

Задача 1: Калькулятор с двумя полями

Разработайте EJB приложение, реализующее калькулятор с двумя полями ввода. Программа должна включать в себя следующие компоненты:

1. Веб-слой:
 - Реализуйте архитектуру MVC2 для управления пользовательским интерфейсом.
 - Создайте форму с двумя полями для ввода чисел и кнопками операций.
2. EJB слой:
 - Реализуйте бизнес-логику калькулятора.
 - Используйте инъекцию зависимостей для управления ресурсами.
3. Функциональность:
 - Обеспечьте выполнение арифметических операций (сложение, вычитание, умножение, деление).
 - Реализуйте обработку ошибок и валидацию ввода.

Задача 2: Калькулятор с одним полем и памятью

Разработайте EJB приложение, реализующее калькулятор с одним полем ввода и функцией памяти. Программа должна включать следующие компоненты:

1. Веб-слой:
 - Реализуйте архитектуру MVC2 для управления пользовательским интерфейсом.
 - Создайте форму с одним полем для ввода чисел и кнопками операций.
2. EJB слой:
 - Реализуйте бизнес-логику калькулятора с функцией памяти.
 - Используйте инъекцию зависимостей для управления ресурсами.
3. Функциональность:
 - Обеспечьте выполнение арифметических операций (сложение, вычитание, умножение, деление).
 - Реализуйте функцию сохранения числа в память.
 - Добавьте возможность запоминания имени пользователя.
 - Убедитесь в корректной работе с памятью и сохранением данных.

Задача 3: Игровое приложение "Угадай число"

Разработайте игровое EJB приложение, которое угадывает случайное число от 1 до 10. Программа должна включать следующие компоненты:

1. Веб-слой:
 - Реализуйте архитектуру MVC2 для управления пользовательским интерфейсом.

- Создайте форму с полем ввода и кнопкой отправки ответа.
2. EJB слой:
- Реализуйте бизнес-логику угадывания числа.
 - Используйте инъекцию зависимостей для управления ресурсами.
3. Функциональность:
- Генерируйте случайное число от 1 до 10.
 - Позволяйте нескольким пользователям одновременно пытаться угадать число.
 - Определяйте победителя как первого успешного угадчика.
 - Реализуйте логику завершения игры после правильного ответа.

Выполнение:

Для работы Java EE использовались следующие программы и компоненты: GlassFish, JDK, NetBeans, EJB.

Был создан веб-проект Java EE. Созданы классы сущностей, сеансовые компоненты, контроллеры для обработки запросов, подготовки и передачи управления в созданные соответствующие jsp-файлы, которые отрисовывают результат.

На рис. 1 представлен контроллер Calculator2fieldController для обработки запросов при работе с калькулятором, имеющим 2 поля. Для обработки введенных чисел реализован выбор вызова метода, соответствующего нажатию кнопки с арифметической операцией.

```
Calculator2fieldController.java x
Источник  История  [Icons]
35  override
41  protected void doGet(HttpServletRequest request, HttpServletResponse response)
42      throws ServletException, IOException {
43      String action = request.getParameter("action");
44      try {
45          switch (action) {
46              case "operat": //фильтр по группе
47                  op(request, response);
48                  break;
49              default: //вывод списка студентов
50                  def(request, response);
51                  break;
52          }
53      } catch (SQLException ex) {
54          throw new ServletException(ex);
55      }
56  }
57  private void op(HttpServletRequest request, HttpServletResponse response)
58      throws SQLException, IOException, ServletException {
59      //получение чисел из полей ввода
60      double num1 = Double.parseDouble(request.getParameter("num1"));
61      double num2 = Double.parseDouble(request.getParameter("num2"));
62      String operation = request.getParameter("group1");
63      List result = new ArrayList();
64      //сохранение введенных чисел в полях
65      result.add(num1);
66      result.add(num2);
67      switch (operation) {
68          case "+":
69              result.add(calculator2fieldModel.add(num1, num2)); //сложение
70              break;
71          case "-":
72              result.add(calculator2fieldModel.subtract(num1, num2)); //вычитание
73              break;
74          case "*":
75              result.add(calculator2fieldModel.multiply(num1, num2)); //умножение
76              break;
77          case "/":
78              result.add(calculator2fieldModel.divide(num1, num2)); //деление
79              break;
80          default:
81              result.add(0);
82              break;
83      }
84      request.setAttribute("result", result);
85  }
```

Рисунок 1

Пример сеансового компонента Calculator2fieldModel для работы с арифметическими операциями над числами, который представляет собой простой интерфейс и скрывает сложность модели от клиента, находится на рис.2.

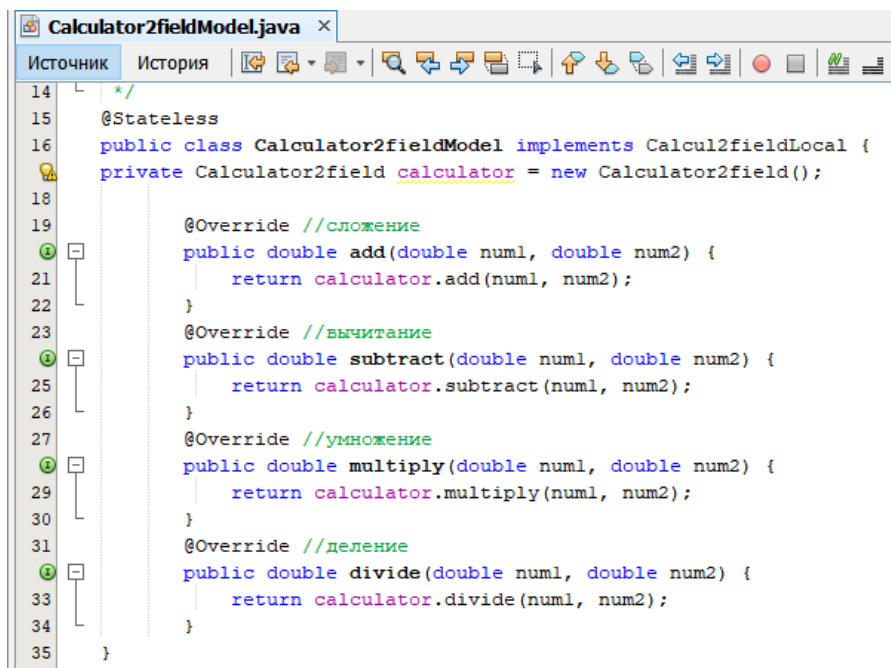


Рисунок 2

Класс Calculator2field с соответствующими геттерами, сеттерами и работы с числами, представлен на рис.3.

