Билет 2

1) Структура JSF

{JSP-страницы с компонентами GUI,Библиотека тегов,Управляемые бины, Доп. Объекты(компоненты, конвертеры, валидаторы), Доп. Теги, Конфигурация – faces-config.xml, Дескриптор развертывания – web.xml }

2) Наследование и полиморфизм в ORM

При объектно-реляционном отображении наследование и полиморфизм тесно связаны.
Три способа реализации:

1. Одна таблица для всех классов.
   Плюсы: простота и производительность
   Минусы: отсутствие null ограничений, не нормализованная таблица
2. Своя таблица на каждый класс
   Плюсы: возможность null ограничений
   Минусы: плохая поддержка полиморфных записей, не нормализованная таблица
3. Своя таблица на каждый подкласс
   Плюсы: нормализованная таблица, возможность null ограничений
   Минусы: низкая производительность

3) EJB калькулятор для 4-х операций

```
1.  @Remote
2.  public interface CalculatorInterface{
3.
4.      public double add(double a, double b);
5.      public double min(double a, double b);
6.      public double mult(double a, double b);
7.      public double div(double a, double b);
8.
9.  }
10.
11.
12. @Stateless(name="calculator")
13. public class Calculator implements CalculatorInterface {
14.     public double add(double a, double b){
15.         return a+b;
16.     }
17.     public double min(double a, double b){
18.         return a-b;
19.     }
20.     public double mult(double a, double b){
21.         return a*b;
22.     }
23.     public double div(double a, double b){
24.         return a/b;
25.     }
26. }
```

Билет 3

1) Контекст управляемых бинов. Конфигурация контекста.

Задается через faces-config.xml или с помощью аннотаций. 6 вариантов конфигурации:
@NoneScoped – контекст не определен, жизненным циклом управляют другие бины.
@RequestScoped (применяется по умолчанию) – контекст-запрос. @ViewScoped – контекст-страница

@SessionScoped – контекст-сессия @ApplicationScoped – контекст-приложение @CustomScoped – бин сохраняется в Map; программист сам управляет его жизненным циклом

2) Связи между сущностями в JPA.

https://devcolibri.com/%D0%BA%D0%B0%D0%BA-%D1%81%D0%B2%D1%8F%D0%B7%D0%B0%D1%82%D1%8C-entity-%D0%B2-jpa/

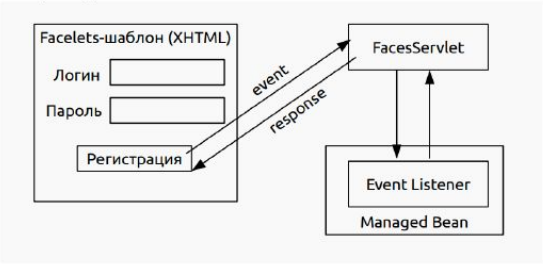
3) Реализовать компонент на React, строку поиска с автодополнением. Массив слов для автодополнения получать через GET запрос с использованием REST API.

```
import React from 'react'
import axios from "axios";


export class Autocomplete extends React.Component{
   constructor(props){
      super(props)
      this.state = {
         list: this.getList()
      }
   }

   getList(){
      axios.get('/someURL')
         .then(result => {
            return result
         })
   }

   render() {
      return(
         <div class={"AutoComplete"}>
            <input list={"options"}/>
            <datalist id={"options"}>
               {this.state.list.map((key, item)=>
                  <option key={key} value={item}/>
               )}
            </datalist>
         </div>
         )
   }
}
```

Билет 4

1) FacesServlet. Конфигурация

Обрабатывает запросы с браузера. Формирует объекты-события и вызывает методы- слушатели.
Конфигурация задаётся в web.xml.




```xml
<web-app xmlns="http://java.sun.com/xml/ns/javaee" version="2.5">
    <servlet>
        <servlet-name>comingsoon</servlet-name>
        <servlet-class>mysite.server.ComingSoonServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>comingsoon</servlet-name>
        <url-pattern>/*</url-pattern>
    </servlet-mapping>
</web-app>
```

2) Hibernate. Задание сущностей, типы соединений, типы языков

ORM-решением для языка Java, является технология Hibernate, которая не только заботится о связи Java классов с таблицами базы данных (и типов данных Java в типы данных SQL), но также предоставляет средства для автоматического построения запросов и извлечения данных и может значительно уменьшить время разработки, которое обычно тратится на ручное написание SQL и JDBC кода. Hibernate генерирует SQL вызовы и освобождает разработчика от ручной обработки результирующего набора данных и конвертации объектов, сохраняя приложение портируемым во все SQL базы данных.
Каждый сохраняемый класс помечается аннотацией @Entity, говорящей Hibernate, что этот класс является сущностью. Помимо того, в каждом классе, помеченном @Entity должно быть поле, имеющее аннотацию @Id, говорящее Hibernate, что это поле может быть использовано как первичный ключ в базе данных и что по значению этого поля Hibernate может отличать один объект от другого.

