**Блок 1: Основы Банковского Дела (Без этого никуда)**

Это фундамент. Если спросят "что такое банк?" или "чем банк занимается?", нужно отвечать уверенно.

1. **Что такое Банк?** (Из Лекции 01 "Введение")
   * **Просто:** Это специальная организация (финансово-кредитное учреждение), которая работает с деньгами.
   * **Главные фишки банка (что он МОЖЕТ делать, а другие нет):**
     + **Привлекать деньги:** Брать деньги у людей и компаний на хранение (депозиты, вклады).
     + **Размещать деньги:** Выдавать эти деньги в долг (кредиты) от своего имени.
     + **Открывать и вести счета:** Создавать "кошельки" для клиентов, где хранятся их деньги и через которые проходят платежи.
   * **Банковская система:** Это все банки в стране + Центральный Банк. Бывает одноуровневая (только ЦБ, редко), двухуровневая (ЦБ + коммерческие банки – как у нас), трехуровневая (например, в США: ФРС + федеральные банки + коммерческие). *Тебе достаточно знать, что есть ЦБ и коммерческие банки.*
2. **Баланс Банка – его "Финансовый Паспорт"** (Лекция 05 "Основные показатели...")
   * Представь весы. На одной чаше – **Активы** (что у банка есть, куда он вложил деньги), на другой – **Пассивы** (откуда банк эти деньги взял). Весы всегда должны быть в равновесии: **Активы = Пассивы**.
   * **Активы (куда банк вложил деньги):**
     + **Деньги:** Наличные в кассе, деньги на счетах в Центробанке или других банках (корреспондентские счета).
     + **Кредиты:** Деньги, которые банк выдал клиентам и ждет их возврата с процентами. *Это основной актив, приносящий доход.*
     + **Ценные бумаги:** Акции, облигации, которые банк купил, чтобы заработать.
     + **Основные средства:** Здания, компьютеры, автомобили.
     + **Недвижимость:** Если банк владеет какой-то недвижимостью для инвестиций.
   * **Пассивы (откуда банк взял деньги):**
     + **Обязательства (что банк должен):**
       - **Депозиты клиентов:** Деньги, которые люди и компании положили в банк. *Это основной пассив.*
       - **Кредиты от других банков или ЦБ:** Если банк сам взял в долг.
       - **Выпущенные ценные бумаги:** Если банк выпустил свои облигации, чтобы привлечь деньги.
     + **Собственный капитал (деньги владельцев банка):**
       - **Уставный фонд:** Первоначальные деньги, которые вложили акционеры при создании банка.
       - **Нераспределенная прибыль:** Прибыль, которую банк заработал, но еще не выплатил акционерам. *Это "подушка безопасности" банка.*
     + **(Временно в пассивах) Доходы и Расходы:** Они влияют на прибыль, а прибыль – на собственный капитал.
   * *Зачем это знать?* Баланс показывает, насколько банк большой, чем он зарабатывает и насколько он устойчив.
3. **Ключевые Показатели Работы Банка** (Лекция 05 "Основные показатели...")
   * **Доходность (Прибыльность):**
     + *Что это?* Способность банка зарабатывать деньги.
     + *Как меряют (упрощенно)?* Прибыль / Средний размер капитала.
     + *Зачем?* Чтобы платить дивиденды акционерам, развиваться.
   * **Ликвидность:**
     + *Что это?* Способность банка быстро превратить свои активы (например, кредиты) в живые деньги, чтобы расплатиться по своим обязательствам (например, вернуть вклад клиенту).
     + *Представь:* У тебя есть квартира (актив), но тебе срочно нужны деньги. Квартиру быстро не продашь. А деньги на счету (высоколиквидный актив) можно снять сразу.
     + *Зачем?* Чтобы банк не оказался в ситуации, когда все клиенты пришли за деньгами, а у него их нет.
     + *Показатели (просто знай названия):* LCR (коэффициент покрытия ликвидности), мгновенная ликвидность, текущая.
   * **Платежеспособность:**
