МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное
образовательное учреждение высшего образования
«Самарский национальный исследовательский университет
имени академика С.П. Королева»
(Самарский университет)

ОТЧЕТ ПО
ЛАБОРАТОРНОЙ РАБОТЕ № 5

«Переопределение и использование стандартных функций класса Object»

по курсу
Объектно-ориентированное программирование

Выполнил: Петрухин Роман,
студент группы 6203-010302D

Оглавление

Пункт №1

Реализация задания №1 представлена на рисунках 1-4.

```
@Override  zhestok1
public String toString() {
    return "(" + this.x + "; " + this.y + ")";
}
```

Рисунок 1

```
@Override  zhestok1
public boolean equals(Object obj) {
    if (this == obj) return true; // Возвращаем true, если у нас тот же объект (т.к. проверяем ссылки)
    if (obj == null || this.getClass() != obj.getClass()) return false; // Разные типы => разные объекты

    return Double.compare(this.x, ((FunctionPoint) obj).x) == 0 // compare возвращает нуль, если координаты равны
            && Double.compare(this.y, ((FunctionPoint) obj).y) == 0; // в противном случае вернёт другую константу

}
```

Рисунок 2

Функция compare() возвращает нуль, если объекты равны, в другом случае - иные значения. Double - класс примитива double, удобен для использования в данном случае.

```
@Override  zhestok1
public int hashCode() {
    return (Double.hashCode(this.x) ^ Double.hashCode(this.y)); // бит = 1, если разные, =0, если одинаковые
}
```

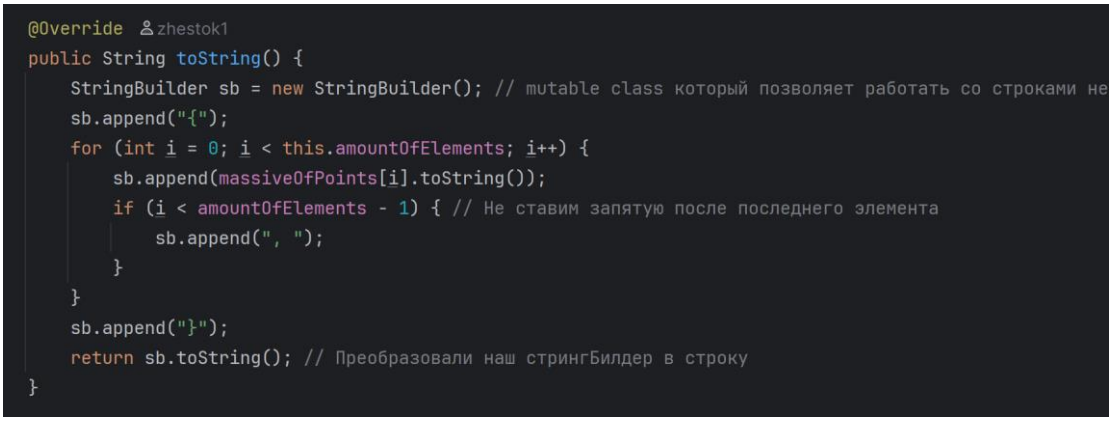
Рисунок 3

```
@Override  zhestok1
public Object clone() {
    return new FunctionPoint(this.x, this.y);
}
```

Рисунок 4

Пункт №2

Реализация пункт №2 представлена на рисунках 5-8.



```
@Override zhestok1
public String toString() {
    StringBuilder sb = new StringBuilder(); // mutable class который позволяет работать со строками не
    sb.append("{");
    for (int i = 0; i < this.amountOfElements; i++) {
        sb.append(massiveOfPoints[i].toString());
        if (i < amountOfElements - 1) { // Не ставим запятую после последнего элемента
            sb.append(", ");
        }
    }
    sb.append("}");
    return sb.toString(); // Преобразовали наш стрингБилдер в строку
}
```