3) Redux Storage, описывающее состояние CRUD интерфейса к списку студентов

```
import {
CREATE,
READ,
UPDATE,
DELETE } from ‘constants’

const initialState = {
//массив студентов
        students = []
}

export function studentsReducer(state = initialState, action){
        switch(action.type){
```

```
            case CREATE:
                return {...state, state.students: [...state.students, action.payload]}
            case UPDATE:
                return {...state, students[action.payload.index]: action.payload}
            case DELETE:
                return {...state, state.students: [
                    ...state.students.slice(0, action.index),
                    ...state.students.slice(action.index + 1)
                ] }
            case READ:
                return {...state, readen: “ну прочитай, браток”}
        }
}
```

Билет 10

1) Валидаторы (все что в слайдах)

Главной целью конвертации и валидации является подготовка данных для обновления объектов модели. Таким образом, к моменту вызова методов, реализующих логику приложения, можно сделать определенные выводы о состоянии модели.
Стандартная валидация
JSF включает в себя три стандартных компонента для валидации:

- DoubleRangeValidator: Проверяет, что значение компонента укладывается в интервал, определяемый нижней границей, верхней границей или и тем, и другим. Значение должно быть числом.
- LongRangeValidator: Проверяет, что значение укладывается в интервал, определяемый нижней границей, верхней границей или и тем, и другим. Значение должно быть числом, преобразуемым к типу long.
- LengthValidator: Проверяет, что длина значения укладывается в интервал, определяемый нижней границей, верхней границей или и тем, и другим. Значение должно быть типа String.

Для создания валидатора необходимо сделать следующее:

- класс, реализующий интерфейс Validator (javax.faces.validator.Validator).
- Реализовать метод validate().
- Зарегистрировать валидатор в файле faces-config.xml или аннотацией @FacesValidator
- Использовать тег <f:validator/> на страницах JSP.

Примеры

```
required="true">
```

```
<f:validateLongRange minimum="1"/>
```

```
<f:validateLength minimum="5" maximum="10" />
```

2) Запросы к бд в JPA. jpql и criteria api

JPQL (Java Persistence query language) это язык запросов, практически такой же как SQL, однако вместо имен и колонок таблиц базы данных, он использует имена классов Entity и их атрибуты. В качестве параметров запросов также используются типы данных атрибутов Entity, а не полей баз данных.
Характерные черты: ● расширение EJB QL; ● SQL-подобный синтаксис; ● в запросах указываются объекты/свойства вместо таблиц/колонок; ● поддерживаются подзапросы.

Criteria API

Характерные черты: ● Объектно-ориентированный API для построения запросов. ● Есть возможность отобразить любой JPQLзапрос в Criteria. ● Поддерживает построение запросов в runtime.
● Необходимо указать сущности, участвующие в запросе (query roots).
● Условие запроса задается через where(Predicate p), где аргумент устанавливает необходимые ограничения.
● Метод select() определяет, что мы получим в результате запроса.

```
Root customer = qdef.from(Customer.class);
qdef.select(customer).where(queryBuilder .equal(customer.get("customerInfo"), ci));
```

3) Написать компонент ejb для списания средств со счета клиента и начисления их на счет банка за одну транзакцию

(это не ответ, тут просто код из презы)

```
@Stateful
@TransactionManagement(BEAN)
public class MySessionBean implements MySession {
 @Resource UserTransaction ut;
 public void method() {
 try {
 ut.begin(); // Открываем транзакцию
 //... какие-то действия
 ut.commit(); // Закрываем транзакцию
 } catch (Exception e) {
 ut.rollback(); // Ошибка — откат транзакции
 }
```

Билет 13

1) Фаза обновления значений компонентов, фаза вызова приложения

Если данные валидны, то значение компонента обновляется. Новое значение присваивается полю объекта компонента.
Управление передается слушателям событий. Формируются новые значения компонентов.

2) ЖЦ spring- приложения

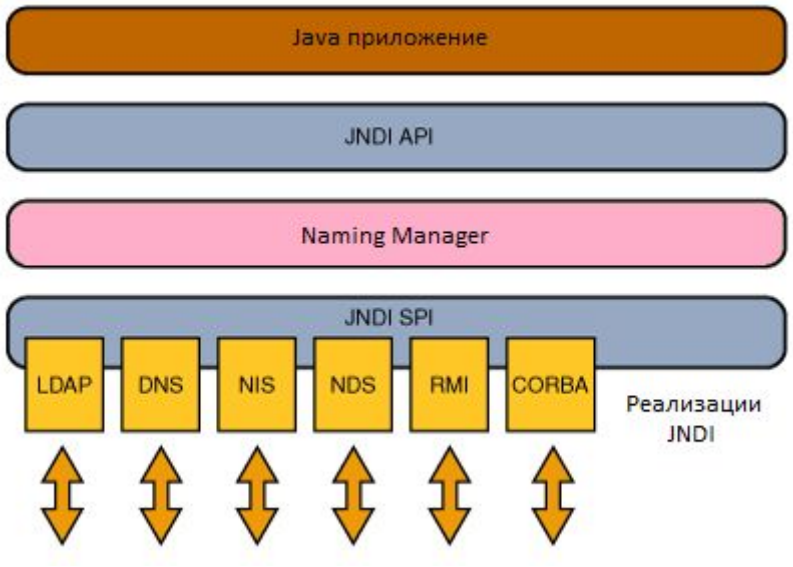
1. Парсинг конфигураций, создание объектов BeanDefinition
2. Настройка объектов BeanDefinition
3. Создание объектов собственных FactoryBean<T>
4. Создание экземпляров бинов
5. Настройка экземпляров бинов

3) Бэкэнд для магазина на GWT RCP(!). Типа поля с названием, ценой, кол-вом и т.д.