     + *Что это?* Способность банка вовремя и полностью платить по всем своим долгам. Тесно связана с ликвидностью и достаточностью капитала.
     + *Зачем?* Это показатель надежности банка.
   * **Достаточность капитала:**
     + *Что это?* Сколько у банка СОБСТВЕННЫХ денег по отношению к его рискованным активам.
     + *Формула (упрощенно):* Собственный капитал / RWA (Активы, взвешенные по риску). RWA – это когда не все активы считаются одинаково рискованными (кредит крупной компании надежнее кредита стартапу).
     + *Норматив:* Обычно Центробанк требует, чтобы было >10%.
     + *Зачем?* Если часть кредитов не вернут (убытки), собственный капитал должен покрыть эти убытки, чтобы банк не разорился и не пострадали вкладчики.
   * **Процентная маржа:** Разница между тем, под какой % банк выдает кредиты, и под какой % привлекает депозиты. Это основной источник дохода банка.
   * **Резервы:** Банк откладывает часть денег "на черный день" – на случай, если выданные кредиты окажутся плохими и их не вернут.
4. **Регулирование Банковской Деятельности** (Лекция 02)
   * **Центральный Банк (ЦБ):**
     + **Регулятор:** Главный надзиратель за всеми банками.
     + **Лицензирование:** Выдает банкам разрешение на работу (лицензию). Без нее банк – не банк.
     + **Нормирование:** Устанавливает правила и нормативы (та же достаточность капитала, ликвидность).
     + **Ставка рефинансирования:** Процент, под который ЦБ дает кредиты коммерческим банкам. Влияет на ставки по кредитам и депозитам для всех.
     + **Резервные требования:** Коммерческие банки обязаны часть привлеченных денег хранить в ЦБ.
   * **Базельские соглашения (Базель I, II, III):**
     + *Что это?* Международные стандарты для банков, разработанные Базельским комитетом по банковскому надзору.
     + *Зачем?* Чтобы банки во всем мире были более устойчивыми и надежными, особенно после кризисов.
     + **Базель I:** Фокус на достаточности капитала (против кредитного риска).
     + **Базель II:** Более сложные подходы к оценке рисков (кредитный, рыночный, операционный), требования к надзору и рыночной дисциплине.
     + **Базель III:** Ответ на кризис 2008 года. Усилил требования к капиталу (качество и количество), ввел новые нормативы по ликвидности (LCR, NSFR), левереджу.
     + *Тебе важно знать, что Базель – это про усиление надежности банков через требования к капиталу и ликвидности.*
5. **Основные Банковские Операции – Что Банк Делает для Клиентов** (Лекция 04)
   * **Расчетный счет:** "Кошелек" клиента в банке для повседневных операций (получить зарплату, оплатить коммуналку).
   * **Вклад (депозит):** Клиент отдает банку деньги на время под процент. Бывают до востребования (можно забрать в любой момент), срочные (на определенный срок).
   * **Кредит:** Банк дает клиенту деньги в долг на время под процент. Бывают потребительские, ипотечные, автокредиты и т.д.
   * **Валютно-обменные операции:** Покупка/продажа одной валюты за другую.
   * **Банковские гарантии:** Банк поручается за клиента. Если клиент не выполнит обязательства (например, не поставит товар), банк заплатит пострадавшей стороне.
   * **Инкассация:** Сбор и перевозка наличных денег (например, из магазина в банк).
   * **Межбанковские операции:**
     + **Корреспондентские счета:** Счета, которые банки открывают друг у друга для взаимных расчетов.
       - **Ностро-счет ("наш у них"):** Счет нашего банка в другом банке. (Пример из лабы: счет нашего банка в Wells Fargo).
       - **Лоро-счет ("их у нас"):** Счет другого банка в нашем банке.
     + **SWIFT:** Международная система передачи финансовых сообщений (как "электронная почта" для банков всего мира). Используется для международных переводов.

