

Билет 1

1. Уровень бизнес-логики в программных системах зачем нужен, какую роль играет, как взаимодействует с другими уровнями.

Отвечает за реализацию предметной области и формирует основную вычислительную нагрузку. Бизнес-логика сложна в проектировании, обычно многопоточная, часто - распределенная (размазана по набору однородных или неоднородных узлов, занимающихся обработкой данных).

Находится между инфраструктурным уровнем (например, уровнем доступа к БД) и сервисным уровнем (REST API).

2. Распределенные транзакции, спецификация XA

Транзакция - группа последовательных операций, представляющая собой логическую единицу работы с данными.

Транзакции бывают:

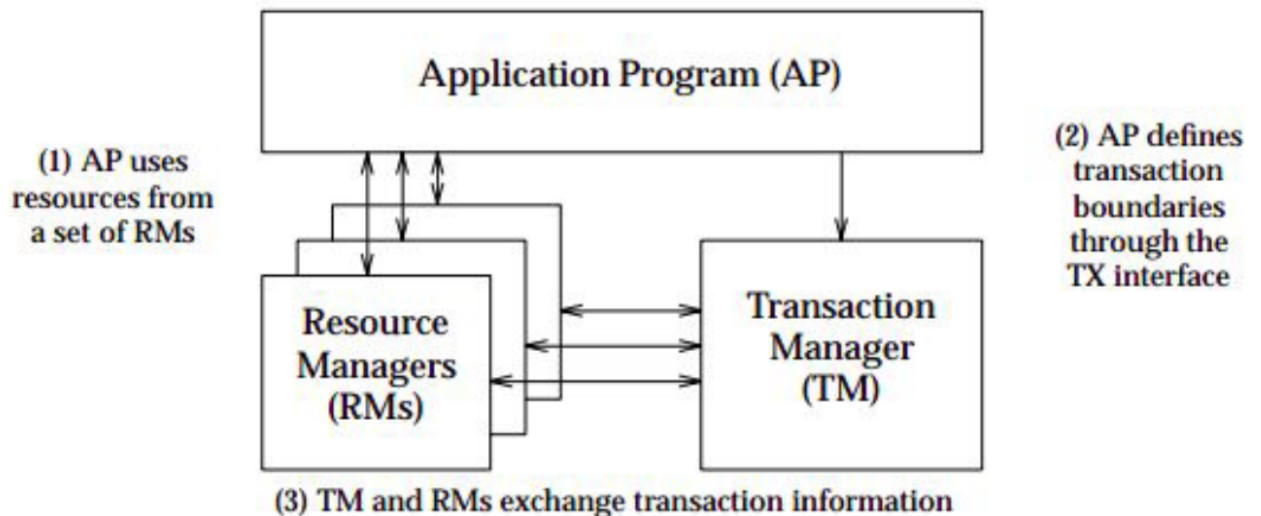
- Последовательные (обычные)
- Параллельные (в многопользовательских системах)
- Распределенные (в распределенных системах)

В параллельных транзакциях сервисы, которые обрабатывают транзакции, функционируют параллельно в разных потоках или процессах и меняют данные независимо друг от друга.

В случае распределенных транзакций все работает аналогично обычным транзакциям, за исключением того, что нужно синхронизировать работу всех источников. Для этого используется менеджер управления распределенными транзакциями.

XA, X/OpenXA (eXtended Architecture) - спецификация распределенных транзакций. Является индустриальным стандартом в реализации менеджеров транзакций. Определяет принципы взаимодействия с транзакционными ресурсами в распределенных ресурсах.

Модель ХА



- Менеджеры ресурсов (RM) - управляют конкретными ресурсами (например, СУБД, файловым хранилищем).
- Менеджер транзакций (TM, он же координатор) - координирует работу RM и принимает решение о фиксации или откате транзакции.
- Прикладная программа (AP) - описывает бизнес-логику, управляет составом транзакций и используемыми в ней RM (внешний ресурс не входит в состав ХА).
- В ХА специфицировано только взаимодействие между RM и TM, а за AP отвечает программист.

Виды транзакций в ХА:

- Локальные -- используется только одно хранилище, роль TM отводится AP. Само определяет, когда коммитить, а когда откатывать транзакцию.
- Глобальные (распределённые):
 - Несколько хранилищ.
 - AP задает границы транзакций.
 - Управлением транзакциями занимается координатор.(что делать с хранилищем). Менеджер транзакций делит одну глобальную транзакцию на несколько выполняющихся локально веток (англ. branch), и при успешной фиксации на каждой из веток осуществляет фиксацию всей глобальной транзакции, и осуществляет откат в противном случае.

Двухфазный коммит имеет два четко разделенных этапа:

- Этап подготовки - координатор посылает сообщение участникам распределенной транзакции о подготовке к транзакции (prepare message). Это сообщение также содержит уникальный номер транзакции TID. Когда участники получают это сообщение, они проверяют смогут ли зафиксировать

транзакцию и отвечают координатору. При этом транзакция выполняется, но не фиксируется и ее состояние сохраняется на диске.