Билет 18

1) JNDI

Java Naming & Directory Interface - это набор Java API, организованный в виде службы каталогов, который позволяет Java-клиентам открывать и просматривать данные и объекты по их именам. Как любое другое Java API, как набор интерфейсов, JNDI не зависит от нижележащей реализации. В дополнению к этому, он предоставляет реализацию service provider interface (SPI), которая позволяет службам каталогов работать в паре с каким-либо фреймворком. Это может быть сервер, файл или база данных[1].



.lookup("Database"); имя объекта, к которому хочешь получить доступ

```
1  //Способ №1 (inject resourse)
2  @Resourse(name="jdbc/OrbisPool")
3  private DataSourse dataSourse;
4
5  //Способ №2
6  private DataSource dataSourse
7  Context ctx=new InitialContext();
8  dataSourse=(DataSourse)ctx.lookup("jdbc/OrbisPool");
```

2) React js

React - библиотека js для создания пользовательских интерфейсов (в основном SPA) из отдельных частей кода - компонентов. Использует специальный язык разметки JSX, позволяющий вставлять куски кода прямо в разметку html и связывать вёрстку и данные React. Использует Virtual DOM. Передача данных происходит от родителей к детям и от детей к родителям. Данные также можно хранить с помощью Redux, который представляет собой единое хранилище

3) Написать бин который выводит количество минут, прошедших со старта сервера

```
@ManagedBean
@ApplicationScoped
public class CounterBean implements Serializable {
long startTime;
public CounterBean() { startTime = System.currentTimeMills()/1000/60; }
public long getMillisecondsAfterRestart() { return System.currentTimeMills()/1000/60 -
startTime; }
public void setMillisecondsAfterRestart() {}}
```

Билет 20

1) Профили платформы JavaEE
2) Типы DI в Spring
3) Написать React компонент, содержащий 2 страницы Главная(«/home») и Новости(«/news»), взаимодействие между которыми осуществляется по гипер-ссылкам

Билет 23

1) Обращение к session bean из managed и unmanaged кода

Managed - жизненным циклом управляет RunTime. Можно обратиться через аннотации. Пример:
@EJB
Unmanaged - не управляет RunTime. Можем обратиться через JNDI, то есть к реестру по определённому имени в реестре:
Context context = InitialContext();
Database database = context.lookup('PostgreSQL')

2) Структура в react. Jsx.(может как-то иначе сформулировано)

React представляет собой дерево из компонентов. Точкой входа (корнем) являет index.js который определяет дальше компоненты. Каждый компонент включает в себя другие компоненты или являются листьями дерева.

3) JSF Manager Bean, после инициализации HTTP-сессии формирующий коллекцию с содержимым таблицы H_УЧЕБНЫЕ_ПЛАНЫ. Для доступа к БД необходимо использовать JDBC-ресурс jdbc/OrbisPool.

Билет 32

1) Проблемы ORM

1. Ошибки в реализации трудно найти, отладить и исправить
2. Ограничения в реализации
3. Могут потребоваться дополнительные таблицы для отображения классов в таблицы
4. Медлительность

2) Vaadin и gwt, сходства и различия

3) Написать конфиг JSF страницы которая принимает xhtml запросы и все, чей url начинается на /faces/

```
<servlet>
	<servlet-name>Faces Servlet</servlet-name>
	<servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
	<load-on-startup>1</load-on-startup>
  </servlet>
	<servlet-mapping>
		<servlet-name>Faces Servlet</servlet-name>
		<url-pattern>*.xhtml</url-pattern>
	 </servlet-mapping>
	<servlet-mapping>
		<servlet-name>Faces Servlet</servlet-name>
		<url-pattern>/faces/*</url-pattern>
	</servlet-mapping>
```

Билет ?

1) JTA

Интерфейс для управления транзакциями в EJB-компонентах.
Позволяет открывать, закрывать и откатывать транзакции.

```
public interface TransactionInterface {

	public openTransaction(UserTransaction trns){}
	public closeTransaction(UserTransaction trns){}
	public rollbackTransaction(UserTransaction trns){}
}

@TransactionManagment(BEAN)
public class Transactions implements TransactionInterface{
	@Resource UserTransaction trns

	public openTransaction(UserTransaction trns){
		trns.open();
}
	public closeTransaction(UserTransaction trns){
		trns.close();
}
	public rollbackTransaction(UserTransaction trns){
		trns.rollback()
}
}
```

