом драгоценных металлов и драгоценных камней**) — это **«Честный ЗНАК» для ювелирных магазинов и ломбардов**. Каждое золотое кольцо, цепочка или серьги с бриллиантом имеют свой цифровой паспорт с момента добычи металла до продажи на кассе.

**3 главных понятия системы**

1. **УИН (Уникальный идентификационный номер).** Код из **16 цифр**, который присваивается каждому отдельному украшению. Даже если на витрине лежат 10 абсолютно одинаковых обручальных колец одного размера, у каждого из них свой УИН.
2. **Двухмерный код DataMatrix (Нано-гравировка и бирка).** На каждое золотое или платиновое изделие лазером наносится микроскопический код прямо на металл (рядом с пробой), а также печатается крупный DataMatrix на бумажной бирке, чтобы его можно было считать обычным 2D-сканером. *(Для серебра требования упрощены — достаточно бирки с УИН).*
3. **Спецификация (Электронная накладная).** Главный документ при передаче партии украшений с завода в магазин (полный аналог УПД в «Честном ЗНАКе»).

**Пошаговый алгоритм работы (на примере ювелирного магазина)**

- **Шаг 1. Приемка (Подтверждение спецификации).** Ювелирный завод прислал посылку с 30 кольцами и оформил в **ГИИС ДМДК** **Спецификацию**. Товаровед берет 2D-сканер и **пикает бирку каждого кольца**. Если все 30 УИН совпали со списком в компьютере — нажимает **«Подтвердить»** (подписывает электронной подписью). Кольца встают на баланс магазина.
- **Шаг 2. Если приехало не то кольцо (Пересорт УИН).** Внешне два кольца одинаковые, но у одного бирка с чужим УИН. Принимать

такую спецификацию целиком нельзя — нужно оформить **акт расхождения**, исключив проблемное кольцо из приемки, иначе на кассе его невозможно будет продать.

- **Шаг 3. Продажа на кассе.** Когда покупатель выбирает кольцо, кассир сканирует код **DataMatrix** с бирки. Онлайн-касса через специальный модуль (**УТМ ДМДК**) отправляет данные в госсистему, и этот конкретный 16-значный **УИН** переходит в статус **«Реализовано»** (выбывает из оборота).
- **Шаг 4. Возврат украшения от покупателя.** Если покупатель вернул кольцо (например, по браку) с целой биркой, кассир пробивает чек возврата, и УИН снова возвращается на баланс магазина.

 **Главные ловушки ГИИС ДМДК**

- **Ловушка «Оторвалась или затерлась бирка».** Без читаемого УИН продавать ювелирное изделие запрещено. Если в магазине оторвали бирку, нельзя просто распечатать новую в Word — нужно зайти в ГИИС ДМДК, привязать старый УИН к новой бирке и распечатать её через специальный принтер этикеток.
- **Ловушка «Учет лома и сдачи в ремонт».** Если покупатель сдает старое золотое кольцо в скупку или в ремонт, система не списывает его просто так — изделие переводится в специальный статус «Лом/отходы» с указанием точного веса чистого металла в граммах.

1. **В кафе привезли 5 мешков риса (по 25 кг) по ФГИС «Зерно». Какой документ нужно оформить при приемке и что сделать, когда повара сварят весь рис?**
   - **Ответ:** При приемке нужно **погасить входящий СДИЗ** на 125 кг. Когда рис израсходуют на кухне, нужно оформить **списание** на 125 кг с причиной «Производство / переработка».
2. **В ювелирный магазин пришло 5 одинаковых золотых цепочек. Можно ли при приемке отсканировать бирку одной цепочки 5 раз?**
   - **Ответ: Нет, это грубое нарушение.** У каждой из 5 цепочек свой уникальный 16-значный **УИН**. Сканировать нужно каждую цепочку по очереди, иначе на балансе магазина встанет 1 цепочка, а 4 останутся «нелегальными».
3. **В продуктовом магазине на кассе продают пачку овсяных хлопьев. Нужно ли кассиру сканировать какой-то спецкод для ФГИС «Зерно»?**
   - **Ответ: Нет.** В отличие от «Честного ЗНАКа» и ГИИС ДМДК, во ФГИС «Зерно» нет поштучных кодов на пачках. Магазин просто гасит входящий СДИЗ при приемке партии, а затем периодически списывает проданные объемы в личном кабинете.

4. **Чем по логике работы отличаются две пары систем: (Меркурий + ФГИС «Зерно») и (Честный ЗНАК + ГИИС ДМДК)?**
   - **Ответ:** Первая пара (**Меркурий** и **Зерно**) работает по принципу **весового/партионного учета** (мы следим за килограммами и партиями, гасим документы при приемке и списываем остатки раз в неделю/месяц). Вторая пара (**Честный ЗНАК** и **ДМДК**) работает по принципу **поштучного учета** (у каждой единицы товара есть уникальный цифровой код, который сканируется на кассе при каждой продаже).

## Контур.Диадок и СБИС (Saby): ЭДО на пальцах

**Жизненная аналогия**

Представьте обычную **Почту России**, только заказное письмо с накладной едет не неделю в бумажном конверте, а долетает за **1 секунду** в браузере. Вместо синей шариковой ручки и круглой печати вы ставите **УКЭП** (усиленную квалифицированную электронную подпись — специальную флешку-токен). По закону такой электронный документ имеет точно такую же силу, как бумага с мокрой печатью.

**В чем разница между Диадоком и СБИС?**

- **Контур.Диадок** — это специализированный сервис **только для обмена документами**. Он легкий, быстрый и интуитивно понятный, как обычная электронная почта (папки «Входящие», «Исходящие», «Требуют подписания»).
- **СБИС (теперь называется Saby)** — это огромный **комбайн «всё в одном»**. Помимо **ЭДО**, внутри него есть сдача отчетности в налоговую, учет кадров, проверка контрагентов, кассовая программа и работа с «**Меркурием**» / «**Честным ЗНАКом**».

**Пошаговый алгоритм работы в ЭДО (Приемка накладной)**

1. **Настройка роуминга (1 раз).** Если у поставщика **СБИС**, а у вас **Диадок**, они общаются через «роуминг» (как звонок с МТС на МегаФон). Перед первой поставкой компании обмениваются приглашениями в сервисе.
2. **Входящий документ (УПД).** Поставщик отгрузил товар и прислал вам **УПД** (Универсальный передаточный документ — это счет-фактура + товарная накладная в одном файле). Документ падает в папку **«Требуют обработки»**.
3. **Проверка.** Вы открываете документ на экране, сверяете цены, количество товара и коды маркировки «Честного ЗНАКа» (если товар маркированный).
4. **Подписание или отказ:**

- **Всё верно** → нажимаете кнопку **«Подписать и отправить»** (вводите пароль от флешки **УКЭП**). Сделка закрыта.
- **Нашли ошибку (не та цена, недовезли коробку)** → нажимаете **«Отказать в подписи»**, в окошке пишете причину (*«Неверная цена в строке 3»* или *«Недовоз 2 шт.»*) и отправляете поставщику, чтобы он прислал исправленный документ.

 **Главная ловушка ЭДО:** Никогда не нажимайте кнопку «Подписать», если не уверены в ценах или количестве! После подписания обеими сторонами документ считается юридически закрытым, и просто «удалить» его нельзя — придется просить поставщика оформлять корректировочный документ (**УКД**).

## WMS и ТСД: как работает современный склад

**Жизненная аналогия**

- **Обычный склад (без WMS)** — это как ваш домашний шкаф, куда вещи кладут «куда влезло». Чтобы найти зимнюю куртку, нужно перерыть все полки.
- **WMS (Warehouse Management System — система управления складом)** — это **электронный навигатор** для огромного склада (как на складах Ozon или Wildberries). Каждая полка и ячейка на складе имеет свой штрихкод и адрес (например: *Зона А — Ряд 05 — Стеллаж 12 — Полка 3*). Система сама решает, куда положить товар и по какому маршруту отправить кладовщика, чтобы он не наматывал лишние километры.
- **ТСД (Терминал сбора данных)** — это **промышленный смартфон в противоударном корпусе со встроенным лазерным сканером** (вместо камеры для фото). На него ставится программа (например, *DataMobile* или *Клеверенс*), которая по Wi-Fi общается с 1С или **WMS**.

**Пошаговый рабочий день кладовщика с ТСД (3 главные операции)**

1. **Приемка (Размещение по ячейкам)**
   - Приехала фура. В 1С загрузилась накладная, и задание прилетело на экран ТСД кладовщика.
   - Кладовщик пикает штрихкод на коробке \(\rightarrow \) ТСД пишет: *«Отнеси в ячейку А-05-12»*.
   - Кладовщик подходит к стеллажу, кладет коробку и **обязательно пикает штрихкод самой ячейки на полке**. Теперь компьютер точно знает, где лежит этот товар.

2. **Сборка заказа (Пикинг)**
   - Клиент заказал 3 разных товара. На экране ТСД появляется маршрут (волна сборки).
   - Навигатор ведет кладовщика по кратчайшему пути: *«Подойди к ячейке А-05-12, возьми 2 шт.»*.
   - Кладовщик сканирует ячейку (подтверждает, что подошел туда), затем сканирует товар (подтверждает, что взял не пересорт). Если взять не ту коробку — ТСД громко запищит и выдаст ошибку на красном экране.
3. **Инвентаризация (Пересчет)**
   - Кладовщик подходит к стеллажу, сканирует адрес ячейки, а затем подряд пикает все товары, которые в ней лежат. Данные мгновенно улетают в 1С, и программа сама сравнивает реальность с учетной базой.

 **Главная ловушка работы с ТСД:** «Скан по памяти» или пиканье одной коробки 10 раз подряд вместо каждой коробки по очереди. Если внутри стопки затесался другой товар или брак, ТСД этого не увидит, и на складе возникнет пересорт, за который отвечает смена.

1. **Поставщик прислал вам УПД в Контур.Диадоке на 10 стульев, а водитель привез только 9. Ваши действия в Диадоке?**
   - **Ответ:** Не подписывать документ! Нажать кнопку **«Отказать в подписи»**, указать причину *«Фактически принято 9 шт., недовоз 1 шт.»* и ждать от поставщика новый исправленный УПД (или УКД).
2. **Чем ТСД отличается от обычного ручного сканера, который стоит на кассе в магазине?**
   - **Ответ:** Ручной сканер просто считывает штрихкод и передает цифры в компьютер по проводу (у него нет своего экрана и мозгов). **ТСД** — это полноценный мини-компьютер с экраном, кнопками и программой внутри, с которым кладовщик автономно ходит по всему складу без проводов.
3. **Зачем кладовщику при раскладке товара на складе с WMS сканировать не только сам товар, но и наклейку на полке стеллажа?**
   - **Ответ:** Чтобы привязать товар к конкретному адресу (ячейке) в базе данных. Без сканирования ячейки система не будет знать, куда именно положили товар, и не сможет построить маршрут для его сборки.

# BAS (Business Automation Software)

Представьте, что **BAS** — это **родной брат-близнец 1С**, который вырос в другой стране. Внутри у него точно такой же «двигатель» (та же платформа, окна, справочники и язык программирования), но все налоги, законы и печатные формы настроены под местные правила.

**Главные конфигурации BAS (что требуют в вакансиях)**

- **BAS Бухгалтерия** — программа для бухгалтеров (налоги, зарплата, банк, отчетность).
- **BAS Управление торговлей** — аналог **1С:УТ** для оптовых складов, магазинов и менеджеров по продажам (заказы, цены, остатки на складах).
- **BAS Розничная торговля** — программа для магазинов у дома и кассиров.
- **BAS Комплексное управление предприятием** — система для средних и крупных заводов, где в одной базе ведут склад, производство и бухгалтерию.

**Как выглядит работа оператора в BAS (по шагам)**

1. **Заведение карточки товара (Номенклатуры).** Оператор создает товар в справочнике, указывает единицу измерения (штуки, килограммы) и **таможенный код товара (УКТ ВЭД)** — обязательный код из десяти цифр для налоговых документов.
2. **Оформление прихода («Поступление товаров и услуг»).** Когда приезжает машина с товаром, оператор заносит накладную от поставщика — товар встает на склад.
3. **Оформление продажи («Реализация товаров и услуг»).** Когда товар отгружают клиенту, оператор создает документ продажи и прямо из него в один клик формирует **Налоговую накладную** для передачи бухгалтеру.

 **Главная ловушка BAS:** Таможенный код товара в карточке. Он обязан попадать в каждый кассовый чек (для алкоголя, сигарет и топлива) и в каждую налоговую накладную. Если оператор ошибется хотя бы в одной цифре этого кода, налоговая заблокирует накладную поставщика или покупателя.

# M.E.Doc (Медок) и Вовремя — электронный документооборот и отчетность (аналоги СБИС и Диадока)

Функции СБИСа и Диадока здесь четко разделены между двумя главными сервисами:

- **M.E.Doc (Медок)** — главный рабочий инструмент бухгалтера (аналог СБИС Отчетность). Через него сдают отчеты в налоговую, но главная его задача — **регистрация налоговых накладных в едином государственном реестре**. Без зарегистрированной в Медке налоговой накладной покупатель не сможет вернуть или зачесть НДС.
- **Вовремя** — легкий облачный сервис в браузере (полный аналог **Контур.Диадока**). Используется для быстрого обмена обычными первичными документами: **расходными накладными** (товарными накладными), **актами выполненных работ** и договорами.

**Пошаговый алгоритм работы с документами**

1. Поставщик формирует в BAS расходную накладную и выгружает её в сервис **Вовремя** (или напрямую через встроенный модуль интеграции).
2. Покупатель заходит в **Вовремя**, проверяет количество и цены, прикладывает свою **квалифицированную электронную подпись (КЭП)** и нажимает **«Подписать»**.
3. Если компания работает с **НДС**, поставщик дополнительно отправляет через **M.E.Doc** налоговую накладную на регистрацию в налоговую службу.

# ПРРО (Программный регистратор расчетных операций) — онлайн-касса в телефоне или компьютере

**Жизненная аналогия**

В России внутри каждой онлайн-кассы стоит специальный дорогой чип — фискальный накопитель. Здесь от обязательных «железных» касс разрешили отказаться в пользу **ПРРО** — это просто **программа на смартфоне, планшете или компьютере** (самые популярные сервисы — **Checkbox**, **Вовремя.Касса**, **Poster**), которая сама связывается с налоговой через интернет.

**Как работает программная касса**

1. Кассир открывает смену в приложении на компьютере или смартфоне с помощью электронной подписи.

2. При продаже товара программа за доли секунды обращается к серверу налоговой службы по интернету, получает от налоговой уникальный **фискальный номер чека** и печатает чек на обычном принтере (или показывает QR-код чека на экране для покупателя).
3. В конце смены кассир нажимает кнопку **«Закрыть смену (Z-отчет)»** — данные автоматически улетают в налоговую.

## E-ТТН и «е-Акциз» — прослеживаемость грузов и алкоголя

- **E-ТТН (Электронная товарно-транспортная накладная)** — государственный реестр грузоперевозок (аналог **ГИС ЭПД**).
  - **Как работает:** Ни одна фура с грузом не может выехать на трассу без зарегистрированной в базе **E-ТТН**. В документе указываются данные отправителя, перевозчика (водителя и номера машины) и получателя. При разгрузке кладовщик подписывает **E-ТТН** своей электронной подписью прямо на планшете или смартфоне.
- **«е-Акциз» (Электронная акцизная марка)** — цифровой аналог российского **«Честного ЗНАКа» + ЕГАИС** для алкоголя, сигарет и жидкостей для электронных сигарет.
  - **Как работает:** Вместо привычных бумажных акцизных марок на каждую бутылку алкоголя или пачку сигарет наносится уникальный двухмерный код **DataMatrix** (электронная марка). Система отслеживает весь путь бутылки: от завода или таможни до сканера на программной кассе (**ПРРО**) в магазине. При продаже на кассе код сканируется, и марка автоматически списывается в базе государства.

## Odoo — международный «конструктор» для бизнеса (аналог 1С)

**Жизненная аналогия**
Если 1С — это готовый многоквартирный дом со всеми коммуникациями, то **Odoo** — это **набор кубиков Lego**. Вы открываете пустую систему и сами включаете нужные приложения: сначала «Склад», потом «Закупки», затем «Кассу» или «Интернет-магазин». Все модули мгновенно связываются друг с другом без программирования.
**Главная фишка: Складской учет по принципу «Двойной записи»**
В отличие от обычных программ, где товар при приемке просто «появляется» на складе, в **Odoo** товар **никогда не возникает из ниоткуда и не исчезает в никуда**. Весь склад работает как перемещение между виртуальными локациями:

- Приемка товара = перемещение из виртуальной зоны *«Поставщики»* в зону *«Наш склад»*.
- Продажа клиенту = перемещение из зоны *«Наш склад»* в виртуальную зону *«Клиенты»*.
- Списание брака = перемещение из зоны *«Наш склад»* в виртуальную мусорку *«Потери / Списание»*.

**Пошаговый алгоритм работы в Odoo (Закупка и приемка)**

1. **Запрос котировок (RFQ) → Заказ на закупку (Purchase Order).** Менеджер создает заказ поставщику (например, 50 стульев) и нажимает «Подтвердить заказ».
2. **Автоматическое задание кладовщику.** В ту же секунду в модуле «Склад» (**Inventory**) сама появляется карточка ожидаемой приемки на 50 стульев.
3. **Валидация (Validate).** Когда машина приезжает, кладовщик сверяет количество (или сканирует штрихкоды) и нажимает кнопку **«Подтвердить» (Validate)**. Стулья перемещаются на склад и сразу становятся доступны для продажи.

 **Главная ловушка Odoo:** Если по факту приехало 45 стульев вместо 50, а кладовщик нажал «Подтвердить», система спросит: *«Создать дозаказ (Backorder) на оставшиеся 5 стульев или забыть про них?»*. Если нажать «Не создавать» по ошибке, система закроет сделку, и 5 недовезенных стульев потеряются из учета.

## SAP S/4HANA (модуль MM) — стандарт мировых корпораций

## Жизненная аналогия

Если **Odoo** — это удобный городской кроссовер, то **SAP** — это **кабина пилота огромного авиалайнера Boeing 747**. Здесь сотни кнопок и строжайшие правила безопасности. В SAP работают гиганты (Nestlé, Coca-Cola, Bosch, крупные торговые сети), потому что система способна переваривать миллионы операций в час без сбоев.
Для операторов, логистов и кладовщиков в SAP главный модуль — **MM (Materials Management — управление запасами и закупками)**.

**3 главных понятия SAP**

1. **T-codes (Коды транзакций).** В SAP опытные сотрудники почти не пользуются меню — они вбивают в верхнюю строку короткий код из букв и цифр, чтобы мгновенно открыть нужное окно. Например: `ME21N` — создать заказ поставщику, `MIGO` — принять или переместить товар на складе, `MMBE` — посмотреть остатки товара.

2. **Принцип «Тройной сверки» (3-Way Match).** SAP не даст бухгалтеру оплатить счет поставщику, пока компьютер не увидит три идеально совпадающих документа:
   - **Заказ на закупку (PO)** — что мы заказывали.
   - **Поступление материала (Goods Receipt)** — что реально принял кладовщик на воротах.
   - **Счет-фактура (Invoice)** — за сколько денег поставщик просит заплатить.
3. **Номер документа материала.** Каждое движение коробки на складе получает уникальный 10-значный номер в базе.

**Главная ловушка SAP:** В **SAP НЕЛЬЗЯ просто удалить или отредактировать** проведенный документ, если вы ошиблись в цифре (как это иногда делают в 1С). Любая ошибка исправляется только через создание нового документа **«Сторно» (Reversal)** — то есть зеркальной операции со знаком минус, чтобы в истории навсегда остался след, кто и когда ошибся.

**Протоколы EDI**

**Жизненная аналогия**

Представьте, что у поставщика стоит **SAP**, у торговой сети (например, Walmart или Auchan) — своя закрытая программа, а у перевозчика — третья. Чтобы эти программы понимали друг друга без участия человека, придумали **EDI (Electronic Data Interchange)** — это **единый дипломатический язык роботов**.

Вместо привычных нам картинок накладных по **EDI** летают строгие текстовые файлы (в формате **XML** или **EDIFACT**), где каждое поле стоит на своем месте.

**3 главных сообщения EDI (Золотой стандарт торговли)**

1. **ORDERS (Заказ на поставку).** Торговая сеть автоматически (когда товар заканчивается на полках) отправляет поставщику файл ORDERS: *«Привезите во вторник к 09:00 на склад №3 ровно 100 коробок сока»*.
2. **DESADV (Despatch Advice — Уведомление об отгрузке).** Полный аналог российского УПД или накладной. Когда фура грузится на заводе, поставщик отправляет в сеть файл **DESADV**. В нем написано не просто «100 коробок», а **какая коробка на какой паллете лежит**.
3. **INVOIC (Коммерческий счет).** Финансовый документ на оплату, который улетает после того, как склад принял товар.

