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 функций класса Object»

по курсу
Объектно-ориентированное программирование

Выполнил: Волынец Никита,
6203-010302D

Оглавление

Задание №1

Я переопределил методы в классе FunctionPoint. Для метода toString() я сделал возвращение строки в формате (x,y) с координатами точки. В методе equals() я предусмотрел проверку что переданный объект не null и того же класса, а также сравнение координат через Double.compare() для корректной работы с числами с плавающей точкой. Для hashCode() я преобразовал каждую координату double в long с помощью Double.doubleToLongBits(), затем разбил каждый long на два int и скомбинировал их через XOR. Метод clone() просто создает новую точку с теми же координатами.

```java
@Override
public String toString(){
    return "("+this.x+ "," + this.y + ")"; // возвращаем строку с координатами точки
}

@Override
public boolean equals(Object o){
    if( o == null || this.getClass() != o.getClass()){ // проверяем что объект не null и того же класса
        return false; // возвращаем false если классы разные
    }
    if(o == this){ // проверяем что это тот же объект
        return true; // возвращаем true если объекты одинаковые
    }
    return Double.compare(this.x, ((FunctionPoint) o).x) == 0 && Double.compare(this.y, ((FunctionPoint) o).y) == 0;
}

@Override
public int hashCode(){
    long xBit = Double.doubleToLongBits(this.x); // получаем битовое представление x
    long yBit = Double.doubleToLongBits(this.y); // получаем битовое представление y

    int x1 = (int)(xBit & 0xFFFFFFFF); // младшие 4 байта x
    int x2 = (int)(xBit >>> 32); // старшие 4 байта x
    int y1 = (int)(yBit & 0xFFFFFFFF); // младшие 4 байта y
    int y2 = (int)(yBit >>> 32); // старшие 4 байта y

    return x1^x2^y1^y2; // комбинируем все части через xor
}

@Override
public Object clone(){
    return new FunctionPoint(this.x, this.y); // создаем новую точку с теми же координатами
}
```

Задание №2

Для ArrayTabulatedFunction я реализовал toString() который формирует строку в формате {(x1;y1), (x2;y2), ...} перебирая массив точек.

```
@Override
public String toString(){
    String res = "{"; // начинаем строку с фигурной скобки
    for (int i = 0; i < point_count; i++){ // проходим по всем точкам массива
        res+= "("+ point_mass[i].getX() + ";" + point_mass[i].getY() + ")"; //
        if(i<point_count -1){ // если это не последняя точка
            res+=", "; // добавляем запятую и пробел
        }
    }
    res += "}"; // закрываем фигурную скобку
    return res; // возвращаем результат
}
```

В equals() я сделал быструю проверку для ArrayTabulatedFunction через прямой доступ к массивам и общую проверку для других TabulatedFunction через методы интерфейса.

```
@Override
public boolean equals(Object o){
    if (this == o) return true; // если это тот же объект возвращаем true
    if (o == null || !(o instanceof TabulatedFunction)) return false; // если объект null или не табулированная функци

    if (o instanceof ArrayTabulatedFunction) { // если объект тоже array tabulated function
        if (this.point_count != ((ArrayTabulatedFunction) o).point_count) { // сравниваем количество точек
            return false; // если разное количество точек возвращаем false
        }
        for (int i = 0; i < point_count; i++) { // проходим по всем точкам массива
            if (Double.compare(this.point_mass[i].getX(), ((ArrayTabulatedFunction) o).point_mass[i].getX()) != 0 || D
                return false; // если координаты не равны возвращаем false
            }
        }
    }
    else { // если объект другой реализации tabulated function
        if (this.getPointsCount() != ((TabulatedFunction) o).getPointsCount()) { // сравниваем количество точек через
            return false; // если разное количество точек возвращаем false
        }
        for (int i = 0; i < point_count; i++) { // проходим по всем точкам
            if (Double.compare(this.getPointX(i), ((TabulatedFunction) o).getPointX(i)) != 0 || Double.compare(this.ge
                return false; // если координаты не равны возвращаем false
            }
        }
    }

    return true; // если все проверки пройдены возвращаем true
}
```

HashCode() комбинирует количество точек и хэш-коды всех точек через XOR.

```java
@Override
public int hashCode(){
    int hash = point_count; // начинаем с количества точек

    for (int i = 0; i < point_count; i++) { // проходим по всем точкам массива
        hash ^= point_mass[i].hashCode(); // комбинируем хэш точки через xor
    }
    return hash; // возвращаем результат
}
```

Clone() создает глубокую копию через клонирование всех точек и создание нового массива.

```java
@Override
public Object clone(){
    FunctionPoint[] points_copy = new FunctionPoint[point_count]; // создаем новый массив для точек
    for(int i =0; i < point_count; i++){ // проходим по всем точкам исходного массива
        points_copy[i] = (FunctionPoint)point_mass[i].clone(); // клонируем каждую точку
    }
    return new ArrayTabulatedFunction(points_copy); // создаем новый объект с скопированными точками
}
```