Рисунок 5

В данном случае я использовал класс StringBuilder. Он очень удобен для создания строк, его плюс заключается в том, что он не требует создания отдельных «временных» объектов «String». Возвращаю объект типа стрингБилдер преобразованный в станартный «String».

```
@Override zhestok1
public boolean equals(Object obj) {
    if (this == obj) return true;
    if (obj == null || !(obj instanceof TabulatedFunction)) return false;

    // Общие проверки для любого TabulatedFunction
    if (this.getPointsCount() != ((TabulatedFunction) obj).getPointsCount()) return false;

    // Оптимизация для того же класса
    if (obj instanceof ArrayTabulatedFunction) {
        for (int i = 0; i < amountOfElements; i++) {
            if (!this.massiveOfPoints[i].equals(((ArrayTabulatedFunction) obj).massiveOfPoints[i]))
                return false;
        }
    } else {
        // Общий случай для других реализаций TabulatedFunction
        for (int i = 0; i < this.getPointsCount(); i++) {
            if (!this.getPoint(i).equals(((TabulatedFunction) obj).getPoint(i)))
                return false;
        }
    }
    return true;
}
```

Рисунок 6

Из нового - «instanceof» - оператор, который проверяет является ли объект экземпляром класса или нет. Вся остальная логика, как мне кажется, довольно стандартна.

```java
@Override  zhestok1
public int hashCode() {
    return Objects.hash((Object) massiveOfPoints);
}
```

Рисунок 7

Функция класса Objects, которая корректно хэширует массив.

```java
@Override  zhestok1
public Object clone() {
    try {
        return new ArrayTabulatedFunction(this.massiveOfPoints.clone());
    } catch (InappropriateFunctionPointException e) {
        throw new RuntimeException(e);
    }
}
```

Рисунок 8

Клонирую каждую точку массива и возвращаю новый объект «ArrayTabulatedFunction» из клонированных точек.

Пункт №3

Реализация пункта №3 представлена на рисунках 9-12.

```java
@Override  zhestok1
public String toString() {
    StringBuilder sb = new StringBuilder();
    sb.append("{");
    for (int i = 0; i < this.getPointsCount(); i++) {
        sb.append(this.getNodeByIndex(i).toString());
        if (i < this.getPointsCount() - 1) {
            sb.append(", ");
        }
    }
    sb.append("}");
    return sb.toString();
}
```

Рисунок 9

Реализация абсолютно аналогична, за исключением обращения к нужному узлу. Здесь это делается при помощи функции getNodeByIndex().

```
@Override  zhestok1
public boolean equals(Object obj) {
    if (this == obj) return true;
    if (obj == null || !(obj instanceof TabulatedFunction)) return false;

    // Общие проверки для любого TabulatedFunction
    if (this.getPointsCount() != ((TabulatedFunction) obj).getPointsCount()) return false;

    // Оптимизация для того же класса
    if (obj instanceof LinkedListTabulatedFunction) {
        for (int i = 0; i < getPointsCount(); i++) {
            if (!(this.getNodeByIndex(i).equals(((LinkedListTabulatedFunction) obj).getNodeByIndex(i))));
            return false;
        }
    } else {
        // Общий случай для других реализаций TabulatedFunction
        for (int i = 0; i < this.getPointsCount(); i++) {
            if (!this.getPoint(i).equals(((TabulatedFunction) obj).getPoint(i)))
                return false;
        }
    }
    return true;
}
```

Рисунок 10

Метод equals реализован идентично его аналогу из «ArrayTabukatedFunctions».

```
@Override  zhestok1
public int hashCode() {
    int hash = 17;
    FunctionNode curr = head.getNext();
    while (curr != head) {
        hash = 31 * hash + curr.getPoint().hashCode();
        curr = curr.getNext();
    }
    return hash;
}
```

Рисунок 11

Эту функцию я сделал в соответствии с тз. Первый элемент - количество, а дальше мы ксорим каждый элемент и получаем нужный нам хэш-код.

```java
@Override  zhestok1
public Object clone()  {
    FunctionPoint[] massOfPoints = new FunctionPoint[this.getPointsCount()];
    FunctionNode curr = head.getNext();

    for (int i = 0; curr != head; i++) {
        massOfPoints[i] = curr.getPoint();
        curr = curr.getNext();
    }

    try {
        return new LinkedListTabulatedFunction(massOfPoints);
    } catch (InappropriateFunctionPointException e) {
        throw new RuntimeException(e);
    }

}
```

Рисунок 12

Эта функция также сделана по тз. Мы «пересобираем» наш список, как и указано в тз. Возвращаем новый список, состоящий из «пересобранных» элементов.