**Важный термин в паре с EDI: Этикетка SSCC**

Вместе с файлом **DESADV** на каждую паллету клеится специальная этикетка со штрихкодом **SSCC** (**Serial Shipping Container Code** — уникальный 18-значный номер паллеты). Когда фура приезжает в

распределительный центр сети, кладовщик не вскрывает паллету — он просто **пикает один штрихкод SSCC**, и система по файлу **DESADV** сразу понимает, какие 40 коробок находятся внутри.

**Задача 1. Odoo (Недовоз товара при приемке)**
**Условие задачи:**
Менеджер оформил в **Odoo** заказ поставщику (**Purchase Order**) на **100 коробок кофе**. Когда фура приехала на склад, кладовщик пересчитал груз и обнаружил только **80 коробок** (20 коробок поставщик пообещал довезти следующей машиной через два дня).
**Как кладовщик должен закрыть приемку в Odoo, чтобы не потерять 20 коробок из учета?**
**Пошаговое решение:**

1. **Открываем карточку приемки (Receipt).** В модуле *Склад (Inventory)* кладовщик находит ожидаемую поставку на 100 коробок.
2. **Меняем количество в графе «Выполнено» (Done).** Вместо 100 вписываем фактическое количество — **80**.
3. **Нажимаем «Подтвердить» (Validate).** Поскольку цифры не совпали, Odoo остановит процесс и выведет всплывающее окно с выбором действия.
4. **Выбираем кнопку «Создать дозаказ» (Create Backorder).**
   - **Логика решения:** В эту секунду 80 коробок официально встанут на склад и будут готовы к продаже, а система автоматически создаст **новую карточку приемки на оставшиеся 20 коробок**, которая будет висеть в списке ожидаемых поставок до приезда второй машины. Если бы кладовщик нажал «Без дозаказа» (**No Backorder**), система бы списала недовоз, и склад забыл бы про долг поставщика.

**Задача 2. SAP S/4HANA (Ошибка в накладной после проведения)**
**Условие задачи:**
Оператор склада в SAP провел поступление товара через транзакцию MIGO (принял 50 мониторов на основной склад). Через час выяснилось, что по ошибке он указал не тот номер склада, и документ уже сохранен в базе. Прямое редактирование в SAP заблокировано.
**Как исправить эту ошибку по стандартам SAP?**
**Пошаговое решение:**

1. **Открываем транзакцию MIGO (Движение материала).**
2. **Меняем режим операции на «Сторнирование» (Cancellation).** В верхнем левом выпадающем списке вместо «**Поступление**» (**Goods Receipt**) выбираем «**Сторнирование**» (**Cancellation**).

3. **Вводим номер ошибочного документа.** Вставляем 10-значный номер того самого документа, который провели час назад, и нажимаем Enter. Система сама подтянет все 50 мониторов со знаком минус (код движения **102**).
4. **Проводим документ (Post) и оформляем правильный приход заново.**
   - **Логика решения:** В **SAP** действует строгий аудиторский след — нельзя просто стереть строчку ластиком. Сначала мы «обнуляем» ошибку зеркальным документом в минус (чтобы на неправильном складе стал 0), а затем заново проводим обычное поступление (**101** вид движения) уже на правильный склад.

**Задача 3. Протоколы EDI + этикетка SSCC (Приемка фуры в крупной сети)**

**Условие задачи:**

Поставщик отправляет в распределительный центр крупной торговой сети фуру, в которой стоит **20 паллет** с соком (на каждой паллете по 50 коробок, всего 1000 коробок). Сеть работает по стандартам EDI.

**Как проходит обмен документами и приемка без вскрытия каждой паллеты?**

**Пошаговое решение:**

1. **Получение заказа (ORDERS).** Накануне отгрузки робот торговой сети присылает поставщику EDI-файл **ORDERS** с запросом на 1000 коробок.
2. **Формирование паллет и отправка DESADV.** Поставщик собирает 20 паллет, клеит на каждую уникальный 18-значный штрихкод **SSCC** и до выезда машины отправляет в сеть файл **DESADV** (уведомление об отгрузке), где внутри прописана связка: *«Паллета SSCC №1 = 50 коробок сока»*.
3. **Быстрая приемка на воротах (Сканирование SSCC).** Когда машина приезжает на склад сети, кладовщик не режет пленку на паллетах и не пересчитывает 1000 коробок. Он просто пикает терминалом сбора данных (**ТСД**) **20 паллетных этикеток SSCC**.
4. **Сверка и выставление счета (INVOIC).**
   - **Логика решения: ТСД** сопоставляет отсканированные коды паллет с присланным файлом **DESADV**. Если все 20 паллет на месте, склад отправляет поставщику подтверждение приемки (**RECADV**), и только после этого поставщик отправляет финальный электронный счет на оплату (**INVOIC**).

**Практикум: «Утренняя приемка в европейском распределительном центре»**

Вы работаете специалистом по учету и логистике (Inventory & Logistics Specialist) на складе. К разгрузочным воротам подъехала фура от поставщика, который привез два вида груза:

1. **10 паллет кофемашин** (по 20 коробок на каждой паллете, всего 200 штук). Документы пришли по **EDI**.
2. **2 паллеты импортного швейцарского сыра** (1000 кг). Груз пересек границу Евросоюза, поэтому он идет через систему **TRACES NT**.

При осмотре фуры кладовщик выявил **две проблемы**:

- **Проблема А (Недовоз).** Из 10 паллет с кофемашинами 9 паллет целые, а у 10-й паллеты порвана пленка и не хватает **5 кофемашин** (по факту приехало 195 штук вместо 200).
- **Проблема Б (Ошибка оператора).** При оформлении сыра в SAP вы по невнимательности оприходовали его на склад «Зона заморозки» (`FROZ`) вместо «Холодильника» (`CHIL`).

**Пошаговое решение с разъяснением логики**

**Шаг 1. Приемка паллет по EDI и сканирование кодов SSCC**

- **Подготовка в системе.** Накануне поставщик получил ваш заказ (`ORDERS`) и перед выездом машины прислал EDI-сообщение **`DESADV`** (уведомление об отгрузке), где перечислены 18-значные коды всех 10 паллет (**SSCC**).
- **Действия на воротах.** Кладовщик берет терминал сбора данных (RF-сканер) и пикает этикетки **SSCC** на первых 9 паллетах. Сканер мгновенно сверяет коды с файлом **`DESADV`** и принимает 180 кофемашин без вскрытия коробок.
- **Обработка 10-й паллеты.** Поскольку упаковка нарушена, кладовщик вскрывает 10-ю паллету и сканирует штрихкоды коробок поштучно, фиксируя реальное количество — **15 штук**.

**Шаг 2. Проведение прихода в SAP S/4HANA (Решаем Проблему А)**

- **Открытие транзакции.** Вы вводите в верхнюю строку SAP код транзакции **`MIGO`** (Движение материала) и выбираете операцию **Goods Receipt (Поступление)** с привязкой к номеру заказа (**`Purchase Order`**).
- **Ввод фактического количества.** Система автоматически подставляет из заказа цифру `200`, но вы меняете её на реальное количество — **`195`**.
- **Фиксация расхождения.** Вы нажимаете кнопку **Post (Провести)**. В эту секунду **SAP** ставит на баланс склада 195

кофемашин, а через **EDI** поставщику автоматически улетает сообщение **RECADV** (отчет о приемке с расхождением на 5 штук).

**Шаг 3. Подтверждение ветсертификата в TRACES NT (Приемка сыра)**

- **Проверка документа.** Вы заходите в браузере в систему **TRACES NT** и открываете входящий сертификат **CHED** (Единый медицинский документ на ввоз) на 1000 кг швейцарского сыра.
- **Сверка и гашение.** Вы сверяете номер пломбы на фуре, температуру в кузове и дату выработки на коробках. После проверки вы нажимаете кнопку **Validate (Подтвердить)**, тем самым официально закрывая таможенно-ветеринарный контроль груза.

**Шаг 4. Исправление ошибки со складом в SAP (Решаем Проблему Б)**

- **Запрет на удаление.** В **SAP** НЕЛЬЗЯ открыть проведенный документ и просто поменять склад **FROZ** на **CHIL**. Для исправления нужен строгий аудиторский след.
- **Оформление сторно.** Вы снова открываете транзакцию **MIGO**, выбираете режим **Cancellation (Сторнирование)** и вводите 10-значный номер ошибочного документа. **SAP** создает зеркальный документ со знаком минус (код движения **102**) и обнуляет сыр на складе заморозки.
- **Повторный ввод.** После сторнирования вы заново проводите обычный приход (**101** вид движения) на 1000 кг сыра, но теперь указываете правильный склад **CHIL**.

**Шаг 5. Тройная сверка и оплата счета через Peppol (Финал)**

- **Получение счета.** На следующий день бухгалтер получает от поставщика электронный счет **INVOIC** через международную сеть **Peppol**.
- **Работа 3-Way Match.** Бухгалтер запускает в **SAP** транзакцию **MIRO** (Проверка счетов). Если поставщик выставил счет на 195 кофемашин (по факту приемки), **SAP** видит совпадение заказа, складского прихода и счета, после чего разрешает банку перевести деньги.

**Практикум: «Приемка импортной партии продуктов и лекарств (TRACES NT + GS1 + EMVS)»**

**Вводная ситуация**

На логистический хаб прибыла сборная фура от зарубежного производителя, в которой везут два вида груза:

1. **500 кг замороженного лосося** (подконтрольный груз для ветеринарной системы **TRACES NT**).

2. **1 000 упаковок рецептурных лекарств** (подконтрольный груз для системы защиты от подделок **EMVS** и стандартов **GS1**).
При проверке на воротах выявились две задачи: нужно корректно закрыть санитарный контроль рыбы и проверить подлинность каждой пачки таблеток перед выпуском в продажу.

**Пошаговое решение с разъяснением логики**

**Шаг 1. Санитарный контроль и закрытие сертификата в TRACES NT**

- **Проверка номера сертификата.** До выезда фуры отправитель оформил в системе **TRACES NT** электронный документ **CHED-P** (единый медицинский документ на ввоз продукции животного происхождения). Оператор находит его в базе по номеру партии.
- **Сверка физических параметров.** Сотрудник сверяет номер пломбы на дверях контейнера, показания датчика температуры в кузове (должно быть не выше `-18 °C`) и маркировку на коробках с данными на экране.
- **Валидация документа.** Если все данные совпали, уполномоченный инспектор или оператор ставит электронную отметку **Validated (Подтверждено)**. В этот момент груз получает официальное разрешение на свободное обращение, а документ закрывается в системе.

**Шаг 2. Расшифровка транспортной этикетки по стандартам GS1**

- **Считывание логистической метки.** На коробках с лекарствами наклеены этикетки международного стандарта **GS1-128**. Оператор сканирует длинный штрихкод терминалом сбора данных.
- **Автоматический разбор данных (Application Identifiers).** Внутри одного штрихкода **GS1** зашиты сразу три параметра, разделенных специальными кодами в скобках:
  - `(01)` **GTIN** — глобальный код товара (что именно лежит в коробке).
  - `(10)` **Batch / Lot** — номер производственной партии.
  - `(17)` **Expiry Date** — точный срок годности в формате *ГГММДД*.
- **Логика процесса.** Программа мгновенно сверяет срок годности и номер партии с заказом, исключая ручной ввод цифр с клавиатуры.

**Шаг 3. Поштучная проверка и отпуск лекарств в системе EMVS**

- **Приемка на складе (Верификация).** На каждой индивидуальной пачке таблеток нанесен двухмерный квадрат **GS1 DataMatrix**, содержащий уникальный серийный номер `(21)` **Serial Number**. Кладовщик сканирует коробки, а система по защищенному каналу

проверяет в национальном репозитории (**NMVS**), числятся ли эти серийные номера как легальные и активные.

- **Обработка брака (Деактивация).** Если при разгрузке 2 пачки таблеток оказались помяты, их нельзя просто выбросить в мусорку — иначе их серийные номера останутся «висеть» в базе как активные. Сотрудник сканирует эти 2 пачки и переводит их в статус **Destroyed (Уничтожено)** или **Damaged (Повреждено)**.
- **Отпуск в аптеке (Вывод из оборота).** Когда оставшиеся 998 пачек попадают в аптеку, провизор при продаже покупателю сканирует DataMatrix на кассе. Система мгновенно меняет статус конкретного серийного номера с ***Active*** на **Supplied (Отпущено)** — повторно продать пачку с таким же кодом в любой другой точке становится технически невозможно.

В любой международной складской программе (**SAP**, **Odoo**, **WMS**) штрихкоды читаются по единым идентификаторам применения GS1:

| Код в скобках | Расшифровка (Термин GS1) | Что означает на практике |
|---|---|---|
| `(00)` | **SSCC** | Уникальный 18-значный номер всей паллеты или контейнера. |
| `(01)` | **GTIN** | Штрихкод самого товара (одинаковый для всех пачек одного вида). |
| `(10)` | **Batch / Lot** | Номер производственной партии (важно при отзыве брака). |
| `(17)` | **Expiration Date** | Срок годности товара (год, месяц, день). |
| `(21)` | **Serial Number** | Уникальный серийный номер конкретной пачки или бутылки. |

## Урок 1. Программирование в SAP: язык ABAP (Отчет по остаткам на складе)

В **SAP** весь код пишется на внутреннем языке **ABAP** (он чем-то напоминает смесь **SQL** и строгого паскаля/C++).

**Жизненная задача**

Кладовщику нужен простой отчет: программа должна зайти в главную складскую таблицу **SAP** (`MARD` — данные о запасах материалов), найти все товары, которых на складе `0001` осталось **меньше 10 штук**, и вывести их артикулы на экран.

**Код программы на ABAP с разбором:**

```
REPORT z_low_stock_alert. " Имя программы (пользовательские всегда начинаются с Z)

" 1. Объявляем внутреннюю таблицу
DATA: lt_stock TYPE TABLE OF mard,
      ls_stock TYPE mard. " Обычная переменная-строка для цикла

" 2. Делаем запрос прямо к базе данных SAP (Open SQL)
SELECT matnr labst
  FROM mard
  INTO CORRESPONDING FIELDS OF TABLE lt_stock
  WHERE werks = '1000'  " Код завода
    AND lgort = '0001'  " Код склада
    AND labst < 10.     " Свободный остаток меньше 10 штук

" 3. Проходим циклом по найденным товарам (аналог range-based for в C++)
LOOP AT lt_stock INTO ls_stock.
  WRITE: / 'Внимание! Мало товара:', ls_stock-matnr,
           'Остаток:', ls_stock-labst.
ENDLOOP.
```

**Урок 2. Программирование в Odoo: Python + ORM (Автоматическая проверка срока годности)**

В **Odoo** вся логика пишется на **Python**. Вместо прямых SQL-запросов здесь используется **ORM** (**объектно-реляционное отображение**) — вы работаете с таблицами базы данных как с обычными классами и объектами.

**Жизненная задача**

При приемке товара на склад кладовщик вводит срок годности партии. Нам нужно дописать правило: *если срок годности уже истек (меньше сегодняшней даты), программа должна заблокировать сохранение и выдать красную ошибку*.

**Код модуля на Python для Odoo:**

python

```
from odoo import models, fields, api
from odoo.exceptions import ValidationError
```

```
# Расширяем стандартную модель партий товара
(stock.lot)
class StockLot(models.Model):
    _inherit = 'stock.lot'

    # Декоратор @api.constrains запускает проверку при
сохранении карточки
    @api.constrains('expiration_date')
    def _check_expiration_date(self):
        today = fields.Date.today()

        for lot in self:
            # Если дата задана и она меньше сегодняшней
— блокируем операцию
            if lot.expiration_date and
lot.expiration_date < today:
                raise ValidationError(
                    f"Ошибка приемки! Срок годности
партии {lot.name} уже истек!"
                )
```

**Главная особенность Odoo:** Разработчик почти никогда не переписывает исходный код самой Odoo. Вместо этого используется механизм наследования (`_inherit = 'stock.lot'`), который аккуратно «приклеивает» вашу новую проверку поверх стандартной программы.

## Урок 3. Интеграция и обмен данными: Парсинг EDI / XML-сообщения DESADV

Когда склад получает по **EDI** электронное уведомление об отгрузке (**DESADV** — аналог накладной), внутри него лежит структурированный текст (**XML** или **JSON**). Задача программиста интеграций — «распарсить» (прочитать) этот файл и загрузить товары в учетную систему.

**Пример входящего XML-документа от поставщика:**

xml

```
<DespatchAdvice>
    <DocumentNumber>INV-2026-99</DocumentNumber>
    <Pallet SSCC="003460123456789012">
```

```
        <Item GTIN="04601234567890" Quantity="50" />
        <Item GTIN="04609876543210" Quantity="20" />
    </Pallet>
</DespatchAdvice>
```

**Как этот файл читается в коде (на примере Python):**

python

```
import xml.etree.ElementTree as ET

# Загружаем пришедший по EDI файл
tree = ET.parse('desadv_incoming.xml')
root = tree.getroot()

doc_num = root.find('DocumentNumber').text
print(f"Принята накладная № {doc_num}")

# Проходим по всем товарам внутри паллеты
for item in root.findall('.//Item'):
    gtin = item.attrib['GTIN']
    qty = int(item.attrib['Quantity'])
    print(f"Товар (GTIN): {gtin} — Количество: {qty}
шт.")
```

**Чтобы запретить приемку товара, у которого до истечения срока годности осталось меньше 3 дней, достаточно вычислить минимально допустимую дату (`today + timedelta(days=3)`) и сравнить с ней дату из карточки товара.**

**Обновленный код для Odoo (Python)**

python

```
from datetime import timedelta # Импортируем инструмент
для работы с интервалами времени
from odoo import models, fields, api
from odoo.exceptions import ValidationError

class StockLot(models.Model):
    _inherit = 'stock.lot'

    @api.constrains('expiration_date')
    def _check_expiration_date(self):
        # Вычисляем пограничную дату: сегодня + 3 дня
        min_allowed_date = fields.Date.today() +
timedelta(days=3)
```

```
        for lot in self:
            # Если срок годности заканчивается раньше,
чем через 3 дня — блокируем приемку
            if lot.expiration_date and
lot.expiration_date < min_allowed_date:
                raise ValidationError(
                    f"Ошибка приемки! У партии
{lot.name} осталось меньше 3 дней срока годности."
                )
```

**Пошаговый разбор логики**

1. **Подключаем `timedelta`.** В Python нельзя просто прибавить число `3` к дате (вызовет ошибку типов). Специальный объект `timedelta(days=3)` превращает число в отрезок времени длиной ровно трое суток.
2. **Считаем границу приемки (`min_allowed_date`).** Если сегодня **17 ноября**, то `min_allowed_date` будет равна **20 ноября**.
3. **Проверяем условие.** Если поставщик привез товар со сроком годности до **19 ноября**, условие `19 ноября < 20 ноября` станет истинным (`True`), и программа мгновенно остановит приемку с понятным сообщением для кладовщика.

**Практика по SAP ABAP (Автоматический расчет скидки при закупке)**

В **SAP** разработчикам часто ставят задачи по автоматизации закупок (модуль MM).

Представьте такую задачу: *при создании заказа поставщику (`ME21N`) компания получает автоматическую скидку **5%**, если сумма заказа превышает **10 000 евро**, и **10%**, если сумма больше **50 000 евро**.*

**Для расчета автоматической скидки в SAP ABAP используется конструкция ветвления `IF / ELSEIF / ELSE`, которая по логике полностью повторяет условия из вашего конспекта по C++.** Чтобы избежать потери копеек при делении, в ABAP используют специальный денежный тип данных `DECIMALS 2`.