**Блок 2: Система Temenos T24 – "Сердце" Банка**

Это та система, с которой ты, возможно, будешь работать.

1. **Что такое T24?** (Лекция 07 "Логическая архитектура")
   * Это **АБС – Автоматизированная Банковская Система**. Огромная программа, которая помогает банку выполнять все его операции. Производитель – компания Temenos.
   * **Структура T24:**
     + **Система:** Сама T24.
     + **Модуль:** Крупный блок функционала (например, "Клиенты", "Счета", "Кредиты").
     + **Приложение:** Конкретный инструмент внутри модуля для выполнения операций (например, приложение CUSTOMER для работы с клиентами, ACCOUNT для счетов). *Часто "модуль" и "приложение" используют как синонимы.*
2. **Как хранятся данные в T24?** (Лекция 07)
   * **Таблица:** Как файл в компьютере, где хранится информация одного типа (например, таблица FBNK.CUSTOMER хранит всех клиентов).
   * **Запись (Record):** Одна строка в таблице. Например, информация о конкретном клиенте Иванове И.И. Каждая запись имеет уникальный ID (ключ).
   * **Поле:** Отдельная характеристика в записи (например, "Имя", "Фамилия", "Адрес").
   * **Множественность (Multi-value):** Некоторые поля могут иметь несколько значений (например, у клиента может быть несколько телефонных номеров).
   * **Подзначения (Sub-value):** Значения внутри множественного поля (встречается реже).
3. **Жизненный Цикл Записи в T24 – ОЧЕНЬ ВАЖНО!** (Лекция 07, 08 "Запись приложения")
   * Когда пользователь создает или изменяет запись, она проходит несколько стадий:
     + **Unauth (Неавторизованная):**
       - Пользователь ввел данные, но они еще не проверены и не утверждены другим сотрудником.
       - Статусы записи: INAU (Input Not Authorised – введена, не авторизована), IHLD (Input Held – на удержании, заморожена).
       - Такие записи хранятся в файлах с суффиксом $NAU (например, FBNK.CUSTOMER$NAU).
     + **Live (Активная, Авторизованная):**
       - Запись проверена, утверждена (авторизована) другим сотрудником.
       - Теперь это "официальная" запись, система ее использует в работе.
       - Хранится в основном файле приложения (например, FBNK.CUSTOMER).
     + **History (Историческая):**
       - Когда Live-запись изменяется и авторизуется, ее предыдущая версия перемещается в историю.
       - Хранится в файлах с суффиксом $HIS (например, FBNK.CUSTOMER$HIS).
       - *Реверс (Reverse):* Если авторизованную запись нужно отменить, она помечается как RNAU (Reversed Not Authorised) в истории, а потом может быть полностью удалена или переведена в другой статус.
   * *Зачем это нужно?* Для контроля, безопасности, отслеживания изменений, возможности отката.
4. **Принцип "Ввод-Авторизация" (Maker-Checker / Двойной контроль)** (Лекция 07)
   * Один сотрудник (Maker, Inputter) вводит данные (создает Unauth-запись).
   * Другой сотрудник (Checker, Authoriser) проверяет и подтверждает (авторизует) эти данные, переводя запись в Live-статус.
   * *Зачем?* Уменьшает риск ошибок и мошенничества.
5. **Основные Приложения T24 и их Назначение (Знать обязательно!)** (Лекция 09 "Приложение CUSTOMER", и др.)
   * **CUSTOMER:** Хранит всю информацию о клиентах банка (физические лица, юридические лица, другие банки, брокеры). Паспортные данные, адреса, контакты, тип клиента (сектор, отрасль).
   * **ACCOUNT:** Управляет всеми счетами в банке (текущие, депозитные, кредитные, внутренние счета банка). Номер счета, валюта, остаток, связанные клиенты.
   * **FUNDS.TRANSFER (FT):** Приложение для выполнения денежных переводов (внутрибанковских, межбанковских, международных). Откуда, куда, сумма, валюта, комиссия.
   * **LIMIT:** Управление лимитами (например, кредитный лимит для клиента, лимит на сумму одной операции).
   * **CURRENCY, COUNTRY, DATES, HOLIDAY:** Системные справочники (валюты и их курсы, страны, календари рабочих и выходных дней). Нужны для корректной работы всех приложений.
   * **USER:** Управление пользователями системы. Кто имеет доступ, какие у него права (роли), какое меню ему доступно, какой пароль.
   * **EB.API (Enterprise Bus Application Programming Interface):** Приложение для регистрации кастомных разработок (например, Hook Routines).
   * **VERSION:** Приложение для создания и настройки версий (экранов) других приложений.
   * **ENQUIRY:** Приложение для создания и настройки запросов к данным.
6. **Взаимодействие с T24**
   * **BrowserWeb:** Основной интерфейс для пользователей, работа через веб-браузер. (скриншоты в большинстве лекций)
   * **Командная строка T24:** (Lesson 03 "Команды")
     + Используется для быстрого доступа к записям или запуска функций.
     + Формат: Приложение Ключ Функция (например, CUSTOMER 190361 I F3 – открыть клиента 190361 в режиме ввода с автоформированием ID, если это новая запись).
     + Основные функции:
       - I (Input) - ввод
       - S (See) - просмотр
       - A (Authorise) - авторизация
       - D (Delete) - удаление (неавторизованных)
       - L - список Live-записей
       - E - список Unauth-записей
       - H - восстановить из истории
       - R - реверс (отмена авторизованной)
   * **Telnet/PuTTY (для jBASE/TAFC):** (Lesson 08 "TAFC/TAFJ") Доступ к серверу на более низком уровне для разработчиков (компиляция, просмотр файлов).

**Блок 3: Кастомизация и Разработка в T24 – "Допиливаем" под Банк**

Стандартный T24 – это хорошо, но каждый банк хочет что-то свое.