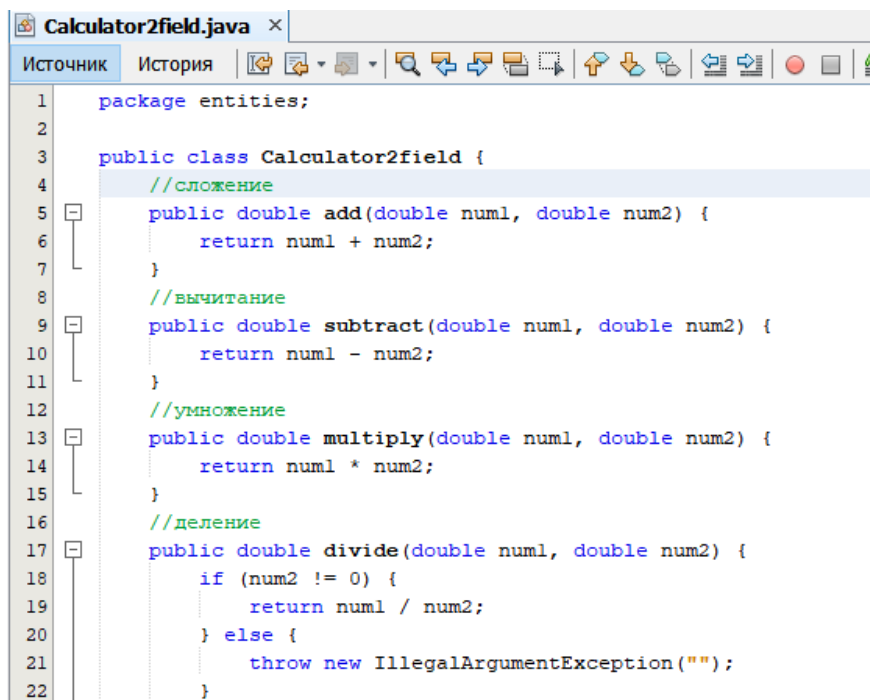


Рисунок 3

Для отображения калькулятора с двумя полями создана страница calculator2field.jsp (рис.4), которая является начальной. На странице отображается калькулятор, который имеет два поля ввода и одно поле вывода, а также кнопки с изображением арифметических операций. На странице присутствует верхняя панель для быстрого переключения между страницами калькуляторов и игры.

```

calculator2fields.jsp
Источники История
18 <nav class="navbar navbar-expand-md navbar-dark" style="background-color: lightgreen; font-size: 1.3rem">
19 <ul class="navbar-nav">
20 <li><a href="calculator2fields.jsp" class="nav-link" style="color: hsl(59, 100%, 5%);">Два поля</a></li></ul>
21 <ul class="navbar-nav">
22 <li><a href="calculator1fields.jsp" class="nav-link" style="color: hsl(59, 100%, 5%);">Одно поле</a></li></ul>
23
24 </nav>
25 </header>
26 <hr>
27 <div class="calculator">
28 <form method="POST" action="Calculator2fieldController?action=operat">
29 <div class="cal_out">
30 <p class="cal_in_Text">Первое число:</p>
31 <input class="cal_in_input" type="text" name="num1" value="${result[0]}"></div>
32 <div class="cal_out">
33 <p class="cal_out_Text">Второе число:</p>
34 <input class="cal_in_input" type="text" name="num2" value="${result[1]}">
35 <p></p> </div>
36 <div class="calculator_keys">
37 <input class="add" type="submit" name="group1" value ="+"><br>
38 <input class="sub" type="submit" name="group1" value ="-"><br>
39 <input class="mul" type="submit" name="group1" value ="*"><br>
40 <input class="div" type="submit" name="group1" value ="/"><br><p></p>
41 </div>
42 <div class="cal_out">
43 <p></p>
44 <p>Результат:</p>
45 <input class="cal_out_input" type="text" value="${result[2]}" readonly><p></p>
46 </div>
47 </form>
48 </div>
49 </body>
50 </html>
51 <style>
52 .calculator {
53   align-items: center;
54   position: relative;
55   top: 50%;
56   left: 20%;
57   max-inline-size: 20rem;
58   border-radius: 10px;
59   background: hsl(156, 41%, 94%);

```

Рисунок 4

На рисунке 5 демонстрируется калькулятор с двумя полями, после введения двух чисел и нажатия на кнопку с арифметической операцией «+», и результат ее выполнения над введенными числами.

