

Санкт-Петербургский национальный исследовательский университет
информационных технологий, механики и оптики

Факультет инфокоммуникационных технологий

Направление подготовки 11.03.02

Лабораторная работа №2

«Основы объектно-ориентированного проектирования в Java»

Выполнил:

Швалов Даниил Андреевич

Группа: К33211

Проверил:

Иванов Сергей Евгеньевич

Санкт-Петербург

2024

Введение

Цель работы:

- создание приложений, реализующих различные операции над элементами массивов;
- получение навыков объектно-ориентированного анализа и проектирование классов для задач из различных предметных областей.

Ход работы

Упражнение 1. Использование аргументов командной строки и разбор массива args

В данном упражнении необходимо создать программу, в которой программа генерирует случайное число от 1 до 5, а пользователь пытается угадать это число. На рисунке 1 показан исходный код получившейся программы. На рисунке 2 приведен пример работы программы.

```
public class GuessingGame {  
    /**  
     * @param args the command line arguments  
     */  
    public static void main(String[] args) {  
        if (args.length == 0 || args[0].equals("help")) {  
            System.out.println("Enter argument n where 1 <= n <= 5");  
            return;  
        }  
  
        int guess = Integer.parseInt(args[0]);  
        if (guess < 1 || guess > 5) {  
            System.out.println("Argument n must be 1 <= n <= 5");  
            return;  
        }  
  
        int randomNum = (int) (Math.random() * 5 + 1);  
        if (guess == randomNum) {  
            System.out.println("You are right!");  
        } else {  
            System.out.println("Hidden number was " + randomNum + ". Try again");  
        }  
    }  
}
```

Рисунок 1 – Исходный код программы

```

classes > java guessinggame.GuessingGame 5
You are right!
classes > java guessinggame.GuessingGame 5
Hidden number was 1. Try again
classes > java guessinggame.GuessingGame 6
Argument n must be 1 <= n <= 5
classes > java guessinggame.GuessingGame 0
Argument n must be 1 <= n <= 5
classes > java guessinggame.GuessingGame help
Enter argument n where 1 <= n <= 5
classes > java guessinggame.GuessingGame
Enter argument n where 1 <= n <= 5

```

Рисунок 2 – Вывод программы

Упражнение 2. Создание пользовательского класса и включение комментариев в код

В данном упражнении необходимо спроектировать классы банковской системы. На рисунке 3 приведен исходный код класса «Account». Он представляет собой информацию об аккаунте и содержит информацию о клиенте. Для тестирования методов класса был написан код, показанный на рисунке 4. Вывод данной программы изображен на рисунке 5.

```

public class Account {

    private int customer;
    private double balance;

    public Account(int customer, int balance) {
        this.customer = customer;
        this.balance = balance;
    }

    public double getBalance() {
        return balance;
    }

    public void deposit(double count) {
        balance += count;
    }

    public void withdraw(double count) {
        balance -= count;
    }

    public int getCustomer() {
        return customer;
    }

    public String getDetails() {
        return "Client id: " + customer + ", balance: " + balance;
    }

}

```

Рисунок 3 – Исходный код класса «Account»

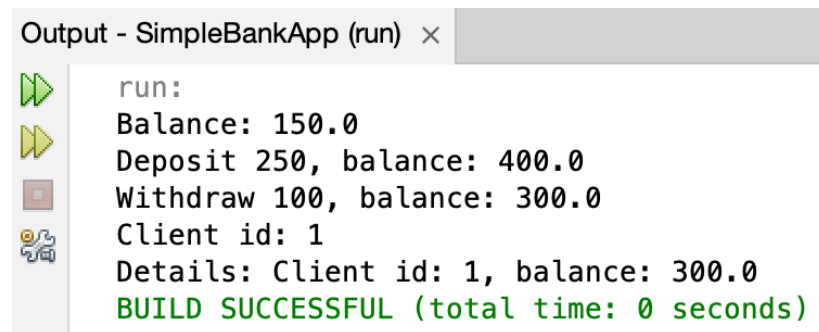
```

public class SimpleBankApp {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        Account account = new Account(1, 150);
        System.out.println("Balance: " + account.getBalance());
        account.deposit(250);
        System.out.println("Deposit 250, balance: " + account.getBalance());
        account.withdraw(100);
        System.out.println("Withdraw 100, balance: " + account.getBalance());
        System.out.println("Client id: " + account.getCustomer());
        System.out.println("Details: " + account.getDetails());
    }
}