1. **Versions (Версии) – Настройка Экранов** (Lesson 04 "Version", Lesson 07 "Version/Enquiry")
   * *Что это?* Это способ изменить стандартный экран приложения T24, не меняя само приложение. Как бы "надеть другую маску" на стандартную форму.
   * *Зачем?*
     + Сделать экран удобнее для пользователя.
     + Адаптировать под конкретные бизнес-процессы банка.
     + Обеспечить ввод только нужных данных.
   * *Что можно настроить для полей в версии:*
     + **Скрыть поле:** Если оно не нужно.
     + **Запретить ввод (NOINPUT):** Поле только для просмотра.
     + **Запретить изменение (NOCHANGE):** Можно ввести один раз, потом не менять.
     + **Сделать обязательным (MANDATORY):** Пользователь не сможет сохранить запись, пока не заполнит это поле.
     + **Значение по умолчанию (DEFAULT.VALUE):** Поле будет автоматически заполнено этим значением.
     + **Автоматическое заполнение:** Например, поле "Сектор" может заполняться само на основе другого поля.
     + **Подсказка (TOOLTIP):** Текст, который появляется при наведении мышки.
     + **Название поля (PROMPT):** Изменить стандартное название поля на экране.
     + **Заголовок (HEADER):** Добавить текст над группой полей.
     + **Стиль (ATTRS):** Изменить цвет, шрифт (например, class-bold:red).
     + **Ссылка (Hyperlink):** Сделать поле ссылкой на сайт или другой ресурс.
   * *Как создается (очень упрощенно):* В приложении VERSION указывается базовое приложение (например, CUSTOMER), и для каждого поля настраиваются нужные свойства.
2. **Enquiries (Запросы) – Получение Информации** (Lesson 05 "Enquiry", Lesson 06, 07 "Version/Enquiry")
   * *Что это?* Инструмент для выборки данных из таблиц T24 и их отображения в нужном формате. Как создание отчета.
   * *Зачем?*
     + Быстро находить нужную информацию.
     + Анализировать данные.
     + Формировать списки для дальнейшей работы.
   * *Основные настройки запроса:*
     + **File Name:** Какую таблицу опрашиваем (например, CUSTOMER).
     + **Field Name:** Какие поля выводим в результат.
     + **Operation:** Условия отбора (например, SECTOR EQ 2001 – сектор равен 2001). EQ (равно), LK (похоже на), GT (больше), LT (меньше) и т.д.
     + **Fixed Selection:** Условия, которые применяются всегда при запуске запроса.
     + **Page Size:** Сколько строк выводить на одной странице (например, 4,20 – начать с 4-й строки, выводить по 20 строк).
     + **Заголовки (Expressions):** Можно добавить статический текст (@(1,1) 'Список клиентов') или вычисляемые значения в заголовок.
     + **Сортировка (Sort By):** По какому полю сортировать результат.
     + **Drill Down (Проваливание):** Возможность из строки результата запроса перейти к другой записи или запустить другой запрос (например, из списка клиентов перейти к счетам выбранного клиента).
   * **%APPLICATION (например, %CUSTOMER):** Для каждого приложения T24 автоматически создает стандартный запрос с таким именем, который выводит все поля. Его можно использовать как основу для своих запросов.
   * **EB.SELECTION.FAVOURITES:** Пользователь может сохранить часто используемые запросы (с их настройками) в "Избранное".
3. **Menus (Меню) – Навигация для Пользователя** (Lesson 03 "Создание меню")
   * *Что это?* Иерархическая структура, которая позволяет пользователю запускать приложения, версии или запросы.
   * *Как устроено:*
     + **HELPTEXT.MENU:** Создание "подменю" или меню нижнего уровня. В нем указываются пункты: название пункта и какую команду T24 он запускает (например, CUSTOMER,SMW.BANK – запустить версию SMW.BANK приложения CUSTOMER).
     + **HELPTEXT.MAINMENU:** Создание меню верхнего уровня. В него добавляются ссылки на созданные HELPTEXT.MENU.
   * *Как привязать к пользователю:* В приложении USER, в профиле конкретного пользователя, в поле Init Application (или аналогичном) указывается ID главного меню (HELPTEXT.MAINMENU), которое будет ему показываться при входе в систему.
4. **Hook Routines (Java-вставки) – Своя Логика в T24** (Lesson 10 "Разработка HookRoutines")
   * *Что это?* Это Java-код, который разработчик пишет, чтобы добавить какую-то свою, уникальную логику в стандартные процессы T24. "Крючок", который цепляется к системе.
   * *Зачем?*
     + **Валидация данных:** Проверять правильность ввода данных по своим правилам (например, ИНН должен быть 10 цифр, или поле "Лимит" не должно превышать ХХХ).
     + **Автоматическое заполнение полей:** Более сложное, чем в VERSION.
     + **Выполнение дополнительных действий:** Например, при сохранении клиента отправить уведомление.
   * **Как это работает (общая идея):**
     + T24 в определенные моменты работы с записью (например, перед сохранением, при проверке поля) может вызвать такой Java-код.
     + **Ключевой Java-класс (интерфейс), который нужно реализовать:** com.temenos.t24.api.hook.system.RecordLifecycle.