**Код программы на языке ABAP (SAP)**

abap

```
REPORT z_po_discount_calc.

" 1. Объявляем переменные (аналог double в C++)
```

```
" TYPE p DECIMALS 2 — упакованное число с двумя знаками
после запятой (стандарт для денег в SAP)
DATA: lv_order_amount   TYPE p DECIMALS 2 VALUE
'65000.00', " Сумма заказа (65 000)
      lv_discount_pct   TYPE i VALUE 0,
" Процент скидки (целое число, как int)
      lv_discount_sum   TYPE p DECIMALS 2,
" Сумма скидки в деньгах
      lv_final_amount   TYPE p DECIMALS 2.
" Итоговая сумма к оплате

" 2. Проверяем условия скидки (начинаем с самого
большого порога!)
IF lv_order_amount > 50000.
  lv_discount_pct = 10.
ELSEIF lv_order_amount > 10000.
  lv_discount_pct = 5.
ELSE.
  lv_discount_pct = 0.
ENDIF.

" 3. Вычисляем сумму скидки и итог
lv_discount_sum = lv_order_amount * lv_discount_pct /
100.
lv_final_amount   = lv_order_amount - lv_discount_sum.

" 4. Выводим результат на экран SAP
WRITE: / 'Сумма заказа без скидки:', lv_order_amount,
       / 'Ваша скидка (%):         ', lv_discount_pct,
       / 'Размер скидки:           ', lv_discount_sum,
       / 'Итого к оплате:          ', lv_final_amount.
```

**Пошаговый разбор логики и «подводных камней»**

1. **Тип данных для денег (`TYPE p DECIMALS 2`).** В C++ мы использовали `double`, но в финансовых системах (SAP, 1C) дробные числа с плавающей точкой для денег запрещены из-за погрешностей округления. Тип `p` (***packed number*** — упакованное число) хранит ровно 2 знака после запятой с банковской точностью.
2. **Порядок проверок в `IF / ELSEIF`.** Обратите внимание, что мы сначала проверяем `> 50000`, и только потом `> 10000`. Если поменять их местами, то заказ на `65000` провалится в первую же ветку (`65000 > 10000`) и получит скидку 5% вместо положенных 10%.
3. **Синтаксис ABAP.** В отличие от C++, где блоки кода берутся в фигурные скобки `{ ... }`, в **ABAP** каждый блок закрывается

специальным словом с приставкой **END** (**IF ... ENDIF.**, **LOOP ... ENDLOOP.**), а в конце каждой команды ставится обычная точка **.** вместо точки с запятой **;**.

**C++ против ABAP**

| Конструкция | Как пишем в C++ | Как пишем в SAP ABAP |
|---|---|---|
| **Целое число** | `int count = 0;` | `DATA lv_count TYPE i VALUE 0.` |
| **Деньги / дробь** | `double price = 99.50;` | `DATA lv_price TYPE p DECIMALS 2 VALUE '99.50'.` |
| **Условие** | `if (x > 10) { ... } else { ... }` | `IF x > 10. ... ELSE. ... ENDIF.` |
| **Вывод на экран** | `std::cout << price << "\n";` | `WRITE: / lv_price.` |
| **Конец команды** | Точка с запятой **;** | Обычная точка **.** |

**Боевая задача: «Умный приемщик электронных накладных (API-обработчик)»**

**Бизнес-задача (как её ставит руководитель на работе)**

К нам от поставщиков по API прилетают электронные накладные в формате **JSON** (полный аналог **EDI-сообщения** `DESADV` или **УПД**). Нам нужно написать функцию **`process_incoming_delivery`**, которая:

1. **Защищает базу от дублей.** Если поставщик случайно отправил одну и ту же накладную дважды — отклоняет её.
2. **Проверяет каждый товар в списке:**
   - Существует ли такой артикул (**`GTIN`**) в нашей базе данных.
   - Не истек ли срок годности товара.
   - Не превышает ли количество разумный лимит (защита от опечатки, например `10000` вместо `10`).
3. **Разделяет приемку (Partial Receipt).** Хорошие товары ставит на баланс склада, а проблемные складывает в «Акт расхождения» для отправки поставщику, формируя итоговый отчет.

**Код программы с подробными комментариями**

python

```
from datetime import datetime, date

# Имитация нашей базы данных (в реальности это таблицы в SAP, 1С или PostgreSQL)
PROCESSED_DOCS = {"DOC-2026-001", "DOC-2026-002"}  # Уже принятые ранее накладные
WAREHOUSE_STOCK = {
    "4601234567890": {"name": "Молоко 3.2%", "qty": 50},
    "4609876543210": {"name": "Сыр Российский", "qty": 12},
}

def process_incoming_delivery(payload: dict) -> dict:
    doc_id = payload.get("document_id")

    # ШАГ 1. Защита от дублей (Идемпотентность — главное правило интеграций!)
    if doc_id in PROCESSED_DOCS:
        return {"status": "ERROR", "message": f"Накладная {doc_id} уже была проведена ранее!"}

    accepted_items = []    # Список принятых товаров
    discrepancies = []     # Акт расхождений (брак, просрочка, неизвестный товар)
    today = date.today()

    # ШАГ 2. Перебираем каждую строчку накладной
    for line in payload.get("items", []):
        gtin = line.get("gtin")
        qty = line.get("quantity", 0)
        exp_date_str = line.get("expiration_date")

        # Проверка А: Есть ли такой товар в нашей номенклатуре (мэппинг)?
        if gtin not in WAREHOUSE_STOCK:
            discrepancies.append({"gtin": gtin, "reason": "Неизвестный товар (не сопоставлен)"})
            continue  # Переходим к следующей строчке накладной

        # Проверка Б: Корректное ли количество?
```

```
        if qty <= 0 or qty > 1000:
            discrepancies.append({"gtin": gtin,
"reason": f"Подозрительное количество: {qty}"})
            continue

        # Проверка В: Контроль срока годности
        exp_date = datetime.strptime(exp_date_str, "%Y-
%m-%d").date()
        if exp_date <= today:
            discrepancies.append({"gtin": gtin,
"reason": f"Товар просрочен ({exp_date_str})"})
            continue

        # ШАГ 3. Если все проверки пройдены —
увеличиваем остаток на складе
        WAREHOUSE_STOCK[gtin]["qty"] += qty
        accepted_items.append({
            "name": WAREHOUSE_STOCK[gtin]["name"],
            "added_qty": qty,
            "new_total_stock":
WAREHOUSE_STOCK[gtin]["qty"]
        })

    # ШАГ 4. Запоминаем номер накладной, чтобы не
провести её второй раз
    PROCESSED_DOCS.add(doc_id)

    # Формируем ответ для отправки поставщику (аналог
RECADV в EDI)
    return {
        "document_id": doc_id,
        "status": "PARTIAL_ACCEPT" if discrepancies
else "FULL_ACCEPT",
        "accepted": accepted_items,
        "discrepancies": discrepancies
    }

# --- ТЕСТИРУЕМ НА РЕАЛЬНЫХ ДАННЫХ ---
incoming_json = {
    "document_id": "DOC-2026-003",
    "items": [
        {"gtin": "4601234567890", "quantity": 20,
"expiration_date": "2026-12-01"}, # Норма
        {"gtin": "4609876543210", "quantity": 5,
"expiration_date": "2025-01-10"}, # Просрочка!
```

```
        {"gtin": "9999999999999", "quantity": 10,
"expiration_date": "2026-12-15"}, # Неизвестный товар!
    ]
}

result = process_incoming_delivery(incoming_json)
print(result)
```

1. **Идемпотентность** (`if doc_id in PROCESSED_DOCS`). В реальной работе из-за сбоев интернета «Меркурий», «Честный ЗНАК» или EDI-провайдер часто присылают один и тот же документ дважды. Если разработчик не поставил проверку по ID документа, склад удвоит остатки, и придется делать инвентаризацию.
2. **Паттерн «Ранний выход»** (`continue`). Обратите внимание, как мы проверяем ошибки: вместо трехэтажных вложенных `if / else` мы проверяем каждую ошибку по очереди и сразу пишем `continue` (пропустить строчку и идти дальше). Это делает код чистым и понятным для коллег.
3. **Частичная приемка вместо падения всей накладной.** Если в накладной на 100 позиций 1 строчка оказалась просроченной, программа не должна «падать» с ошибкой и останавливать весь склад. Она принимает 99 хороших позиций, а 1 проблемную складывает в массив `discrepancies` (акт расхождения).
4. **Структурированный ответ** (`return {...}`). Функция не просто печатает текст на экран, а возвращает словарь (JSON-объект), который другая программа (например, 1С или веб-интерфейс) сразу отобразит в виде красивой таблицы.

# SAP

**1. Организационная структура (Где физически лежит товар)**
В **SAP** нельзя просто «принять 10 коробок на склад». Система всегда требует указать точные координаты компании сверху вниз — от главного холдинга до конкретной полки.

- **Мандант** (**Client**) — самый верхний уровень, изолированная база данных всей корпорации (например, `Mandant 100` — для реальной работы, `Mandant 200` — для обучения и тестов). Данные из одного манданта никогда не пересекаются с другим.

- **Балансовая единица (Company Code)** — отдельное юридическое лицо со своим **ИНН** и бухгалтерским **балансом** (например, ООО «Альфа» — код `1000`).
- **Завод (Plant)** — физический объект: фабрика, распределительный центр или крупный филиал (например, Завод в таком то городе — код `1010`).
- **Склад (Storage Location)** — конкретное помещение внутри завода, где хранится товар (например, `0001` — основной склад, `0002` — зона брака).

## 2. Главные модули SAP (Как поделены обязанности)

Вся система разделена на функциональные блоки.

| Код модуля | Расшифровка | За что отвечает в бизнесе |
|---|---|---|
| **MM** | Materials Management | Закупки, управление запасами, приемка товара и работа с поставщиками. |
| **SD** | Sales and Distribution | Продажи, заказы клиентов, отгрузка со склада и выставление счетов покупателям. |
| **WM / EWM** | Warehouse Management | Умное адресное хранение на складе (управление ячейками, погрузчиками и ТСД). |
| **FI / CO** | Financial Accounting / Controlling | Бухгалтерия, расчет налогов, банковские платежи и управленческий учет затрат. |
| **PP** | Production Planning | Производство: рецептуры (спецификации), техкарты и выпуск готовой продукции. |

## 3. Основные данные против Переменных (Главный закон SAP)

В SAP вся информация делится на два типа. Понимание этой разницы — база для любого аналитика и разработчика.

1. **Основные данные (Master Data)** — это **фундамент**, который заводится один раз и меняется редко.
   - *Примеры:* Карточка товара (**Material Master**), карточка поставщика (**Business Partner**), справочник складов.

- *Жизненная аналогия:* Это паспорт человека — фамилия и дата рождения не меняются каждый день.

2. **Переменные / Транзакционные данные (Transactional Data)** — это **ежедневное движение**, которое опирается на основные данные.
   - *Примеры:* Заказ на закупку, накладная на приемку, счет на оплату.
   - *Жизненная аналогия:* Это билет на поезд — в него вписывают данные из паспорта, но сам билет создается под конкретную поездку.

## 4. Главный процесс закупки (Цикл P2P: от заявки до оплаты)

В **SAP** этот стандартный процесс называется **Procure-to-Pay (P2P)**. Вот как он проходит через 4 последовательных документа и 4 транзакции:

1. **Заявка на закупку (`ME51N` — Purchase Requisition).** Начальник склада видит, что заканчивается бумага, и создает внутреннюю заявку для отдела закупок: *«Нам нужно 100 пачек бумаги к пятнице»*.
2. **Заказ поставщику (`ME21N` — Purchase Order).** Менеджер по закупкам выбирает поставщика, согласовывает цену и превращает заявку в официальный внешний заказ, который улетает поставщику.
3. **Поступление материала (`MIGO` — Goods Receipt).** Машина приехала на ворота. Кладовщик открывает `MIGO`, вводит номер заказа и проводит приемку. В этот момент SAP присваивает операции **Вид движения `101`** (стандартный приход на склад) и увеличивает остаток.
4. **Проверка и проводка счета (`MIRO` — Invoice Verification).** Бухгалтер получает счет от поставщика и вводит его в `MIRO`. Система автоматически сверяет три цифры: *цену в заказе (`ME21N`)*, *количество на складе (`MIGO`)* и *сумму в счете (`MIRO`)*. Если всё сошлось, счет уходит на оплату в банк.

## 5. Три главных «подводных камня» SAP для новичков

- **Ловушка «Виды движения» (Movement Types).** В SAP каждое действие с товаром имеет трехзначный цифровой код. Код `101` — это обычный приход, `102` — отмена прихода (сторно), `201` — списание на внутренние нужды, `311` — перемещение со склада на склад внутри одного завода. Перепутать код — значит отправить товар не на тот бухгалтерский счет.

- **Ловушка «Блокировка объекта» (Lock Object).** В SAP над одним заказом не могут работать два человека одновременно. Если ваш коллега открыл заказ на редактирование и ушел пить кофе, система заблокирует документ. Посмотреть, кто именно «держит» документ, и снять зависшую блокировку можно через транзакцию SM12.
- **Ловушка «Фоновые задания» (Background Jobs).** Тяжелые отчеты и обмен данными по EDI в SAP не запускаются кнопкой в реальном времени, чтобы не вешать систему. Они крутятся по расписанию ночью или каждые 15 минут. Проверить, не упал ли ночной обмен с поставщиками, можно в транзакции SM37.

# Урок 2. Главный цикл продаж: Order-to-Cash (O2C)

Если в модуле закупок (MM) мы покупаем товар у поставщиков, то в модуле продаж (**SD — Sales and Distribution**) мы продаем этот товар клиентам. Этот сквозной процесс называется **Order-to-Cash** (от заказа клиента до получения денег в кассу).

**Жизненная аналогия**

Представьте заказ в интернет-магазине:

1. Вы положили кроссовки в корзину и оформили заказ (на складе их для вас зарезервировали).
2. Кладовщик собрал коробку и передал её курьеру (кроссовки физически уехали со склада).
3. Вам на почту пришел электронный чек, а деньги списались с карты.

В SAP этот путь состоит из **3 главных документов и 3 транзакций**:

**Шаг 1. Заказ клиента (VA01 — Sales Order)**

- **Что делаем.** Менеджер по продажам принимает звонок от покупателя, открывает транзакцию VA01 и создает заказ: *«Клиент ООО „Вектор“ покупает 20 мониторов по 300 евро»*.
- **Что делает SAP под капотом.** Система автоматически запускает **проверку доступности (ATP — Available-to-Promise)**. Она заглядывает на склад, видит, что мониторы есть, и ставит на них «бронь», чтобы другой менеджер не продал их дважды. На этом этапе остаток на складе ещё не уменьшается.

**Шаг 2. Исходящая поставка и списание со склада (VL01N — Outbound Delivery)**

- **Что делаем.** Наступает день отгрузки. Кладовщик открывает транзакцию **VL01N**, привязывает её к заказу клиента, собирает 20 мониторов (пикинг) и нажимает главную кнопку — **Post Goods Issue (Проводка отпуска материала)**.
- **Что делает SAP под капотом.** В эту секунду срабатывает **вид движения 601** (отгрузка клиенту). Физический остаток мониторов на складе уменьшается на 20 штук, а в бухгалтерии автоматически формируется проводка о списании себестоимости товара.

**Шаг 3. Фактурирование / Выставление счета (VF01 — Billing Document)**

- **Что делаем.** После того как машина с мониторами уехала к клиенту, бухгалтер или оператор открывает транзакцию **VF01** и на основании поставки создает **фактуру (счет-фактуру / инвойс)**.
- **Что делает SAP под капотом.** Документ автоматически улетает в финансовый модуль (**FI**), где у клиента возникает официальный долг перед нашей компанией (дебиторская задолженность), который закроется, когда деньги придут на расчетный счет.

**Две главные ловушки в цикле продаж SAP (SD)**

- **Ловушка «Кредитный лимит» (Credit Block).** В **SAP** каждому клиенту можно задать предел долга (например, не более 10 000 евро). Если клиент ещё не оплатил прошлую поставку и пытается сделать новый заказ в **VA01**, система автоматически заблокирует документ. Снять эту блокировку может только финансовый контролер через транзакцию **VKM3**.
- **Ловушка «Запрет на фактурирование до отгрузки».** Вы не сможете выписать счет клиенту в **VF01**, пока кладовщик не нажмет кнопку отпуска товара в **VL01N**. **SAP** жестко следит за тем, чтобы компания не выставляла счета за товар, который всё ещё лежит на её собственном складе.

## Урок 3. Уровень «Немного сверху»: Как устроена база данных SAP (Главные таблицы)

Чтобы вырасти из обычного пользователя в **Junior-аналитика или ABAP-разработчика**, нужно знать, где физически лежат все эти заказы и остатки.

В **SAP** вся информация хранится в реляционных таблицах, названия которых состоят из **4 немецких букв** (потому что **SAP** создали в Германии). Посмотреть любую таблицу изнутри можно через транзакцию **SE16N**.

**Карточка товара (Material Master):**

- **MARA** — общие данные о товаре для всей корпорации (артикул, вес, единица измерения, группа товара).
- **MARC** — данные о товаре на конкретном заводе (какой завод производит или закупает этот товар).
- **MARD** — текущие остатки товара на конкретном складе (именно из этой таблицы мы брали остатки в нашем коде на ABAP).

**Закупки (Purchase Orders):**

- **EKKO** — «шапка» заказа поставщику (номер заказа, дата, код поставщика, валюта).
- **EKPO** — «строки» (позиции) заказа поставщику (какие именно товары заказаны, в каком количестве и по какой цене).

**Продажи (Sales Orders):**

- **VBAK** — «шапка» заказа клиента (кто покупает, дата доставки).
- **VBAP** — «строки» заказа клиента (список проданных товаров).

**Жизненная аналогия для запоминания таблиц:** Представьте бумажный чек из магазина. Верхняя часть чека (название магазина, дата, время) — это таблица **шапки** (**EKKO** или **VBAK**). А список покупок ниже (хлеб, молоко, сыр) — это таблица **позиций** (**EKPO** или **VBAP**). Они связаны между собой по номеру документа.

**Практика для закрепления (Уроки 2 и 3)**

Представьте, что вы работаете Junior-аналитиком поддержки SAP. К вам обращается менеджер по продажам с жалобой:
*«Я создал заказ клиента в VA01, всё сохранилось. Потом передал его на склад, кладовщик в VL01N собрал коробки и отгрузил их. Но когда бухгалтер пытается выставить счет в VF01, система пишет ошибку: „Документ не релевантен для фактурирования“ (отгрузка не завершена). Что забыл сделать кладовщик внутри VL01N?»*

**Кладовщик забыл нажать кнопку Post Goods Issue (Проводка отпуска материала) внутри документа исходящей поставки (VL01N / VL02N).** Без этого действия система считает, что коробки всё ещё лежат на складе, и блокирует выставление счета бухгалтером.

**Пошаговый разбор ошибки и как её исправить**

1. **Что произошло на самом деле.** Кладовщик создал документ поставки в VL01N, вписал количество собранных коробок (выполнил пикинг) и просто нажал «Сохранить» (иконку дискеты). В SAP

сохранение поставки создает только черновик накладной, но **не списывает товар со склада**.

2. **Почему бухгалтер получил ошибку в VF01.** В SAP действует жесткий контроль: нельзя выставить счет (VF01) за товар, который официально не покинул ворота склада. Пока не нажата кнопка отпуска, статус поставки висит как *«Не проведено»*.
3. **Как исправить ситуацию:**
   - Открываем транзакцию VL02N (Изменение исходящей поставки) и вводим номер документа.
   - Проверяем, что в графе *Количество комплектования (Picked Qty)* стоит нужное число.
   - Нажимаем главную кнопку **Post Goods Issue (Проводка ОМ)** в верхней панели.
   - В эту секунду срабатывает **вид движения 601**, остаток на складе уменьшается, а бухгалтер может спокойно открывать VF01 и выставлять счет.

## Урок 4. Производство (PP), адресный склад (EWM) и уровень «Сверху» (Отладка)

Двигаемся дальше по программе SAP: разберем, как на заводе из сырья делают готовую продукцию, как работает умный склад и как аналитики ищут ошибки под капотом программы.

**1. Производство в SAP (Модуль PP — Production Planning)**

По логике этот модуль повторяет производственную транзакцию из «Меркурия», но на уровне крупного завода. Он держится на двух главных справочниках (Master Data):

- **Спецификация (BOM — Bill of Materials, транзакция CS01).** Это **рецептура** готового изделия. Например, чтобы собрать 1 велосипед, нужно взять: 1 раму, 2 колеса, 1 руль и 1 цепь.
- **Технологическая карта (Routing, транзакция CA01).** Это **список операций и время работы станков**. Например: 15 минут на сварку рамы в цехе №1, 20 минут на покраску и 10 минут на сборку.

**Как проходит выпуск продукции (CO01 → MIGO):**

1. Плановик создает **Производственный заказ (CO01)** на 100 велосипедов. SAP сам умножает рецептуру (BOM) на 100 и автоматически резервирует на складе 200 колес и 100 рам.
2. Когда сборка завершена, оператор проводит выпуск продукции (вид движения **101** для готовых велосипедов и **261** для списания использованных колес и рам).

**2. Умный склад в SAP (Модуль EWM — Extended Warehouse Management)**

Если обычный модуль **MM** знает только, что на складе **0001** лежит 500 колес, то **EWM** знает **точное местоположение каждой коробки с точностью до сантиметра**.