Задание №3

В LinkedListTabulatedFunction toString() проходит по узлам списка через getNodeByIndex() и формирует строковое представление.

```java
@Override
public String toString(){
    String res = "{"; // начинаем строку с фигурной скобки
    for (int i = 0; i < pointcount; i++){ // проходим по всем точка
        res+= this.getNodeByIndex(i).getPoint().toString(); // доба
        if(i<pointcount -1){ // если это не последняя точка
            res+=", "; // добавляем запятую и пробел
        }
    }
    res += "}"; // закрываем фигурную скобку
    return res; // возвращаем результат
}
```

Equals() имеет быструю проверку для LinkedListTabulatedFunction через прямое сравнение узлов и общую проверку для других реализаций.

```java
@Override
public boolean equals(Object o){
    if (this == o) return true; // если это тот же объект возвращаем true
    if (o == null || !(o instanceof TabulatedFunction)) return false; // если объект null или не табулированная

    if (o instanceof LinkedListTabulatedFunction) { // если объект тоже linked list
        LinkedListTabulatedFunction other = (LinkedListTabulatedFunction) o; // приводим к linked list
        if (this.pointcount != ((LinkedListTabulatedFunction) o).pointcount) { // сравниваем количество точек
            return false; // если разное количество точек возвращаем false
        }
        FunctionNode curThis = this.head.getNext(); // начинаем с первого узла текущего списка
        FunctionNode curOther = other.head.getNext(); // начинаем с первого узла другого списка

        while (curThis != this.head) { // пока не вернемся к голове
            if (Double.compare(curThis.getPoint().getX(), curOther.getPoint().getX()) != 0 || Double.compare(cur
                return false; // если координаты не равны возвращаем false
            }
            curThis = curThis.getNext(); // переходим к следующему узлу текущего списка
            curOther = curOther.getNext(); // переходим к следующему узлу другого списка
        }
    }
    else { // если объект другой реализации tabulated function
        if (this.getPointsCount() != ((TabulatedFunction) o).getPointsCount()) { // сравниваем количество точек
            return false; // если разное количество точек возвращаем false
        }
        for (int i = 0; i < pointcount; i++) { // проходим по всем точкам
            if (Double.compare(this.getPointX(i), ((TabulatedFunction) o).getPointX(i)) != 0 || Double.compare(t
                return false; // если координаты не равны возвращаем false
            }
        }
    }

    return true; // если все проверки пройдены возвращаем true
}
```

HashCode() обходит узлы списка и комбинирует их хэш-коды.

```java
@Override
public int hashCode(){
    int hash = pointcount; // начинаем с количес

    FunctionNode cur = head.getNext(); // начина
    while(cur!=head) { // пока не вернемся к гол
        hash ^= cur.getPoint().hashCode(); // ко
        cur = cur.getNext(); // переходим к след
    }
    return hash; // возвращаем результат
}
```

Clone() создает массив копий точек и использует конструктор для создания нового списка что эффективнее клонирования узлов.

```java
@Override
public Object clone(){
    FunctionPoint[] point_mass = new FunctionPoint[pointcount]; // с
    FunctionNode cur = head.getNext(); // начинаем с первого узла
    int i =0; // индекс для массива
    while(cur != head){ // пока не вернемся к голове
        point_mass[i] = (FunctionPoint)cur.getPoint().clone(); // кл
        cur = cur.getNext(); // переходим к следующему узлу
        i++; // увеличиваем индекс
    }
    return new LinkedListTabulatedFunction(point_mass); // создаем н
}
```

Задание №4

Я добавил наследование от Cloneable в интерфейс TabulatedFunction и объявил метод clone() что делает все реализации клонируемыми с точки зрения JVM.