     + **Важные методы этого класса (чтобы ты мог назвать):**
       - validateRecord(String application, String recordId, TStructure record, TStructure lastLiveRecord): Вызывается при проверке всей записи перед авторизацией. Зд есь основные проверки.
       - defaultFieldValues(String application, String recordId, TStructure record, TStructure lastLiveRecord): Для установки значений по умолчанию.
       - updateLookupTable(...): Если нужно что-то записать в свою таблицу при работе с основной.
       - getOverrideMessage(...): Если валидация не прошла, можно выдать свое сообщение (Override).
   * **Пример из лабораторной (Lesson 10):**
     + Проверка поля LIMIT.REF в приложении ACCOUNT.
     + Условие: поле должно начинаться с "REF".
     + Если не начинается, выдать Override-сообщение ACC.LIMIT.REF.
   * **Процесс разработки и внедрения Hook Routine (упрощенная схема):**
     + **Написать Java-код:** В среде разработки (Design Studio, основана на Eclipse) создать Java-класс, который наследуется от RecordLifecycle и реализует нужный метод (например, validateRecord).
     + **Собрать JAR-файл:** Скомпилировать Java-код и упаковать его в JAR-архив.
     + **Загрузить JAR на сервер T24:** Скопировать JAR-файл в специальную директорию на сервере T24 (обычно jboss/modules/com/temenos/t24/main/lib/).
     + **Прописать JAR в module.xml:** В конфигурационном файле module.xml (в той же папке на сервере) добавить путь к новому JAR-файлу, чтобы система его "увидела".
     + **Зарегистрировать рутину в T24:** В приложении EB.API создать запись, где указать:
       - ID записи (например, ACC.LIMIT.REF.VAL).
       - Тип рутины (например, VALIDATION).
       - Имя Java-класса (полное, с пакетом, например, bstu.bank.SmwLimitRefValidate).
       - Имя метода, который будет вызываться (например, validateRecord).
     + **Привязать рутину к приложению/версии:** В приложении VERSION, в настройках нужной версии (например, ACCOUNT,DEMO), в поле Input Routine (или аналогичном, зависит от типа рутины) указать ID записи из EB.API (тот самый ACC.LIMIT.REF.VAL).
     + **Протестировать:** Запустить версию ACCOUNT,DEMO, попробовать ввести некорректное значение в LIMIT.REF и увидеть Override-сообщение.
5. **TAFC/jBASE и Программирование в T24 (для общего понимания)** (Lesson 08 "TAFC/TAFJ", "Temenos Programming Standards")
   * **jBASE:** Это специальная база данных (многомерная) и язык программирования (jBC – jBASE Basic), на котором исторически написана большая часть T24.
   * **TAFC (Temenos Application Framework C version):** Среда компиляции и выполнения программ, написанных на jBASE.
   * **TAFJ (Temenos Application Framework Java version):** Более новая среда, позволяет T24 работать на Java-платформе и интегрироваться с Java-разработками (те же Hook Routines). *Твои Hook Routines работают через TAFJ.*
   * **Разработка на jBASE (если вдруг спросят, что видел):**
     + **PuTTY/Telnet:** Подключение к серверу T24.
     + **JED имя.файла ID.записи:** Редактор для jBASE-программ (например, JED SMW.BP HELLO). .BP – это каталог для исходных кодов программ.
     + **EB.COMPILE имя.файла ID.записи:** Команда для компиляции jBASE-программы.
     + **SET JBCDEV\_BIN=... и SET JBCDEV\_LIB=...:** Установка переменных окружения, чтобы система знала, где лежат компиляторы и библиотеки.
     + **Структура jBASE-программы (очень кратко):**
       - SUBROUTINE ИМЯ\_ПРОГРАММЫ
       - $INSERT I\_COMMON, $INSERT I\_EQUATE, $INSERT I\_F.имя.приложения – подключение стандартных файлов с определениями переменных и функций.
       - Дальше идет логика программы (чтение файлов, обработка данных).
       - RETURN – конец программы.
     + **Файлы данных и словари:** Данные хранятся в файлах (например, FBNK.CUSTOMER). Описание полей этих файлов хранится в словарях (FBNK.CUSTOMER]D).
   * **jBASE SQL Engine:** Возможность выполнять SQL-подобные запросы к данным jBASE (например, SELECT MNEMONIC, ADDRESS FROM FBNK.CUSTOMER).
6. **LOCAL.TABLE, LOCAL.REF.TABLE, STANDARD.SELECTION – Добавление Своих Полей** (Lesson 07 "Version/Enquiry")
   * *Что это?* Инструменты, которые позволяют "легально" добавлять свои, кастомные поля в стандартные приложения T24.
   * **LOCAL.TABLE:** Здесь описывается новое поле (его название, тип, длина). Например, создаем поле CONTROL.TYPE.
   * **LOCAL.REF.TABLE:** Здесь это поле "привязывается" к конкретному приложению (например, к ACCOUNT). После этого поле становится доступно в этом приложении. *Важно: после авторизации записи в LOCAL.REF.TABLE ее нельзя просто так удалить, если поле используется.*
   * **STANDARD.SELECTION:** После добавления через LOCAL.REF.TABLE, информация о новом поле автоматически появляется здесь. Это нужно, чтобы поле можно было использовать в запросах (ENQUIRY).
   * *Результат:* В приложении ACCOUNT (и его версиях) появится новое поле CONTROL.TYPE.