- **Складское место (Storage Bin).** Самая маленькая единица склада — конкретная ячейка на стеллаже (например, `01-02-04`: Ряд 1, Стеллаж 2, Полка 4).
- **Складская задача (Warehouse Task).** Электронный приказ для терминала сбора данных (**ТСД**) кладовщика: *«Возьми 1 паллету из зоны приемки `GR-ZONE` и отвези её в ячейку `01-02-04`»*.

**3. Уровень «Немного сверху»: Инструменты поиска ошибок (Для аналитика и разработчика)**

Когда в **SAP** что-то ломается или бухгалтерия не может закрыть месяц, специалисты используют три «секретных» инструмента диагностики:

- **Просмотр IDoc-сообщений (Транзакция `WE02`).** Именно в формате **IDoc** (**Intermediate Document**) **SAP** общается с внешним миром — получает **EDI**-заказы от торговых сетей и отправляет данные на склады. Если накладная от поставщика «потерялась» по дороге в **SAP**, аналитик открывает **`WE02`**, видит красный светофор и читает текст технической ошибки.
- **Анализ «дампов» / падений программы (Транзакция `ST22`).** Если пользователь нажал кнопку, а **SAP** внезапно вылетел с критической ошибкой (**ABAP Runtime Error**), система записывает подробный снимок памяти в **`ST22`**. Разработчик открывает этот лог и видит точную строчку кода, на которой споткнулась программа.
- **Отладчик кода (Debugger — команда `/h`).** Самый знаменитый прием в **SAP**: если перед нажатием любой кнопки в программе вбить в верхнюю командную строку **`/h`** и нажать Enter, SAP остановит выполнение прямо перед действием и откроет код программы построчно, позволяя аналитику или программисту заглянуть внутрь переменных в реальном времени.

1. **Структура:** Мандант → Балансовая единица → Завод → Склад.
2. **Закупки (MM):** `ME51N` (Заявка) → `ME21N` (Заказ) → `MIGO` (Приемка, код `101`) → `MIRO` (Счет).
3. **Продажи (SD):** `VA01` (Заказ клиента) → `VL01N` (Отгрузка, код `601`) → `VF01` (Фактура).
4. **Таблицы БД (`SE16N`):** `MARA` / `MARD` (товары и остатки), `EKKO` / `EKPO` (закупки), `VBAK` / `VBAP` (продажи).
5. **Производство и диагностика:** `BOM` (рецептура), `EWM` (ячейки), `WE02` (обмен EDI), `ST22` (логи ошибок), `/h` (отладчик).

**Кейс 1. «Зависшая приемка на воротах» (Модуль MM + Закупки)**
Водитель привез 100 коробок товара. Кладовщик открывает транзакцию **MIGO**, вводит номер заказа поставщику (Purchase Order), но SAP выдает ошибку: *«Заказ не содержит релевантных позиций»* (или не дает провести приемку). Вы проверяете заказ в ME23N и видите, что ваш коллега из отдела закупок открыл этот заказ 20 минут назад, ушел на обед и оставил окно открытым на своем компьютере.

- **Вопрос:** Через какую транзакцию вы проверите, кто именно заблокировал заказ, и снимете зависшую блокировку?

**Разбор Кейса 1. «Зависшая приемка на воротах» (Блокировка объекта)**
**Короткий ответ для конспекта:** Чтобы проверить, кто заблокировал документ, и снять зависшую блокировку, используется транзакция **SM12**.
**Пошаговое решение и рассуждения аналитика:**

1. **В чем суть проблемы.** В SAP работает механизм защиты данных: если один сотрудник открыл заказ на изменение (ME22N), система ставит на этот заказ «цифровой замок» (**Lock Entry**). Пока замок висит, склад не может принять товар по этому заказу в MIGO.
2. **Открываем транзакцию SM12.** Вводим в командную строку SM12 (Просмотр и удаление блокировок).
3. **Находим зависший документ.** В поле *Имя пользователя* стираем свое имя (чтобы увидеть блокировки всех коллег) и нажимаем **Enter**. В списке находим строку с номером нашего заказа и логином закупщика, который ушел на обед.
4. **Снимаем блокировку.** Выделяем нужную строчку и нажимаем иконку **Корзины (Удалить)**. Сразу после этого кладовщик может обновить **MIGO** и провести приемку.

 **Важное правило безопасности:** Перед удалением блокировки в SM12 всегда проверяйте время её создания (и желательно свяжитесь с коллегой). Если удалить блокировку в ту секунду, когда человек реально редактирует документ, можно повредить данные в базе.

**Кейс 2. «Потерянный электронный заказ от торговой сети» (Интеграции + EDI)**

Крупный клиент отправил вам электронный заказ по EDI, но в модуле продаж (VA03) новый заказ не появился. Руководитель просит срочно проверить, дошло ли вообще это сообщение до SAP и на какой строчке оно споткнулось.

- **Вопрос:** Какую транзакцию нужно открыть для проверки входящих EDI-сообщений (IDoc) и поиска текста ошибки?

**Чтобы починить «красный» IDoc в SAP, аналитик открывает внутреннюю структуру сообщения в транзакции WE02, находит сломанный сегмент данных, устраняет причину в справочнике и повторно запускает документ через BD87 или WE19.** Это один из самых востребованных навыков для специалиста поддержки и Junior-консультанта.

**1. Как устроен IDoc изнутри (Жизненная аналогия)**

Представьте, что **IDoc** — это **заказное бумажное письмо в конверте**, которое пришло от клиента в ваш офис. Любой IDoc в SAP всегда состоит из трех обязательных частей:

1. **Управляющая запись (Control Record — «Конверт»).** Здесь написано, *кто* отправил письмо (код клиента), *кому* оно предназначено (ваш завод) и *что это за документ* (например, ORDERS05 — заказ на продажу).
2. **Записи данных (Data Records — «Лист бумаги внутри конверта»).** Это сама начинка заказа, разбитая на блоки-сегменты. Например, сегмент E1EDK01 хранит шапку заказа, а сегменты E1EDP01 и E1EDP19 — конкретные строчки товаров, их количество и артикулы.
3. **Записи статусов (Status Records — «Штампы почты»).** История прохождения документа: во сколько он пришел, кто его проверял и на каком этапе произошла ошибка.

---

**2. Пошаговый алгоритм: Как найти и починить ошибку в Кейсе 2**

Допустим, торговая сеть прислала заказ, а в WE02 он горит красным цветом (статус 51). Вот как шаг за шагом действует аналитик:

**Шаг 1. Читаем текст ошибки в WE02**

- Открываем красный IDoc двойным кликом в транзакции **WE02**.
- В левом дереве раскрываем папку **Записи статусов (Status Records)** и нажимаем на самую нижнюю красную строчку.

- Внизу экрана появляется текст: *«Материал клиента ART-9988 не найден в таблице соответствия для клиента 500120»*.
- **Логика:** Торговая сеть прислала заказ со своим внутренним артикулом ART-9988, а SAP не знает, какой наш товар скрывается под этим кодом.

**Шаг 2. Проверяем, что именно прислал клиент в сегментах**

- В том же окне WE02 раскрываем папку **Записи данных (Data Records)**.
- Находим сегмент **E1EDP19** (идентификация товара) и нажимаем на иконку просмотра.
- Видим, что клиент заказал 100 штук товара ART-9988. Мы связываемся с менеджером по продажам и узнаем, что под кодом ART-9988 клиент подразумевает наше «Молоко 3.2%» (код в SAP 100450).

**Шаг 3. Устраняем причину в справочнике SAP (VD51)**

- Чтобы **SAP** научился понимать артикулы этого клиента (полный аналог сопоставления номенклатуры в 1С), аналитик открывает транзакцию **VD51** (Инфо-запись «Клиент-Материал»).
- Вписывает связку: *Клиент 500120* + *Артикул клиента ART-9988* = *Наш материал 100450* и нажимает **Сохранить**.

**Шаг 4. Повторно проталкиваем заказ в систему (BD87)**

- Открываем транзакцию **BD87** (Обработка IDoc), вводим номер нашего красного **IDoc** и нажимаем **Выполнить (F8)**.
- Выделяем строчку со статусом 51 и нажимаем кнопку **Обработать (Process)**.
- Поскольку связка в справочнике теперь есть, **SAP** успешно переваривает документ, меняет светофор на зеленый (статус **53**) и автоматически создает заказ клиента в VA01.

**3. Секретный инструмент аналитика: Тестовая утилита WE19**

Иногда клиент присылает **IDoc** с опечаткой внутри самого файла (например, забыл указать дату доставки или перепутал единицу измерения КГ вместо ШТ). Ждать, пока клиент перешлет правильный файл, долго.

Для таких случаев в **SAP** есть транзакция **WE19** (Инструмент тестирования **IDoc**):

1. Вы вводите номер сломанного **IDoc** в WE19.
2. Программа открывает все сегменты письма в режиме **прямого редактирования**.
3. Вы руками исправляете опечатку прямо внутри сегмента и нажимаете кнопку **Стандартный вход (Standard Inbound)**.
4. **SAP** создает точную копию этого **IDoc** с исправленными данными и проводит заказ в базу.

**Главная ловушка утилиты WE19:** Когда вы исправляете документ через WE19, SAP создает **новый IDoc с новым номером**, а старый красный IDoc остается висеть в базе. Чтобы в отчетах не болтались «мертвые» ошибки, старый красный IDoc нужно перевести в статус 68 (Обработка не требуется) через стандартную программу **RBDMANI2** или отчет **RC1_IDOC_SET_STATUS**.

**Шпаргалка по главным статусам IDoc**

| Код статуса | Цвет светофора | Расшифровка | Что делать аналитику |
|---|---|---|---|
| **53** | Зеленый | Документ успешно проведен | Всё отлично, заказ или накладная уже в системе. |
| **51** | Красный | Ошибка приложения | Открыть WE02, прочитать текст ошибки, исправить справочник и перезапустить в BD87. |
| **64** | Желтый | Готов к передаче в приложение | Документ дошел до SAP, но ждет своей очереди в фоновом задании (SM37). |
| **03** | Зеленый | Данные переданы в порт (исходящий) | Ваш ответ успешно улетел из SAP во внешнюю систему или EDI-провайдеру. |

**Часть 1. Как разделены роли в SAP («Всё вместе или отдельно?»)**
В отличие от небольших внедрений 1С, где один программист часто и настраивает склад, и пишет код, и чинит принтеры, в мире SAP труд четко поделен на **три профессии**:

1. **Функциональный консультант / Аналитик (Без программирования)**
   - **Что делает:** Общается с бизнесом, настраивает логику складов, закупок и продаж через специальные меню настроек (**SPRO**), разгребает ошибки в **IDoc** и пишет технические задания (**ТЗ**) для программистов.
   - **Зачем ему знать код:** Чтобы включить отладчик (**/h**), прочитать чужой код и понять, почему программа выдала ошибку, но сам с нуля он программы не пишет.
2. **ABAP-разработчик (Чистое программирование)**

- **Что делает:** Получает готовое **ТЗ** от консультанта и пишет код на языке **ABAP** — создает новые отчеты, печатные формы накладных, сложные проверки при приемке и шлюзы обмена данными. В настройки складов он обычно не лезет.

3. **Basis-администратор (Системный инженер)**
   - **Что делает:** Следит за серверами, производительностью базы данных, обновлениями системы и управляет учетными записями и правами пользователей.

## Урок 5: Права доступа и полномочия (SU01, SU53, PFCG)

Разбор ошибок с правами доступа — это ежедневная рутина любого специалиста поддержки и Junior-аналитика.

**Жизненная аналогия**

Представьте огромный офисный центр с электронными пропусками. У охранника на входе есть пропуск только на 1-й этаж, у бухгалтера — на 3-й этаж и в архив, а у директора — везде. Если кладовщик попытается зайти в бухгалтерию, турникет пискнет красным и не пустит.

В **SAP** у каждого сотрудника есть своя **Учетная запись**, к которой привязаны **Роли** (наборы разрешений: какие транзакции можно запускать и какие заводы просматривать).

**3 главных инструмента для работы с правами**

1. **SU01 (Ведение пользователей).** Главный пульт администратора. Здесь создают нового сотрудника, задают ему временный пароль, разблокируют учетную запись (если сотрудник три раза ввел неверный пароль) и прикрепляют нужные роли.
2. **PFCG (Ведение ролей).** Конструктор пропусков. Здесь консультант или администратор собирает «роль» — например, *«Кладовщик склада 0001»*, куда разрешает доступ только к транзакциям **MIGO** и **VL01N**, но запрещает **MIRO** (финансовые счета).
3. **SU53 (Анализ ошибок полномочий) — главный инструмент спасателя.** Когда пользователь пытается нажать кнопку, а **SAP** пишет внизу красным: *«У вас нет полномочий на выполнение этой операции»*, именно **SU53** мгновенно показывает, какого именно «ключа» не хватило.

**Пошаговая практика: «У кладовщика не работает приемка товара»**

**Вводная ситуация**

К вам в поддержку звонит новый кладовщик: *«Я открываю транзакцию `MIGO`, ввожу номер заказа, нажимаю Провести, а система выдает ошибку полномочий и не дает принять товар!»*.

**Как аналитик решает эту задачу за 3 минуты:**

1. **Просим пользователя запустить SU53.** Говорим кладовщику: *«Ничего не закрывай, вбей в верхнюю командную строку `/nSU53` и нажми Enter, а потом пришли мне скриншот экрана»*.
   - *(Примечание: приставка `/n` перед любой транзакцией закрывает текущее зависшее окно и сразу открывает новое).*
2. **Читаем отчет в SU53.** На скриншоте мы видим красную строчку с техническим кодом ошибки — это **Объект полномочий** (Authorization Object). Например, написано: `M_MSEG_WMB` (Полномочия на завод), значение `WERKS = 1010`.
3. **Расшифровываем причину.** Это означает, что кладовщику выдали права на приемку товара на заводе `1000`, а заказ оформлен на новый филиал — завод `1010`, куда у него нет доступа.
4. **Передаем задачу на добавление прав.** Вы передаете этот скриншот администратору безопасности (или сами добавляете завод `1010` в роль кладовщика через **PFCG**), после чего приемка сразу начинает работать.

### Итоговая шпаргалка по безопасности SAP

| Транзакция | Назначение | Кто и когда использует |
|---|---|---|
| **SU01** | Карточка пользователя | Сбросить забытый пароль, снять блокировку после 3 неудачных попыток входа. |
| **SU53** | Проверка последней ошибки прав | Первый шаг при любой жалобе пользователя «Система не пускает / нет прав». |
| **PFCG** | Редактор ролей | Настроить список разрешенных транзакций, заводов и складов для должности. |
| **/n + код** | Быстрый переход | Команда в верхней строке (например, `/nMIGO`), закрывающая текущее окно. |
| **/o + код** | Открыть в новом окне | Команда (например, `/oSE16N`), открывающая новую вкладку SAP без закрытия старой. |

# Урок 6. Настройка системы без программирования (Кастомизация через SPRO)

Когда компания покупает «чистый» **SAP**, он похож на пустую квартиру без мебели. Чтобы система начала работать по правилам конкретного бизнеса (например, запретила принимать товар без номера заказа или добавила новый склад в таком то городе), **функциональный консультант** открывает главный пульт настроек — транзакцию **SPRO**.

**Жизненная аналогия**

Представьте настройки в вашем смартфоне: вы не переписываете код операционной системы, а просто передвигаете ползунки (включить Wi-Fi, поменять мелодию звонка, запретить уведомления). В **SAP** всё устроено точно так же, только вместо 20 пунктов меню здесь **тысячи веток дерева настроек**, которое называется **IMG (Implementation Guide — Руководство по внедрению)**.

**Как устроена работа консультанта в SPRO (3 главных принципа)**

1. **Дерево настроек (SPRO → SAP Reference IMG).** Консультант раскрывает нужную ветку, например: *Управление материальными потоками (MM) → Закупки → Заказ на поставку → Определение параметров экрана*.
2. **Транспортные запросы (Transport Requests — транзакция SE09 / SE10).** В **SAP** нельзя менять настройки сразу в боевой базе, где работают люди. Вся работа идет по строгому конвейеру из трех систем (**Ландшафт SAP**):
   - **DEV (Development — Система разработки).** Здесь консультант меняет настройки в SPRO, а программист пишет код. Любое изменение автоматически записывается в специальный контейнер — **Транспортный запрос**.
   - **QAS (Quality Assurance — Тестовая система).** Запрос переносится сюда, чтобы аналитик и заказчик проверили, ничего ли не сломалось.
   - **PRD (Production — Продуктивная / Боевая система).** Только после успешного теста администратор переносит запрос в реальную базу, где сидят кладовщики и бухгалтеры.
3. **Настройка полей документа (Обязательные / Скрытые / Только чтение).** Самая частая задача консультанта — изменить поведение полей на экране. Например, сделать поле «Срок годности» в **MIGO** **обязательным для заполнения**, чтобы кладовщик физически не мог провести приемку без даты.

**Пошаговая практика: «Открытие нового склада в компании»**
**Вводная ситуация**
Компания арендовала новое помещение под бракованный товар на заводе `1000`. Руководитель ставит задачу консультанту **SAP**: *«С завтрашнего утра кладовщики должны иметь возможность принимать и перемещать брак на новый склад с кодом `0099` и названием „Склад брака“»*.
**Как консультант решает задачу в `SPRO` за 4 шага:**

1. **Заходим в пульт настроек.** Вводим транзакцию **`SPRO`** и нажимаем кнопку **SAP Reference IMG**.
2. **Идем по ветке оргструктуры.** Раскрываем дерево: *Структура предприятия → Определение → Управление материальными потоками → Ведение склада (Storage Location)*.
3. **Создаем новый склад.** Вводим код завода **`1000`**, нажимаем кнопку **Новые записи (New Entries)**, вписываем код **`0099`** и название **`Склад брака`**.
4. **Сохраняем в Транспортный запрос.** Нажимаем «Сохранить» (дискету). **SAP** выводит окно с просьбой выбрать **Транспортный запрос**. Мы создаем новый запрос с понятным именем *«Создание склада брака 0099 для завода 1000»*, сохраняем и передаем номер запроса на перенос из **DEV** в **QAS** и **PRD**.

**Две главные ловушки настройки SAP (`SPRO`)**

- **Ловушка «Прямая правка в боевой системе (PRD)».** В продуктивной системе **`SPRO`** закрыто на замок — менять настройки напрямую там запрещено. Исключение составляют только «текущие настройки» (например, курсы валют или открытие нового бухгалтерского периода в транзакции **`MMPV`**), которые меняются каждый месяц.
- **Ловушка «Забытый транспортный запрос».** Если консультант сделал 10 настроек в **`DEV`**, проверил их там, но забыл перенести свой запрос в боевую систему **`PRD`** (или перенес их в неправильном порядке), у пользователей в понедельник утром вылетят критические ошибки.

**Шпаргалка по новым транзакциям**

| Транзакция | Назначение простыми словами | Зачем нужна в работе |
|---|---|---|
| **`SPRO`** | Главное меню настроек SAP (IMG) | Настройка складов, заводов, видов документов и обязательных полей без кода. |
| **`SE09` / `SE10`** | Органайзер | Контроль переноса настроек и |

| | | |
|---|---|---|
| | транспортных запросов | кода из тестовой среды (DEV) в боевую (PRD). |
| **MMPV** | Закрытие/открытие периода склада | Ежемесячная операция: разрешает кладовщикам проводить документы в новом месяце. |

## Урок 7. Интеграция склада и бухгалтерии (Модуль FI и автоматические проводки)

В небольших программах кладовщик принимает товар в своей базе, а бухгалтер в конце месяца вручную переносит цифры в бухгалтерскую программу. В **SAP** модули логистики (**MM / SD**) и финансов (**FI — Financial Accounting**) сшиты намертво: в ту секунду, когда кладовщик нажимает кнопку «Провести» на складе, SAP в фоновом режиме сам пишет проводку в бухгалтерию.

**Жизненная аналогия**

Представьте кассовый аппарат в магазине. Кассир просто сканирует штрихкод пачки молока и нажимает «Оплата», но внутри программы одновременно происходят два события: со склада списывается 1 пачка молока (складской учет), а в денежный ящик виртуально прибавляется 100 рублей (финансовый учет). В **SAP** этот механизм называется **автоматическим определением счетов (транзакция OBYC)**.

**Как работает связка «Склад → Бухгалтерия» по шагам**

Когда кладовщик принимает 10 мониторов по 200 евро (итого 2 000 евро) в транзакции **MIGO**, **SAP** создает **два связанных документа**:

1. **Документ материала (Material Document)** — фиксирует физику: *«На склад 0001 приехало 10 мониторов»*.
2. **Бухгалтерский документ (Accounting Document)** — фиксирует деньги:
   - **Дебет (Плюс на складе):** Счет запасов (товаров) увеличивается на `+2 000 EUR`.
   - **Кредит (Наш долг поставщику):** Временный транзитный счет **ПМ/ПС (GR/IR — Goods Receipt / Invoice Receipt)** увеличивается на `-2 000 EUR`.