- Этап фиксации - после того, как координатор получил все ответы, он решает зафиксировать или прервать начатую транзакцию в соответствии с правилом глобальной фиксации. Если все участники глобальной транзакции ответили, что могут ее зафиксировать, то он посылает участникам сообщение о фиксации транзакции, в противном случае все участники получают сообщение об откате транзакции.

- Вложенные:
 - Транзакция запускается внутри существующей.
 - Изменения фиксируются только при фиксации "верхней" транзакции.

3. **Диаграмма BPMN 2.0 для бизнес-процесса управления автомашина по продаже кофе (кофе-машина). Автомат должен уметь приготавливать несколько видов кофе, добавлять в кофе сливки и сахар и принимать к оплате наличные и карты и выдавать сдачу.**

<https://drive.google.com/file/d/1FcBR5zZ-QivXPxe4VO8fiK0eOmy7iy9E/view?usp=sharing> можете исправлять если что, я хз, как это адекватно сделать

Билет 2

1. **Spring Boot зачем нужен, как применяется, отличие от Spring обычного.**

Spring Boot — это, по сути, расширение среды Spring, которое устраняет стандартные конфигурации, необходимые для настройки приложения Spring.

Spring Boot:

- Набор утилит для автоматизации процессов настройки, создания и развертывания приложения.
- Избавляет от необходимости повторять рутинные действия.
- Позволяет более эффективно управлять зависимостями и упростить конфигурацию приложения.

Spring Boot делает менее заёбным притягивание зависимостей, включает внутри себя сервер Tomcat'a, имеет профилирование (Spring Boot Profile) и прочее.

2. **Программное и декларативное выполнение транзакций в spring.**

Декларативное – аннотация @Transactional

Программное – TransactionTemplate:

1. Инициализируем через TransactionalManager

```
// test annotations
class ManualTransactionIntegrationTest {

    @Autowired
    private PlatformTransactionManager transactionManager;

    private TransactionTemplate transactionTemplate;

    @BeforeEach
    void setUp() {
        transactionTemplate = new TransactionTemplate(transactionManager);
    }

    // omitted
}
```

2. Хуярим как то так (если нам не всрался результат)

```
void givenAPayment_WhenNotExpectingAnyResult_ThenShouldCommit() {
    transactionTemplate.execute(new TransactionCallbackWithoutResult() {
        @Override
        protected void doInTransactionWithoutResult(TransactionStatus status) {
            Payment payment = new Payment();
            payment.setReferenceNumber("Ref-1");
            payment.setState(Payment.State.SUCCESSFUL);

            entityManager.persist(payment);
        }
    });

    assertThat(entityManager.createQuery("select p from Payment p").getResultList()).hasSize(1);
}
```

3. А если всрался – так

```
void givenAPayment_WhenNotDuplicate_ThenShouldCommit() {
    Long id = transactionTemplate.execute(status -> {
        Payment payment = new Payment();
        payment.setAmount(1000L);
        payment.setReferenceNumber("Ref-1");
        payment.setState(Payment.State.SUCCESSFUL);

        entityManager.persist(payment);

        return payment.getId();
    });

    Payment payment = entityManager.find(Payment.class, id);
    assertThat(payment).isNotNull();
}
```

Мы также можем менять конфигурацию транзакций (как и с `@Transactional` передавая ей параметры)

For example, we can set the `transaction isolation level`:

```
transactionTemplate = new TransactionTemplate(transactionManager);  
transactionTemplate.setIsolationLevel(TransactionDefinition.ISOLATION_REPEATABLE_READ);
```

Similarly, we can change the transaction propagation behavior:

```
transactionTemplate.setPropagationBehavior(TransactionDefinition.PROPGATION_REQUIRES_NEW);
```

На почитать: <https://www.baeldung.com/spring-programmatic-transaction-management>

P.S. Также программное управление транзакций можно напрямую реализовать через *TransactionManager* (если ты мазохист или долбаёб)

3. Нарисовать BPMN 2.0 диаграмму: 1 ноября и 1 апреля должны отчислять студентов у которых хотя бы одна Академ. задолженность.

[expulsion](#)

Может исправлять Егор Кривоносов :)

Билет 3

1. BPMN и моделирование бизнес процессов.

Моделирование бизнес процессов:

- Набор формальных подходов к описанию бизнес-процессов.
- Бизнес-процесс разбивается на состояния и переходы между ними.
- Существует ПО для визуальной отрисовки схем бизнес-процессов.
- Существует ПО для генерации кода по описанию бизнес-процесса

В то время как BPM является предметом или философией управления (КОНЦЕПЦИЕЙ), которая может быть применена к бизнесу, BPMN является официальной стандартной нотацией, созданной этим предметом, которая используется во всем мире.

Другими словами, BPMN призвана визуально поддерживать и стандартизировать реализацию управления бизнес-процессами. BPMN — это набор символов и графических представлений, которые вместе показывают весь поток процесса, его действия, события, шлюзы, соединения и другие элементы.

2. Политика безопасности в корпоративных приложениях, зачем, за что отвечают, особенности реализации на уровне Бизнес-логики

- “Закрыть” веб-приложение http basic auth обычно недостаточно!
- Политики задают “сквозные” правила доступа к компонентам архитектуры приложения.
- Применимы не только для веб-приложений.
- Вторая важная задача – отображение набора прав доступа на набор ролей конкретного пользователя.

