

L^AT_EX

Написать исходный код CDI бина, реализующего паттерн «команда» (Command Pattern)

```
interface Command {void execute();}
@Named(value="cmd1") @ApplicationScoped
public class Cmd1 implements Command {
    void execute() { ... };
}
@Named(value="cmd2") @ApplicationScoped
public class Cmd2 implements Command {
    void execute() { ... };
}
@Named @ApplicationScoped
public class MyBean implements Command {
    private final Command cmd1, cmd2;
    @Inject
    public MyBean(@Named("cmd1") Command cmd1,
        ↪ @Named("cmd2") Command cmd2) {
        ↪ this.cmd1 = cmd1; this.cmd2 = cmd2;
    }
    public void cmd(int n) {if (n == 1)
        ↪ cmd1.execute(); if (n == 2)
        ↪ cmd2.execute();}
```

Написать веб-приложение на JSF (xhtml + CDI-бин) со списком студентов и бин, который будет реализовывать логику отчисления студентов. Напротив каждого имени студента должна быть кнопка "отчислить". Обновление должно производиться при помощи AJAX

```
@Named
@ApplicationScoped
public class StudentBean implements Serializable {
    private List<String> students;
    public List<String> getStudents(){ return students;}
    public void expelStudent(String studentName) {
        students.remove(studentName);}

<h:body><h:form>
<ui:repeat value="#{studentBean.students}"
    ↪ var="student"> #{student}
<h:commandButton value="Expel" action="#{stude
    ↪ ntBean.expelStudent(student)}">
<f:ajax execute="@this" render="@form" />
</h:commandButton>
<br/></ui:repeat></h:form></h:body>
```

Интерфейс JSF (xhtml страница + CDI), реализующий ввод паспортных данных (серия, номер, дата, место выдачи)

```
@ManagedBean @SessionScoped @Setter @Getter
public class PassportInputBean {
    private String series; private String number;
    private Date date; private String place;
    @ManagedProperty(value = "#{repoBean}")
    private PassportRepo repo;
    public void storePassportData() {
        repo.store(series, number, date, place);}
    <h:form>Enter series
    <h:inputText type="text"
        ↪ value="#{passportInputBean.series}"
        required="true"/>Enter number
    <h:inputText type="text"
        ↪ value="#{passportInputBean.number}">
    Enter date<h:inputText
        ↪ value="#{passportInputBean.date}">
    <f:convertDateTime pattern="yyyy-MM-dd"/>
    </h:inputText> Enter born place
    <h:inputText type="text"
        ↪ value="#{passportInputBean.place}">
    <h:commandButton value="Send"
        ↪ action="#{passportInputBean.storePas
        sportData()}"> </h:form>
```

Написать страницу JSF, которая бы выводила сначала 10 простых чисел, а затем с помощью Ajax запроса ещё 10.

```
@ApplicationScoped @Named public class
    ↪ PrimeNumber{@Getter
private final List<Long> primes = new ArrayList<>();
public PrimeNumber(){nextPrimes();}
public nextPrimes(){//add in primes next 10 prime
    ↪ numbers}}

<h:dataTable id="table"
    ↪ value="#{primeNumber.primers}" var="prime">
    <h:column> <h:outputText value="#{prime}"/>
    </h:column> </h:dataTable> <h:form>
    <h:commandButton value="Next 10"
        ↪ action="#{primeNumber.nextPrimes}">
    <f:ajax execute="@form" render="table"/>
    </h:commandButton> </h:form>
```

JSF страница, динамически подгружаемая и выводящая новостную ленту с новостями формата: автор, заголовок, дата, иллюстрация, аннотация и полный текст(показывается при нажатии на соответствующую строчку)

```
<h:body> <h:dataTable value="#{newsBean.newsList}"
    ↪ var="news" border="1">
    <h:column> <f:facet name="header">Author</f:facet>
    <h:outputText value="#{news.author}" /> </h:column>
    <h:column> <f:facet name="header">Title</f:facet>
    <h:outputText value="#{news.title}" /> </h:column>
    <h:column> <f:facet name="header">Date</f:facet>
    <h:outputText value="#{news.date}" /> </h:column>
    <h:column> <f:facet name="header">Image</f:facet>
    <h:graphicImage value="#{news.image}" width="100"
        ↪ height="100" alt="News Image" />
    </h:column> <h:column> <f:facet
        ↪ name="header">Summary</f:facet>
    <h:outputText value="#{news.summary}" /> </h:column>
    <h:column> <f:facet name="header">Full
        ↪ Text</f:facet>
    <h:commandLink value="Show Full Text"
        ↪ action="#{newsBean.showFullText(news)}">
    <f:setPropertyActionListener
        ↪ target="#{newsBean.selectedNews}"
        ↪ value="#{news}" />
    </h:commandLink> </h:column> </h:dataTable>
    <h:panelGroup rendered="#{not empty
        ↪ newsBean.selectedNews}">
    <h3>Full Text of News:</h3>
    <h:outputText
        ↪ value="#{newsBean.selectedNews.fullText}"
        ↪ escape="false" />
    </h:panelGroup> </h:body>
```