**Зачем нужен транзитный счет ПМ/ПС (GR/IR)?**

В жизни машина с товаром часто приезжает во вторник, а бумажный или электронный счет на оплату бухгалтер получает только в пятницу. Счет **GR/IR** служит «мостом»: во вторник он фиксирует, что товар уже у нас, но счет ещё не пришел. В пятницу, когда бухгалтер проводит счет в **MIRO**, транзитный счет **GR/IR** закрывается в ноль, а долг переносится на реальный счет конкретного поставщика.

**Главные финансовые таблицы и транзакции**

Если в бухгалтерии не сходятся цифры с остатками на складе, аналитик проверяет финансовые таблицы в **SE16N**:

- **BKPF** — «шапка» бухгалтерского документа (номер документа, дата проводки, валюта, код балансовой единицы).
- **BSEG** — «строки» бухгалтерского документа (конкретные бухгалтерские счета, суммы по дебету и кредиту).
- **FB03** — транзакция для красивого просмотра любого бухгалтерского документа (из MIGO в неё можно провалиться напрямую, нажав кнопку *«Финансовые документы»*).
- **OBYC** — главный пульт настройки связи склада и бухгалтерии (здесь консультант прописывает, на какой именно бухгалтерский счет падают деньги при приемке сырья, готовой продукции или брака).

**Две главные финансовые ловушки в SAP**

- **Ловушка «Закрытый бухгалтерский период» (Ошибка в OB52).** В SAP есть два независимых календаря: складской (MMPV) и бухгалтерский (OB52). Даже если склад открыл новый месяц, но бухгалтер забыл открыть период в OB52, кладовщик при приемке товара в MIGO получит красную ошибку: *«Период проводки 011 2026 не открыт»*.
- **Ловушка «Ненастроенный класс оценки» (Valuation Class).** У каждого товара в карточке (MARA / MBEW) указан его «класс оценки» (например: *Сырье*, *Готовый товар*, *Упаковка*). Именно по этому коду SAP понимает, на какой счет в OBYC положить деньги. Если при создании нового товара оператор забыл указать класс оценки, кладовщик не сможет принять этот товар на склад.

## (Урок 8): Отладка кода своими руками (Команда `/h`)

Это финальный штрих, который превращает обычного пользователя в специалиста поддержки или Junior-аналитика. Представьте, что SAP выдал непонятную ошибку, и вам нужно заглянуть «под капот» программы, не имея прав программиста.

**Пошаговый алгоритм работы с отладчиком (`/h`):**

1. Открываем проблемную транзакцию (например, **`MIGO`** или **`VA01`**) и заполняем все поля, но **последнюю кнопку «Сохранить» или «Провести» пока не нажимаем**.
2. В верхнюю командную строку (где обычно пишем коды транзакций) вводим **`/h`** и нажимаем **Enter**. Внизу экрана появится сообщение: *«Включена отладка»*.
3. Теперь нажимаем ту самую кнопку **«Провести»**. Программа замирает, и открывается окно **ABAP Debugger** с исходным кодом.
4. Нажимаем клавишу **`F8`** (Выполнить до точки остановки) или ставим **точку прерывания (Breakpoint)** на команду `MESSAGE` (в меню *Breakpoints → Breakpoint at → Breakpoint at Message*).
5. Программа сама прокрутит тысячи строк кода и **остановится ровно на той строчке `IF ...`, где сработала ваша ошибка**. Аналитик видит конкретное условие (например: `IF lv_plant <> '1000'. MESSAGE e001...`), и причина проблемы становится ясна за 10 секунд.

1. **Архитектура:** Мандант → БЕ → Завод → Склад (`SPRO`).
2. **Логистика закупок (MM):** `ME51N` → `ME21N` → `MIGO` (`101`) → `MIRO` (счет `GR/IR`).
3. **Логистика продаж (SD):** `VA01` → `VL01N` (`601`) → `VF01`.
4. **Интеграции и диагностика:** `WE02` / `BD87` (IDoc), `SM12` (блокировки), `SU53` (права), `SE16N` (таблицы `MARA`, `MARD`, `EKKO`, `BKPF`), `/h` (отладчик).

# JAVA

1. **Шаг 1. Входные данные (Сбор информации).** Мы определяем, какие «коробки» (переменные) нам нужны и откуда придут данные (с клавиатуры от пользователя или из базы).

2. **Шаг 2. Анализ и логика (Мозги программы).** Мы решаем, что сделать с этими данными: посчитать по формуле, сравнить (`больше / меньше / равно` через **`if`**) или повторить действие много раз в цикле (**`for`** / **`while`**).
3. **Шаг 3. Вывод результата (Ответ нам).** Мы красиво печатаем итог на экран.

## Урок 1. Каркас программы на Java и конвейер «Вход → Логика →Выход»

Главное отличие **Java** от **C++** в том, что в **Java** **всё обязано лежать внутри класса (`class`)**. В **C++** вы могли написать функцию **`main()`** просто в пустом файле, а в **Java** даже самая маленькая программа оборачивается В КЛАСС (**class**).

**Разберем на живой задаче: «Калькулятор доставки»**

**Условие:** Пользователь вводит сумму заказа в рублях. Если заказ от **2000 рублей** и выше — доставка бесплатная (`0` руб.), если меньше — доставка стоит **300 рублей**. Программа должна посчитать и вывести итоговую сумму к оплате.

**Код программы на Java (разложенный по шагам):**

```
import java.util.Scanner; // 1. Подключаем инструмент для чтения с клавиатуры (аналог #include <iostream>)

public class DeliveryCalculator { // В Java весь код всегда живет внутри класса

    public static void main(String[] args) { // Главная точка входа (аналог int main() в C++)

        // ========================================
        // ШАГ 1. ВХОДНЫЕ ДАННЫЕ (Готовим "коробки")
        // ========================================
        Scanner scanner = new Scanner(System.in); // Включаем "слушатель" клавиатуры (аналог std::cin)

        System.out.print("Введите сумму заказа: "); // Просим ввести число (аналог std::cout)
        double orderSum = scanner.nextDouble();     // Считываем дробное число в переменную

        // ========================================
```

```
        // ШАГ 2. АНАЛИЗ И ЛОГИКА (Думаем и сравниваем)
        // ==========================================
        int deliveryCost = 0; // Заводим коробку для цены доставки

        if (orderSum >= 2000) {
            deliveryCost = 0;   // Условие "больше или равно" — доставка бесплатная
        } else {
            deliveryCost = 300; // Условие "меньше" — доставка 300 рублей
        }

        double totalAmount = orderSum + deliveryCost; // Считаем общую сумму

        // ==========================================
        // ШАГ 3. ВЫВОД РЕЗУЛЬТАТА (Печатаем ответ)
        // ==========================================
        System.out.println("Стоимость доставки: " + deliveryCost + " руб.");
        System.out.println("Итого к оплате: " + totalAmount + " руб.");

        scanner.close(); // Хорошая привычка: закрываем сканер в конце
    }
}
```

| Что хотим сделать | Как пишем в C++ | Как пишем в Java |
|---|---|---|
| Вывести текст с переносом строки | `std::cout << "Привет\n";` | `System.out.println("Привет");` |
| Вывести текст без переноса | `std::cout << "Привет";` | `System.out.print("Привет");` |
| Считать целое число | `int x; std::cin >> x;` | `int x = scanner.nextInt();` |
| Считать строку с пробелам | `std::getline(std::cin, str);` | `String str = scanner.nextLine();` |

| и | | |
|---|---|---|
| **Строка текста (тип данных)** | `std::string name = "Иван";` | `String name = "Иван";` ***(с большой буквы!)*** |
| **Условие (больше / меньше)** | `if (a > b) { ... } else { ... }` | `if (a > b) { ... } else { ... }` ***(один в один!)*** |
| **Цикл со счетчиком** | `for (int i = 0; i < 5; i++)` | `for (int i = 0; i < 5; i++)` ***(один в один!)*** |

**1. Вывод с переносом строки против вывода без переноса (`println` vs `print`)**

- **Как распознать в условии (слова-маркеры):**
  - «**Выведи списком**», «**каждое значение с новой строки**», «**выведи отчет**» → берем `System.out.println(...)`.
  - «**Выведи в одну строку через пробел**», «**сделай приглашение к вводу, чтобы курсор остался рядом**» → берем `System.out.print(...)`.
- **Практический пример:**

java
```
System.out.print("Введите пин-код: "); // Курсор мигает сразу после двоеточия на этой же строке
// А если выводим чек:
System.out.println("Товар 1: Хлеб");   // Перенесет курсор вниз
System.out.println("Товар 2: Молоко"); // Напечатается на новой строке
```

**2. Чтение чисел против чтения текста с пробелами (`nextInt()` / `nextDouble()` vs `nextLine()`)**

- **Как распознать в условии (слова-маркеры):**
  - «Пользователь вводит возраст, количество штук, номер месяца» (целое число) → `scanner.nextInt()`.
  - «Вводится цена, вес в килограммах, температура, процент скидки» (дробное число) → `scanner.nextDouble()`.

- «**Вводится ФИО, адрес доставки, название товара, комментарий**» (текст из нескольких слов) → **`scanner.nextLine()`**.

**Практический пример (с учетом ловушки буфера):**

java
```
Scanner scanner = new Scanner(System.in);

System.out.print("Сколько коробок приехало? ");
int boxes = scanner.nextInt(); // Считали число
scanner.nextLine();            // ОБЯЗАТЕЛЬНО очистили оставшийся Enter!

System.out.print("Введите комментарий кладовщика: ");
String comment = scanner.nextLine(); // Теперь спокойно читаем целую фразу с пробел
```

**3. Условия: больше, меньше, равно (`if / else if / else`)**

- **Как распознать в условии (слова-маркеры):**
  - «**Если... то... иначе...**», «**при превышении лимита**», «**проверить попадание в диапазон**», «**выбрать тариф в зависимости от веса**».
  - Увидел слова **«если»**, **«в случае»**, **«не менее»**, **«превышает»** → стопроцентный **`if`**.
- **Практический пример (Контроль температуры в холодильнике на складе):**

java
```
double temp = 4.5; // Температура в кузове машины

// Шаг 1: проверяем норму (от 0 до +2 градусов)
if (temp >= 0 && temp <= 2) {
    System.out.println("Температура в норме. Принимаем товар.");
}
// Шаг 2: если не норма, проверяем другую ветку (слишком тепло)
else if (temp > 2) {
    System.out.println("БРАК! Нарушен температурный режим, оформляем возврат.");
}
// Шаг 3: всё остальное (меньше 0 — товар замерз)
else {
    System.out.println("ОШИБКА! Товар заморожен.");
}
```

**4. Сравнение строк (`.equals()` вместо `==`)**

- **Как распознать в условии (слова-маркеры):**
  - «Проверить пароль», «**если пользователь ввел слово `stop`**», «если статус заказа равен **`PAID`**», «если роль — **`ADMIN`**».
  - Как только сравниваем **слова (текст)** → забываем про **`==`** и ставим **`.equals()`**.
- **Практический пример (Проверка статуса накладной):**

java
```
String status = "PAID"; // Статус из базы данных

// Правильно: спрашиваем у строки status, совпадает ли она со словом "PAID"
if (status.equals("PAID")) {
    System.out.println("Заказ оплачен, можно отгружать со склада!");
} else {
    System.out.println("Ожидаем оплату.");
}
```

**5. Циклы (`for` против `while`)**

- **Как распознать в условии (слова-маркеры):**
  - **Цикл `for` (со счетчиком):** «Повторить 10 раз», «пройтись по всем товарам в списке», «посчитать сумму за 7 дней недели», «вывести числа от 1 до N». То есть **заранее известно количество шагов**.
  - **Цикл `while` (пока верно условие):** «Спрашивать пароль, пока не введет верный», «считывать товары, пока пользователь не введет `0`», «работать, пока на складе есть остаток». То есть **заранее неизвестно, сколько попыток понадобится**.

**Практический пример (Приемка 5 коробок по очереди через `for`):**

java
```
// Знаем точное число коробок (5 штук) — берем for
for (int i = 1; i <= 5; i++) {
    System.out.println("Сканируем коробку №" + i + " — принята!");
}
```

*Программа просит кассира вводить цены товаров с клавиатуры (дробные числа) **до тех пор, пока он не введет число** 0. В конце программа выводит общую сумму чека.»*

1. Какой метод `Scanner` возьмем для чтения цены? → `nextDouble()` (так как цена дробная).
2. Какой цикл возьмем (`for` или `while`)? → `while`, потому что мы не знаем заранее, сколько товаров в корзине у покупателя (3 или 50) — ориентируемся на маркер *«до тех пор, пока не введет 0»*.

СТРУКТУРА КОДА:

1. **Где хранить данные? (Память).** Если значение одно — берем переменную (`int`, `double`, `String`). Если значений много (список товаров в накладной, цены в чеке) — берем коллекцию (`ArrayList`).
2. **Сколько раз повторить? (Циклы).** Если нужно обработать весь список товаров — ставим цикл (`for` или `while`).
3. **Что отфильтровать или проверить? (Условия).** Внутри цикла проверяем каждый элемент через `if` (просрочен ли товар, превышает ли цена лимит).
4. **Что отдать в итоге? (Результат).** Сохраняем итог в переменную-накопитель (сумму, счетчик брака) и выводим на экран или возвращаем из метода.

## Урок 2. Списки (`ArrayList`) и обработка потоков данных

В реальной работе никто не заводит 50 переменных `price1`, `price2`… `price50`. Для хранения списка однотипных данных в Java используется `ArrayList` — это полный аналог `std::vector` из вашего конспекта по C++.

**Как распознать в условии, что нужен `ArrayList`:**

- Слова-маркеры: *«список покупок»*, *«база сотрудников»*, *«история заказов»*, *«сохранить все введенные числа»*, *«найти самый дорогой товар среди всех»*.

**Главные команды `ArrayList` (Сравнение с C++ `std::vector`):**

- Создать список: `ArrayList<Double> prices = new ArrayList<>();` *(в C++: `std::vector<double> prices;`)*
- Добавить элемент в конец: `prices.add(150.5);` *(в C++: `prices.push_back(150.5);`)*
- Узнать размер списка: `prices.size();` *(один в один как в C++!)*
- Достать элемент по индексу: `prices.get(0);` *(в C++: `prices[0]` или `prices.at(0)`)*

**Практическая рабочая задача: «Анализатор накладной на складе»**
**Условие задачи**
Кладовщик сканирует цены пришедших товаров (дробные числа) и вводит их в программу. Когда товары заканчиваются, он вводит `0`.
Программа должна:

1. Сохранить все цены в список.
2. Посчитать **общую сумму** накладной.
3. Найти **самый дорогой товар** (максимум).
4. Посчитать, **сколько товаров дороже 1000 рублей** (подпадают под особый контроль).

**Ход мыслей программиста перед написанием кода:**

1. *Память:* Количество товаров заранее неизвестно → создаем `ArrayList<Double> prices`.
2. *Ввод:* Читаем цены до нуля → берем цикл `while (true)` с выходом `break`, если ввели `0`.
3. *Логика:* Проходим по списку циклом `for-each`. Заводим три «коробки» для итогов: `totalSum` (сумма), `maxPrice` (максимум) и `expensiveCount` (счетчик дорогих товаров).
4. *Вывод:* Печатаем красивые итоги через `System.out.println`

**Готовый код на Java:**
java

```
import java.util.ArrayList;
import java.util.Scanner;

public class InvoiceAnalyzer {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // ==========================================
        // ШАГ 1. ВХОДНЫЕ ДАННЫЕ (Собираем список)
        // ==========================================
        ArrayList<Double> prices = new ArrayList<>();
        System.out.println("Вводите цены товаров
(введите 0 для завершения):");

        while (true) {
            double inputPrice = scanner.nextDouble();
            if (inputPrice == 0) {
                break; // Увидели 0 — выходим из цикла
ввода
            }
```

```
            if (inputPrice > 0) {
                prices.add(inputPrice); // Кладем цену
в наш динамический список
            }
        }

        // Защита от пустого ввода (если сразу нажали
0)
        if (prices.isEmpty()) {
            System.out.println("Список товаров пуст.");
            return;
        }

        // ========================================
        // ШАГ 2. АНАЛИЗ И ЛОГИКА (Считаем метрики)
        // ========================================
        double totalSum = 0;
        double maxPrice = prices.get(0); // Берем
первый товар как временный максимум
        int expensiveCount = 0;

        // Современный цикл по списку (как range-based
for в C++)
        for (double price : prices) {
            totalSum += price; // Прибавляем к общей
сумме

            if (price > maxPrice) {
                maxPrice = price; // Нашли товар дороже
— обновляем максимум
            }

            if (price > 1000) {
                expensiveCount++; // Считаем товары
дороже 1000 руб.
            }
        }

        // ========================================
        // ШАГ 3. ВЫВОД РЕЗУЛЬТАТА
        // ========================================
        System.out.println("\n--- Отчет по накладной --
-");
        System.out.println("Всего позиций: " +
prices.size());
```

```
            System.out.println("Общая сумма: " + totalSum +
    " руб.");
            System.out.println("Самый дорогой товар: " +
    maxPrice + " руб.");
            System.out.println("Товаров дороже 1000 руб.: "
    + expensiveCount + " шт.");

            scanner.close();
        }
    }
```

**Две важные ловушки при работе со списками (`ArrayList`)**

1. **Нельзя писать примитивы в угловых скобках.** В C++ вы писали `std::vector<int>` или `std::vector<double>`. В Java в угловых скобках нельзя писать `int` или `double` с маленькой буквы — компилятор выдаст ошибку. Всегда используем классы-обертки с большой буквы: **`ArrayList<Integer>`**, **`ArrayList<Double>`**, **`ArrayList<String>`**.
2. **Проверка на пустой список (`prices.isEmpty()`).** Прежде чем писать `prices.get(0)` (взять первый элемент), всегда проверяйте, не пустой ли список. Если список пуст, попытка достать нулевой элемент «уронит» программу с ошибкой `IndexOutOfBoundsException`.

**Компьютер не умеет догадываться — он делает ровно то, что вы прописали в условиях, а любая «дыра» в логике превращается в баг.** Если вы описали ветку «больше 1000» и «меньше 1000», но забыли про случай «ровно 1000», программа либо проигнорирует это число, либо пойдет по неверному пути.

Чтобы никогда не упускать такие стыки и писать железобетонную логику, программисты используют **три главных правила проверки условий**.

**Правило 1. «Стыки на границах» (Строгое `>` против нестрогого `>=`)**
Представьте шкалу чисел, разбитую на зоны (например, скидка зависит от суммы чека: до 1000 руб. — нет скидки, от 1000 до 5000 руб. — скидка 5%, свыше 5000 руб. — скидка 10%).
Если написать так:

```
if (sum < 1000) { ... }
else if (sum > 1000 && sum < 5000) { ... }
```

else if (sum > 5000) { ... }

**Где здесь дыра?**
Если покупатель наберет товаров ровно на **1000** или ровно на **5000** рублей, ни одно из условий не сработает!

**Как правильно (метод сплошного забора):** Границы должны плотно примыкать друг к другу без зазоров с помощью знаков **<=** или **>=** :

```
if (sum < 1000) {
    // Всё, что от 0 до 999.99
} else if (sum < 5000) {
    // Сюда попадет всё от 1000.00 до 4999.99 (потому что < 1000 уже отсеялось выше!)
} else {
    // Сюда попадет всё от 5000.00 и до бесконечности
}
```

**Правило 2. «Защита от дурака» (Валидация на самом входе)**
В нашем коде выше было условие `if (inputPrice > 0)`. Зачем мы его поставили?
Потому что пользователь или кассир может случайно нажать минус и ввести `-500` рублей. Если не отсечь отрицательные числа сразу на входе, программа вычтет `-500` из общей суммы накладной, и бухгалтерия получит неверный отчет.
**Золотой шаблон на входе (Guard Clause — «Охранник на воротах»):**
Перед тем как считать формулы, всегда проверяйте: *а адекватные ли данные вообще пришли?*

- Цена или вес не могут быть отрицательными (`price < 0`).
- Список товаров не должен быть пустым (`prices.isEmpty()`).
- Делитель не может быть равен нулю (`count == 0`).