```

Рисунок 4 – Код для тестирования класса «Account»



Output - SimpleBankApp (run) x

```

run:
Balance: 150.0
Deposit 250, balance: 400.0
Withdraw 100, balance: 300.0
Client id: 1
Details: Client id: 1, balance: 300.0
BUILD SUCCESSFUL (total time: 0 seconds)

```

Рисунок 5 – Вывод программы

Затем был добавлен класс «АТМ», представляющий банкомат. Он содержит метод для вывода информации о счете клиента. Исходный код класса «АТМ» приведен на рисунке 6. С помощью кода, показанного на рисунке 7, класс «АТМ» был протестирован. Вывод программы приведен на рисунке 8.

```

public class ATM {
    static public void out(Account account) {
        System.out.println("Details: " + account.getDetails());
    }
}

```

Рисунок 6 – Исходный код класса «АТМ»

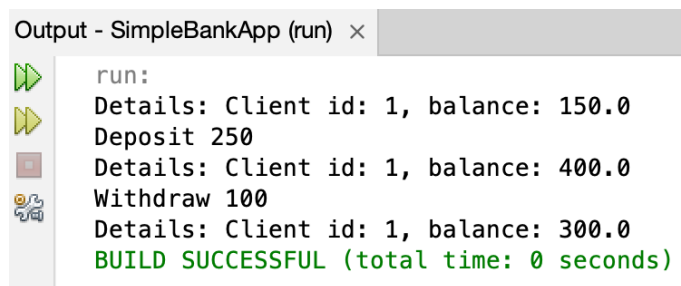
```

public class SimpleBankApp {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        Account account = new Account(1, 150);
        ATM.out(account);
        account.deposit(250);
        System.out.println("Deposit 250");
        ATM.out(account);
        account.withdraw(100);
        System.out.println("Withdraw 100");
        ATM.out(account);
    }
}

```

Рисунок 7 – Кол для тестирования класса «АТМ»



```

Output - SimpleBankApp (run) x
run:
Details: Client id: 1, balance: 150.0
Deposit 250
Details: Client id: 1, balance: 400.0
Withdraw 100
Details: Client id: 1, balance: 300.0
BUILD SUCCESSFUL (total time: 0 seconds)

```

Рисунок 8 – Вывод программы

После этого в класс «Account» были добавлены документирующие комментарии. К каждому методу было добавлено описание, что делает данный метод. Для тех методов, у которых есть параметры или возвращаемое значение, были дописаны дополнительные комментарии. На рисунке 9 приведен получившийся исходный код с комментариями.

```

public class Account {
    private int customer;
    private double balance;

    public Account(int customer, int balance) {
        this.customer = customer;
        this.balance = balance;
    }

    /**
     * Метод <em>getBalance</em> возвращает текущий баланс пользователя
     *
     * @return Текущий баланс пользователя
     */
    public double getBalance() {
        return balance;
    }

    /**
     * Метод <em>deposit</em> обеспечивает пополнение денег на счет
     *
     * @param count Количество пополняемых средств
     */
    public void deposit(double count) {
        balance += count;
    }

    /**
     * Метод <em>withdraw</em> обеспечивает снятие денег со счета
     *
     * @param count Количество снимаемых средств
     */
    public void withdraw(double count) {
        balance -= count;
    }

    /**
     * Метод <em>getCustomer</em> возвращает идентификатор пользователя
     *
     * @return Идентификатор пользователя
     */
    public int getCustomer() {
        return customer;
    }

    /**
     * Метод <em>getDetails</em> возвращает детальную информацию о
     * пользователе
     *
     * @return Детальная информация о пользователе
     */
    public String getDetails() {
        return "Client id: " + customer + ", balance: " + balance;
    }
}