**Блок 4: Практические Сценарии (Лабораторные Работы)**

Постарайся вспомнить, что ты делал в этих работах.

1. **Корреспондентские Отношения и Международные Расчеты** (Lesson 02 "Корреспондентский счет")
   * **Сценарий:** Наш банк (например, BSTUBY3CXXX) хочет проводить операции в долларах США. Для этого ему нужен банк-корреспондент в США (например, Wells Fargo Bank, WFBIUS6SXXX), который имеет прямой доступ к долларовой системе.
   * **Шаги:**
     1. **Договор:** Наш банк заключает договор с Wells Fargo.
     2. **Регистрация Wells Fargo как клиента:** В T24 нашего банка создаем запись для Wells Fargo в приложении CUSTOMER.
     3. **Открытие Ностро-счета:**
        + В приложении ACCOUNT.CLASS находим или создаем класс для Ностро-счетов (например, категория NOSTRO, сектор BANKS).
        + В приложении ACCOUNT создаем счет для учета долларов нашего банка, которые будут "лежать" в Wells Fargo. ID этого счета, например, 86967. Валюта USD.
     4. **Связка Ностро-счета с реальным счетом:** В приложении NOSTRO.ACCOUNT создаем запись. Указываем наш Ностро-счет (86967), банк-корреспондента (Wells Fargo), и номер НАШЕГО реального счета в Wells Fargo (например, 222333444). Также указываем, для каких приложений этот корсчет будет доступен (например, FT – для переводов, ALL – для всех).
     5. **Зачисление денег клиенту через корсчет:**
        + Клиент нашего банка (SMW60 LLC, счет 86975) ждет 10000 USD от компании XYZ Corp.
        + XYZ Corp платит через свой банк, указывая реквизиты: Банк получателя – наш банк (BSTUBY3CXXX), Банк-корреспондент – Wells Fargo (WFBIUS6SXXX), Счет в банке-корреспонденте – 222333444 (наш счет в Wells Fargo), Получатель – SMW60 LLC, счет 86975.
        + Wells Fargo получает деньги, зачисляет их на наш счет 222333444 и отправляет нам SWIFT-сообщение.
        + Наш банк, получив SWIFT, делает проводку в T24 через FUNDS.TRANSFER (FT): Дебет Ностро-счета 86967 (деньги "ушли" с нашего корсчета), Кредит счета клиента 86975 (деньги "пришли" клиенту).
     6. **Списание денег со счета клиента через корсчет (аналогично, но наоборот).**
   * *Главная идея:* Ностро-счет – это "зеркало" нашего счета в другом банке.
2. **Создание TAB-Приложения** (Lesson 08 "Menu, Version, Enquiry, Tab")
   * *Что это?* Способ объединить на одном экране несколько разных элементов T24 (версии, запросы, меню, ссылки) в виде вкладок.
   * *Инструмент:* Screen Designer (с опцией Tabbed Screen).
   * *Пример из лабы:* Создать "Рабочее место для менеджера по корпоративным клиентам" (XXX.CORP).
     1. Вкладка 1 "New Customer": Версия для создания нового корпоративного клиента (CUSTOMER,XXX.CORP.CUS).
     2. Вкладка 2 "New Account": Версия для открытия счета корпоративному клиенту (ACCOUNT,XXX.CORP.ACC).
     3. Вкладка 3 "Customers": Запрос для просмотра списка корпоративных клиентов (ENQUIRY %XXX.CORP.CUS).
     4. Вкладка 4 "Accounts": Запрос для просмотра счетов корпоративных клиентов (ENQUIRY %XXX.CORP.ACC).
     5. Вкладка 5 "Funds Transfer": Версия для выполнения переводов (FUNDS.TRANSFER,XXX.CORP.FT).
   * *Настройки для версий и запросов в рамках TAB-приложения:*
     1. Значения полей по умолчанию (например, SECTOR = 2001, ACCOUNT.OFFICER = 61, TARGET = 1 для корпоративных клиентов).
     2. Ограничения на отображение (например, показывать только корпоративных клиентов/счета).
     3. Сортировка результатов поиска (например, по ID в порядке убывания).

**Советы по ответам на собеседовании:**

* **"Расскажите о своем опыте с T24"**: Начни с того, что изучал архитектуру, основные модули (CUSTOMER, ACCOUNT, FT), принципы работы (жизненный цикл записи, ввод-авторизация). Потом переходи к кастомизации: создавал Versions для настройки экранов, Enquiries для получения данных, Menus для навигации. Обязательно упомяни разработку Hook Routines на Java для кастомной валидации (приведи пример с LIMIT.REF или любой другой, который помнишь). Скажи, что знакомился с jBASE и TAFC/TAFJ. Если помнишь лабораторные, кратко опиши, что делал (корсчета, TAB-приложение).
* **"Что такое Hook Routine?"**: Это Java-программа, которая позволяет добавить свою бизнес-логику в стандартные процессы T24. Например, для дополнительной проверки данных, которые вводит пользователь. Она "цепляется" к определенному этапу жизненного цикла записи (например, к этапу валидации). Расскажи упрощенно шаги создания (код -> JAR -> деплой -> регистрация в EB.API -> привязка к VERSION).
* **"Чем Version отличается от Enquiry?"**: Version – это настройка **экрана** для ввода или просмотра **одной записи** приложения (как она выглядит, какие поля обязательны). Enquiry – это инструмент для **выборки и отображения списка многих записей** по определенным критериям (как отчет).
* **"Объясните жизненный цикл записи в T24"**: Unauth (ввели, не проверили,
* NAU)−>Авторизациядругимпользователем−>Live(проверена,активна,основнойфайл)−>ИзменениеLive−записи−>СтараяLiveуходитвHistory(NAU) -> Авторизация другим пользователем -> Live (проверена, активна, основной файл) -> Изменение Live-записи -> Старая Live уходит в History (NAU)−>Авторизациядругимпользователем−>Live(проверена,активна,основнойфайл)−>ИзменениеLive−записи−>СтараяLiveуходитвHistory(