**Правило 3. «Изолированные проверки» против «Взаимоисключающих веток»**
Посмотрите на код из предыдущего урока — там внутри цикла стояли **два отдельных** `if`, а не `if / else if`:

```
// Проверка 1: Ищем самый дорогой товар
if (price > maxPrice) {
    maxPrice = price;
}
```

```
// Проверка 2: Считаем товары дороже 1000 руб.
if (price > 1000) {
    expensiveCount++;
}
```

**Почему здесь нельзя было ставить `else if`?**
Потому что это **два независимых вопроса** к одному и тому же товару! Допустим, приехал товар за `2500` рублей. Он **одновременно** и самый дорогой (**`> maxPrice`**), и дороже тысячи (**`> 1000`**). Если бы мы написали **`else if`**, программа обновила бы максимум, но **пропустила бы** счетчик дорогих товаров.

- **Используем `if` / `else if` / `else`**, когда нужно выбрать **только один** вариант из нескольких (например: тариф *Эконом*, *Стандарт* или *Премиум* — товар не может быть в двух тарифах сразу).
- **Используем несколько отдельных `if` подряд**, когда товар может подходить под **несколько правил одновременно** (например: и дорогой, и хрупкий, и импортный).

Когда вы написали все условия, мысленно прогоните через программу **4 контрольных числа**:

1. **Нормальное число** из середины (например, `1500`).
2. **Пограничное число** ровно на стыке (например, `1000` или `0`).
3. **Некорректное число** (например, `-50` или пустую строку).
4. **Огромное число** (например, `1000000`)

# Урок 3. Методы (Функции) в Java: собираем программу из аккуратных кирпичиков

Представьте, что вы открыли ресторан, и один повар одновременно чистит картошку, жарит мясо, моет посуду и рассчитывает гостей на кассе. Через час на кухне начнется хаос. В программировании то же самое: если писать весь код на 500 строк внутри одного **`main()`**, разобраться в нем будет невозможно.
Чтобы навести порядок, код разбивают на **Методы** (в C++ они назывались *Функциями*). Каждый метод — это «узкий специалист», который отвечает строго за **одну конкретную задачу**.
**Как распознать в условии, что нужен отдельный метод:**

- Слова-маркеры: *«вынеси проверку в отдельную функцию»*, *«напиши метод, который принимает... и возвращает...»*, или когда одно и то же действие (например, расчет скидки или проверка пароля) повторяется в программе 3 раза в разных местах.

В **Java** метод всегда пишется внутри класса (обычно рядом с **main**). Пока мы работаем без создания объектов, перед типом возвращаемого значения ставится слово **static**.

**1. Метод, который считает и возвращает ответ (return)**

- **Жизненная аналогия:** Вы отдаете калькулятору два числа (цену и скидку), он внутри себя считает и выдает вам готовый ответ.
- **Синтаксис:** **public static ТипОтвета имяМетода(Тип1 параметр1, Тип2 параметр2)**

java

```
// Метод принимает цену и процент скидки, а возвращает итоговую цену (double)
public static double calculateDiscountPrice(double price, int discountPercent) {
    // Защита на входе (Guard Clause): если скидка некорректная — возвращаем исходную цену
    if (discountPercent <= 0 || discountPercent > 100) {
        return price;
    }
    double discountAmount = price * (discountPercent / 100.0);
    return price - discountAmount; // Возвращаем результат и мгновенно выходим из метода
}
```

**2. Метод-действие, который ничего не возвращает (void)**

- **Жизненная аналогия:** Вы нажимаете кнопку «Распечатать чек». Принтер печатает бумажку, но в руки вам никакое число обратно не возвращает.
- **Синтаксис:** Вместо типа ответа пишем слово **void** (пустота)

```
// Метод просто красиво печатает шапку чека на экран
public static void printReceiptHeader(String storeName) {
   System.out.println("================================");
   System.out.println("   МАГАЗИН: " + storeName);
   System.out.println("================================");
}
```

**Практическая рабочая задача: «Валидатор приемки товара на складе»**

Соберем всё вместе: разделим программу на понятные методы, соблюдая наш конвейер **«Вход → Логика → Выход»**.

**Условие задачи**

Написать программу для приемки партии товара. Вынести проверку качества в отдельный метод `isBatchValid`, который принимает **температуру в кузове** (`double temp`) и **срок годности в днях** (`int daysLeft`), а возвращает `boolean` (`true` — партия годная, `false` — брак).

**Четкие правила бизнеса (без дыр на границах):**

1. Температура должна быть строго в диапазоне **от `0.0` до `+4.0` градусов включительно**.
2. До конца срока годности должно оставаться **не менее `5` дней** (`daysLeft >= 5`).

```
import java.util.Scanner;

public class WarehouseValidator {

    public static void main(String[] args) {
        // ==========================================
        // ШАГ 1. ВХОДНЫЕ ДАННЫЕ
        // ==========================================
        Scanner scanner = new Scanner(System.in);

        System.out.print("Введите температуру в кузове (°C): ");
        double truckTemp = scanner.nextDouble();

        System.out.print("Сколько дней осталось до конца срока годности? ");
        int shelfLifeDays = scanner.nextInt();

        // ==========================================
        // ШАГ 2. ВЫЗОВ МЕТОДА ПРОВЕРКИ (Логика спрятана внутри!)
        // ==========================================
        boolean isAccepted = isBatchValid(truckTemp, shelfLifeDays);

        // ==========================================
        // ШАГ 3. ВЫВОД РЕЗУЛЬТАТА
```

```
        // ========================================
        if (isAccepted) {
            System.out.println("РЕЗУЛЬТАТ: Партия
прошла контроль! Разгружаем на склад.");
        } else {
            System.out.println("РЕЗУЛЬТАТ: Ошибка
приемки! Оформляем возврат поставщику.");
        }

        scanner.close();
    }

    // Отдельный метод-эксперт, который отвечает только
за проверку правил
    public static boolean isBatchValid(double temp, int
daysLeft) {
        // Проверяем температуру (от 0.0 до 4.0
включительно)
        boolean isTempOk = (temp >= 0.0 && temp <=
4.0);

        // Проверяем остаточный срок годности (не менее
5 дней)
        boolean isDateOk = (daysLeft >= 5);

        // Возвращаем true только если ОДНОВРЕМЕННО
выполнены оба условия (&&)
        return isTempOk && isDateOk;
    }
}
```

**Два главных плюса такого кода (Зачем мы вынесли `isBatchValid`)**

1. **Читаемость метода `main`.** Посмотрите на `main()`: он читается как обычная инструкция на русском языке — *считали температуру, считали дни, вызвали `isBatchValid`, напечатали результат*.
2. **Легкость изменений на работе.** Если завтра руководство скажет: *«Теперь принимаем товар только со сроком годности от 7 дней»*, вам не придется перерывать всю программу. Вы откроете один маленький метод `isBatchValid` и поменяете цифру `5` на `7` в одной строчке.

**Важное отличие от C++:** В C++ вам приходилось писать прототипы функций над `main()`, потому что компилятор читал файл строго сверху вниз. В **Java** порядок не важен: метод `isBatchValid` можно написать как **под** `main()`, так и **над** ним — компилятор сам найдет его внутри класса.

**Границы класса в Java жестко задаются фигурными скобками** `{ ... }`

**1. Как определять границы класса (Физически и логически)**
Чтобы никогда не путаться, где начинается и заканчивается класс, опирайтесь на **три простых правила**:

- **Правило одной пары скобок** `{ ... }`**.** Класс начинается со строки `public class ИмяКласса {` и заканчивается самой последней закрывающей скобкой `}` в конце файла. Всё, что написано внутри этих скобок, принадлежит этому классу.
- **Правило запрета на «голый» код.** Внутри границ класса могут лежать только **переменные (свойства)** и **методы (действия)**. Написать `if` или `for` просто внутри класса нельзя — компилятор выдаст ошибку, так как любая логика обязана жить внутри какого-то метода.
- **Правило «Один файл = Один класс».** В реальной разработке каждый публичный класс лежит в своем отдельном файле с таким же именем. Например, класс `Product` лежит в файле `Product.java`, а класс `Order` — в файле `Order.java`.

Как понять, где провести логическую границу между двумя классами?

- В Java действует принцип (**Принцип единственной ответственности**). Если данные описывают товар (**название**, **цена**, **срок годности**) — это границы класса `Product`. Если данные описывают поставщика (**ИНН**, **адрес**, **телефон**) — это границы другого класса `Supplier`.

**2. Как разбивают большие программы (Сравнение 1С и Java)**
Если представить программу на 50 000 строк, то циклы там находятся на самом нижнем уровне. Вся архитектура делится на **4 этажа сверху вниз**:

1. **Пакеты (`Packages`) — как разделы меню в 1С.** Это обычные папки на диске, которые группируют файлы по смыслу. Например, папка `warehouse` (склад), папка `accounting` (бухгалтерия) и папка `integration` (обмен данными).

2. **Классы (Classes) — как справочники, документы и общие модули в 1С.** Внутри каждой папки лежат классы двух видов: классы-данные (карточка товара, накладная) и классы-обработчики (общие модули в 1С, которые считают налоги или проверяют брак).
3. **Методы (Methods) — как процедуры и функции внутри модуля 1С.** Внутри каждого класса лежит 5–10 коротких методов по 15–20 строк. Один метод проверяет температуру, второй считает скидку, третий сохраняет документ.
4. **Циклы и условия (`for`, `while`, `if`) — кирпичики внутри метода.** Они живут только внутри конкретного метода, чтобы перебрать список товаров или сравнить две цифры.

**3. Архитектура «Трех слоев» (Как строят любую рабочую программу)**

Когда программист садится писать реальный проект (хоть интернет-магазин, хоть складской учет), он всегда делит классы на **три независимых слоя**:

- **Слой 1. Модели данных (`Model` / `Entity`).** Классы-коробки, которые просто хранят информацию без сложной логики. Примеры — `Product` **(Товар)**, `Invoice` **(Накладная)**.
- **Слой 2. Бизнес-логика (`Service`).** Классы-работяги (полный аналог общих модулей в 1С), где живут все правила компании. Например, класс `WarehouseService` содержит методы «принять партию», «списать просрочку» и «проверить остаток».
- **Слой 3. Ввод и вывод (`Controller` / `UI` / `Main`).** Классы, которые общаются с человеком (читают кнопки с экрана или клавиатуры), передают данные в слой логики и печатают красивый ответ.

// Схематичный пример границ двух разных классов в одном проекте:

```
// 1. Границы класса-данных (файл Product.java)
public class Product {
    String name;      // Поле класса (свойство)
    double price;     // Поле класса (свойство)
} // <-- Здесь класс Product полностью закончился!

// 2. Границы класса-логики (файл PriceService.java)
public class PriceService {
    // Метод живет строго внутри границ PriceService
    public double applyDiscount(double price) {
        if (price > 1000) { // Условие живет строго внутри метода
            return price * 0.9;
```

```
        }
        return price;
    } // <-- Конец метода
} // <-- Здесь класс PriceService полностью закончился!
```

## Урок 4. Классы и Объекты на практике (Воплощаем вашу схему в коде)

В этом уроке мы превратим вашу схему из трех частей в рабочий код.

- **Класс (`Class`)** — это **чертеж** или пустой бланк анкеты (например, «Карточка товара»).
- **Объект (`Object`)** — это **конкретный экземпляр**, созданный по этому чертежу (например, «Молоко 3.2% за 90 руб.» или «Сыр Российский за 650 руб.»).

**Собираем класс `Product` строго по вашей структуре:**

1. **Часть 1. Шапка / Свойства (Поля класса)** — описываем, *из чего состоит* товар.
2. **Часть 2. Сборка (Конструктор)** — специальный метод, который заполняет поля при создании нового товара.
3. **Часть 3. Действия и условия (Методы)** — вся математика и проверки (`if`), которые умеет делать товар.

Код программы: Класс `Product` и запуск в `Main`

```
//
==========================================================
// ФАЙЛ 1: Product.java (Чертеж товара — ваша 3-частная схема)
//
==========================================================
public class Product {

    // --- ЧАСТЬ 1. ШАПКА / ИДЕНТИФИКАЦИЯ СУЩНОСТИ (Поля класса) ---
    String name;      // Название товара
    double price;     // Цена за 1 штуку
    int quantity;     // Количество на складе

    // --- ЧАСТЬ 2. КОНСТРУКТОР (Как товар рождается в памяти) ---
    // Имя конструктора ВСЕГДА совпадает с именем класса!
    public Product(String name, double price, int quantity) {
        this.name = name;         // Слово "this." означает: "запиши в поле ЭТОГО объекта"
        this.price = price;
        this.quantity = quantity;
```

```
    }

    // --- ЧАСТЬ 3. ДЕЙСТВИЯ И УСЛОВИЯ (Логика задачи) ---
    // Действие 1: Посчитать общую стоимость партии с учетом скидки
    public double calculateTotalCost() {
        double total = price * quantity;

        // Условие бизнеса: если берем 10 штук и больше — даем скидку 10%
        if (quantity >= 10) {
            total = total * 0.9;
        }
        return total; // Возвращаем готовую цифру наверх
    }

    // Действие 2: Проверить, пора ли дозаказывать товар (осталось меньше 5 шт.)
    public boolean needReorder() {
        return quantity < 5; // Вернет true (да, пора) или false (нет, еще хватит)
    }
}
```

**Как выглядит этот конвейер шаг за шагом**

Когда вы начинаете писать любую функцию в **Java**, держите в голове именно эту цепочку из четырёх этапов:

- **1. Входные данные (Приём заказа):** Метод получает значения в круглых скобках, например `(int amount)`. Это сырьё, с которым предстоит работать.
- **2. Защитные условия / Фильтры (`if`):** Сразу на входе мы отсекаем весь мусор и неверные сценарии. Если данные плохие (например, сумма меньше нуля), мы сразу выводим ошибку и прерываем работу через команду `return`.
- **3. Основная логика (Полезная работа):** Если все проверки пройдены и данные безопасны, мы спокойно выполняем нужное действие — вычитаем баланс, меняем координаты героя или сохраняем файл.
- **4. Вывод и завершение:** Показываем результат пользователю на экране (или возвращаем готовое значение в другую часть программы), после чего метод заканчивает работу.

**Жизненная аналогия: проходная на заводе**

Представьте охранника на входе в секретный цех:

1. **Вход:** Подошёл человек с пропуском.
2. **Условия (`if`):** Охранник проверяет: пропуск не просрочен? Фотография совпадает? Если нет — сразу «**До свидания**» (`return`), дальше человек не идёт.
3. **Логика:** Если всё чисто, охранник открывает турникет и записывает время входа в журнал.
4. **Вывод и завершение:** На экране турникета загорается зелёная стрелка «**Проходите**».

**Почему такой порядок доводит код до автоматизма**
Когда вы всегда ставите проверки `if` в самом начале метода, код не превращается в запутанную «лесенку» из вложенных условий. Вы точно знаете: если программа дошла до середины метода, значит, все входные данные уже проверены и ничего не сломается.

## Тема: Наследование и Полиморфизм (Шаблоны и универсальные кнопки)

В реальной разработке никто не пишет каждый класс с нуля, если объекты похожи друг на друга. Для этого придумали наследование (чтобы не дублировать код) и полиморфизм (чтобы управлять разными вещами одной командой).

**1. Наследование (`extends`) — «Чертёж-основа и доработка»**

- **Жизненная аналогия:** Представьте базовый чертёж автомобиля. У любой машины есть руль, колёса и кнопка «Завестись». Чтобы сделать грузовик или гоночный болид, инженеры не рисуют руль и колёса заново — они берут базовый чертёж и просто дописывают к нему кузов или спойлер.
- **Как работает в Java:** Мы создаём общий родительский класс `Vehicle` (Транспорт) с общими полями и методами. Затем создаём дочерний класс `Truck` (Грузовик) с помощью слова `extends` («расширяет»). Он автоматически получает всё, что умеет базовый транспорт, плюс свои уникальные функции.

**2. Полиморфизм — «Одна кнопка для разных устройств»**

- **Жизненная аналогия:** Пульт управления или педаль газа. Вы нажимаете на педаль газа в легковой машине, грузовике или электрокаре. Действие одно и то же («нажать педаль»), но под капотом у каждого транспорта срабатывает свой механизм: где-то льётся бензин, где-то подаётся ток.
- **Как работает в Java:** Родительский класс объявляет общее действие (например, `move()` — двигаться). Каждый наследник

переписывает этот метод под себя с помощью пометки `@Override` («переопределяю»). Программа может собрать в один список разные машины и просто дать всем команду `move()`, не вникая, кто именно перед ней.

```
// 1. БАЗОВЫЙ ЧЕРТЁЖ (Родитель)
public class Vehicle {
    public void move() {
        System.out.println("Транспорт едет");
    }
}

// 2. НАСЛЕДНИК (Берёт всё от Vehicle и меняет поведение под себя)
public class SportsCar extends Vehicle {

    // Полиморфизм: та же команда move(), но работает по-своему
    @Override
    public void move() {
        System.out.println("Спорткар мчится со скоростью 300 км/ч!");
    }
}
```

**Главный смысл для проектирования**

- **Наследование** спасает от копипасты: всё общее выносим В ОДИН родительский класс.
- **Полиморфизм** спасает от гигантских цепочек `if-else`: вместо того чтобы проверять «если это машина, то так, если самолёт, то этак», мы просто вызываем общую команду `move()`, и каждый объект сам знает, как её выполнить.

**Полиморфизм нужен для того, чтобы основной код был коротким, чистым и гибким, а при добавлении новых функций старый код не приходилось переписывать и ломать.**

Как это выглядит на практике: «Без полиморфизма» против «С полиморфизмом»

Представьте, что вы пишете игру, где по экрану бегут 50 разных врагов: зомби, роботы и драконы. На каждом кадре всем нужно дать команду «атаковать».

• **БЕЗ ПОЛИМОРФИЗМА** (длинно и хрупко): Пришлось бы писать огромную проверку на каждый тип врага. Если завтра вы придумаете нового врага (например, вампира), вам придётся лезть в главный код игры и дописывать туда ещё один **else if**, рискуя случайно сломать старую логику.

• **С ПОЛИМОРФИЗМОМ** (коротко и безопасно): У вас есть один общий список врагов **List<Enemy>**. Вы просто запускаете цикл на две строчки, который говорит каждому **enemy.attack()**. Зомби укусит, робот выстрелит, а дракон дыхнёт огнём. Если завтра появится вампир — главный цикл игры вообще не изменится.

// КАК ДЕЛАЕТСЯ С ПОЛИМОРФИЗМОМ (Идеальный короткий скелет)

```
for (Vehicle car : garage) {

    car.move(); // Одна строчка заменяет десятки проверок if-else!

}
```

• **ИНКАПСУЛЯЦИЯ** = замок и охранник (чтобы не сломали данные).
• **НАСЛЕДОВАНИЕ** = копирование базового чертежа (чтобы не писать одно и то же дважды).

• **ПОЛИМОРФИЗМ** = универсальный пульт (одна команда запускает разные действия без кучи условий if-else).

всё **ООП** придумано ради порядка, защиты от случайных поломок и возможности расширять программу без переписывания старого кода.

## Тема: Интерфейсы и Абстрактные классы

В **Java** НЕЛЬЗЯ унаследовать один класс сразу от двух родителей (множественное наследование запрещено, чтобы не было путаницы). Но что делать, если нужно дать объекту сразу несколько разных ролей? Тут на помощь приходят интерфейсы и абстрактные классы.

**1. Интерфейс (`interface`) — «Контракт» или «Навык»**

- **Жизненная аналогия:** Розетка в стене или водительские права. Розетке всё равно, что вы в неё воткнёте — утюг, ноутбук или зарядку телефона. Главное условие (контракт): у прибора должна быть вилка нужного стандарта. В коде интерфейс работает так же — это список обязательных кнопок (методов), которые класс обязан у себя сделать.
- **Как работает в Java:** Интерфейс не содержит готового кода внутри методов, только их названия (пустые заглушки). Класс подключает интерфейс словом `implements` («реализует») и обязуется написать, как именно эта кнопка работает у него. При этом один класс может подключить сколько угодно интерфейсов (например, Утка может и `Flyable` — летать, и `Swimmable` — плавать).

**2. Абстрактный класс (`abstract class`) — «Недостроенный дом»**

- **Жизненная аналогия:** Заготовка автомобиля на конвейере. У неё уже есть готовый кузов, рама и колёса (обычные методы с кодом), но ещё нет двигателя (абстрактный метод без кода). Ездить на такой заготовке нельзя — сначала нужно создать конкретную модель и поставить туда мотор.
- **Как работает в Java:** Это обычный родительский класс, в котором часть методов уже написана, а часть оставлена пустыми с пометкой `abstract`. Наследник обязан дописать эти пустые методы.

```
// 1. ИНТЕРФЕЙС (Просто правило / навык без готового кода)
public interface Flyable {
    void fly(); // Обязательное умение, но как именно — решает сам класс
}

// 2. АБСТРАКТНЫЙ КЛАСС (Заготовка: часть уже есть, часть надо дописать)
public abstract class Bird {
    public void breathe() { // Готовый метод для всех птиц
        System.out.println("Птица дышит");
    }
    public abstract void makeSound(); // Пустой метод: каждая птица поёт по-своему
}

// 3. РЕАЛЬНЫЙ КЛАСС (Берёт заготовку Bird и подключает навык Flyable)
public class Duck extends Bird implements Flyable {
    @Override
    public void makeSound() {
```

```
        System.out.println("Кря-кря!");
    }

    @Override
    public void fly() {
        System.out.println("Утка летит над озером");
    }
}
```

- **Абстрактный класс (`extends`)** отвечает на вопрос **«КТО ЭТО?»** (Утка — это птица). Можно выбрать только одного родителя.
- **Интерфейс (`implements`)** отвечает на вопрос **«ЧТО УМЕЕТ ДЕЛАТЬ?»** (Умеет летать, умеет плавать). Можно подключить сколько угодно навыков.