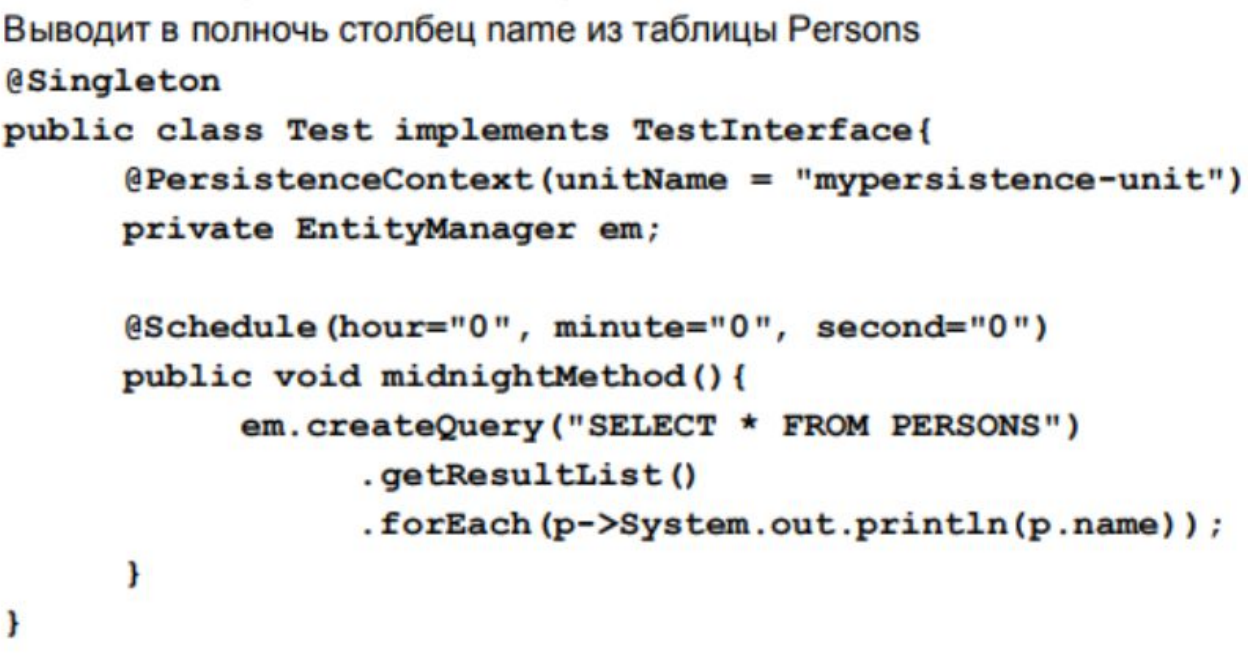
2) React Router

Router определяет набор маршрутов и, когда к приложению, приходит запрос, то Router выполняет сопоставление запроса с маршрутами. И если какой-то маршрут совпадает с URL запроса, то этот маршрут выбирается для обработки запроса.
Для выбора маршрута определен объект Switch. Он позволяет выбрать первый попавшийся маршрут и его использовать для обработки. Без этого объекта Router может использовать для обработки одного запроса теоретически несколько маршрутов, если они соответствуют строке запроса.
Каждый маршрут представляет объект Route. Он имеет ряд атрибутов. В частности, для маршрута устанавливаются два атрибута:
path: шаблон адреса, с которым будет сопоставляться запрошенный адрес URL
component - тот компонент, который отвечает за обработку запроса по этому маршруту

Switch - для поиска первого совпавшего пути (прим. сначала идет “/’, потом “/main”, мы ищем “/main”, откроет “/”
BrowserHistory - сохраняется в историю, имеет человеческие назавния
HashHistory - не сохраняется, имеет хэши в URL

```
<Router>
        <Switch>
          <Route exact path="/" component={Main} />
          <Route path="/about" component={About} />
          <Route component={NotFound} />
        </Switch>
      </Router>
```

3) Написать EJB , который " просыпается " в полночь и выводит содержимое таблицы н_люди



```
Выводит в полночь столбец name из таблицы Persons
@Singleton
public class Test implements TestInterface{
      @PersistenceContext(unitName = "mypersistence-unit")
      private EntityManager em;

      @Schedule(hour="0", minute="0", second="0")
      public void midnightMethod(){
            em.createQuery("SELECT * FROM PERSONS")
                  .getResultList()
                  .forEach(p->System.out.println(p.name));
      }
}
```

Билет ?

1) ORM

3) Servlet + html. Посчитать и вывести количество сессий в текущий момент

```
<%! private int count = 0;
        private Set<String> sessions; %>
<%
sessions.add(request.getSession().getId());
%>
<%= sessions.size()%>
```

Билет ?

1) RMI. RMI в javaEE

Java API, позволяющий вызывать методы удаленных объектов

2)инициализация Spring Beans

In Spring, the objects that form the backbone of your application and that are managed by the Spring IoC container are called beans. A bean is an object that is instantiated, assembled, and otherwise managed by a Spring IoC container.

```
public class Address {
    private String street;
    private int number;

    public Address(String street, int number) {
        this.street = street;
        this.number = number;
    }

    // getters and setters
}
@Configuration
@ComponentScan(basePackageClasses = Company.class)
public class Config {
    @Bean
    public Address getAddress() {
        return new Address("High Street", 1000);
    }
}
```

3)Компонент для React, реализующий интерфейс ввода данных для банковской карты: номер (16 цифр); имя (латинские символы); срок действия(mm/yy); защитный код (3 цифры)

```
import {React} from 'react'
```