Интерфейс на React, формирующий две страницы с разными URL: Главную (/home) и Новости (/news). Переход между страницами должен осуществляться посредством гиперссылок.

```
export function App(props) {
    return (<BrowserRouter>
    <Routes><Route path="/home"
        ↪ element={<div><h1>Home</h1> <a
        ↪ href="/news" > news</a></div>} />
    <Route path="/news" element={<div><h1>news</h1> <a
        ↪ href="/home" > home</a></div>} />
    </Routes></BrowserRouter>);}
```

Привести фрагмент кода управляемого бина, увеличивающего на 1 значение, отображаемое на кнопке при каждом клике по ней

```
@ManagedBean @ApplicationScoped
public class MyBean implements Serializable {
    private int value = 0; public void increment() {
        value++;} public int getValue() {return value;}
    public void setValue(int value) {this.value =
        ↪ value;}
    <h:commandButton action= "#{myBean.increment}"
        ↪ value = "#{bean.value}"/>
```

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

```
@Named("myBean") @SessionScoped class MyBean {
    private List<String> plans;
    private List<String> getPlans() {return plans;}
    @Resource(name="jdbc/OrbisPool",type=DataSource.class)
    private DataSource dataSource; @PostConstruct
    private void loadPlans() {
        try (var conn = dataSource.getConnection()) {
            var stmt = conn.createStatement();
            var rs = stmt.executeQuery("SELECT name FROM
                ↪ plans");
            plans = new ArrayList<>(); while (rs.next()) {
                plans.add(rs.getString("name"));}}}
```

Написать React компонент формирующий таблицу пользователей, данные приходят в props

```
function UsersTable(props) { return (
    <table className="table"> <thead>
    <tr><td>Name</td><td>Surname</td></tr>
    </thead> <tbody> { props.data.map((user, i) => (
    <tr key={i}> <td>{user.name}</td>
    <td>{user.surname}</td> </tr>))} </tbody> </table>)}>
```

Написать CDI Bean калькулятор, поддерживающий 4 базовые операции для целых чисел

```
@Named(value="calc")
@ApplicationScoped
public class Calc implements Serializable{
    public int add(int a, int b) { return a + b; }
    public int sub(int a, int b) { return a - b; }
    public int mul(int a, int b) { return a * b; }
    public int div(int a, int b) { return a / b; }
```

Сделать бин который показывает время в минутах со старта сервера

```
@Named("serverTimer") @ApplicationScoped
public class ServerTimer { private long start;
    public ServerTimer() {start =
        ↪ System.currentTimeMillis();}
    public long getTime() {return
        ↪ (System.currentTimeMillis() - start) /
        ↪ 60000;}}
```

Написать managed bean и задать ему score такой же как у бина otherBean

```
@ManagedBean @ApplicationScoped
public class OtherBean {
    @ManagedProperty(value="#{myBean}")
    @Getter private MyBean myBean;}
@ManagedBean @NoneScoped @Named
public class MyBean{}
```

Vue.js простейший чат бот, который на любое сообщение отвечает сам дурак

```
<template> <input v-model="message_value"
  ↳ placeholder="message"/>
<button @click="send">send</button>
<li v-for="item in message_history" :key="item">
  {{item}}</li></template>
<script> export default { data() { return {
  message_value: '', message_history: []}},methods: {
  send(){this.message_history.push(this.message_value);
  this.message_history.push("sam durak");
  this.message_value = "";}}}</script>
```

Конфигурация faces-config, задающая managed bean с именем myBean, которым будет управлять сам программист

```
<faces-config version="2.2" ...> <managed-bean>
  <managed-bean-name>myBean</managed-bean-name>
  <managed-bean-class>MyBean</managed-bean-class>
  <managed-bean-scope>custom</managed-bean-scope>
</managed-bean> </faces-config>
```