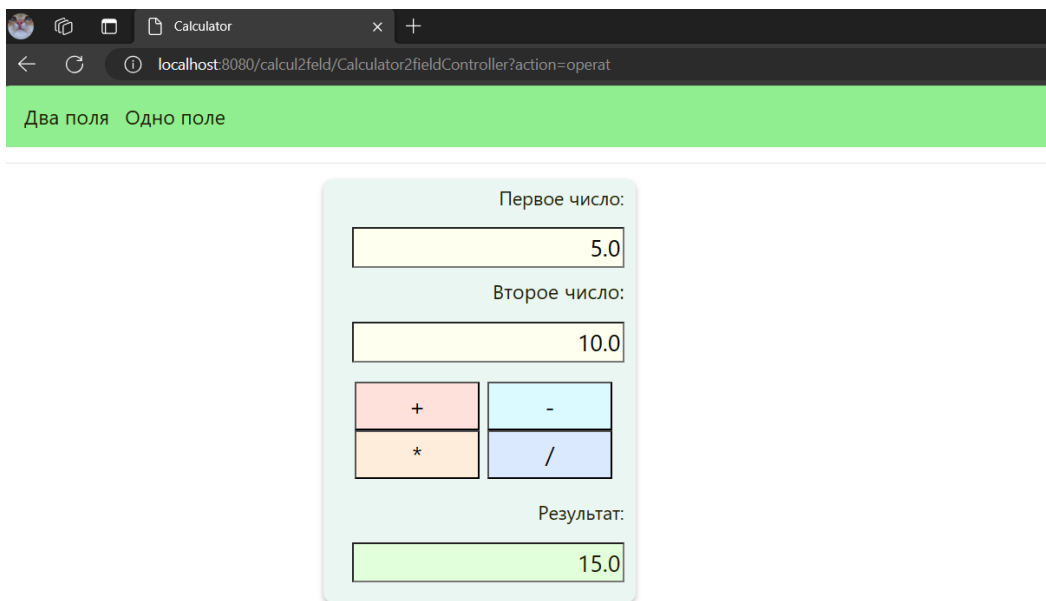


Рисунок 5

На рисунке 6 демонстрируется калькулятор с двумя полями, после введения двух чисел и нажатия на кнопку с арифметической операцией «/», и результат ее выполнения над введенными числами.

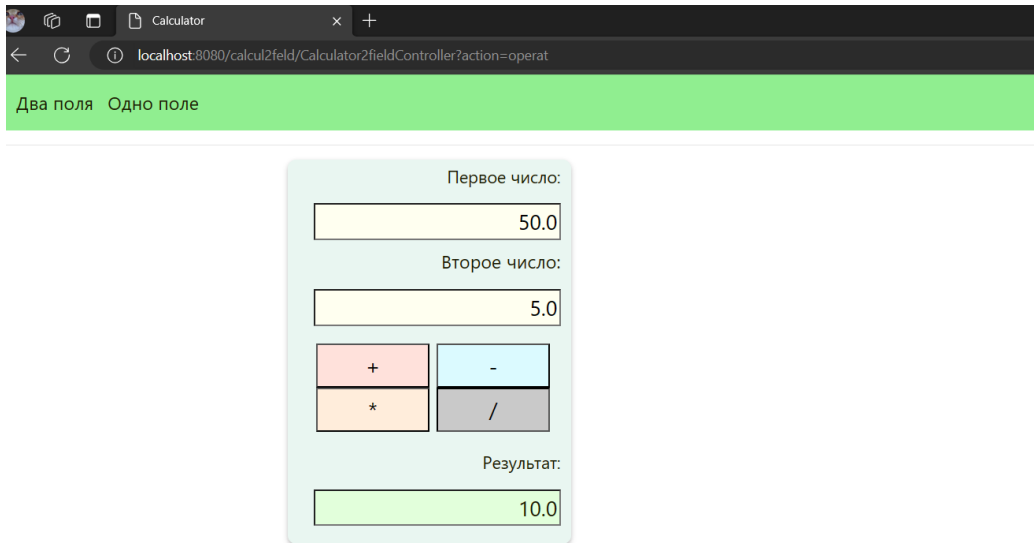


Рисунок 6

На рис.7 представлен контроллер Calculator1fieldController для обработки запросов при работе с калькулятором, имеющим 1 поле ввода чисел и поле для ввода имени. Решение выражения происходит путем обработки введенной строки.

```

Calculator1fieldController.java
Источник  История
23  */
24  @Named(value = "calculator1fieldController")
25  @Dependent
26  @WebServlet(name = "Calculator1fieldController", urlPatterns = {"/Calculator1fieldController"})
27  public class Calculator1fieldController extends HttpServlet {
28
29      @EJB
30      private Calcul1fieldLocal calculator1fieldModel;
31
32      public Calculator1fieldController() {
33      }
34
35      @Override
36      protected void doGet(HttpServletRequest request, HttpServletResponse response)
37          throws ServletException, IOException {
38          //получение выражения с поля ввода
39          String inputValue = request.getParameter("input");
40          //получение имени клиента
41          String fio = request.getParameter("fio");
42          double result = 0;
43          //вычисление выражения
44          result = calculator1fieldModel.calculate(inputValue);
45          //передача результата вычислений и имени клиента обратно на форму
46          request.setAttribute("result", result);
47          request.setAttribute("fio", fio);
48          RequestDispatcher dispatcher = request.getRequestDispatcher("/calculator1fields.jsp");
49          dispatcher.forward(request, response);
50
51      }
52
53      @Override
54      protected void doPost(HttpServletRequest request, HttpServletResponse response)
55          throws ServletException, IOException {
56          doGet(request, response);

```

Рисунок 7

Пример сеансового компонента Calculator1fieldModel для работы с арифметическими операциями над числами, который представляет собой простой интерфейс и скрывает сложность модели от клиента, находится на рис.8.

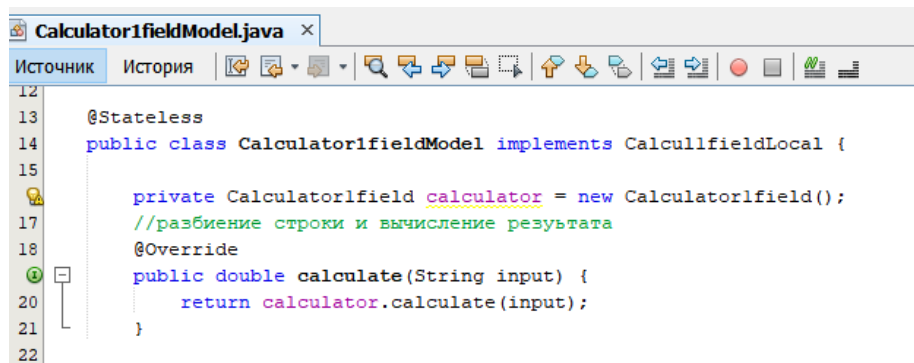


Рисунок 8

Класс Calculator1field, который обрабатывает строку, расщепляя ее на числа и арифметические знаки, и вычисляя значение, представлен на рис.9.