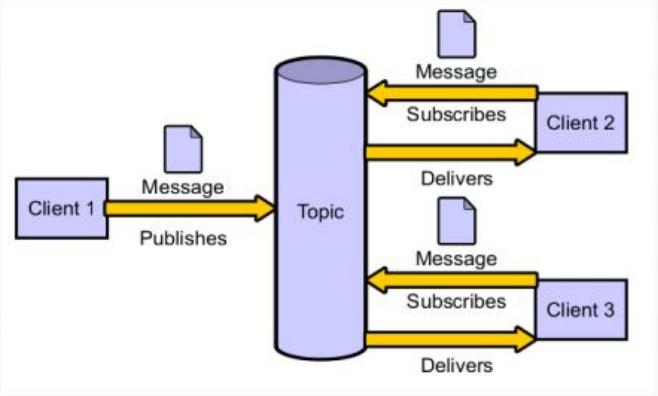
```
export class CreditCard extends React.Component{


  constructor(props){
    super(props)
     this.state = {
       number = '',
       data = '',
       owner = '',
       CVV = ''
    }
//Так же все методы надо привязать,чтобы использовать this в них, приведу один
    this.onChange = this.onChange.bind(this)
  }

//Все аналогичны, напишу один
  onChange(e) {
//Вот тут делаете валидацию или добавляете / если это Data
    this.state.number = e.value
  }

  onClick(e) {
    sendData(this.state.number, this.state.date, this.state.CVV, this.state.owner)
  }


  render() {
    return(
    <input onChange={this.onChangeNumber(event)} value={this.state.number}/>
    <input onChange={this.onChangeData(event)} value={this.state.data}/>
    <input onChange={this.onChangeOwner(event)} value={this.state.owner}/>
    <input onChange={this.onChangeCVV(event)} value={this.state.CVV}/>
    <button onClick={this.onClick() }/>
    )
  }
}
```

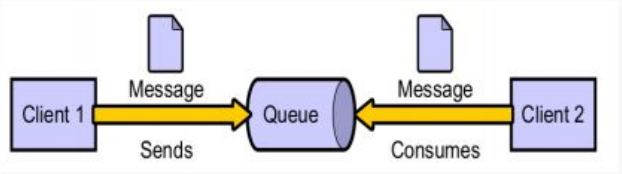
Билет ?

1) JMS и реализация в Java

Позволяет организовать асинхронный обмен между компонентами системы

Две модели доставки сообщений подписка”topic” and “queue”





2) State в React redux и flux

3) JPQL запрос из таблицы, который выводит всех сотрудников старше 18 лет по полу age (Date)

JPQL (Java Persistence query language) это язык запросов, практически такой же как SQL, однако вместо имен и колонок таблиц базы данных, он использует имена классов Entity и их атрибуты. В качестве параметров запросов так же используются типы данных атрибутов Entity, а не полей баз данных. SELECT p FROM Person p WHERE ((YEAR(CURRENT_DATE)- YEAR(p.birthDate))>18)

Билет ? (возможно это к первой рубежке)

1) фазы обработки запроса в jsf

2)IoC и CDI в Spring

3)написать сервис jms для отправки спама, тем кто подписан на ресурс jms/blablabla

```
@ApplicationScoped
public class Sender
{
   @Resource(mappedName = "jms/blablabla")
   private Topic topic;

   @Inject
   private JMSContext context;

   public void sendMessage(String txt) {
      context.createProducer().send(topic, txt);
   }
}
```

Билет ? (возможно это к первой рубежке)

1. Шаблоны проектирование - что такое и для чего. Отличие от архитектурных

2. Создание веб-сервисов на spring. Spring restful.

3. Html форма для отправки паспортных данных

Метод get
Выпадающий список
Радиобаттон
Чекбокс
Текстовое поле

Билет ?

1) JNDI. Способы обращения (не уверен в формулировке вопроса)

2) React.js. Ключевые особенности

3) Реализовать бин, который считает количество минут со старта приложения (или рестарта сервера); И приведён пример кода: ... <h:outputText value="#{counterBean.millisecondsAfterRestart}"/> ...

Билет ?

1) 3 фаза жизни jsf

Фаза валидации значений компонентов
Вызываются валидаторы, зарегистрированные для компонентов преставления.
Если значение компонента не проходит валидацию, формируется сообщение об ошибке и сохраняется в FacesContext.

2) IoC DI в Spring

3) сервер JMS, который осуществляет рассылку спама всем подписчикам топика "lol"

Билет ?

1) 1 фаза ЖЦ jsf

Фаза формирования представления
JSF Runtime формирует представление (начиная с UIViewRoot):

- Создаются объекты компонентов.
- Назначаются слушатели событий, конвертеры и валидаторы.
- Все элементы представления помещаются в FacesContext.

Если это первый запрос пользователя к странице JSF, то формируется пустое представление.
Если это запрос к уже существующей странице, то JSF Runtime синхронизирует состояние компонентов представления с клиентом.

2) Spring, отличия от java ee

3) Сконфигурировать entity через xml

```
<class name="Passport" table="PASSPORTS">
        <id name="id" type="int" column="id">
            <generator class="native"/>
        </id>
        <property name="series" type="string" column="SERIES"/>
        <property name="no" type="string" column="NO"/>
        <property name="issueDate" type="timestamp" column="ISSUE_DATE"/>
        <one-to-one name="owner" class="ru.easyjava.data.hibernate.entity.Person"
cascade="save-update"/>
    </class>
```

Билет ?