Написать на Angular интерфейс, который проверяет если ли в куки sessionId и если нет, отправляет пользователя на аутентификацию по логину и паролю

```
@Component({selector: 'app',
  template: '<form *ngIf="!hasCookie">
  <h1>auth</h1><input type="text" name="username"
  ↳ placeholder="name">
  <input type="password" name="password"
  ↳ placeholder="password">
  <input type="submit" value="sing in"></form>'})
class AppComponent {get hasCookie(): boolean {
  return document.cookie.indexOf("jSessionid=") !=
  ↳ -1;}}
```

REST - контроллер на Spring Web MVC, представляющий CRUD-интерфейс к таблице со списком покемонов

```
@RestController class PokemonResource {
  @Autowired private PokemonRepository repository;
  @GetMapping("/pokemons") List<Pokemon> all() {
    return repository.findAll();
  }
  @GetMapping("/{pokemons/{id}}")
  Pokemon one(@PathVariable Long id) {
    return repository.findOne(id);
  }
  @PostMapping("/pokemons")
  Long createNew(@RequestBody Pokemon pokemon) {
    return repository.save(pokemon);
  }
  @DeleteMapping("/{pokemons/{id}}")
  void delete(@PathVariable Long id) {
    repository.delete(id);
  }
}
```

Конфигурация, чтобы JSF обрабатывал все запросы приходящие с .xhtml и со всех URL, начинающихся с /faces/

```
<web-app><servlet>
  <servlet-name>Faces Servlet</servlet-name>
  <servlet-class>javax.faces.webapp.FacesServlet
  </servlet-class></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></web-app>
```

RestController, который реализует перевод градусов Цельсия в Фаренгейты и обратно

```
@RestController class TempController {
  @GetMapping("/convert/c/f")
  double cToF(@RequestParam double c) {return c * 1.8
  ↳ + 32.0;}
  @GetMapping("/convert/f/c")
  double fToC(@RequestParam double f) {return (f -
  ↳ 32.0) / 1.8;}}
```

Angular компонент, который позволяет поделиться чем-то в VK, Twitter, Facebook (API для соцсетей можно описать текстом) API принимает логин и пароль, а также сообщение, которое будет опубликовано на личной странице после прохождения авторизации.

```
import { Component } from '@angular/core';
import { HttpClient } from '@angular/common/http';
@Component({ selector: 'my-app',
  template: <div class="form-group">
  <label>Login</label>
  <input class="form-control" name="username"
  ↳ [(ngModel)]="username" /> </div>
  <div class="form-group"><label>Password</label>
  <input class="form-control" name="password"
  ↳ [(ngModel)]="password" /></div>
  <div class="form-group"><label>Message</label>
  <input class="form-control" name="message"
  ↳ [(ngModel)]="message" /></div>
  <div class="form-group">
  <button class="btn btn-default"
  ↳ (click)="submit()">Send</button></div>})
export class AppComponent {username: string="";
  password: string="";message: string="";
  http: HttpClient;
  submit(){const body = {login: username, password:
  ↳ password, message: message;
  this.http.post('link to API', body);}}
```

Компонент для React, формирующий строку с автодополнением. Массив значений для автодополнения должен получаться с сервера посредством запроса к REST API

```
function Autocomplete() {
  const [candidates, setCandidates] = useState([])
  async function updateList(e) {
    let res = await fetch("http://api.itmo.ru/stu
    dents/search?q=" + e.target.value);
    setCandidates(await res.json()); return <div>
    <input type="text" onChange={updateList}>
    <ul>{candidates.map((text, i) => <li
    ↳ key={i}>{text}</li>)}</ul></div>;}
```

Написать на vue.js интерфейс для навигации по текстовому документу, представленному в виде большой строки, должны быть реализованы переходы на следующую и предыдущую страницу

```
<template>
<div><p>{{ this.document.substring(this.start,
  ↳ this.start + this.shift) }}</p></div>
<button @click="next">Next</button>
<button @click="prev">Back</button>
</template><script> export default {
  data :function() {return {start : 0, shift : 10,
  ↳ document : "lotofNumbers"}},
  methods :{next : function(){this.start +=
  ↳ this.shift;}, prev : function(){
  this.start -= this.shift;}}}</script>
```

Интерфейс на Angular, который выводит интерактивные часы с обновлением каждую секунду

```
@Component({ selector: 'my-app',
  template: <div>{{time}}</div>})
export class AppComponent{
  time: ""
  setInterval(()=>this.time = new Date(),1000)}
```