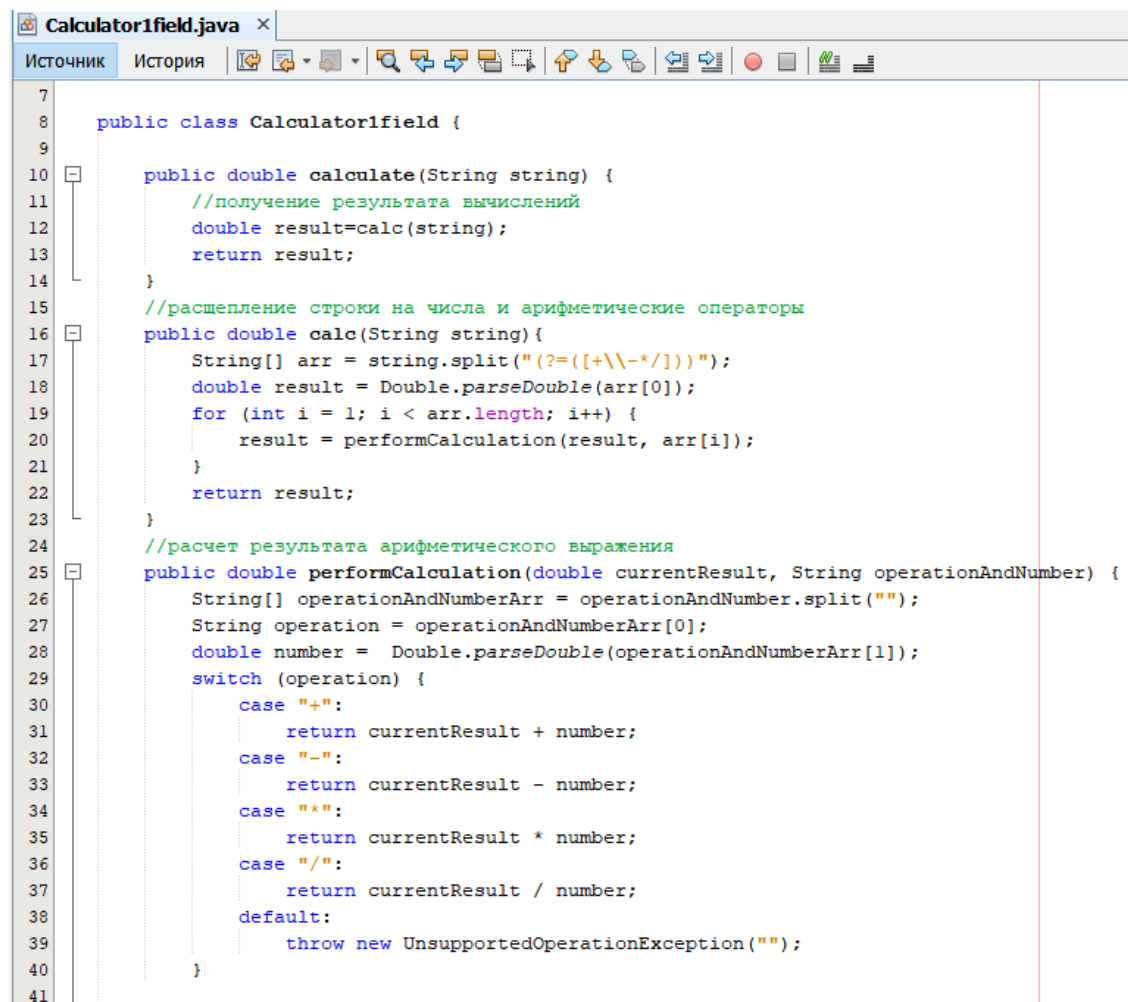


Рисунок 9

Для отображения калькулятора с одним полем создана страница calculator1field.jsp (рис.10). На странице отображается калькулятор, который имеет поле ввода выражения и поле ввода имени, а также кнопки с изображением

арифметических операций, цифр и кнопка равно. Также присутствует поле ввода имени. На странице присутствует верхняя панель для быстрого переключения между страницами калькуляторов и игры.

```

calculator1fields.jsp
История
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
<ul class="navbar-nav">
  <li><a href="calculator2fields.jsp" class="nav-link" style="color: hsl(59, 100%, 5%);">Два поля</a></li></ul>
  <ul class="navbar-nav">
    <li><a href="calculator1fields.jsp" class="nav-link" style="color: hsl(59, 100%, 5%);">Одно поле</a></li></ul>
  </nav>
</header>
<hr>
<div class="calculator">
  <div>
    <form method="POST" action="Calculator1fieldController">
      <input class="calculator_output" type="text" id="input" name="input" value="${result}" readonly />
    </div>
    <div class="calculator_keys">
      <input type="button" class="calculator_key calculator_key--operator" value="+" onclick="appendToInput('+')"/>
      <input type="button" class="calculator_key calculator_key--operator" value="-" onclick="appendToInput('-')"/>
      <input type="button" class="calculator_key calculator_key--operator" value="/" onclick="appendToInput('/')"/>
      <input type="button" class="calculator_key calculator_key--operator" value="*" onclick="appendToInput('*')"/>
      <input type="button" class="calculator_key" value="7" onclick="appendToInput('7')"/>
      <input type="button" class="calculator_key" value="8" onclick="appendToInput('8')"/>
      <input type="button" class="calculator_key" value="9" onclick="appendToInput('9')"/>
      <input type="button" class="calculator_key" value="4" onclick="appendToInput('4')"/>
      <input type="button" class="calculator_key" value="5" onclick="appendToInput('5')"/>
      <input type="button" class="calculator_key" value="6" onclick="appendToInput('6')"/>
      <input type="button" class="calculator_key" value="1" onclick="appendToInput('1')"/>
      <input type="button" class="calculator_key" value="2" onclick="appendToInput('2')"/>
      <input type="button" class="calculator_key" value="3" onclick="appendToInput('3')"/>
      <input type="button" class="calculator_key" value="0" onclick="appendToInput('0')"/>
      <input type="button" class="calculator_key" value="." onclick="appendToInput('.')"/>
      <input type="button" class="calculator_key" value="C" onclick="clearInput()"/>
      <input type="submit" class="calculator_key calculator_key--enter" value="=" />
    </div>
    <p class="cal_Text">Введите имя:</p>
    <input class="calculator_outName" type="text" name="fio" value="${fio}" />
  </div>
</form>
</div>
<script>
function appendToInput(value) {
  var input = document.getElementById("input");
  input.value += value;
}
function clearInput() {
  var input = document.getElementById("input");
  input.value = "";
}

```

Рисунок 10

На рисунке 11 демонстрируется калькулятор с одним полем до введения чисел.