HIS).

* **Будь готов нарисовать простой банковский баланс (Активы/Пассивы) или объяснить его на пальцах.**
* **Говори проще.** Не используй сложные термины, если не уверен в них на 100%. Лучше объяснить суть простыми словами.

**Самое главное на оставшееся время:**

1. **Пробегись глазами по этому тексту еще раз.**
2. **Сконцентрируйся на выделенных жирным ключевых понятиях.**
3. **Представь, что тебе нужно объяснить это человеку, который вообще не знает, что такое T24.**

У тебя все получится! Удачи!

Понял, добавил презентацию "Screen Hooks" к остальным материалам.

Эта презентация очень важна, так как она **детально описывает механизм Hook Routines (хуков)**, который мы уже затрагивали.

**Ключевые моменты из "Screen Hooks", на которые стоит обратить ОСОБОЕ внимание:**

1. **Exit Points / Trigger Points (Слайд 5):**
   * T24 спроектирована с возможностью расширения. "Exit points" – это точки в стандартном потоке выполнения приложения, где можно "подключить" свой пользовательский код (хук-рутину).
   * Система выходит из обычного потока, выполняет твою рутину, а затем возвращается.
2. **Типы валидаций и этапы, где они срабатывают (Слайд 5):**
   * **Unauthorized Stage (До авторизации):**
     + **ID Level Validation:** Проверка самого ID записи.
     + **Record Level (CheckRec routine):** Проверка значений в записи *перед* отображением пользователю или установка значений по умолчанию.
     + **Auto Field routines:** Установка значений по умолчанию/изменение значений в поле *перед* отображением записи.
     + **Field Validations (validation hooks):** Дополнительные проверки полей (стандартные проверки изменить нельзя).
     + **Record Validations (Input routines):** Дополнительные проверки на уровне всей записи, здесь можно генерировать Override-сообщения. (Твой пример из Lesson 10 как раз сюда относится).
     + **Processing after commit of a record (After Unauth routines):** Обработка *после* сохранения неавторизованной записи (например, запись данных в другие файлы).
   * **Authorization Stage (Во время авторизации):**
     + **Before Authorised Write (Before auth routines):** Проверки *непосредственно перед* тем, как запись станет Live. Можно записывать данные в новые файлы.
     + **After Authorised Write (Auth routines):** Действия *после* того, как запись стала Live и все обновления файлов завершены.
3. **Поля в VERSION для подключения хуков (Слайд 7 - таблица):**
   * **ID.RTN:** Хук для валидации ID записи. (Java метод: checkId)
   * **CHECK.REC.ROUTINE:** Хук уровня записи для проверки/установки значений по умолчанию перед отображением. (Java метод: defaultFieldValues)
   * **AUT.NEW.CONTENT:** Хук уровня поля для установки/изменения значений по умолчанию перед отображением. (Java метод: defaultFieldValues)
   * **VALIDATION.RTN:** Хук уровня поля для валидации данных в поле.
   * **INPUT.RTN:** Хук уровня записи для выполнения дополнительных валидаций. **(Это то, что ты делал в Lesson 10, метод validateRecord)**.
   * **DEFAULT.ROUTINE:** Вызывается при валидации записи и "горячей" валидации поля.
   * **BEFORE.AUTH.RTN:** Вызывается перед финальным обновлением файлов на этапе авторизации. **(Метод updateCoreRecord или updateLookupTable)**.
   * **AUTH.ROUTINE:** Вызывается после финального обновления файлов на этапе авторизации. **(Метод postUpdateRequest)**.
4. **Метод validateRecord() (Слайды 8, 9):**
   * **Назначение:** Позволяет разработчику вернуть validationResponse, который содержит ошибки и/или дополнения (enrichments). Вызывается, когда пользователь валидирует запись.
   * **Где указывается в T24:** Поле INPUT.ROUTINE в VERSION или VERSION.CONTROL, или поле CROSSVAL.PROC в EB.TABLE.PROCEDURES.
   * **Параметры метода (чтобы ты понимал, что приходит на вход):**
     + application: Имя приложения.
     + currentRecordId: ID текущей обрабатываемой записи.
     + currentRecord: Сама текущая запись (в виде объекта TStructure).
     + unauthorisedRecord: Последняя неавторизованная версия записи.
     + liveRecord: Последняя Live-версия записи.
     + transactionContext: Контекст транзакции.
   * **Возвращаемое значение:** TValidationResponse – объект, содержащий информацию об ошибках (errors), переопределениях (overrides) и обогащениях (enrichments).
5. **Метод updateCoreRecord() (Слайды 10, 11):**
   * **Назначение:** Позволяет разработчику определить одну или несколько записей, которые должны быть созданы/изменены во время текущей транзакции (при авторизации основной записи). Например, при создании депозита автоматически создать связанный счет.
   * **Когда вызывается:** Когда запись авторизуется.
   * **Где указывается в T24:** Поле BEF.AUTH.PROC в EB.TABLE.PROCEDURES или BEFORE.AUTH.RTN в VERSION / VERSION.CONTROL.
   * **Параметры похожи на validateRecord**, но добавляются versionNames (имена версий для создаваемых записей), isZeroAuth (флаг нулевой авторизации), currentRecordIds и currentRecords (для множественных записей).
   * **Возвращаемое значение:** TValidationResponse (хотя тут основной эффект – создание/изменение других записей).
6. **Метод postUpdateRequest() (Слайды 12, 13):**
   * **Назначение:** Позволяет отправить запрос на обновление записей любой таблицы в асинхронном режиме (т.е. не немедленно, а когда система освободится).
   * **Как работает:** Запрос помещается в очередь OFS.MESSAGE.QUEUE, а специальная служба OFS.MESSAGE.SERVICE его обрабатывает.
   * **Где указывается в T24:** Поле AFTER.AUTH.PROC в EB.TABLE.PROCEDURES или AUTH.ROUTINE в VERSION / VERSION.CONTROL.
   * **Параметры:** Включают transactionData (данные транзакции для формирования запроса) и currentRecords (записи, которые были обновлены).
   * **Возвращаемое значение:** nothing (ничего).
7. **Практический пример (Practice 5.1 - validateRecord, Слайды 15-19):**
   * **Задача:** Выдать Override или Error сообщение, если валюта дебета и кредита не совпадают в транзакции FUNDS.TRANSFER.
   * **Решение:**
     + В методе validateRecord получить значения полей валюты дебета (ftRecord.getDebitCurrency()) и валюты кредита (ftRecord.getCreditCurrency()).
     + Сравнить их значения (!debitCurrency.getValue().equals(creditCurrency.getValue())).
     + Если не совпадают, сформировать список параметров для Override-сообщения и установить его для поля валюты кредита (creditCurrency.setOverride(parameters.toString())).
     + Вернуть ftRecord.getValidationResponse().
     + **Регистрация в EB.API:** Создать запись (например, FT.CCY.DIFF.ERROR), указать Source Type = Method, класс, метод (validateRecord), пакет.
     + **Привязка к VERSION:** В версии FUNDS.TRANSFER,ACVERSION на вкладке API добавить созданную запись EB.API как Input Routine (After Commit) (или другой подходящий тип хука, судя по скрину это Input Routine).