Написать JSF страницу с многострочным полем, в которое можно вводить только строчные символы латиницы.

```
<h:head> <script> function validateInput() {
  var textArea =
  ↳ document.getElementById('inputTextArea');
  var regex = /^[a-z\s]*$/; if
  ↳ (!regex.test(textArea.value)) {
  textArea.value = '';}</script></h:head>
<h:body><h:form>
<h:inputTextArea id="inputTextArea" rows="5"
  ↳ cols="30" oninput="validateInput()"
  ↳ /></h:form></h:body>
```

Приложение на Angular, реализующее форму для заполнения бланка на отчисление по собственному желанию. Форма должна принимать на вход имя пользователя и дату, и формировать заполненный бланк заявления (на клиентской стороне)

```
@Component({selector: 'app',template: '<form>
<input type="text" name="firstName"
  ↳ placeholder="Name"
  ↳ [(ngModel)]="firstName">
<input type="text" name="lastName"
  ↳ placeholder="lastName"
  ↳ [(ngModel)]="lastName">
<input type="date" name="lastName"
  ↳ [(ngModel)]="date"></form>
<main><h1>PSJ</h1>
<h2>{{ firstName }} {{ lastName }} {{ date }} ,
  ↳ PSJ</h2>
</main>'})
class AppComponent {firstName = "" lastName = ""
  date = "01-09-2021"}
```

Реализовать бронь авиабилетов на jsf

```
@ManagedBean @SessionScoped @Setter @Getter
public class Ticket {private int id;
  private int price;private String source;
  private String dest;@ManagedProperty(value =
  ↳ "#{repoBean}")
  private TicketRepo repo;
  private List<Ticket> available;
  @PostConstruct void init(){
  available = repo.getAvaiable();
  public void book(int id) {repo.book(id);
  available = repo.getAvaiable();}
  <h1>Book avia ticket<></h1>
  <ui:repeat value="#{tickets.available}"
  ↳ var="ticket">
  <div class="ticket">
  #{ticket.source} - #{ticket.destination}
  Number: #{ticket.id} Cost: #{ticket.price} e.g.
  <h:commandButton action="#{tickets.book(ticket.id)}"
  value="book"></div></ui:repeat>
```

JSF страничка с данными из бина

```
@Named("myBean") @ManagedBean class MyBean {
  int value; public int getValue(){return value;}}
<h:outputText value="#{myBean.value}"/>
```

Интерфейс реализации логин+пароль на React. На стороне сервера- Rest API

```
function App(props) {
  const [login, setLogin] = useState("")
  const [password, setPassword] = useState("")
  async function onSubmit() {
    let res = await fetch("api/login", {
      headers: { 'Content-Type': "application/json" },
      body: JSON.stringify({login: login, password:
      ↳ password})
    })
    let token = await res.text();
    localStorage.setItem("token", token)
    return <form onSubmit={onSubmit}>
    <input type="text" placeholder="login" required
    ↳ value={login}>
    onChange={e => setLogin(e.target.value)}>
    <input type="password" placeholder="Pass" required
    ↳ value={password}>
    onChange={e => setPassword(e.target.value)}>
    <input type="submit" value="Sign in"/>
    </form>;}
```

Интерфейс на Vue, реализующий простой телефонный набиратель (кнопки 0-9, *, #, кнопка набрать номер и экран)

```
<template><div class="screen"
  ↪ v-text="display"></div>
<div class="keypad"><button v-for="n in numbers"
  ↪ v-text="n"@click="typeN(n)"></button></div>
<button @click="ring">Ring</button> </template>
<script>export default {data() {return {
display: '',numbers: [0-9, '*', '#']}},
methods: {typeN(n) {this.display += n;},
ring() {alert("Ringing" + this.display);}}}
</script>
```

Написать на JSF правило навигации в faces-config.xml со страницы page1.jsp на page2.jsp с использованием этой кнопки: `<h:commandButton action="goToPage"value="Go to Page 2"/>`

```
<navigation-rule>
<from-view-id>/page1.jsp</from-view-id>
<navigation-case>
<from-outcome>goToPage</from-outcome>
<to-view-id>/page2.jspl</to-view-id>
<redirect/></navigation-case></navigation-rule>
```

Написать интерфейс для ввода данных банковской карты на React

```
export function App() {
const [name, setName] = React.useState('');
const [number, setNumber] = React.useState('');
function handleSubmit(event) {
event.preventDefault(); console.log('name:', name);
console.log('number:', number);}
return (<form onSubmit={handleSubmit}><div>
<label>Name</label><input type="text" maxLength={5}
value={name} onChange={(e)=>setName(e.target.value)}
/></div><div><label>CVS Number</label>
<input type="text" maxLength={5} value={number}
onChange={(e) => setNumber(e.target.value)}></div>
<button type="submit">Submit</button></form>);}
```