Задание №5

```java
import functions.*;

public class Main {
    public static void main(String[] args) {
        // создаем тестовые точки
        FunctionPoint[] points1 = {new FunctionPoint(0.0, 1.0), new
FunctionPoint(1.0, 3.0), new FunctionPoint(2.0, 5.0)};
        FunctionPoint[] points2 = {new FunctionPoint(0.0, 1.0), new
FunctionPoint(1.0, 3.0), new FunctionPoint(2.0, 5.0)};
        FunctionPoint[] points3 = {new FunctionPoint(0.0, 1.0), new
FunctionPoint(1.0, 3.5),new FunctionPoint(2.0, 5.0)}; // отличается y

        // тестирование toString()
        System.out.println("=== toString() ===");
        ArrayTabulatedFunction arrayFunc1 = new
ArrayTabulatedFunction(points1);
        LinkedListTabulatedFunction listFunc1 = new
LinkedListTabulatedFunction(points1);

        System.out.println("Array: " + arrayFunc1.toString());
        System.out.println("LinkedList: " + listFunc1.toString());
        System.out.println();

        // тестирование equals()
        System.out.println("=== equals() ===");
        ArrayTabulatedFunction arrayFunc2 = new
ArrayTabulatedFunction(points2);
        LinkedListTabulatedFunction listFunc2 = new
LinkedListTabulatedFunction(points2);
        ArrayTabulatedFunction arrayFunc3 = new
ArrayTabulatedFunction(points3);

        System.out.println("array1 == array2: " +
arrayFunc1.equals(arrayFunc2)); // true
```

```
        System.out.println("array1 == list1: " +
arrayFunc1.equals(listFunc1));   // true
        System.out.println("array1 == array3: " +
arrayFunc1.equals(arrayFunc3)); // false
        System.out.println("list1 == list2: " +
listFunc1.equals(listFunc2));     // true
        System.out.println();

        // тестирование hashCode()
        System.out.println("=== hashCode() ===");
        System.out.println("array1 hashCode: " +
arrayFunc1.hashCode());
        System.out.println("array2 hashCode: " +
arrayFunc2.hashCode());
        System.out.println("list1 hashCode: " + listFunc1.hashCode());
        System.out.println("list2 hashCode: " + listFunc2.hashCode());
        System.out.println("array3 hashCode: " +
arrayFunc3.hashCode());

        // проверка согласованности equals и hashCode
        System.out.println("hashCode equals check:");
        System.out.println("array1.hash == array2.hash: " +
(arrayFunc1.hashCode() == arrayFunc2.hashCode()));
        System.out.println("array1.hash == list1.hash: " +
(arrayFunc1.hashCode() == listFunc1.hashCode()));
        System.out.println();

        // изменение объекта и проверка изменения хэша
        System.out.println("=== hashCode() after modification ===");
        int originalHash = arrayFunc1.hashCode();
        arrayFunc1.setPointY(1, 3.001); // незначительное изменение
        int modifiedHash = arrayFunc1.hashCode();
        System.out.println("Original hash: " + originalHash);
        System.out.println("Modified hash: " + modifiedHash);
        System.out.println("Hash changed: " + (originalHash !=
modifiedHash));
        System.out.println();

        // тестирование clone()
        System.out.println("=== clone() ===");
        ArrayTabulatedFunction arrayClone = (ArrayTabulatedFunction)
arrayFunc1.clone();
        LinkedListTabulatedFunction listClone =
(LinkedListTabulatedFunction) listFunc1.clone();

        System.out.println("Array original: " +
arrayFunc1.toString());
        System.out.println("Array clone: " + arrayClone.toString());
        System.out.println("LinkedList original: " +
listFunc1.toString());
        System.out.println("LinkedList clone: " +
listClone.toString());

        // проверка глубокого клонирования
        System.out.println("=== Deep clone test ===");
        arrayFunc1.setPointY(0, 999.0); // изменяем оригинал
        listFunc1.setPointY(0, 888.0);  // изменяем оригинал

        System.out.println("After modification:");
        System.out.println("Array original: " +
arrayFunc1.toString());
        System.out.println("Array clone: " + arrayClone.toString());
// не должен измениться
```

```
        System.out.println("LinkedList original: " +
listFunc1.toString());
        System.out.println("LinkedList clone: " +
listClone.toString()); // не должен измениться

        System.out.println("Array clone unchanged: " +
(arrayClone.getPointY(0) != 999.0));
        System.out.println("LinkedList clone unchanged: " +
(listClone.getPointY(0) != 888.0));

    }
}
```

```
=== toString() ===
Array: {(0.0;1.0), (1.0;3.0), (2.0;5.0)}
LinkedList: {(0.0,1.0), (1.0,3.0), (2.0,5.0)}

=== equals() ===
array1 == array2: true
array1 == list1: true
array1 == array3: false
list1 == list2: true

=== hashCode() ===
array1 hashCode: 1075576835
array2 hashCode: 1075576835
list1 hashCode: 1075576835
list2 hashCode: 1075576835
array3 hashCode: 1075314691
hashCode equals check:
array1.hash == array2.hash: true
array1.hash == list1.hash: true

=== hashCode() after modification ===
Original hash: 1075576835
Modified hash: 161897530
Hash changed: true

=== clone() ===
Array original: {(0.0;1.0), (1.0;3.001), (2.0;5.0)}
Array clone: {(0.0;1.0), (1.0;3.001), (2.0;5.0)}
LinkedList original: {(0.0,1.0), (1.0,3.0), (2.0,5.0)}
LinkedList clone: {(0.0,1.0), (1.0,3.0), (2.0,5.0)}
=== Deep clone test ===
After modification:
Array original: {(0.0;500.0), (1.0;3.001), (2.0;5.0)}
Array clone: {(0.0;1.0), (1.0;3.001), (2.0;5.0)}
LinkedList original: {(0.0,676.0), (1.0,3.0), (2.0,5.0)}
LinkedList clone: {(0.0,1.0), (1.0,3.0), (2.0,5.0)}
Array clone unchanged: true
LinkedList clone unchanged: true
```