Пункт №4

```java
public interface TabulatedFunction extends Function, Cloneable, Externalizable
```

Рисунок 13

Для того чтобы сделать все объекты типа TabulatedFunction были клонированы я просто сделал так чтобы TabulatedFunction. расширял Cloneable. Метод clone() соответственно тоже внесен в интерфейс.

Пункт №5

Код последних пунктов представлен на рисунках 14-16, а main 17-18.

```java
// Создаем тестовые данные
double[] testData = {0.0, 1.0, 4.0, 9.0, 16.0};

// Создаем объекты
ArrayTabulatedFunction attf1 = new ArrayTabulatedFunction( leftX: 0.0,  rightX: 4.0, testData);
ArrayTabulatedFunction attf2 = new ArrayTabulatedFunction( leftX: 0.0,  rightX: 4.0, testData);
LinkedListTabulatedFunction lttf1 = new LinkedListTabulatedFunction( leftX: 0.0,  rightX: 4.0, testData);
LinkedListTabulatedFunction lttf2 = new LinkedListTabulatedFunction( leftX: 0.0,  rightX: 4.0, testData);

// 1. Тест toString()
System.out.println("Тест toString() :");
System.out.println("Array: " + attf1.toString());
System.out.println("List: " + lttf1.toString());
System.out.println();
```

Рисунок 14

```
    // 2. Тест equals()
    System.out.println("Тест equals():");
    System.out.println("array1 == array2: " + attf1.equals(attf2));
    System.out.println("list1 == list2: " + lltf1.equals(lltf2));
    System.out.println("array1 == list1: " + attf1.equals(lltf1));
    System.out.println();

    // 3. Тест hashCode()
    System.out.println("Тест hashCode():");
    System.out.println("array1 hash: " + attf1.hashCode());
    System.out.println("array2 hash: " + attf2.hashCode());
    System.out.println("list1 hash: " + lltf1.hashCode());
    System.out.println("Согласованность: " + (attf1.hashCode() == attf2.hashCode()));
    System.out.println();

    // 4. Тест изменения хеша
    System.out.println("Изменение хеша:");
    int oldHash = lltf1.hashCode();
    FunctionPoint point = new FunctionPoint( x: 1,  y: 999);
    lltf1.setPoint( index: 1, point);// Меняем точку в lltf1
    int newHash = lltf1.hashCode();
    System.out.println("Старый хеш: " + oldHash + ", Новый хеш: " + newHash);
    System.out.println("Хеш изменился: " + (oldHash != newHash));
    System.out.println();
```

Рисунок 15

```
    // 5. Тест clone()
    System.out.println("Тест clone():");
    ArrayTabulatedFunction arrayClone = (ArrayTabulatedFunction) attf1.clone();
    LinkedListTabulatedFunction listClone = (LinkedListTabulatedFunction) lltf1.clone();

    System.out.println("array1 == clone: " + (attf1 == arrayClone));
    System.out.println("list1 == clone: " + (lltf1 == listClone));

    // Проверка глубокого копирования
    double originalY = attf1.getPoint( index: 0).getY();
    attf1.getPoint( index: 0).setY(999.0); // Меняем оригинал

    System.out.println("Глубокое копирование: " + (arrayClone.getPoint( index: 0).getY() != 999.0));
    System.out.println("Клон не изменился: " + (arrayClone.getPoint( index: 0).getY() == originalY));
}
```