1) Структура JSF-приложения. Компоненты JSF. Иерархия компонент

Структура JSF-приложения:

- JSP/XHTML-страницы, содержащие компоненты GUI
- библиотеки тегов
- управляемые бины
- доп объекты (компоненты, конвертеры, валидаторы)
- доп теги
- конфигурация - faces-config.xml (опционально)
- дескриптор развертывания - web.xml

Интерфейс строится из компонентов.
Компоненты расположены на Facelets-шаблонах или страницах JSP.
Компоненты реализуют интерфейс javax.faces.component.UIComponent.
Можно создавать собственные компоненты.
Компоненты на странице объединены в древовидную структуру — представление.
Корневым элементов представления является экземпляр класса javax.faces.component.UIViewRoot.

2) JPA - особенности, недостатки и преимущества, отличия от JDBC и от его использования с ORM.

3) Интерфейс на GWT, проверяющий, аутентицифирован ли пользователь. И, если нет, предлагающий ему аутентифицироваться путем ввода логина и пароля.

Билет ?

1) Конвертер jsf

Используются для преобразования данных компонента в заданный формат. Реализуют интерфейс javax.faces.convert.Converter. Существуют стандартные конвертеры для основных типов данных.

f:convertNumber: используется для преобразования строки в число.
f:convertDateTime: используется для преобразования строки в формат даты.

В стандартную поставку JSF входит множество стандартных конвертеров данных, благодаря чему большая часть конвертации происходит автоматически.
Стандартные конвертеры

- javax.faces.BigDecimal
- javax.faces.BigInteger
- javax.faces.Boolean
- javax.faces.Byte
- javax.faces.Character
- javax.faces.DateTime
- javax.faces.Double
- javax.faces.Float

Для создания специализированного конвертера необходимо следующее:
●Создать класс, реализующий интерфейс Converter
●Реализовать метод getAsObject(), который будет вызываться для преобразования строкового значения поля в объект
●Реализовать метод getAsString, который будет вызываться для получения строкового представления объекта
●Зарегистрировать конвертеры в файле faces-config.xml, используя элемент <converter>.

2) Интерфейс entitymanager и его методы.

Интерфейс EntityManager управляет CRUD-операциями. Этот класс существенно облегчает операции создания, чтения, обновления и удаления экземпляров классов, снабженных тегом [Table]. Позволяет работать с сущностями, сохраненными в Persistance Context.
EntityManager em = Persistance.createEntityManagerFactory("...").createEntityManager();
Методы:

- find
- getReference
- merge
- refresh
- remove
- flush
- close

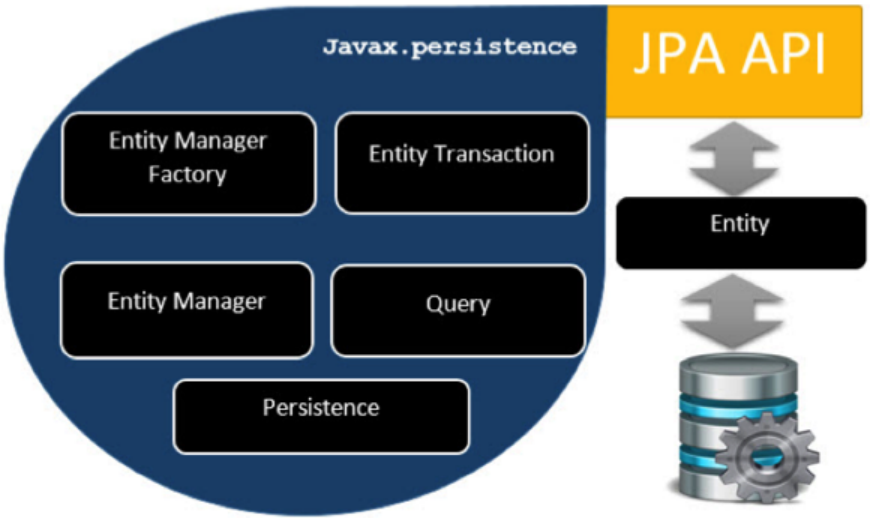
Все классы, которые могут быть сохранены в базе данных называются entity (сущность) и на них налагаются определённые требования:
- Наличие публично доступного конструктора без аргументов
- Класс, его методы и сохраняемые поля не должны быть final
- Если объект Entity класса будет передаваться по значению как отдельный объект (detached object), например через удаленный интерфейс (through a remote interface), он так же должен реализовывать Serializable интерфейс.
Сохраняемые поля должны быть доступны только с использованием методов класса.

3) Виджет на gwt, для оповещений в vk, Twitter (вроде как-то так звучал)

Билет ?

1) ajax в jsp

2) Архитектура JPA



EntityManagerFactory - This is a factory class of EntityManager. It creates and manages multiple EntityManager instances.
EntityManager - It is an Interface, it manages the persistence operations on objects. It works like factory for Query instance.
Entity - Entities are the persistence objects, stores as records in the database.
EntityTransaction - It has one-to-one relationship with EntityManager. For each EntityManager, operations are maintained by EntityTransaction class.
Persistence - This class contain static methods to obtain EntityManagerFactory instance.
Query - This interface is implemented by each JPA vendor to obtain relational objects that meet the criteria.