```

Рисунок 9 – Исходный код класса «Account»

Затем был настроен javadoc, чтобы в сгенерированную документацию добавлялись авторы и версии. Для этого в параметры javadoc были добавлены флаги «-author» и «-version» (рисунок 10). После генерации документации была создана стра-

ница, показанная на рисунке 11. Как видно, на ней отображена та информация, которая была добавлена в комментарии.

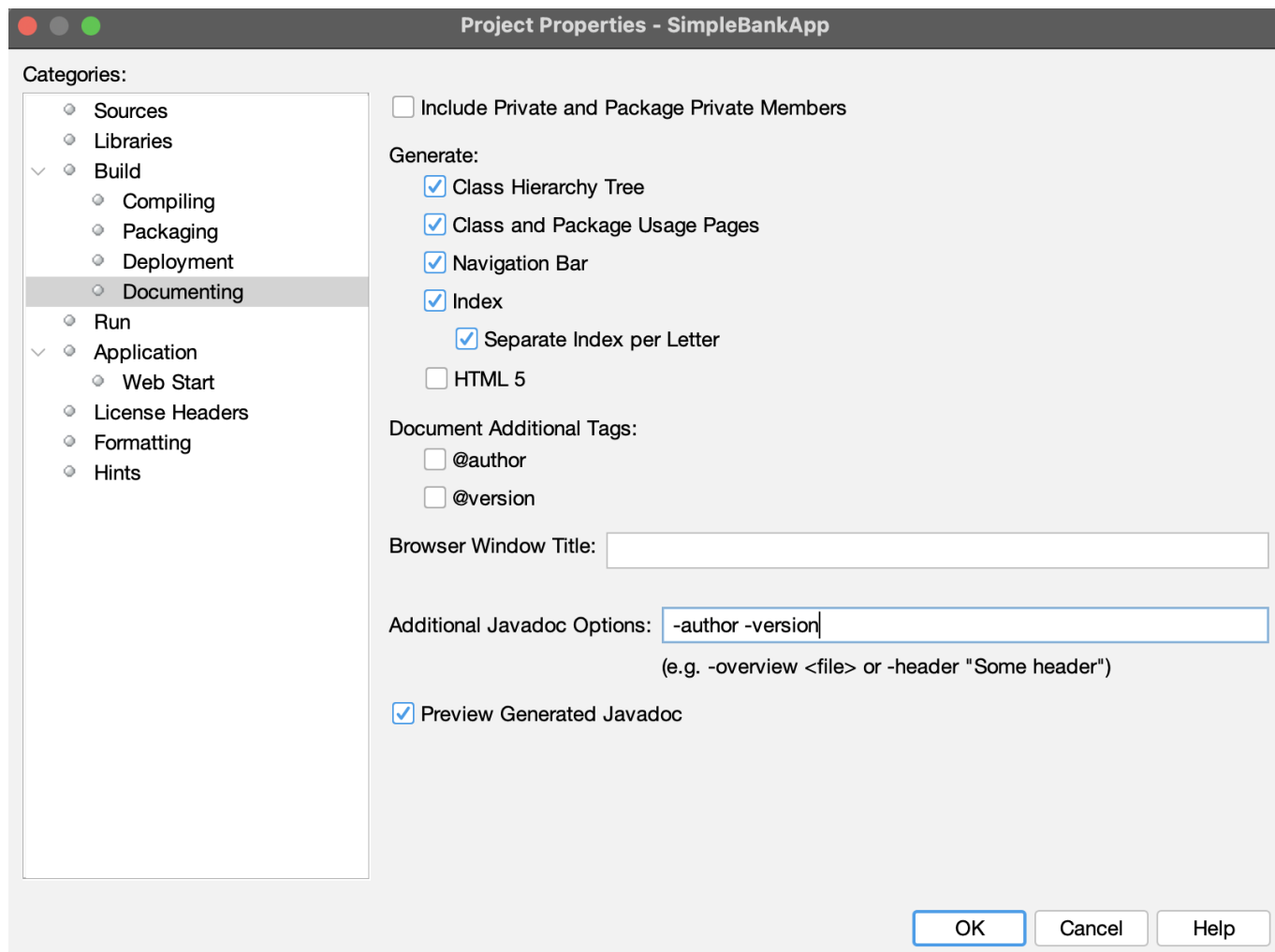


Рисунок 10 – Настройка javadoc

Package simplebankapp
Class Account
java.lang.Object
simplebankapp.Account

```
public class Account  
extends java.lang.Object
```

Объект класса Account имитирует банковский счет

Version:

0.0.1

Author:

Daniil Shvalov

Constructor Summary

Constructors

| Constructor | Description |
|---|-------------|
| <code>Account(int customer, int balance)</code> | |

Method Summary

All Methods Instance Methods Concrete Methods

| Modifier and Type | Method | Description |
|-------------------|-------------------------------------|--|
| void | <code>deposit(double count)</code> | Метод <i>deposit</i> обеспечивает пополнение денег на счет |
| double | <code>getBalance()</code> | Метод <i>getBalance</i> возвращает текущий баланс пользователя |
| int | <code>getCustomer()</code> | Метод <i>getCustomer</i> возвращает идентификатор пользователя |
| java.lang.String | <code>getDetails()</code> | Метод <i>getDetails</i> возвращает детальную информацию о пользователе |
| void | <code>withdraw(double count)</code> | Метод <i>withdraw</i> обеспечивает снятие денег со счета |

Methods inherited from class java.lang.Object

`clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Рисунок 11 – Сгенерированная документация

Упражнение 3. Отладка приложений в интегрированной среде разработки.