На уровне бизнес-логики реализуются как раз таки с использованием ролей и привилегий. Роль определяет то кем является пользователь, а привилегия то, что пользователь может делать.

Роль - высокоуровневая залука, может выдаваться новым пользователям, например, администратором. Каждой роли соответствует набор привилегий - низкоуровневых залуп.

С помощью привилегий мы можем разграничивать доступ к различным ресурсам (разрешать чтение, изменение только определённой привилегии) и прочее

3. Написать класс, реализующий транзакцию, которая получает Почку, Seller и Buyer и совершает акт купли-продажи. Причем продавец не может суммарно продать, а покупатель купить более 2 почек.

```
class KidneyService {
    @Autowired
    private KidneyRepository kidneyRepo;
    @Autowired
    private SellerRepository sellerRepo;
    @Autowired
    private BuyerRepository buyerRepo;

    @Transactional
    public void kidneyTrading(Kidney kidney, Seller s, Buyer b) {
        Seller seller = sellerRepo.findSellerById(s.getId());
        Buyer buyer = buyerRepo.findSellerById(b.getId());

        List<Kidney> sellerSoldKidneys = seller.getSoldKidneys();
        List<Kidney> buyerBoughtKidneys = buyer.getBoughtKidneys();

        if (sellerSoldKidneys.size() >= 2 || buyerBoughtKidneys.size() >= 2)
            throw new KidneyLimitExceededException("Fuck trading"); //extends RuntimeException

        sellerSoldKidneys.add(kidney);
        buyerBoughtKidneys.add(kidney);

        kidney.setOwner(buyer);
        // Не обязательно

        kidneyRepo.save(kidney);
        sellerRepo.save(seller);
        buyerRepo.save(buyer);
    }
}
```

P.S. Я нихуя не понял, что хотят, можно реализовать еще 1000 разными способами в зависимости от того, что Цопа блядь имеет в виду

Билет 4

1. BPMN

Моделирование бизнес-процессов - деятельность по представлению процессов предприятия, позволяющая анализировать, улучшать и автоматизировать текущие бизнес-процессы. Набор формальных подходов к описанию бизнес-процессов.

Бизнес-процесс разбивается на состояния и переходы между ними. Существует ПО для визуальной отрисовки схем бизнес-процессов. Существует ПО для генерации кода по описанию бизнес-процесса.

Система условных обозначений для моделирования бизнес-процессов. Транслируется в XML. Специфицирована Object Management Group, актуальная версия стандарта -- 2.0.2 (2014 г.). Моделирование осуществляется путём составления диаграмм (моделей), состоящих из стандартизированных элементов.

BPMN (Business Process Management Notation) – это язык моделирования бизнес-процессов, который является промежуточным звеном между формализацией/визуализацией и воплощением бизнес-процесса. Представляет собой описание графических элементов, используемых для построения схемы протекания бизнес-процесса.

Как минимум, такая схема нужна, чтобы выстроить в соответствии с ней бизнес процесс и понятно регламентировать его для всех участников. Немаловажным является то, что моделирование BPMN позволяет впоследствии провести автоматизацию бизнес-процессов в соответствии с имеющейся схемой.

Важно отметить, что одной из причин создания BPMN явилась необходимость построения простого механизма для проектирования как простых, так и сложных моделей бизнес- процессов. Для удовлетворения двух этих противоречащих требований был применен подход систематизации графических элементов нотации по категориям. Результатом явился небольшой перечень категорий нотаций, позволивший людям, работающим с диаграммами BPMN, без труда распознавать основные типы элементов и осуществлять корректное чтение схем.

Преимущества BPMN

- Первое – простота трансляции диаграмм в исполняемые модели с помощью языка формального описания бизнес-процессов.
- Описание элементов BPMN является понятным для большинства участников бизнес-процессов и часто не требует никаких дополнительных разъяснений. С помощью простого графического выражения можно составить конкретные регламенты, которые будут исполняться сотрудниками.

- Наряду с тем, что описание нотации BPMN 2.0 позволяет добиться понимания сотрудниками того, как происходят бизнес-процессы, данную нотацию поддерживают большинство современных инструментов бизнес-моделирования, что позволяет импорт готовых схем бизнес-процессов в BPM-системы.

2. Spring security (роли и т.д.)

Фреймворк аутентификации, авторизации и ролевого разграничения доступа в составе Spring.

Обеспечивает сквозное разграничение доступа на всех уровнях.

Полномочий нет - только роли.

Есть возможность интеграции с JAAS.

1. Описание основных используемых аннотаций

Controller – специальный тип класса, применяемый в MVC приложениях. Похож на обычный сервлет `HttpServlet`, работающий с объектами `HttpServletRequest` и `HttpServletResponse`, но с расширенными возможностями от Spring Framework.

Repository – указывает, что класс используется для задания перечня необходимых работ по поиску, получению и сохранению данных. Аннотация может использоваться для реализации шаблона DAO.

Service – указывает, что класс является сервисом для реализации бизнес логики.

Configuration – эта аннотация используется для классов, которые определяют bean-компоненты.