3) Интерфейс авторизации(логин+пароль) на React. На стороне сервера - REST API

Билет ?

1) Управляемые бины, конфигурация и что-то еще

Содержат параметры и методы для обработки данных с компонентов, событий и валидации данных. Жизненным циклом управляет JSF Runtime Environment. Доступ из JSF-страниц осуществляется с помощью элементов EL.
Для объявлении класса управляемым бином используется аннотация @ManagedBean(name = «beanName», eager = «true\false»).
name: указывает уникальное имя класса (бина) в JSF. Если имя не указано, то имя совпадает с именем класса, где первая буква в нижнем регистре.
eager: указывает на время создания бина. Если стоит true, то бин будет создан на старте приложения, если false, то бин будет создан при первом запросе к нему.

- faces-config.xml:

```
<managed-bean>
        <managed-bean-name>customer</managed-bean-name>
        <managed-bean-class>CustomerBean</managed-bean-class>
        <managed-bean-scope>request</managed-bean-scope>
```

```
        <managed-property>
                <property-name>areaCode</property-name>
                <value>#{initParam.defaultAreaCode}</value>
        </managed-property>
</managed-bean>
```

- аннотации:

```
@ManagedBean(name="customer")
@RequestScoped
public class CustomerBean {
        ...
        @ManagedProperty(value="#{initParam.defaultAreaCode}" name="areaCode")
        private String areaCode;
        ...
}
```

2) JPA

3) Написать приложение на Vaadin, которое будет автомаически заполнять приказ об отчислении.

Билет ?

1) JMS. Модели доставки сообщений

2) GWT. Основные преимущества и недостатки

3) Реализуйте CRUD-интерфейс к таблице Н_ВИДЫ_УЧЕБНОЙ_ДЕЯТЕЛЬНОСТИ с помощью Spring data

Билет ?

1) Модель JMS

2) GWT - особенности,плюсы и недостатки

3) Интерфейс реализация CRUD для Н_ВИДЫ_ДИСЦИПЛИН SpringData

Билет ?

1) Location Transparency, реализация в Java EE

2) Rest в Spring. Spring RESTful

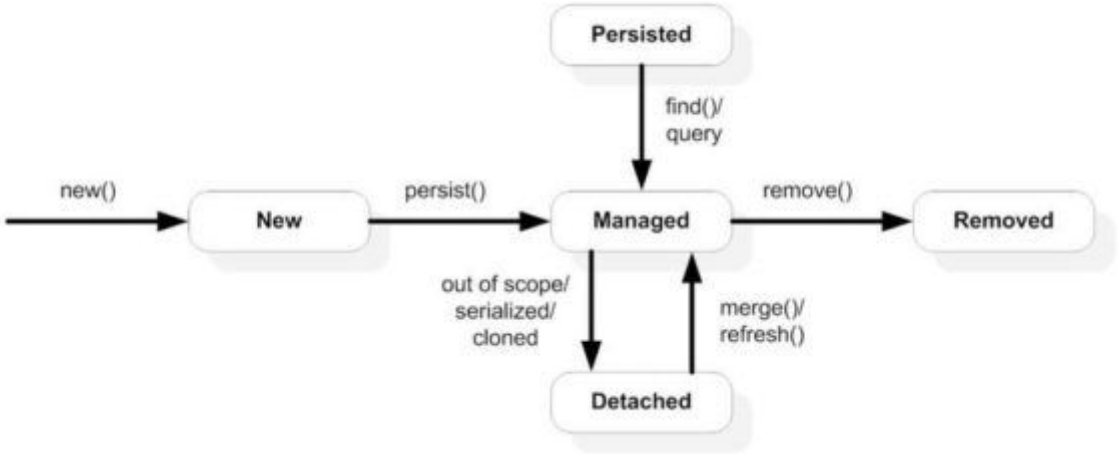
3) Создать бин, конфигурируемый аннотациями, с именем myBean, контекст которого равен контексту другого бина - myOtherBean

```java
@ManagedBean(name="SampleBean")
public class SampleBean implements Serializable {

    @ManagedProperty(value="#{AnotherSampleBean}")
    private AnotherSampleBean anotherSampleBean;

}
```

Билет ?

1) JSF-конвертеры

2) Интерфейс EntityManager и его методы



EntityManager - основной интерфейс ORM, служит для управления персистентными сущностями. Экземпляр EntityManager содержит в себе "персистентный контекст" – набор экземпляров сущностей, загруженных из БД или только что созданных.

persist() - вводит новый экземпляр сущности в персистентный контекст. При коммите транзакции командой SQL INSERT в БД будет создана соответствующая запись.

merge() - переносит состояние отсоединенного экземпляра сущности в персистентный контекст следующим образом: из БД загружается экземпляр с тем же идентификатором, в него переносится состояние переданного Detached экземпляра и возвращается загруженный Managed экземпляр. Далее надо работать именно с возвращенным Managed экземпляром. При коммите транзакции командой SQL UPDATE в БД будет сохранено состояние данного экземпляра.

remove() - удалить объект из базы данных, либо, если включен режим мягкого удаления, установить атрибуты deleteTs и deletedBy.