Рисунок 16

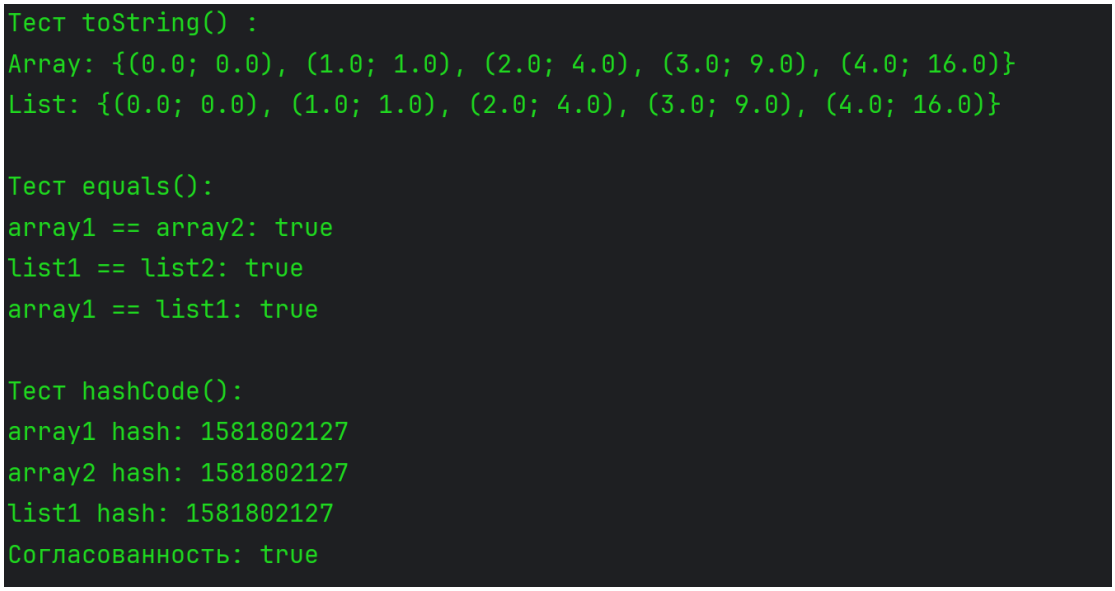


Рисунок 17

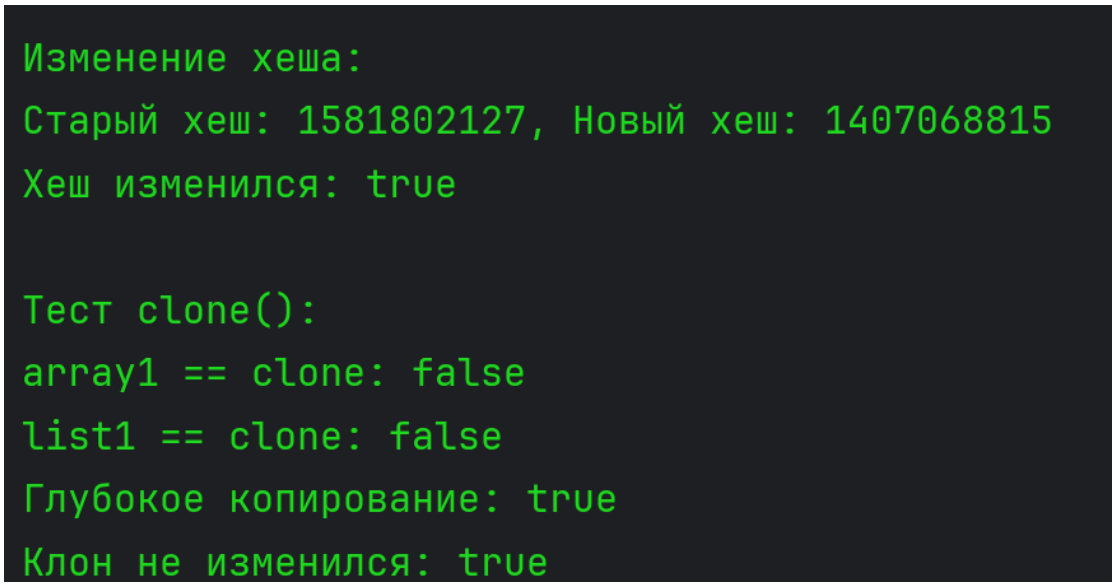


Рисунок 18

Список использованной литературы:

1) https://javarush.com/quests/lectures/questsyntaxpro.level12.lecture01 - Про классы обёртки примитивов.
2) https://javarush.com/quests/lectures/questmultithreading.level01.lecture07 - Про Cloneable
3) https://javarush.com/quests/lectures/questsyntaxpro.level09.lecture06 - Про StringBuilder