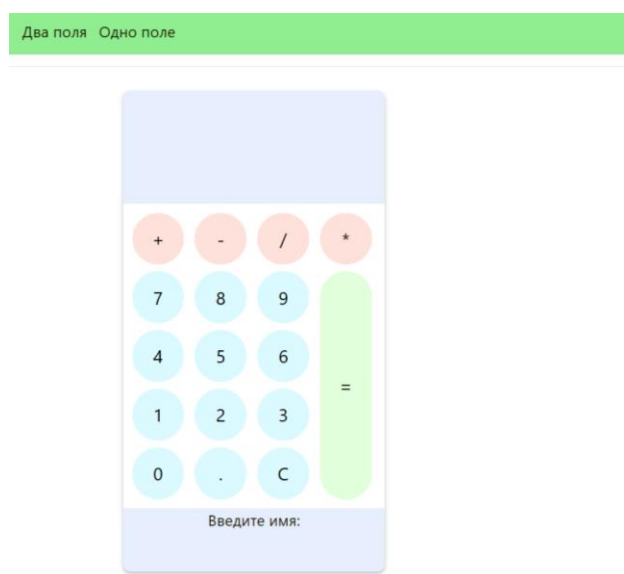


Рисунок 11

На рисунке 12 демонстрируется калькулятор с одним полем после введения двух чисел, посредством нажатия кнопок с цифрами и арифметическими операциями, и нажатия на кнопку равно, и результат выполнения. Также введенное имя остается на экране после нажатия кнопки равно.



Рисунок 12

На рис. 13 представлен контроллер PlayerController для обработки запросов при работе с игрой. Для запуска игры предусмотрена кнопка «Угадать»

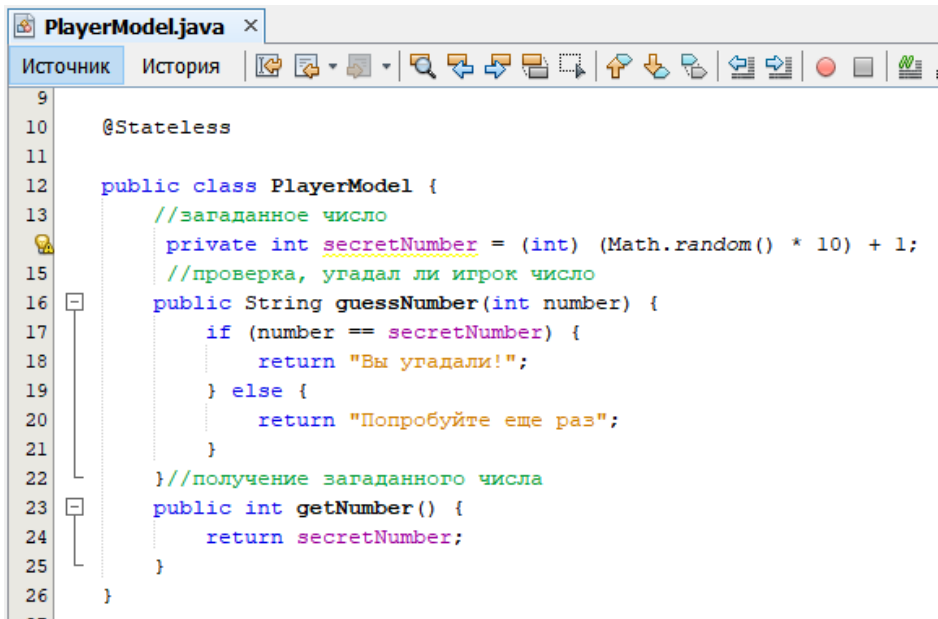
```

PlayerController.java x
Источник  История
22
23 @WebServlet(name = "PlayerController", urlPatterns = {"/PlayerController"})
24 public class PlayerController extends HttpServlet {
25
26     @EJB
27     private PlayerModel playerModel;
28
29     @Override
30     protected void doGet(HttpServletRequest request, HttpServletResponse response)
31         throws ServletException, IOException {
32         try {
33             Player player;
34             int i = 0;
35             //получение числа-ответа от каждого игрока
36             while (i <= 3) {
37                 player = new Player(i);
38                 int number = player.getNumber();
39                 request.setAttribute("player" + i, number);
40                 String result = playerModel.guessNumber(number);
41                 //сопоставление заданного числа с выданным игроком
42                 request.setAttribute("result" + i, result);
43                 i++;
44             }
45             //вывод заданного числа
46             request.setAttribute("secret", playerModel.getNumber());
47             request.getRequestDispatcher("/game.jsp").forward(request, response);
48
49         } catch (NumberFormatException ex) {
50         }
51     }
52
53     @Override
54     protected void doPost(HttpServletRequest request, HttpServletResponse response)
55         throws ServletException, IOException {
56         doGet(request, response);

```

Рисунок 13

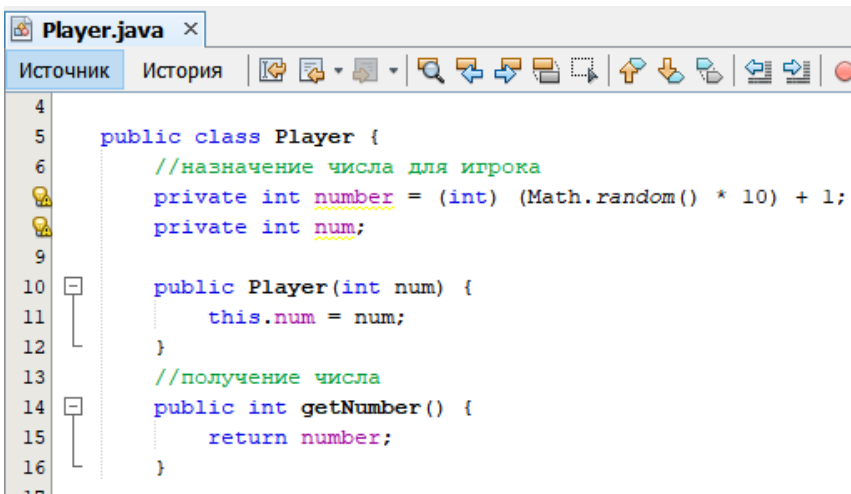
Пример сеансового компонента PlayerModel для назначения загаданного числа и проверки чисел игроков, который представляет собой простой интерфейс и скрывает сложность модели от клиента, находится на рис.14.



```
9
10 @Stateless
11
12 public class PlayerModel {
13     //загаданное число
14     private int secretNumber = (int) (Math.random() * 10) + 1;
15     //проверка, угадал ли игрок число
16     public String guessNumber(int number) {
17         if (number == secretNumber) {
18             return "Вы угадали!";
19         } else {
20             return "Попробуйте еще раз";
21         }
22     } //получение загаданного числа
23     public int getNumber() {
24         return secretNumber;
25     }
26 }
```