В данном упражнении необходимо научиться использовать отладчик. Для этого был создан класс «Shirt», при создании которого поля класса заполняются некоторыми значениями по умолчанию. Также в данный класс был добавлен метод для вывода информации о классе. Исходный код класса приведен на рисунке 12.

После этого был создан класс «ShirtTest», который тестирует работу класса «Shirt». Исходный код данного класса приведен на рисунке 13.


```

public class Shirt {

    private int shirtID;
    private String description;
    private char colorCode;
    private double price;
    private int quantityInStock;

    public Shirt() {
        this.shirtID = 0;
        this.description = "description";
        this.colorCode = 'U';
        this.price = 0;
        this.quantityInStock = 0;
    }

    public void displayShirtInformation() {
        System.out.println("Shirt ID: " + shirtID);
        System.out.println("Description: " + description);
        System.out.println("Color code: " + colorCode);
        System.out.println("Price: " + price);
        System.out.println("Quantity in stock: " + quantityInStock);
    }
}

```

Рисунок 12 – Исходный код класса «Shirt»

```

public class ShirtTest {

    public static void main(String[] args) {
        Shirt myShirt;
        myShirt = new Shirt();
        myShirt.displayShirtInformation();
    }
}

```

Рисунок 13 – Исходный код класса «ShirtTest»

Для тестирования работы программы на 15 строке была добавлена точка останова (рисунок 14). С помощью кнопки «Debug Project» (рисунок 15) была запущена отладка программы. После этого на экране появилась панель с информацией о значении переменных программы (рисунок 16).

```

11 public class ShirtTest {
12
13     public static void main(String[] args) {
14         Shirt myShirt;
15         myShirt = new Shirt();
16         myShirt.displayShirtInformation();
17     }
18 }
19
20