**Что это значит для твоей подготовки:**

* **Хуки – это способ кастомизации T24 на уровне кода.** Это глубже, чем Versions и Enquiries.
* **Ты должен понимать, на каких этапах обработки записи можно вмешаться** (до отображения, при валидации поля, при валидации всей записи, до/после авторизации).
* **Метод validateRecord – самый частый для проверок.** Ты его реализовывал.
* **Процесс внедрения хука:** Java-код -> JAR -> module.xml -> EB.API -> VERSION. Это важно понимать.
* **Разница между Override и Error:** Error – фатальная ошибка, не дает сохранить. Override – предупреждение, которое пользователь может проигнорировать (если разрешено).

**Теперь, с учетом этой презентации, твой ответ про Hook Routines может быть еще полнее:**

"Hook Routines – это Java-классы, которые позволяют разработчикам расширять стандартную функциональность T24, встраивая собственный код в предопределенные "точки выхода" (exit points) на различных этапах жизненного цикла записи. Например, можно добавить кастомную валидацию данных при вводе записи (используя метод validateRecord интерфейса RecordLifecycle), установить значения полей по умолчанию, или выполнить какие-то действия до или после авторизации записи (используя методы updateCoreRecord или postUpdateRequest). Для внедрения хука необходимо написать Java-код, скомпилировать его в JAR-файл, разместить на сервере, прописать в module.xml, зарегистрировать в приложении EB.API и затем привязать к нужной VERSION или полю в VERSION (например, через INPUT.RTN). Это позволяет гибко адаптировать систему под специфические требования банка."

Этот материал очень хорошо дополняет Lesson 10. Просмотри слайды 5 и 7 внимательно, чтобы запомнить, какие типы хуков бывают и за что отвечают поля в VERSION.

Теперь я точно готов к финальному анализу!