Рисунок 14

Класс Player, который назначает игрокам числа, представлен на рис.15.



```
4
5 public class Player {
6     //назначение числа для игрока
7     private int number = (int) (Math.random() * 10) + 1;
8     private int num;
9
10    public Player(int num) {
11        this.num = num;
12    }
13    //получение числа
14    public int getNumber() {
15        return number;
16    }
17 }
```

Рисунок 15

Для отображения игры создана страница game.jsp (рис.16). На странице отображаются поля, в которых появляются числа игроков, по нажатию кнопки, а также вывод загаданного числа и пояснения для каждого игрока, угадал ли он число. На странице присутствует верхняя панель для быстрого переключения между страницами калькуляторов и игры.

```

16 <body >
17 <header>
18 <nav class="navbar navbar-expand-md navbar-dark" style="background-color: lightgreen; font-size: 1.3rem">
19 <ul class="navbar-nav">
20 <li><a href="calculator2fields.jsp" class="nav-link" style="color: hsl(59, 100%, 5%);">Два поля</a></li></ul>
21 <ul class="navbar-nav">
22 <li><a href="calculator1fields.jsp" class="nav-link" style="color: hsl(59, 100%, 5%);">Одно поле</a></li></ul>
23 <ul class="navbar-nav">
24 <li><a href="game.jsp" class="nav-link" style="color: hsl(59, 100%, 5%);">Игра</a></li></ul>
25 </nav>
26 </header>
27 <hr>
28 <div class="game">
29 <form method="POST" action="PlayerController">
30 <h2>Игроки угадывают число число</h2>
31
32 <p class="game_Text">Игрок 1 <input class="game_out" type="text" name="player1" value="{player1}" readonly /></p>
33 <p class="game_Text">Игрок 2 <input class="game_out" type="text" name="player2" value="{player2}" readonly /></p>
34 <p class="game_Text">Игрок 3 <input class="game_out" type="text" name="player3" value="{player3}" readonly /></p>
35
36 <input class="key" type="submit" value="Угадать">
37 <p></p>
38 </form>
39 </div>
40 <div class="gameResult">
41 <p class="game_Text">Загаданное число: {secret}</p>
42 <p class="game_Text">Игрок 1: {result1}</p>
43 <p class="game_Text">Игрок 2: {result2}</p>
44 <p class="game_Text">Игрок 3: {result3}</p>
45 </div>
46 </body>
47 </html>
48 <style>
49 .game {
50 align-items: center;
51 position: relative;
52 top: 50%;
53 left: 20%;
54 max-inline-size: 20rem;
55 border-radius: 10px;
56 background: hsl(156, 41%, 94%);

```

Рисунок 16

На рисунке 17 демонстрируется игра до нажатия кнопки «Угадать».

Два поля Одно поле Игра

Игроки угадывают число число

Игрок 1

Игрок 2

Игрок 3

Загаданное число:

Игрок 1:

Игрок 2:

Игрок 3:

Рисунок 17

На рисунке 18 демонстрируется игра после нажатия кнопки «Угадать», здесь отображаются числа каждого игрока, загаданное число и результат угадывания числа каждым игроком.

**Игроки угадывают
число число**

Игрок 1

Игрок 2

Игрок 3

Загаданное число: 1

Игрок 1: Попробуйте еще раз

Игрок 2: Вы угадали!

Игрок 3: Попробуйте еще раз

Рисунок 18

Ответы на вопросы:

1. Дайте определение EJB. Роль EJB в приложениях уровня предприятия.

Enterprise JavaBeans (EJB) является серверной архитектурой компонентов для Java Platform, Enterprise Edition (Java EE). EJB позволяет быстро и просто разрабатывать распределенные, транзакционные, безопасные и переносимые приложения на основе технологии Java 1.

EJB обеспечивает множество преимуществ для разработчика и системного администратора. Во-первых, использование EJB позволяет создавать модульные и масштабируемые приложения, которые могут быть развернуты на разных серверах. Во-вторых, EJB предоставляет механизмы для управления транзакциями, безопасностью и конкурентным доступом к данным. Это значительно упрощает разработку сложных приложений и повышает их надежность

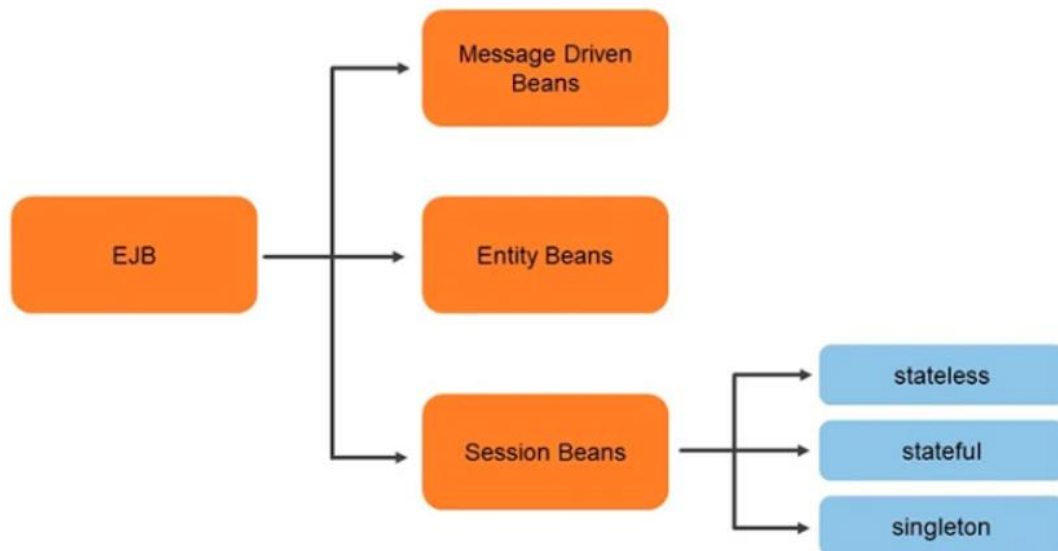
2. Классификация EJB.