Если переданный экземпляр находится в Detached состоянии, сначала выполняется merge().

find() - загружает экземпляр сущности по идентификатору.

3) Написать виджет на GWT для репостов с соцсетей (вызовы api соцсетей можно написать текстом)

Билет ?

1) RMI в java EE

2) Инициализация Spring beans

3) React компонент для хранения информации о банковской карте

Билет ?

1) Жизненный цикл Session beans

2) Jsx, особенности, применение в реакт

3) Написать правило навигации, для перехода с одной страницы на другую по нажатию кнопки

<h:commandButton action="otherPage" value="Submit" />

```
<navigation-rule>
  <from-view-id>/start.xhtml</from-view-id>
  <navigation-case>
     <from-outcome>otherPage</from-outcome>
     <to-view-id>/page.xhtml</to-view-id>
     <redirect/>
  </navigation-case>
</navigation-rule>
```

Билет ?

1) Бизнес-логика, ejb, структура и классы. Локальные и удаленные что-то.

2) SPA

3) Что-то с БД, JPA Criteria api. Найти всех студентов со средним баллом ниже 4.0 и удалить

Билет ?

1) stateless stateful singleton session bean. Сходство отличия итп

Session Beans
синхронная обработка вызовов
Вызываются через API. Могут вызываться локально и удаленно.
Могут быть endpoint-ами для веб-сервисов.
CDI - @EJB
Не обладают свойством персистентности
Можно формировать пулы бинов (кроме @Singleton)

Разработка:

1. создание Business interface
   содержит заголовки всех методов, реализующий бизнес-логику компонента.
   @Local / @Remote
   ```
   import javax.ejb.*
   @Remote
   public interface Hello{
           public String sayHello();
   }
   ```
2. создание класса компонента, реализующего этот интерфейс
   компонент должен реализовывать все методы этого интерфейса. Может содержать методы, реагирующие на события жизненного цикла компонента.
   ! public class верхнего уровня
   ! не final/abstract
   ! public конструктор()
   ! без переопределения метода finalize()
   ```
   import javax.ejb.*
   @Stateless
   public class HelloBean implements Hello{
           public String sayHello(){
                   return “Hello!”;
           }
   }
   ```
3. конфигурация компонента с помощью аннотаций или дескриптора развертывания.

Обращение через Dependency Injection

Stateless Session Bean (@Stateless)

не сохраняют состояние между обращениями клиента, не привязаны к клиенту, хорошо масштабируются

Stateful Session Bean (@Stateful)

привязаны к клиенту, можно сохранять контекст в полях класса, хуже масштабируются.

Singleton (@Singleton)

контейнер генерирует один экземпляр бина

2) компоненты react. Что такое итп. Умные и глупые компоненты

Мои глупые компоненты:

не зависят от остальной части приложения, например Flux actions или stores
часто содержатся в this.props.children
получают данные и колбэки исключительно через props
имеют свой css файл
изредка имеют свой state
могут использовать другие глупые компоненты
примеры: Page, Sidebar, Story, UserInfo, List

Мои умные компоненты:

оборачивает один или несколько глупых или умных компонентов
хранит состояние стора и пробрасывает его как объекты в глупые компоненты
вызывает Flux actions и обеспечивает ими глупые компоненты в виде колбэков

никогда не имеют собственных стилей
редко сами выдают DOM, используйте глупые компоненты для макета
примеры: UserPage, FollowersSidebar, StoryContainer, FollowedUserList

А теперь мои умные и глупые:

Умные - могут изменять данные
Глупые - не могут

3) jsf поле многострочного ввода в которое можно ввести только строчные английские буквы

```
<h:inputTextarea row="10">
    <f:validateRegex pattern="[^A-ZA-Я]" />
</h:inputText>
```

Ангуляр (вопросов по нему нет, но Цопа обещал)

Сервисы:

Сервисы в Angular представляют довольно широкий спектр классов, которые выполняют некоторые специфические задачи, например, логгирование, работу с данными и т.д.

В отличие от компонентов и директив сервисы не работают с представлениями, то есть с разметкой html, не оказывают на нее прямого влияния. Они выполняют строго определенную и достаточно узкую задачу.

Стандартные задачи сервисов:

- Предоставление данных приложению. Сервис может сам хранить данные в памяти, либо для получения данных может обращаться к какому-нибудь источнику данных, например, к серверу.
- Сервис может представлять канал взаимодействия между отдельными компонентами приложения
- Сервис может инкапсулировать бизнес-логику, различные вычислительные задачи, задачи по логгированию, которые лучше выносить из компонентов. Тем самым код компонентов будет сосредоточен непосредственно на работе с представлением. Кроме того, тем самым мы также можем решить проблему повторения кода, если нам потребуется выполнить одну и ту же задачу в разных компонентах и классах

DI в Angular

Язык TypeScript

Строго типизированный, объектно-ориентированный, компилируемый язык программирования компилируется в js. (хз что про него еще добавить, если что знаете, напишите)

Крестики-Нолики на Angular

Получить список всех игр в крестики-нолики с сервера при загрузке компонента