Autowired – аннотация позволяет автоматически установить значение поля. Функциональность этой аннотации заключается в том, что нам не нужно заботиться о том, как лучше всего Bean'у передать экземпляр другого Bean'a. Spring сам найдет нужный Bean и подставит его значение в свойство, которое отмечено аннотацией.

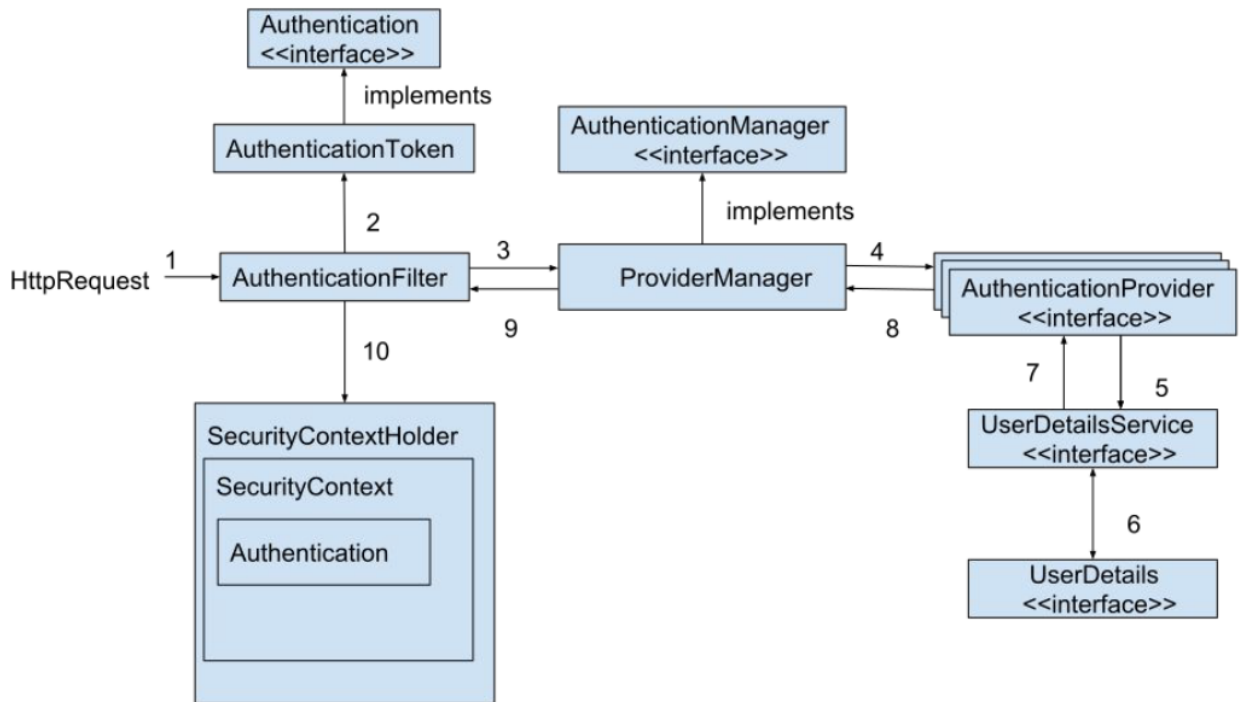
Немного информации о Spring Security

Самым фундаментальным объектом является **SecurityContextHolder**. В нем хранится информация о текущем контексте безопасности приложения, который включает в себя подробную информацию о пользователе (принципале), работающим с приложением. Spring Security использует объект **Authentication**, пользователя авторизованной сессии.

«Пользователь» – это просто Object. В большинстве случаев он может быть приведен к классу **UserDetails**. **UserDetails** можно представить, как адаптер между БД пользователей и тем что требуется Spring Security внутри **SecurityContextHolder**.

Для создания **UserDetails** используется интерфейс **UserDetailsService**, с единственным методом:

```
UserDetails loadUserByUsername(String username) throws UsernameNotFoundException
```



- Запрос, проходя по цепочке фильтров, отлавливается фильтром, настроенный на аутентификацию запросов на определённый URL с определённым типом запроса
- Данный фильтр создаёт объект типа Authentication и направляет его в AuthenticationManager, дефолтной реализацией которого является ProviderManager. Если необходимо, вы всегда можете написать свою реализацию.
- AuthenticationManager возвращает авторизованный объект типа Authentication с вложенными в него дополнительными сведениями о пользователе. ProviderManager делает это с помощью заданного набора AuthenticationProvider'ов, каждый из которых имеет метод supports, определяющий применим ли данный провайдер к данной реализации Authentication
- Authentication возвращает то же самое, что и AuthenticationManager, т.е. проводит авторизацию пользователя, устанавливает его роли и другую информацию. Как правило для этого используется интерфейс UserDetailsService или его расширение UserDetailsManager, которые возвращают объект типа UserDetails, содержащий информацию о пользователе.
- В случае неудачной авторизации пробрасывается исключение типа AuthenticationException, в случае удачной возвращается авторизованный объект типа Authentication, который сохраняется в SecurityContext.