## Тема: Строки (`String`) и сравнение объектов

В отличие от **C++**, где строка может меняться прямо в памяти, в Java строки устроены хитрее ради безопасности и экономии ресурсов.

**1. Почему `String` в Java неизменяемый (Immutable)**

- **Жизненная аналогия:** Текст, выбитый на каменной плите, или распечатанный чек. Вы не можете стереть одну букву в чеке и вписать другую — придётся напечатать новый чек.
- **Как работает в Java:** Когда вы создаёте строку `String text = "Привет";`, а потом пишете `text = text + " мир";`, старая строка в памяти НЕ меняется. Java создаёт в памяти совершенно новую строку `"Привет мир"`, а старую со временем удаляет сборщик мусора.
- **Зачем так сделали:** Чтобы строки можно было безопасно передавать в разные потоки и использовать как пароли или ключи в базах данных — никто не сможет незаметно подменить буквы внутри уже созданной строки.
- **Что делать, если нужно часто склеивать текст:** Если вы собираете большой текст в цикле (например, 10 000 раз), обычный `String` создаст 10 000 лишних копий и забьёт память. Для склейки

текста используют специальный «черновик» — класс **StringBuilder** (он работает как обычная тетрадь с ластиком).

**2. Главное правило сравнения: == против .equals()**

Новички после **C++** часто допускают здесь классическую ошибку:

- **Знак == СРАВНИВАЕТ ССЫЛКИ (адреса в памяти).** Это как спросить: «Эти два ключа открывают одну и ту же квартиру?». Даже если две строки содержат одинаковое слово **"Java"**, но созданы по-разному, **==** может вернуть **false**, потому что они лежат в разных ячейках памяти.
- **Метод .equals() сравнивает содержимое (смысл).** Это как спросить: «Написан ли на этих двух листах бумаги одинаковый текст?». Поэтому строки и любые объекты в Java всегда сравнивают только через **.equals()**.

**3. Связка equals и hashCode (Паспорт и номер полки)**

У каждого объекта в **Java** есть два встроенных метода, которые приходят от главного прародителя всех классов — класса **Object**:

- **hashCode() — быстрый предварительный фильтр (номер полки в гардеробе).** Превращает объект в одно целое число. Если у двух объектов разные хэш-коды, они гарантированно разные — дальше можно даже не проверять.
- **equals() — точная сверка (проверка паспорта).** Срабатывает, если у двух объектов совпал **hashCode**. Она проверяет поля по очереди, чтобы убедиться, что объекты действительно одинаковые.

**Готовый скелет для вашего конспекта**

Перепишите этот блок — он спасёт вас от самых частых багов и вопросов на интервью:

```
// 1. ПРАВИЛЬНОЕ СРАВНЕНИЕ СТРОК И ОБЪЕКТОВ
String s1 = new String("Java");
String s2 = new String("Java");

System.out.println(s1 == s2);      // false! (Разные адреса в памяти)
System.out.println(s1.equals(s2)); // true!  (Одинаковый текст внутри)

// 2. БЫСТРАЯ СКЛЕЙКА ТЕКСТА В ЦИКЛЕ (Вместо обычного String)
StringBuilder sb = new StringBuilder();
```

```
sb.append("Привет, ");
sb.append("Aston!");
String result = sb.toString(); // Превращаем черновик в готовую строку
```

- Строки (`String`) менять нельзя — для массовой склейки берём `StringBuilder`.
- Примитивы (`int`, `boolean`) сравниваем через `==`, а строки и объекты — только через `.equals()`.
- Золотое правило контракта: если два объекта равны по `equals()`, у них обязан быть одинаковый `hashCode()`.

Вместо того чтобы на каждом шаге выкидывать старую строку и печатать новую в памяти, мы собираем всё по кусочкам в одном «черновике» (`StringBuilder`), а в самом конце одним движением превращаем его в готовую чистую строку (`String`).

**Жизненная аналогия для закрепления**

- **Обычный `String` в цикле:** Вы пишете слово на листе бумаги. Чтобы дописать ещё одну букву, вы берёте *новый* чистый лист, переписываете туда старое слово, дописываете новую букву, а старый лист комкаете и кидаете в мусорку. Если букв 1000 — у вас полная комната скомканной бумаги (забитая память).
- **`StringBuilder`:** Вы берёте одну большую доску или тетрадь, спокойно дописываете туда все 1000 букв подряд, а когда текст полностью готов — один раз фотографируете его (`toString()`), получая итоговый `String`.

**Короткое правило в конспект**

- **1–2 склейки** (например, `"Привет, " + name`) — спокойно пишем обычный + со `String`.
- **Много склеек в цикле** (собираем отчёт, большой текст, логи) — обязательно берём `StringBuilder`.

Когда квартира забита скомканными бумагами, программе приходится останавливать работу и выносить этот мусор (срабатывает **Garbage Collector**), из-за чего приложение начинает тормозить. А `StringBuilder` позволяет работать на одном листе без создания лишнего мусора.

# Тема: Исключения (Exceptions) — «Страховка и подушка безопасности»

В отличие от **C**, где ошибка часто приводит к тихому падению программы или мусору в памяти, и похоже на обработку ошибок в **Rust/Go**, в Java при любой внештатной ситуации программа «**выбрасывает исключение**» (**Exception**) — специальный объект-сигнал тревоги.

**1. Жизненная аналогия: каскадёр и страховочная сетка**
Представьте каскадёра, который прыгает с крыши.

- **Блок `try` («попробовать»):** Это сам трюк. Мы говорим программе: «Попробуй выполнить этот опасный кусок кода» (например, открыть файл на диске или разделить число на ноль).
- **Блок `catch` («поймать»):** Это страховочная сетка внизу. Если внутри `try` что-то пошло не так, программа не разбивается насмерть (не закрывается с чёрным экраном), а падает в сетку `catch`. Там мы спокойно пишем пользователю: «Файл не найден, проверьте флешку», и программа продолжает жить дальше.
- **Блок `finally` («в любом случае»):** Это уборка площадки после трюка. Этот код сработает всегда — и если трюк удался, и если каскадёр упал в сетку. Обычно там закрывают открытые файлы или соединения с базой данных.

**2. Два главных вида исключений**
В **Java** все ошибки делятся на два лагеря, и это обязательно спрашивают на собеседованиях:

- **Checked (Проверяемые) — «Предупреждение синоптиков»:** Это внешние проблемы, в которых программист не виноват, но обязан к ним подготовиться заранее (нет интернета, пропал файл, отключилась база данных). **Java** даже НЕ разрешит запустить программу, пока вы не обернёте такой код в `try-catch` или не предупредите о риске словом `throws`.
- **Unchecked / Runtime (Непроверяемые) — «Ошибка самого водителя»:** Это баги в логике кода (деление на ноль, выход за пределы массива, обращение к пустоте `null` — знаменитый `NullPointerException`). Компилятор не заставляет их ловить, потому что такие ошибки нужно НЕ прятать в сетку, а просто исправлять в коде.

```
try {
    // ОПАСНЫЙ КОД (Пробуем выполнить)
```

```
    int result = 10 / 0; // Ошибка! Деление на ноль
    System.out.println("Эта строчка уже не выполнится");

} catch (ArithmeticException e) {
    // СТРАХОВОЧНАЯ СЕТКА (Ловим конкретную ошибку)
    System.out.println("Ой, на ноль делить нельзя! Программа спасена.");

} finally {
    // УБОРКА (Выполнится в любом случае: была ошибка или нет)
    System.out.println("Закрываем ресурсы и работаем дальше.");
}
```

`try` — зона риска, где может случиться сбой.
`catch` — план Б, если сбой всё-таки произошёл (чтобы программа не упала).
`finally` — действие, которое выполнится при любом исходе (обычно закрытие файлов).
`throw` — команда, чтобы *самому* кинуть ошибку, если данные не прошли проверку (например, при вводе отрицательного баланса в банкомате).

## Тема: Коллекции (Склады и картотеки для данных)

Обычный массив в **Java** (как и в **C++**) неудобен: вы задали ему размер на 10 ячеек, и расширить его уже нельзя. Поэтому в **Java** придумали **Коллекции** — умные резиновые контейнеры, которые сами растут, сжимаются и умеют быстро искать данные.
Всю тему коллекций на собеседованиях сводят к четырём главным «складам».

**1. `ArrayList` — «Резиновый поезд или стеллаж с номерами»**

- **Жизненная аналогия:** Нумерованные места в кинотеатре или стеллаж, где все коробки стоят вплотную друг за другом. Если места закончились, программа автоматически покупает зал побольше и пересаживает всех туда.
- **Когда берём:** Когда нужно просто хранить список чего-либо (имена пользователей, товары в корзине) и **быстро доставать элемент по его номеру** (индексу).
- **Минус:** Если захотите втиснуть нового человека в самое начало ряда, всем остальным придётся сдвинуться на одно кресло вправо — это долго.

**2. `LinkedList` — «Хоровод или цепочка вагонов»**

- **Жизненная аналогия:** Детский хоровод, где каждый держит за руку только соседа слева и соседа справа. Чтобы поставить нового человека в начало или середину, никого двигать не нужно — достаточно расцепить две руки и вклиниться между ними.
- **Когда берём:** Когда нужно **постоянно добавлять или удалять элементы в начале и середине списка**.
- **Минус:** Чтобы найти 50-го человека в хороводе, роботу придётся пройти по цепочке от самого первого, считая каждого — поиск по номеру работает медленно.

**3. `HashSet` — «Мешок уникальных билетов (Фейсконтроль)»**

- **Жизненная аналогия:** Мешок, в который нельзя положить две одинаковые вещи. Если вы положите туда паспорт Ивана, а через минуту попробуете кинуть его дубликат — `HashSet` просто проигнорирует дубликат. Порядок вещей внутри при этом не важен.
- **Когда берём:** Когда нужно **убрать все повторы** из списка или за доли секунды проверить: «Есть ли уже этот логин в базе?».

**4. `HashMap` — «Гардероб с номерками или словарь» (Главный вопрос на собеседовании!)**

- **Жизненная аналогия:** Обычный гардероб или телефонная книга. Вы отдаёте куртку (Значение / `Value`) и получаете уникальный номерок (Ключ / `Key`). Чтобы забрать куртку, вам не нужно перебирать все вешалки в гардеробе — вы сразу идёте по номерку.
- **Как устроен внутри :** Под капотом `HashMap` — это массив «корзин» (ячеек). Когда вы кладёте пару «Ключ + Значение», Java считает от ключа специальное число (`hashCode`), по нему мгновенно вычисляет номер корзины и кладёт данные туда. Если два разных ключа случайно попали в одну корзину (это называется **коллизия**), они выстраиваются там в небольшой список (а если их становится больше 8 — превращаются в сбалансированное дерево для ускорения поиска).

```
// 1. СПИСОК ПО НОМЕРАМ (Самый частый в работе)
List<String> names = new ArrayList<>();
names.add("Алексей");
names.get(0); // Мгновенно достали первого

// 2. СПИСОК БЕЗ ПОВТОРОВ (Уникальные значения)
Set<String> uniqueLogins = new HashSet<>();
uniqueLogins.add("admin");
uniqueLogins.add("admin"); // Дубликат не добавится, останется только один "admin"
```

```
// 3. СЛОВАРЬ «КЛЮЧ -> ЗНАЧЕНИЕ» (Мгновенный поиск по ключу)
Map<String, Integer> ages = new HashMap<>();
ages.put("Иван", 25);        // Положили по ключу "Иван"
int age = ages.get("Иван");  // Мгновенно нашли 25 по ключу
```

• Нужен обычный список и быстрое чтение — берём **ArrayList**.

• Часто вставляем и удаляем в начале/середине — берём **LinkedList**.

• Нужны только уникальные элементы без дублей — берём **HashSet**.

• Ищем данные по имени, ID или артикулу («Ключ — Значение») — берём **HashMap**.

**1. Приём «Осмотр каждого элемента» (Цикл `for-each`)**

Когда у вас есть список (например, `ArrayList` с товарами или врагами в игре), робот должен пройти по каждому элементу и что-то с ним сделать.

- **Жизненная аналогия:** Контролёр идёт вдоль ряда кресел в кинотеатре и проверяет билет у каждого зрителя по очереди.
- **Скелет в конспект:**

```
for (String name : names) {
    System.out.println("Привет, " + name); // Робот сам подставит сюда каждого по очереди
}
```

**2. Приём «Проверка перед действием» (`contains` и `containsKey`)**

Робот буквален: если вы попросите его достать из `HashMap` ключ, которого там нет, он вернёт пустоту (`null`), и программа может упасть. Поэтому перед тем как брать, мы всегда ставим «охранника».

- **Жизненная аналогия:** Прежде чем открыть шкафчик в раздевалке, вы проверяете, есть ли вообще такой номер в зале.
- **Скелет в конспект:**

```
if (ages.containsKey("Иван")) {
    System.out.println("Возраст: " + ages.get("Иван"));
} else {
    System.out.println("Такого человека в базе нет!");
```

```
}
```

**3. Приём «Подсчёт повторов» (Главный хит на собеседованиях!)**
Очень частая задача на отборах: дан огромный список слов или заказов, нужно посчитать, сколько раз встречается каждое слово. Здесь `HashMap` работает как бухгалтерский учёт.

- **Жизненная аналогия:** Вы разбираете корзину с фруктами. Берёте яблоко — ставите в блокноте напротив слова «Яблоко» палочку (1). Берёте ещё одно яблоко — меняете цифру на 2.
- **Скелет в конспект:**

```
Map<String, Integer> counter = new HashMap<>();
for (String word : wordsList) {
    // getOrDefault: если слова ещё нет — берём 0, иначе текущее число, и прибавляем 1
    counter.put(word, counter.getOrDefault(word, 0) + 1);
}
```

**4. Приём «Сортировка по порядку» (`Collections.sort`)**
Сам по себе `ArrayList` хранит вещи в том порядке, в котором вы их кинули. Но если нужно выстроить имена по алфавиту, а числа от меньшего к большему, мы вызываем встроенного сортировщика.

- **Жизненная аналогия:** Библиотекарь расставляет стопку книг на полке строго по алфавиту.
- **Скелет в конспект:**

```
Collections.sort(names); // Сортирует список на месте (от А до Я или от 1 до 100)
```

**Как это связывается с проектированием**
Когда вы получаете реальную задачу, вы просто выбираете нужный приём из конспекта:

- Нужно убрать дубли из `ArrayList`? Пересыпаем его в `HashSet` (`Set<String> clean = new HashSet<>(dirtyList);`).
- Нужно посчитать статистику? Берём `HashMap` и приём №3.
- Нужно вывести всё на экран? Берём цикл `for-each` из приёма №1.

# Тема: Stream API и Лямбды — «Заводской конвейер»

В предыдущей теме мы использовали циклы `for`, чтобы перебирать списки. Но в современной **Java** списки обрабатывают через **Stream API** — это делает код в разы короче и читаемее.

**1. Жизненная аналогия: конвейер на заводе по сортировке яблок**

Представьте, что у вас есть корзина с яблоками (`List<Apple>`). Вы высыпаете её на конвейерную ленту (`stream()`). Вдоль ленты стоят рабочие, каждый выполняет одно маленькое действие:

- **Первый рабочий (`filter` — фильтр):** Отбрасывает зелёные и гнилые яблоки, пропускает дальше только красные и спелые.
- **Второй рабочий (`map` — преобразование):** Моет каждое оставшееся яблоко или нарезает его на дольки (превращает один объект в другой, например, объект «Человек» превращает просто в строку с его именем).
- **Упаковщик в конце ленты (`collect` — сборка):** Складывает готовые обработанные яблоки в новую чистую коробку (`List`). Пока в конце не стоит упаковщик, конвейер даже не запустится!

**2. Что такое Лямбда-выражение (`->`)?**

Это просто короткая запись инструкции для рабочего на конвейере без создания лишних методов. Знак стрелки `->` читается как **«сделай с этим элементом вот что»**.

Например: `name -> name.startsWith("A")` переводится для робота так: *«Возьми очередное имя и проверь, начинается ли оно на букву А»*.

```
List<String> names = List.of("Алексей", "Иван", "Анна", "Петр");

// Запускаем конвейер: берём имена на букву "А", переводим в заглавные и кладём в новый список
List<String> result = names.stream()
        .filter(name -> name.startsWith("A")) // 1. Отсеиваем (остаются Алексей и Анна)
        .map(name -> name.toUpperCase())      // 2. Преобразуем (АЛЕКСЕЙ, АННА)
        .collect(Collectors.toList());        // 3. Упаковываем в новый список
```

# Тема: Дженерики (<T>) — «Прозрачная коробка с этикеткой»

Вы уже видели эти треугольные скобки в прошлых примерах: `List<String>` или `Map<String, Integer>`. Зачем они нужны? Чтобы робот ещё до запуска программы следил, что вы не положили кирпич в коробку для яблок.

**1. Жизненная аналогия: коробка с наклейкой**

- **Без дженериков (Старый подход):** Представьте непрозрачную картонную коробку без надписей. Вы можете кинуть туда и яблоко, и гаечный ключ, и носок. Программа не будет ругаться. Но когда вы сунете туда руку вслепую, чтобы достать яблоко и откусить, вы рискуете сломать зубы о гаечный ключ — программа рухнет во время работы.
- **С дженериками (<String>):** Вы клеите на коробку строгую этикетку «ТОЛЬКО ЯБЛОКИ» (`List<Apple>`). Теперь, если кто-то по ошибке попытается засунуть туда гаечный ключ, программа заблокирует это действие сразу на этапе написания кода (подчеркнёт красным), ещё до запуска. А когда вы достаёте предмет из коробки, вы на 100% уверены, что это яблоко.

**2. Как это выглядит в коде (Буква <T>)**

Когда программисты пишут свой универсальный класс-коробку, они ещё не знают, что именно туда положит другой разработчик (строки, числа или игровых персонажей). Поэтому вместо конкретного типа ставят заглушку `<T>` (от слова *Type* — тип).

**Готовый скелет для вашего конспекта**

Зафиксируйте этот короткий пример — он объясняет всю суть треугольных скобок:

```
// 1. УНИВЕРСАЛЬНАЯ КОРОБКА (Создаём один раз для любых вещей)
public class Box<T> {
    private T item; // Внутри лежит нечто типа T

    public void put(T item) {
        this.item = item;
    }

    public T get() {
        return item;
    }
}

// 2. ИСПОЛЬЗОВАНИЕ (Клеим конкретную этикетку при создании)
Box<String> textBox = new Box<>();
```

**textBox.put("Привет");** // Работает!
// **textBox.put(123);** // ОШИБКА! Робот не даст положить число в коробку для строк.

Треугольные скобки **<...>** — это **фейсконтроль на тип данных** (защита от случайного мусора в списках).

Внутри скобок пишут только классы с большой буквы (**<String>**, **<Integer>**, **<User>**), примитивы вроде **<int>** туда ставить нельзя.

## Тема: Многопоточность — «Несколько поваров на одной кухне»

Обычно программа работает в одном потоке (**main**) — робот выполняет строчки строго по очереди. Но если нужно одновременно скачивать файл, отрисовывать графику и обрабатывать клики мыши, одного робота мало. Мы нанимаем целую бригаду.

**1. Жизненная аналогия: кухня и туалетная кабинка**

- **Потоки (Thread) — это повара на кухне.** Один режет салат, второй жарит мясо, третий варит суп. Работа идёт в три раза быстрее.
- **Главная проблема (Гонка данных / Race Condition):** Представьте, что два повара одновременно схватили одну и ту же солонку или начали записывать заказ на одном листе бумаги. В итоге суп пересолен в два раза, а цифры в чеке перемешались.
- **Решение №1: synchronized («Замок на двери»).** Представьте туалетную кабинку в самолёте или примерочную. Зашёл один поток — закрыл щеколду изнутри. Пока он не закончит работу с общим ресурсом (например, списание денег со счёта) и не выйдет, остальные потоки терпеливо ждут в очереди перед дверью.
- **Решение №2: volatile («Общее табло на стене»).** Чтобы каждый повар работал быстро, он держит копию рецепта в своём кармане (в кэше процессора). Но если шеф-повар изменил рецепт, повар может этого не заметить. Слово **volatile** заставляет всех смотреть не в свой карман, а строго на главное табло в центре кухни (в основную оперативную память), где всегда висит самая свежая версия переменной.