EJB — это обычный Java класс, отмеченный одной из специальных аннотаций. Такие классы называют бинами. В зависимости от того, какой аннотацией отмечен класс, он будет являться представителем того или иного типа EJB (бинов). Есть три основных типа бинов:

- Message Driven Beans (бины, управляемые сообщениями);
- Entity Beans (объектные бины) — определены в спецификации JPA (Java Persistence API) и используются для хранения данных;
- Session Beans (сессионные бины).

Последние (сессионные бины) подразделяются на несколько подвидов:

- stateless (без состояния);
- stateful (с поддержкой текущего состояния сессии);
- singleton (один объект на все приложение; начиная с версии EJB 3.1).



3. Методы доступа к EJB (DI & JNDI).

Доступ к EJB может осуществляться двумя основными способами: через инъекцию зависимостей (DI) и через JNDI (Java Naming and Directory Interface).

Инъекция зависимостей позволяет автоматически внедрять зависимости в компоненты. В контексте EJB, это означает, что EJB могут быть внедрены в другие компоненты, такие как servlet или другие EJB.

JNDI используется для поиска EJB по имени. В JNDI есть три пространства имен: java:global, java:module и java:app.

- java:global используется для поиска удаленных Enterprise-бинов. Адреса JNDI имеют следующую форму:
java:global[/application name]/module name/enterprise bean name[/interface name].
- java:module используется для поиска локальных Enterprise-бинов внутри одного и того же модуля. Адреса JNDI, использующие пространство имён java:module, имеют следующую форму:
java:module/enterprise bean name[/interface name].
- java:app используется для поиска локальных Enterprise-бинов, упакованных в одном приложении. Адреса JNDI, использующие пространство

имён `java:app`, имеют следующую форму: `java:app[/module name]/enterprise bean name[/interface name]`.

4. Business Interface & No-interface views.

Бизнес-интерфейсы определяют методы бизнес-логики, которые могут быть вызваны клиентами. В EJB 3.x бизнес-интерфейсы могут быть локальными или удаленными.

Локальный бизнес-интерфейс позволяет клиентам, работающим в той же JVM, вызывать методы бина. Для этого используется аннотация `@Local` на интерфейсе бизнес-логики.

Удаленный бизнес-интерфейс позволяет клиентам, работающим в разных JVM или даже на разных машинах, вызывать методы бина. Для этого используется аннотация `@Remote` на интерфейсе бизнес-логики.

No-Interface View

No-Interface View представляет собой способ доступа к бинам без использования интерфейса бизнес-логики. В этом случае все публичные методы класса бина автоматически становятся доступными для клиентов. Для использования No-Interface View используется аннотация `@LocalBean` на классе бина.

В этом случае клиенты могут вызывать методы бина напрямую, без использования интерфейса бизнес-логики. Однако клиент и бин должны быть упакованы в одном и том же приложении (EAR).

Remote Interfaces

Удаленные интерфейсы позволяют клиентам, работающим в разных JVM или даже на разных машинах, вызывать методы бина. Для этого используется аннотация `@Remote` на интерфейсе бизнес-логики. Клиенты могут получить ссылку на удаленный интерфейс бина через инъекцию зависимостей или через JNDI.

Здесь `ExampleRemote` - это удаленный бизнес-интерфейс Enterprise-бина. Этот код автоматически внедряет EJB, который реализует интерфейс `ExampleRemote`.

Важно отметить, что выбор между использованием бизнес-интерфейсов, No-Interface View или удаленных интерфейсов зависит от конкретной ситуации и требований к приложению.

5. Типы клиентов EJB: local, remote, web service.

EJB (Enterprise Java Beans) предоставляют три основных типа клиентов для доступа к бинам: локальный, удаленный и веб-сервис.

Локальный клиент может обращаться к бинам EJB, которые работают в той же JVM. Это означает, что клиент и бин должны быть упакованы в одном и том же приложении (EAR). Для доступа к бинам EJB локальным клиентом используется локальный бизнес-интерфейс.

Удаленный клиент может обращаться к бинам EJB, которые работают в разных JVM или даже на разных машинах. Для этого используется удаленный бизнес-интерфейс.

Веб-сервис — это способ предоставления бинам EJB удаленного доступа. Веб-сервисы используют протоколы, такие как SOAP или REST, для обмена сообщениями между клиентами и бинами EJB. Веб-сервисы могут быть доступны как локальные, так и удаленные.

6. Программирование local доступа.

Для программирования локального доступа к EJB (Enterprise Java Beans) используют локальный бизнес-интерфейс. Локальный клиент может обращаться к бинам EJB, которые работают в той же JVM. Это означает, что клиент и бин должны быть упакованы в одном и том же приложении (EAR).

Определение локального бизнес-интерфейса для бина EJB:

```
@Local
public interface LocalBusinessInterface {
    // методы бизнес-логики
}
```

Здесь LocalBusinessInterface — это локальный бизнес-интерфейс бина EJB.

Для доступа к бинам EJB локальным клиентом можно использовать аннотацию @EJB и указать имя локального бизнес-интерфейса.

```
@EJB
LocalBusinessInterface localBusinessInterface;
```

Здесь LocalBusinessInterface — это локальный бизнес-интерфейс бина EJB, к которому вы хотите получить доступ.

Также можно использовать JNDI для получения ссылки на локальный бизнес-интерфейс бина EJB. Для этого используется метод lookup интерфейса `javax.naming.InitialContext`. Например:

```
LocalBusinessInterface localBusinessInterface =  
(LocalBusinessInterface) InitialContext.lookup("java:module/LocalBusinessInt  
erface");
```

Здесь `LocalBusinessInterface` — это локальный бизнес-интерфейс бина EJB, к которому нужно получить доступ.