Ключевые объекты контекста Spring Security:

- SecurityContextHolder, в нем содержится информация о текущем контексте безопасности приложения, который включает в себя подробную информацию о пользователе(Principal) работающем в настоящее время с приложением.
- SecurityContext, содержит объект Authentication и в случае необходимости информацию системы безопасности, связанную с запросом от пользователя.
- Authentication представляет пользователя (Principal) с точки зрения Spring Security.
- GrantedAuthority отражает разрешения выданные пользователю в масштабе всего приложения, такие разрешения (как правило называются «роли»), например ROLE_ANONYMOUS, ROLE_USER, ROLE_ADMIN.
- UserDetails предоставляет необходимую информацию для построения объекта Authentication из DAO объектов приложения или других источников данных системы безопасности. Объект UserDetails содержит имя пользователя, пароль, флаги: isAccountNonExpired, isAccountNonLocked, isCredentialsNonExpired, isEnabled и Collection — прав (ролей) пользователя.
- UserDetailsService этот сервис просто вытаскивает пользователя по полученным данным из хранилища (БД, памяти, properties-файла, нужно подчеркнуть).

Разграничение доступа с помощью hasAuthority()

```
@Profile({"production"})
@EnableWebSecurity
@EnableGlobalMethodSecurity(prePostEnabled = true)
public class ProductionSecurityConfig extends WebSecurityConfigurerAdapter {

    @Override
    public void configure(HttpSecurity http) throws Exception {
        http
            .csrf().disable()
            .authorizeRequests()
            .antMatchers("/admin/**").hasAnyAuthority(Role.SUPER_USER.getAuthority())
            .antMatchers("/student/**").hasAnyAuthority(Role.STUDENT.getAuthority(),
                Role.SUPER_USER.getAuthority())
            .antMatchers("/university/**").hasAnyAuthority(Role.UNIVERSITY.getAuthority(),
                Role.SUPER_USER.getAuthority())
            .antMatchers("/public/**").permitAll()
            .antMatchers("/**").authenticated();
    }
}
```

Аннотация @PreAuthorize

```
@GetMapping("/{certId}")
@PreAuthorize("hasAuthority(T(ru.edu.portfolio.domain.ApiPermission).COURSE_CERTIFICATE_READ)")
public List<SomeDTO> read(@PathVariable long certId) {
    // Some logic
}

@GetMapping("/{certId}")
@PreAuthorize("authentication.details.SSLAuth")
public List<SomeDTO> readWithAuth(@PathVariable long certId) {
    // Some logic
}
```

3. Spring MVC rest. Штука, которая переводит деньги, и проверяет не превышен ли лимит переводов за месяц.

Я хз что он тут хочет, поэтому вот микс всего что можно, мейби нужно просто контроллер и вызов сервиса

```
@RestController
public class MoneyController {
    final UserRepository userRepository;

    @Autowired
    public MoneyController(UserRepository userRepository) {
        this.userRepository = userRepository;
    }

    @PostMapping("/transfer")
    @Transactional
    public void transferMoney(@RequestParam long from, @RequestParam long to, @RequestParam long amount) {
        User sender = userRepository.findById(from);
        User receiver = userRepository.findById(to);

        if (sender.getTransferLimit() < sender.calculateMonthlyTransferredAmount() + amount) {
            throw new TransferLimitExceededException();
        }

        if (sender.getBalance() < amount) {
            throw new NotEnoughMoneyException();
        }

        sender.setBalance(sender.getBalance() - amount);
        receiver.setBalance(receiver.getBalance() + amount);
    }
}
```

Билет 5

1. Основные концепции использующиеся в разработке бизнес логики.

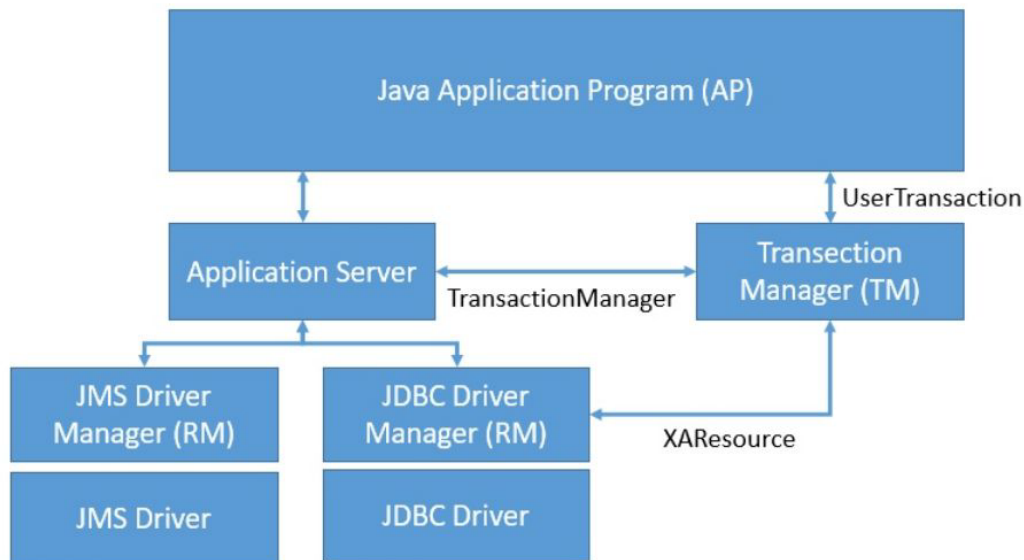
Бизнес логика:

- Формирует основную вычислительную нагрузку на “железо”.
- Сложна в проектировании, обычно многопоточная, часто – распределенная.
- При разработке часто используется компонентный подход.