```

Рисунок 14 – Точка останова

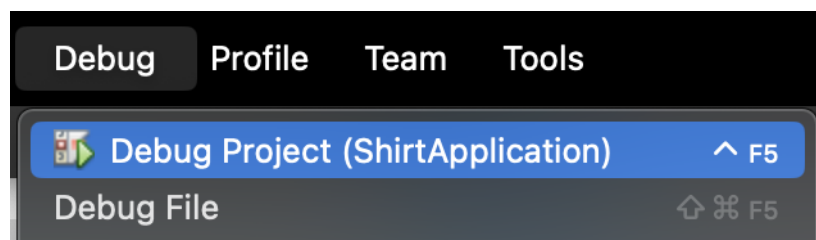


Рисунок 15 – Кнопка для запуска отладки программы

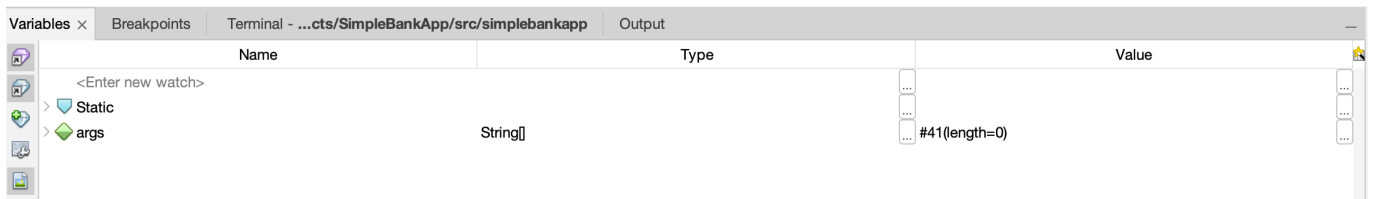


Рисунок 16 – Информация о переменных

После нажатия на кнопку «Step Over», на панели появились переменные класса «Shirt» (рисунок 17). Затем, после нажатия на кнопку «Step Into», на панели появились переменные метода «displayShirtInformation» (рисунок 18). Данные переменные были изменены так, как показано на рисунке 19. После продолжения выполнения программы, как видно на рисунке 20, были выведены те значения переменных, которые были указаны в отладчике, а не те, которые были изначально.

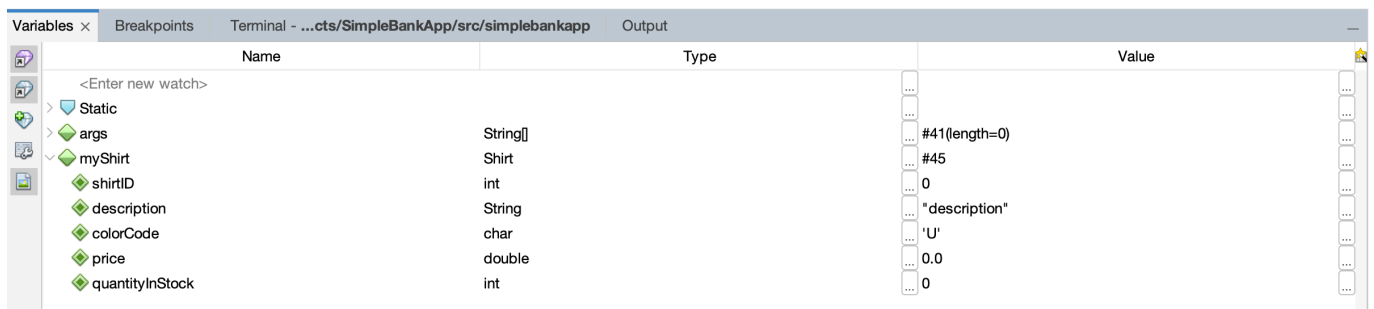


Рисунок 17 – Переменные класса «Shirt»