Локальный доступ к бинам EJB имеет ограничение: он работает только в рамках одной JVM и одного приложения (EAR). Если нужно обращаться к бинам EJB из разных JVM или приложений, то потребуется использовать удаленный доступ или веб-сервисы.

7. Программирование remote доступа.

Создаем интерфейс, который будет служить удаленным бизнес-интерфейсом. Этот интерфейс должен быть аннотирован с `@Remote`:

```
import javax.ejb.Remote;  
  
@Remote  
public interface MyRemoteInterface {  
    String myRemoteMethod();  
}
```

Реализуем этот интерфейс в сессионном бине. Сессионный бин должен быть аннотирован с `@Remote`:

```
import javax.ejb.Remote;  
import javax.ejb.Stateless;  
  
@Stateless  
@Remote(MyRemoteInterface.class)  
public class MySessionBean implements MyRemoteInterface {  
    @Override  
    public String myRemoteMethod() {  
        return "Hello, world!";  
    }  
}
```



```
}
```

Теперь можно использовать этот сессионный бин удаленно. Для этого вам потребуется получить ссылку на бин, используя JNDI (Java Naming and Directory Interface). Это может быть сделано следующим образом:

```
InitialContext context = new InitialContext();  
  
MyRemoteInterface myBean = (MyRemoteInterface)  
context.lookup("java:global/myapp/MySessionBean");
```

После получения ссылки на бин, вы можете вызывать методы удаленного бизнес-интерфейса:

```
String result = myBean.myRemoteMethod();
```

Для доступа к удаленному бину, бин должен быть развернут на сервере приложений Java EE, таком как WildFly или GlassFish. Кроме того, вам потребуется настроить JNDI, чтобы клиент мог найти бин на сервере.

8. Жизненный цикл Stateless bean.

Stateless Bean - это бин, который не хранит состояние. Это означает, что каждый раз, когда бин используется, он создается заново и не сохраняет информацию о предыдущих использованиях. Это полезно для бинов, которые не требуют сохранения состояния между вызовами. Stateless Bean инициализируется только один раз во время инициализации ApplicationContext.

Жизненный цикл Stateless Bean:

- **Инициализация:** Stateless Bean инициализируется только один раз во время инициализации ApplicationContext. Это происходит, когда Spring контейнер создает экземпляр бина.
- **Использование:** Каждый раз, когда бин используется, он создается заново и не сохраняет информацию о предыдущих использованиях. Это полезно для бинов, которые не требуют сохранения состояния между вызовами.
- **Уничтожение:** Stateless Bean не уничтожается, когда ApplicationContext закрывается. Это связано с тем, что Stateless Bean не сохраняет состояние и не требует освобождения ресурсов.

В контексте Spring Framework, жизненный цикл бина начинается с этапа инициализации и заканчивается на этапе уничтожения. Во время этапа инициализации, бин создается и инициализируется. Во время этапа уничтожения, бин уничтожается и освобождаются все ресурсы, которые он использовал.

9. Жизненный цикл Stateful bean.

Stateful Bean - это бин, который хранит состояние. Это означает, что бин сохраняет информацию о предыдущих использованиях и может использовать эту информацию при последующих вызовах. Stateful Bean создается каждый раз, когда требуется объект (например, при использовании оператора «new» в Java)

Жизненный цикл Stateful Bean:

- **Инициализация:** Stateful Bean создается каждый раз, когда требуется объект (например, при использовании оператора «new» в Java). Это происходит, когда Spring контейнер создает экземпляр бина.
- **Использование:** Каждый раз, когда бин используется, он сохраняет информацию о предыдущих использованиях и может использовать эту информацию при последующих вызовах.
- **Уничтожение:** Stateful Bean уничтожается и освобождаются все ресурсы, которые он использовал, когда ApplicationContext закрывается. Это связано с тем, что Stateful Bean сохраняет состояние и требуется освобождение ресурсов.

10. Жизненный цикл Singleton bean.

Singleton Bean - это бин, который создается только один раз и затем повторно используется. Это означает, что для каждого запроса клиента используется один и тот же экземпляр бина. Singleton Bean инициализируется во время инициализации ApplicationContext и затем повторно используется в течение всего времени существования этого ApplicationContext.

Жизненный цикл Singleton Bean:

- **Инициализация:** Singleton Bean инициализируется только один раз во время инициализации ApplicationContext. Это происходит, когда Spring контейнер создает экземпляр бина.
- **Использование:** Singleton Bean используется повторно в течение всего времени существования ApplicationContext. Для каждого запроса клиента используется один и тот же экземпляр бина.
- **Уничтожение:** Singleton Bean уничтожается и освобождаются все ресурсы, которые он использовал, когда ApplicationContext закрывается. Это связано с тем, что Singleton Bean хранится в BeanFactory и требуется освобождение ресурсов.

11. Тестирование EJB.

Тестирование EJB (Enterprise Java Beans) включает в себя различные аспекты, включая модульное тестирование, интеграционное тестирование и тестирование на уровне пользовательского интерфейса.

Модульное тестирование используется для тестирования отдельных компонентов кода в изоляции. В контексте EJB, это может включать в себя тестирование бинов, которые не взаимодействуют с внешними ресурсами, такими как базы данных или внешние веб-службы. Для модульного тестирования EJB можно использовать JUnit в сочетании с библиотеками мокирования, такими как Mockito.

Интеграционное тестирование используется для тестирования взаимодействия между различными компонентами системы. В контексте EJB, это может включать в себя тестирование взаимодействия между различными бинами EJB и внешними ресурсами, такими как базы данных или внешние веб-службы. Для интеграционного тестирования EJB можно использовать Spring Boot Test .

Тестирование **на уровне пользовательского интерфейса (UI)** используется для тестирования взаимодействия пользователя с приложением. В контексте EJB, это может включать в себя тестирование взаимодействия пользователя с веб-интерфейсом, предоставляемым EJB. Для тестирования на уровне пользовательского интерфейса EJB можно использовать инструменты, такие как Selenium или Selenide.