2. Выполнение. Распределенные транзакции в Spring и Jakarta EE.

Java EE предоставляет JTA (Java Transaction API):

- API для управления распределенными транзакциями.
- Специфицирован, входит в состав Java EE.
- Контейнер содержит менеджер транзакций, манипулирующий ресурсами.
- Реализует стандарт X/OpenXA

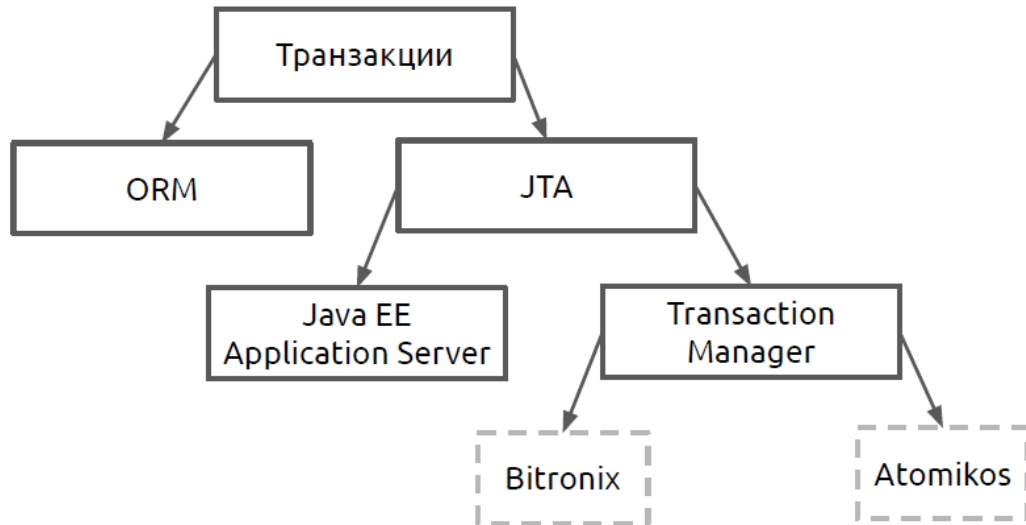


Особенности:

- Поддержка реализуется на уровне сервера приложений (DataSource) и JDBC-драйвера (XAResource и др.).
- На несовместимых с XA JDBC-драйверах распределённые транзакции работать не будут!
- Два высокоуровневых режима управления транзакциями (в EJB) -- программный (BMT – реализует программист, ручками начиная, завершая и отменяя транзакцию

с помощью UserTransaction) и декларативный (CMT – реализуется с помощью аннотаций (@TransactionAttribute), за выполнение отвечает контейнер).

Транзакции в Spring



- Как и в Java EE, два режима – программный и декларативный.
- Разработчики Spring рекомендуют использовать декларативный режим.
- Режим “по умолчанию” использует менеджер транзакций ORM-фреймворка.
- Можно использовать транзакции JTA, для этого нужен провайдер -- сервер приложений Java EE или отдельный менеджер транзакций.

Spring JTA:

- Приложение на базе Spring Boot может работать с распределенными транзакциями JTA.
- Два варианта реализации:
 - Использование сервера приложений Java EE с поддержкой JTA.
 - Использование встроенного менеджера транзакций с поддержкой JTA (Bitronix, Atomikos).

3. **ВРМН для управления ядерным реактором. Исключение несанкционированной ядерной реакции с помощью отрицательной обратной связи. (ВРМН для ядерного реактора. Должен реализовывать аварийную остановку)**

Билет 6

1. Описание bpmn 2.0. принципы построения, основные элементы.

В спецификации BPMN 2.0 содержится более подробное, нежели содержащееся в спецификации BPMN 1.2, описание возможностей и областей использования нотации, а именно:

- a. Определение механизма изменения как расширений модели бизнес-процесса, так и для расширений графических элементов,
- b. Детализация состава и корреляции События,
- c. Расширение определения пользовательских действий,

Основные элементы:

- Пул и Дорожки
- Действия
- Шлюзы или Развилки
- События
- Потоки
- Arteфакты

BPMN - это набор символов и графических представлений, которые вместе показывают весь процесс, его действия, события, шлюзы, соединения и другие элементы

2. Spring Security. Что, зачем, почему. Аннотации, файлы конфигурации.

Spring Security это **Java/JavaEE framework**, предоставляющий механизмы построения систем аутентификации и авторизации, а также другие возможности обеспечения безопасности для корпоративных приложений, созданных с помощью SpringFramework.

Аннотация **@Secured** используется для указания списка ролей в методе. Следовательно, пользователь может получить доступ к этому методу только в том случае, если у него есть хотя бы одна из указанных ролей.