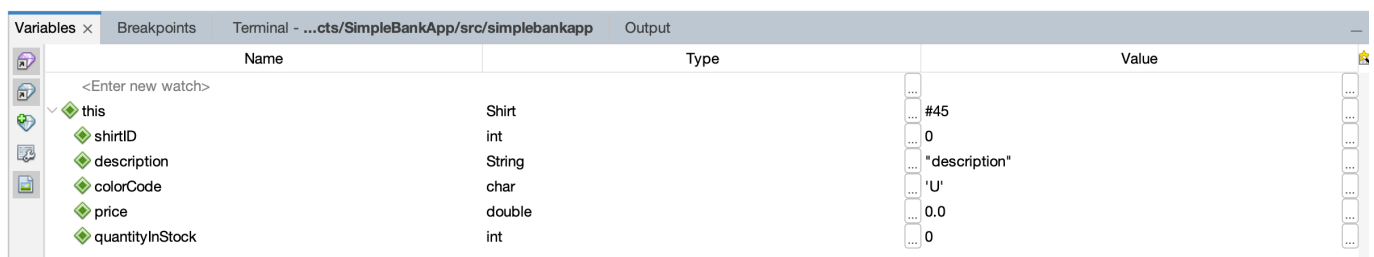


Рисунок 18 – Переменные метода «displayShirtInformation»

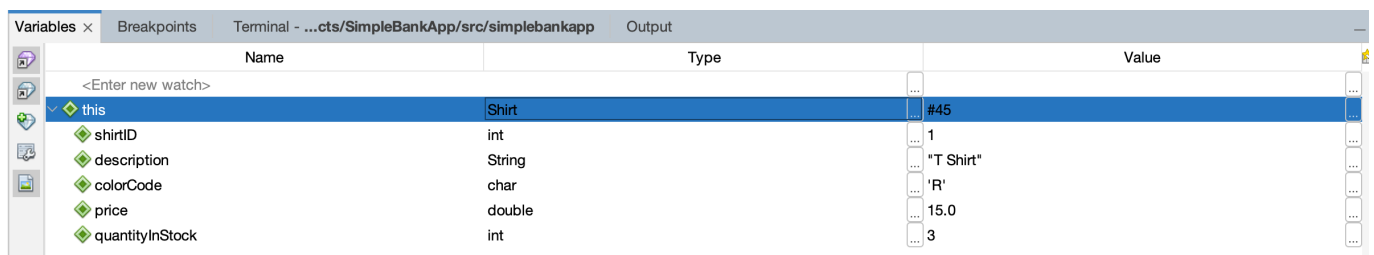


Рисунок 19 – Измененные переменные метода «displayShirtInformation»

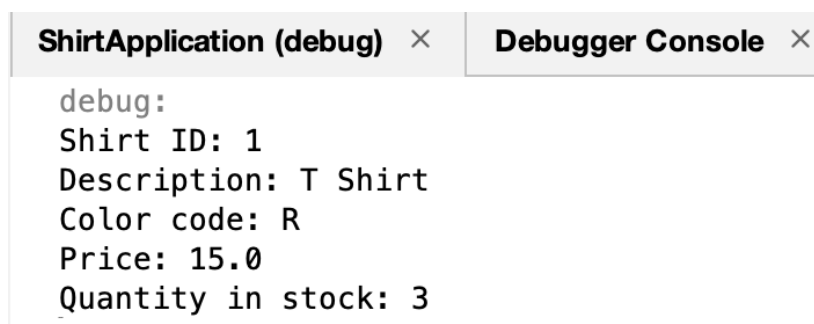


Рисунок 20 – Вывод программы

Заключение

В данной лабораторной работе была создана программа с использованием аргументов командной строки. Также были созданы несколько приложений с использованием объектно-ориентированного программирования.

Цель, поставленная в начале работы, достигнута, задачи выполнены.