**2. Как запускают отдельный поток в Java**

В современной **Java** не создают потоки вручную на каждый чих, а используют **пул потоков (ExecutorService)** — это как бригада из 5 штатных рабочих, которым вы просто подкидываете задачи в общую корзину.

**Готовый скелет для вашего конспекта**
Перепишите этот короткий шаблон — он показывает и запуск задачи в отдельном потоке, и защиту от одновременного доступа:

```
// 1. ЗАПУСК ЗАДАЧИ В ОТДЕЛЬНОМ ПОТОКЕ (Через лямбду)
Thread worker = new Thread(() -> {
    System.out.println("Я работаю параллельно в отдельном потоке!");
});
worker.start(); // Команда «Поехали!» (именно start, а не run!)

// 2. ЗАЩИТА ОТ ОДНОВРЕМЕННОГО ДОСТУПА (Замок на методе)
public synchronized void withdrawMoney(int amount) {
    // Пока один поток внутри этого метода, второй ждёт за дверью
    balance = balance - amount;
}
```

Поток запускается только командой `.start()` (если вызвать `.run()`, никакой параллельности не будет, код выполнится в старом потоке).
`synchronized` — щеколда на двери (защищает кусок кода от одновременного входа двух потоков).
`volatile` — запрещает потокам прятать переменную у себя в кэше, заставляет всегда читать свежее значение из общей памяти.

- `synchronized` — это строгая живая очередь у двери: пока первый поток не закончит свою работу с данными и не выйдет, второй внутрь не зайдёт. Никакой толкотни и путаницы.
- `volatile` — это запрет на «устаревшие черновики» в быстром кэше процессора. Поток обязан каждый раз заглядывать в общую главную память (RAM), чтобы видеть самое свежее, актуальное значение, которое только что обновил другой поток.
- `.start()` — это кнопка запуска нового рабочего (скобки там обычно пустые `worker.start()`, а само задание мы передали чуть выше, когда создавали `new Thread(...)`).

## Устройство памяти (JVM, Стек, Куча и Сборщик мусора)

**1. Как запускается Java (JVM — «Переводчик и виртуальный компьютер»)**

- **Жизненная аналогия:** Переводчик-синхронист на международной встрече. Когда вы пишете код на C++, он переводится сразу в язык конкретного процессора (только под Windows или только

под Linux). В **Java** ваш код сначала переводится в универсальный **байт-код** (эсперанто для машин), а затем специальная программа **JVM (Java Virtual Machine)** переводит его на лету для любого компьютера. Благодаря этому работает главный принцип Java: *«Написано один раз — работает везде»*.

**2. Где хранятся данные: Стек (Stack) и Куча (Heap)**
Вся оперативная память программы делится на два главных склада (как в C++ и Go):

- **Стек (Stack) — «Рабочий стол с быстрыми заметками».** Очень быстрая, но маленькая память. У каждого потока свой личный стол. Здесь лежат простые примитивные переменные (`int a = 5;`), порядок вызова методов и **ярлыки-ссылки** на большие объекты. Как только метод закончил работу, робот мгновенно смахивает всё со стола в мусор.
- **Куча (Heap) — «Огромный общий склад».** Большая память, общая для всех потоков. Здесь лежат все настоящие тяжеловесные объекты, созданные через слово `new` (строки, списки, игровые персонажи, банковские счета). А на рабочем столе (в Стеке) лежит только бумажка с адресом полки на этом складе.

**3. Сборщик мусора (Garbage Collector / GC) — «Ночной уборщик»**

- **Жизненная аналогия:** Уборщик в офисе. В C++ вы сами должны были удалять ненужные объекты (`delete`), иначе память заканчивалась. В **Java** вы просто создаёте объекты через `new`. Как только на объект в Куче не осталось ни одной ссылки из Стека (никто про него больше не помнит), приходит **Garbage Collector**, сам находит такие «бесхозные» коробки на складе и выкидывает их, освобождая место.

ИТОГ И ПОЭТАПНОСТЬ СОЗДАНИЯ КОДА:

1. **Робот абсолютно буквален.** Он читает код сверху вниз, строчка за строчкой, без фантазии и догадок. Что не прописано детально — того для него не существует.
2. **Данные и защита (Склад и охранники).** В начале класса мы описываем переменные (`private`), запирая их от дурачества. Примитивы (`int`) ложатся на быстрый стол (Stack), сложные объекты (`String`, `ArrayList`, `HashMap`) — на общий склад (Heap).
3. **Входные данные и фильтры (`if` — Guard Clauses).** Когда запускается действие (метод), первым делом на входе стоят охранники `if`. Они проверяют данные: если пришёл бред — робот сразу прерывает работу (`return`).

4. **Испытательный полигон (`try-catch`).** Если предстоит опасная операция (работа с сетью, файлами или деление), мы заключаем её в бронекапсулу `try-catch`. Даже если внутри случится сбой, основная программа не рухнет.
5. **Основная работа (Конвейеры и Полиморфизм).** Проверенные данные пропускаются через логику: циклы `for`, конвейеры `Stream API` или универсальные команды (`полиморфизм`), где каждый объект сам знает, как выполнить приказ. Если работают несколько поваров (`Thread`), мы вешаем на общие двери щеколду (`synchronized`).
6. **Вывод и уборка.** Робот показывает результат на экране (`System.out.println`) или возвращает ответ (`return`), очищает свой рабочий стол (**Stack**), а бесхозный мусор на складе (Heap) позже сам вынесет **Garbage Collector**.

**Жизненная аналогия: ящик с инструментами**
Ваш конспект из 6 пунктов — это **чемодан мастера на все руки**, где лежат молоток, отвёртка, дрель, уровень, изолента и защитные очки.

- Если вам нужно просто забить один гвоздь в стену — вы берёте из чемодана **только молоток**. Вы не тащите с собой дрель и защитные очки, но гвоздь всё равно держится отлично.
- Если вам нужно повесить тяжёлую полку над трубами — вы достаёте **и уровень, и дрель, и очки**.

Точно так же и в коде: вы держите в голове все 6 пунктов как карту возможностей, но в конкретную задачу вставляете только те блоки, которые реально защищают или упрощают именно её.

**1. Когда включаем «Испытательный полигон» (`try-catch`)**
Этот блок нужен везде, где программа обращается к чему-то **за пределами своего кода**, ведь внешний мир ненадёжен:

- **Работа с файлами на диске:** Вы делаете загрузку сохранения в игре или открытие картинки в графическом редакторе. Пользователь мог случайно удалить файл, выдернуть флешку или переименовать папку — без `try-catch` программа вылетит с чёрным экраном, а с ним вежливо напишет: *«Файл сохранения не найден»*.
- **Интернет и базы данных:** Приложение отправляет запрос на сервер (например, проверяет баланс карты или загружает погоду). В этот момент может пропасть Wi-Fi или зависнуть сервер банка. `try-catch` перехватит обрыв связи и покажет кнопку *«Повторить попытку»*.
- **Ввод данных от живого человека:** Вы просите пользователя ввести свой возраст цифрами, а он по приколу написал слово *«двадцать»*. При попытке превратить слово в число программа споткнётся, а `try-catch` поймает ошибку и попросит ввести именно цифры.

**2. Когда включаем «Охранников на входе» (`if` — Guard Clauses)**
Охранники нужны там, где есть **правила игры и деньги**, которые нельзя нарушать:

- **Финансы и магазины:** Оформление заказа, перевод денег, применение промокода. Охранник проверяет, что сумма больше нуля, товара хватает на складе, а промокод ещё не истёк.
- **Регистрация и формы:** Пользователь придумывает пароль. Охранник на входе сразу отсекает пароли короче 8 символов или без цифр, не пуская их дальше в базу данных.

**3. Когда включаем «Полиморфизм и Интерфейсы» (Универсальные кнопки)**
Этот блок достают, когда в программе появляется **много однотипных вещей с разным поведением**, и вы знаете, что в будущем их станет ещё больше:

- **Платёжные системы на сайте:** Сегодня клиент платит банковской картой, завтра вы добавите СБП, а через месяц — бонусные баллы. Вместо того чтобы переписывать всю кассу магазина, вы делаете один интерфейс `Payment` («Оплатить») и просто подключаете новые способы как картриджи.
- **Инструменты в графическом редакторе:** У вас есть Кисть, Ластик, Заливка и Карандаш. Все они реагируют на движение мыши (`onMouseDrag`), но каждый рисует по-своему.

**4. Когда включаем «Бригаду поваров» (`Thread` и `synchronized`)**
Многопоточность нужна только в двух случаях: **чтобы экран не зависал** или **когда одновременно приходят тысячи людей**:

- **Кнопка «Скачать» в приложении или игре:** Если скачивать огромный файл в основном потоке, окно программы намертво зависнет и перестанет реагировать на мышку, пока файл не докачается. Поэтому загрузку отдают отдельному рабочему потоку, а основной поток продолжает плавно рисовать анимацию.
- **Бронирование билетов или распродажа:** На сайте остался последний билет на концерт, и на кнопку «Купить» в одну миллисекунду нажали 500 человек. Здесь включают замок `synchronized`, чтобы билет достался строго первому успевшему, а не продался 500 раз.

Код работает **только с вашими внутренними цифрами** в одном окне — пишем минимальный скелет (Ввод -> Действие -> Вывод).
Код лезет **в файлы, интернет, чужой ввод, деньги или работает на толпу пользователей** — достаём из чемодана `try-catch`, охранников `if` и **`многопоточность`**.

Если перепутать местами «охранников» (`if`) и «испытательный полигон» (`try-catch`), код всё равно запустится, но будет работать медленно и некрасиво. В программировании это даже считается грубой ошибкой стиля (антипаттерном).

| Блок (Инструмент) | С какими данными работает | Главный вопрос блока | Пример из жизни (Аэропорт) |
|---|---|---|---|
| **1. `private` (Инкапсуляция)** | Внутренние переменные класса (баланс, здоровье, пароль). | *«Кому разрешено менять эти цифры напрямую?»* | **Сейф в кассе.** Пассажир не может сам залезть в сейф и поменять цену билета — только через кассира. |
| **2. `if` (Охранник на входе)** | Предсказуемые условия, границы чисел, пустота (`null`), права доступа. | *«Подходят ли эти данные под наши правила?»* | **Паспортный контроль.** Проверяем заранее: не просрочен ли паспорт, есть ли виза. Если нет — разворачиваем сразу (`return`). |
| **3. `try-catch` (Полигон)** | Внешний мир: файлы на диске, сеть, база данных, перевод текста в число. | *«Что делать, если внешний мир внезапно сломается?»* | **Погодные условия и турбулентность.** Паспорт в порядке, мы взлетели, но внезапно началась гроза — срабатывает аварийный протокол. |
| **4. Полиморфизм / Интерфейсы** | Наборы однотипных объектов с разным поведением (виды оплаты, враги, кисти). | *«Как дать одну команду разным исполнителям?»* | **Команда диспетчера «На взлёт!».** Её получают и маленький кукурузник, и огромный Боинг |

| | | | |
|---|---|---|---|
| | | | — каждый взлетает по-своему. |
| **5. Коллекции и `Stream API`** | Большие группы данных (списки клиентов, корзина товаров, логи). | *«Как быстро найти, отфильтровать или посчитать пачку данных?»* | **Багажная лента.** Сотни чемоданов едут мимо: отбираем только красные (`filter`) и клеим на них бирки (`map`). |
| **6. `synchronized` / `volatile`** | Общие данные, к которым одновременно тянутся несколько потоков. | *«Как не дать двум рабочим испортить одну деталь?»* | **Взлётно-посадочная полоса.** Полоса одна, а самолётов много — диспетчер пускает их строго по очереди (`synchronized`). |

**Как не перепутать блоки при написании кода: 3 золотых правила**
Чтобы при проектировании рука сама тянулась к нужному шаблону из конспекта, проверяйте себя по трём простым тестам:

- **Тест 1: «Это можно проверить обычным сравнением (>, <, ==, `null`)?»**
  - Если **ДА** — это всегда **Охранник (`if`)**. Никакого `try-catch` сюда не ставим.
  - *Пример:* «Хватает ли денег на карте?», «Не пустое ли имя?», «Больше ли нуля скорость?».
- **Тест 2: «Зависит ли успех от чего-то вне моей программы (диск, интернет, чужой ввод)?»**
  - Если **ДА** — это всегда **Полигон (`try-catch`)**. Даже если вы проверили всё через `if`, файл всё равно может оказаться повреждён, а интернет — пропасть.
  - *Пример:* «Прочитать файл `save.dat`», «Превратить строку от пользователя в число», «Отправить запрос в базу».
- **Тест 3: «Тянутся ли к этой переменной сразу два потока (повара)?»**
  - Если **ДА** — ставим замок **`synchronized`** (если меняем данные) или метку **`volatile`** (если просто читаем свежий флаг

состояния, например `isGameRunning`). Если поток один — эти слова вообще не пишем.

```
public class TicketService {
    // 1. PRIVATE: Защитили количество мест от прямого взлома
    private int freeSeats = 10;

    // 6. SYNCHRONIZED: Очередь, чтобы два покупателя не купили последнее место одновременно
    public synchronized void buyTicket(String passengerName, String seatCountText) {

        // 2. ОХРАННИК (if): Проверяем предсказуемые правила
        if (passengerName == null || passengerName.isEmpty()) {
            System.out.println("Охранник: Введите имя пассажира!");
            return;
        }

        // 3. ПОЛИГОН (try-catch): Внешний ввод текста превращаем в число (вдруг ввели буквы!)
        try {
            int requestedSeats = Integer.parseInt(seatCountText);

            // Ещё один ОХРАННИК (if): Проверяем логику чисел
            if (requestedSeats <= 0 || requestedSeats > freeSeats) {
                System.out.println("Охранник: Неверное количество мест!");
                return;
            }

            // Полезная работа (всё проверено и безопасно)
            freeSeats = freeSeats - requestedSeats;
            System.out.println("Билет оформлен для: " + passengerName);

        } catch (NumberFormatException e) {
            System.out.println("Полигон: Количество мест нужно писать цифрами!");
        }
    }
}
```

**Почему многопоточность вредна для простых задач**

Создание нового потока (`Thread`) в **Java** — это дорогая операция для операционной системы. Она требует выделения отдельной памяти

под стек, регистрации потока в системе и переключения контекста процессора.

- **Жизненная аналогия:** Представьте, что вам нужно принести из соседней комнаты одну чашку. Если вы пойдёте за ней сами (один поток), это займёт 5 секунд. Если вы решите «ускорить» процесс, позвоните в клининговую компанию, оформите договор, дождётесь курьера и попросите его принести чашку (запустили многопоточность) — вы потратите час времени и кучу денег ради пустякового дела.
- **Где проходит граница:** Если операция занимает доли миллисекунды (сложить числа, проверить пароль, отфильтровать список из 100 элементов), один поток справится быстрее, чем система успеет даже создать второй поток. **Многопоточность** включают ТОЛЬКО ДЛЯ «**ТЯЖЕЛЫХ**» задач: загрузка файлов по сети, сложные расчёты на миллионы элементов или одновременное обслуживание сотен пользователей.

- **Вопрос 1: «Что будет, если у потока вызвать метод `.run()` вместо `.start()`?»**
  - *Правильный ответ:* Программа не сломается, но **никакой многопоточности не произойдёт**. Метод `.run()` выполнится как обычная функция в том же самом главном потоке по очереди. Чтобы нанять нового «повара» и запустить код параллельно, нужно вызывать строго `.start()`.
- **Вопрос 2: «Зачем в Java вообще нужны примитивы (`int`, `boolean`), если есть удобные объекты (`Integer`, `Boolean`)?»**
  - *Правильный ответ:* Ради **экономии памяти и скорости**. Примитив `int` весит всего 4 байта и лежит в быстром Стеке, а объект `Integer` весит в 4 раза больше (около 16 байт) и создаёт нагрузку на сборщик мусора. Если нужно просто посчитать цикл до миллиона, берём лёгкий примитив `int`.
- **Вопрос 3: «Почему нельзя везде поставить `synchronized` и `volatile`, чтобы программа вообще никогда не ломалась в многопоточности?»**
  - *Правильный ответ:* Потому что **упадёт скорость работы**. Если повесить щеколду `synchronized` на каждый метод, все потоки выстроятся в одну длинную очередь и будут ждать друг друга — программа фактически снова станет однопоточной, только с лишними расходами на проверку замков.
- **Вопрос 4: «Если у двух объектов одинаковый `hashCode()`, значит ли это, что они равны по `.equals()`?»**
  - *Правильный ответ:* **Нет, не обязательно.** Это как два разных человека, у которых совпал номер полки в гардеробе (коллизия). А вот наоборот работает всегда: если два объекта равны по `.equals()`, их `hashCode()` обязан быть одинаковым.

`ArrayList` = **Кинотеатр** (ищем по номеру места: 0, 1, 2...).
`HashSet` = **Фейсконтроль** (пропускает только уникальные вещи, дубли выкидывает).
`HashMap` = **Гардероб** (отдаём номерок-Ключ -> мгновенно получаем куртку-Значение).

GIT:

1. **git pull** # Утром: скачали свежий чистовик с сервера

2. **git checkout -b my-task** # Взяли чистый черновик (новую ветку) под задачу

... (пишем код в Java) ...

3. **git add .** # Сложили все изменённые файлы в стопку на отправку

4. **git commit -m "Сделал задачу"** # Сделали точку сохранения с понятной подписью

5. **git push origin my-task** # Отправили свой черновик на GitHub/GitLab для проверки

**1. Сборщики проектов (Maven / Gradle) — «Завхоз и снабженец»**
Когда вы пишете сложную программу, вам не нужно изобретать велосипед и писать с нуля код для отправки сообщений, работы с датами или подключения к базе данных. Всё это уже написали другие программисты и выложили в интернет в виде готовых библиотек.

- **Жизненная аналогия:** Вы открываете ресторан. Вместо того чтобы самому лепить тарелки из глины и выращивать пшеницу, вы пишете список покупок на одном листе бумаги. Отдаёте этот лист завхозу (**Maven**), а он сам едет на оптовый склад в интернет, закупает нужные продукты, проверяет сроки годности и раскладывает всё по полкам на вашей кухне.
- **Что нужно знать в конспекте:**
  - Главный файл настроек в Maven называется **`pom.xml`** (это и есть тот самый «список покупок»).
  - Раздел **`<dependencies>`** (зависимости) — место внутри **`pom.xml`**, куда вы просто копируете название нужной библиотеки, а Maven сам её скачивает и подключает к вашей Java-программе.

- Также Maven одной кнопкой превращает ваш код из десятков файлов в один готовый рабочий файл программы (обычно с расширением `.jar`), который можно запустить на любом компьютере.

**2. Базы данных и SQL — «Вечный картотечный шкаф»**
Все списки и переменные в Java (`ArrayList`, `HashMap`) живут только в оперативной памяти компьютера. Как только программа закрылась или выключили свет — все данные бесследно исчезают.

- **Жизненная аналогия:** Оперативная память **Java** — это ваша школьная доска: пока идёт урок, на ней написаны задачи, но в конце дня всё стирают тряпкой. База данных (например, **PostgreSQL**) — это огромный железный архив в подвале с папками, где информация хранится годами даже при выключенном электричестве. А язык **SQL** — это вежливый язык запросов к архивариусу.
- **4 главные команды SQL для вашего конспекта (аббревиатура CRUD):**
  - `INSERT` (*Create* — Создать): «Архивариус, заведи новую карточку клиента Ивана и положи в папку».
  - `SELECT` (*Read* — Прочитать): «Найди и покажи мне всех клиентов из Санкт-Петербурга старше 25 лет».
  - `UPDATE` (*Update* — Обновить): «Найди карточку Ивана и исправь в ней номер телефона».
  - `DELETE` (*Delete* — Удалить): «Достань карточку Ивана и выкинь её в шредер».

1. **Maven** скачивает нужные инструменты и собирает ваш проект.
2. **Java (ваш скелет робота)** принимает запрос от пользователя, проверяет его **Охранниками (`if`)**, защищает от сбоев в **Полигоне (`try-catch`)** и обрабатывает данные.
3. **База данных (SQL)** надёжно сохраняет результат работы, чтобы он не пропал после перезагрузки.
4. **Git** хранит историю всех ваших правок и позволяет сдать код на проверку наставнику.