Аннотация **@PreAuthorize** проверяет данное выражение перед вводом метода, в то время как аннотация **@PostAuthorize** проверяет его после выполнения метода и может изменить результат.

Подробнее: <https://habr.com/ru/post/470786/>

3. REST API для системы быстрых платежей с проверкой месячного ограничения переводов в 100 000 р

Я хз что нужно кроме рау, так что скопирую из билета выше

```

@RestController
public class MoneyController {
    final UserRepository userRepository;

    @Autowired
    public MoneyController(UserRepository userRepository) {
        this.userRepository = userRepository;
    }

    @PostMapping("/transfer")
    @Transactional
    public void transferMoney(@RequestParam long from, @RequestParam long to, @RequestParam long amount) {
        User sender = userRepository.getById(from);
        User receiver = userRepository.getById(to);

        if (sender.getTransferLimit() < sender.calculateMonthlyTransferredAmount() + amount) {
            throw new TransferLimitExceededException();
        }

        if (sender.getBalance() < amount) {
            throw new NotEnoughMoneyException();
        }

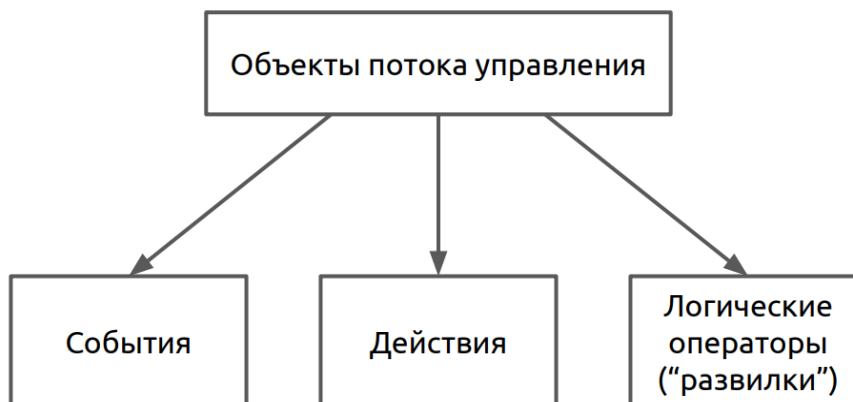
        sender.setBalance(sender.getBalance() - amount);
        receiver.setBalance(receiver.getBalance() + amount);
    }
}

```

Билет 7

1. Объекты потока-управления BPMN 2.0

Объекты потока управления - управляют логикой реализации бизнес-процесса.





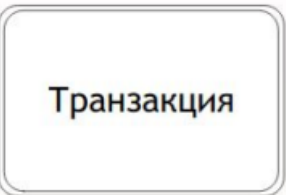


События - Изображаются окружностью.

- Означают какое-либо происшествие.
- Инициируют действия или являются их результатами.
- Могут быть классифицированы на:
 - начальные (start);
 - промежуточные (intermediate);
 - завершающие (end).
- Различают также события обработки и генерации.

Действия - Изображаются прямоугольниками со скругленными углами.

- Делятся на две категории:
 - задания;
 - подпроцессы.
- Могут содержать маркеры.







Категории действий

 <p>Задача</p>	<p>Задача - единица работы. Если задача помечена символом,  то задача является подпроцессом и может быть детализирована.</p>
 <p>Транзакция</p>	<p>Транзакция - набор логически связанных действий. Для транзакции может быть определен протокол выполнения.</p>
 <p>Событийный подпроцесс</p>	<p>Событийный подпроцесс помещается внутри другого процесса. Он начинает выполняться, если иницируется его начальное событие. Событийный подпроцесс может прерывать родительский подпроцесс или выполняться параллельно с ним.</p>
 <p>Вызывающее действие</p>	<p>Вызывающее действие является точкой входа для глобально определенного подпроцесса, который повторно используется в данном процессе.</p>

Маркеры действий и типы задач








Маркеры действий

Маркер отражает поведение действия во время выполнения:

-  Маркер подпроцесса
-  Маркер цикла
-  Маркер параллельных множественных экземпляров (МЭ)
-  Маркер последовательных множественных экземпляров (МЭ)
-  Маркер ad hoc
-  Маркер компенсации

Типы задач

Тип определяет природу действия, которое будет выполнено:

-  Задача отправки сообщения
-  Задача получения сообщения
-  Пользовательская задача
-  Неавтоматизированная задача
-  Задача-бизнес-правило
-  Задача-сервис
-  Задача-сценарий

Логические операторы (“развилки”) - Изображаются ромбами.

- Представляют собой точки принятия решений в процессе.
- Позволяют организовать ветвление и синхронизацию потоков управления.

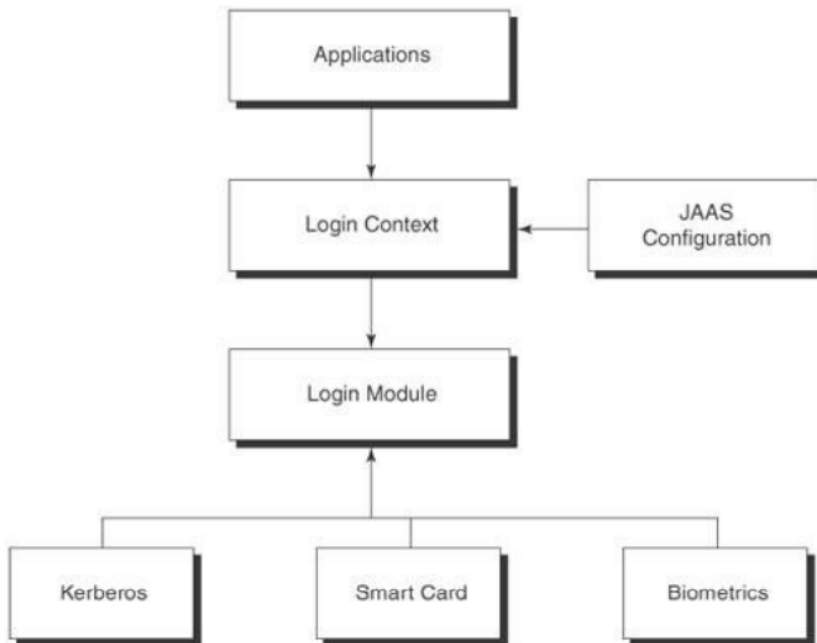
ИЛИ, Исключающее ИЛИ, Оператор И

2. JAAS (спизжено с конспекта)

- Стандарт API и реализация сервиса информационной безопасности.
- Входит в состав Java SE, появилась в Java 1.3.
- С точки зрения системного администратора -- набор конфигурационных файлов.
- С точки зрения программиста -- API для проверки ролей и полномочий.

JAAS — это про определение того, кто запрашивает доступ к ресурсу, и про вынесение решения, а может ли он этот доступ получить.

Архитектура



Конфигурационные файлы JAAS

- *.login.conf -- определяет, какие логин-модули необходимо задействовать в программе.
- *.policy -- определяет, какими привилегиями наделены пользователи или программы.

Объекты JAAS

- Subject -- субъект аутентификации; носитель или обладатель прав.
- Субъект -- источник (source) запроса (request) на выполнение некоторого действия.
- Principal -- представление субъекта с некоторой точки зрения.
- Credential -- то, чем субъект подтверждает, что это он.

Login Module

- API, реализация которой осуществляет аутентификацию и авторизацию пользователей в приложении.
- Содержит 5 абстрактных методов:
 - initialize.
 - login.
 - commit.
 - abort(первая фаза аутентификации завершилась неудачей).
 - logout(выход из системы).

Access Control Policy File

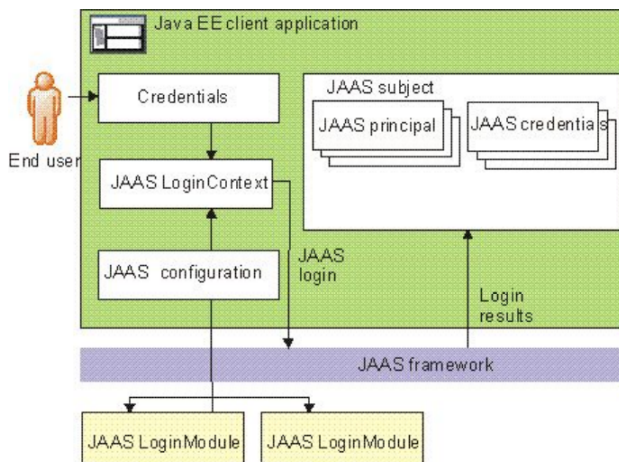
```

/** Java 2 Access Control Policy for the JaasAzn Application */
/** Code-Based Access Control Policy for JaasAzn */
grant codebase "file:./JaasAzn.jar" {
    permission javax.security.auth.AuthPermission
        "createLoginContext.JaasSample";
    permission javax.security.auth.AuthPermission "doAsPrivileged";
};

/** User-Based Access Control Policy for the SampleAction class
 ** instantiated by JaasAzn
 **/
grant codebase "file:./SampleAction.jar",
    Principal javax.security.auth.kerberos.KerberosPrincipal
    "your_user_name@your_realm" {
    permission java.util.PropertyPermission "java.home", "read";
    permission java.util.PropertyPermission "user.home", "read";
    permission java.io.FilePermission "foo.txt", "read";
};

```

Использование JAAS в Java EE



3. Написать класс (валидатор) для автобуса, который позволяет покупать пассажиру билет. Деньги не списываются, если у человека недостаточно средств. Если человек уже оплатил и второй раз прикладывает карту, то средства не списываются. А также обнуляется билет по окончании дня у данного пассажира.

```

@RestController
public class BusController {
    public static final long PRICE = 666;

    private final PassengerRepository passengerRepository;

    @Autowired

```

```
public BusController(PassengerRepository passengerRepository) {
    this.passengerRepository = passengerRepository;
}

@PostMapping
public void pay(long passengerId) {
    User passenger = passengerRepository.getById(passengerId);

    if (passenger.getBalance() < PRICE ) {
        throw new NotEnoughMoneyException();
    }

    if (passenger.getLastPaidDay() == LocalDateTime.now().toLocalDate()) {
        return; // уже оплачено
    }

    passenger.setBalance(passenger.getBalance() - PRICE);
}
}
```