

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

«Расширение возможностей классов»

по курсу
Объектно-ориентированное программирование

Выполнила: Элибекян Анжелина,
студент группы 6203-010302D

Оглавление

Содержание

<u>Задание №1</u>	<u>2</u>
<u>Задание №2</u>	<u>3</u>
<u>Задание №3</u>	<u>5</u>
<u>Задание №4.....</u>	<u>8</u>
<u>Задание №5</u>	<u>8</u>

Задание №1

В данном задании в классе FunctionPoint я переопределила методы : String toString() - класс должен возвращать текстовое описание точки; boolean equals(Object o) - класс должен возвращать true, если переданный объект также является точкой и его координаты совпадают с координатами объекта, метод которого вызываем; int hashCode() - должен возвращать значение хэш-кода для объекта точки; Object clone() - должен возвращать объект-копию для объекта точки.

```
//переопределение метода для текстового описания точки
@Override
public String toString() {
    return "(" + this.x + ";" + this.y + ")";
}

//переопределение метода для сравнения с текущей точкой
@Override
public boolean equals(Object o) {
    //проверка на ссылочное равенство
    if (this == o) return true;
    //проверка на совместимость классов
    if (o == null || getClass() != o.getClass()) return false;

    //сравниваем координаты x и y для корректного сравнения double
    return Double.compare(this.x, ((FunctionPoint) o).x) == 0 && Double.compare(this.y, ((FunctionPoint) o).y) == 0
}

//переопределение метода для вычисления хэш-кода точки
@Override
public int hashCode() {
    //преобразование double в long
    long x_bits = Double.doubleToLongBits(this.x);
    long y_bits = Double.doubleToLongBits(this.y);
    //разбиваем 64бит long на два 32бит int
    // ^ операция XOR между старшими и младшими 32 битами
    int x_hash = (int) (x_bits ^ (x_bits >> 32));
    int y_hash = (int) (y_bits ^ (y_bits >> 32));
    //объединяем хэш координат через XOR
    return x_hash ^ y_hash;
}
```

```

// переопределение метода для создания копии точки
@Override
public Object clone() {
    // создание нового объекта с теми же координатами
    return new FunctionPoint(this.x, this.y);
}

```

Задание №2

Переопределяю методы в классе ArrayTabulatedFunction.

```

@Override
public String toString(){
    //создаем строку с открывающей скобкой
    String result = "{";
    //проходим по всем точкам
    for (int i = 0; i < count; i++){
        result += "(" + points[i].getX() + "; " + points[i].getY() + ")";
        //если точка не последняя, тогда добавляем запятую и пробел
        if (i < count - 1){
            result += ", ";
        }
    }
    //закрываем скобку и возвращаем результат
    return result + "}";
}

// сравниваем табулированную ф-ию с другим объектом на равенство
@Override
public boolean equals(Object o){
    // если объект тот же - возвращаем true
    if (this == o) return true;
    // если наш объект не табулированная ф-ия возвращаем false
    if (!(o instanceof TabulatedFunction)) return false;
    if (o instanceof ArrayTabulatedFunction) { // если объект тот же
        ArrayTabulatedFunction other = (ArrayTabulatedFunction) o;
        if (this.count != other.count){ // сравниваем кол-во точек
            return false;
        }
        for (int i = 0; i < count; i++){ // проходим по всем точкам массива
            if (!this.points[i].equals(other.points[i])){ // сравниваем точки с помощью equals
                return false;
            }
        }
    }
    else { // если объект другой реализации TabulatedFunction

```

```

    }

    else { // если объект другой реализации TabulatedFunction
        TabulatedFunction other = (TabulatedFunction) o;
        if (this.getPointsCount() != other.getPointsCount()) { // сравниваем кол-
            return false; // если разное количество точек возвращаем false
        }

        for (int i = 0; i < count; i++) { // проходим по всем точкам
            FunctionPoint thisPoint = this.getPoint(i); // получаем точку текуще
            FunctionPoint otherPoint = other.getPoint(i); // получаем точку друг
            if (!thisPoint.equals(otherPoint)) { // сравниваем точки
                return false; // если точки не равны возвращаем false
            }
        }
    }

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

@Override
public int hashCode() {
    int hash = count; // кол-во точек
    for (int i = 0; i < count; i++) {
        //вычисляем хэш для каждой точки и объединяем через XOR
        hash ^= points[i].hashCode();
    }
    return hash;
}
@Override
public Object clone(){
    //создаем временный массив для копий точек
    FunctionPoint[] copiedPoints = new FunctionPoint[count];
    //выполняем глубокое копирование: создаем новые объекты точек
    for (int i = 0; i < count; i++) {
        //создаем копию каждой точки через конструктор копирования
        copiedPoints[i] = new FunctionPoint(points[i]);
    }
    //создаем и возвращаем новый объект через конструктор
    return new ArrayTabulatedFunction(copiedPoints);
}

```

Задание №3

Аналогично переопределяю методы в классе LinkedListTabulatedFunction. Метод `toString()` возвращает строковое представление функции в формате `{(x1; y1), (x2; y2), ...}` путем обхода всех узлов списка и сборки строки с координатами точек. Метод `equals()` сравнивает функцию с другим объектом на равенство: если объект тоже `LinkedListTabulatedFunction`, происходит прямое сравнение узлов списка, если другой тип `TabulatedFunction` - сравнение через `getPoint()`, возвращает `true` только если все точки идентичны. Метод `hashCode()` вычисляет хэш-код функции путем комбинирования через XOR количества точек и хэш-кодов всех отдельных точек. Метод `clone` создает глубокую копию списка копируя все точки в массив и передавая его в конструктор нового `LinkedListTabulatedFunction` обеспечивая независимую копию с новыми объектами точек.

```
@Override
public String toString(){
    // создаем строку с открывающей скобкой
    String result = "{";
    // проходим по всем точкам
    FunctionNode current = head.next;
    while (current != head){
        result += "(" + current.point.getX() + "; " + current.point.getY() + ")";
        // если есть следующая точка (не голова), тогда добавляем запятую и пробел
        if (current.next != head){
            result += ", ";
        }
        current = current.next;
    }
    // закрываем скобку и возвращаем результат
    return result + "}";
}

// сравниваем табулированную ф-ию с другим объектом на равенство
@Override
public boolean equals(Object o){
    // если объект тот же - возвращаем true
    if (this == o) return true;
    // если наш объект не табулированная ф-ия возвращаем false
    if (!(o instanceof TabulatedFunction)) return false;

    TabulatedFunction other = (TabulatedFunction) o;

    if (o instanceof LinkedListTabulatedFunction) { // если объект тоже LinkedListTabulatedFunction
        LinkedListTabulatedFunction otherList = (LinkedListTabulatedFunction) o;
        if (this.count != otherList.count){ // сравниваем кол-во точек
            return false;
        }
    }
}
```

```

        return false;
    }
    //проходим по спискам
    FunctionNode thisNode = this.head.next;
    FunctionNode otherNode = otherList.head.next;
    while (thisNode != this.head){ // проходим по всем точкам списка
        if (!thisNode.point.equals(otherNode.point)){ // сравниваем
            return false;
        }
        //переходим к следующим точкам в обоих списках
        thisNode = thisNode.next;
        otherNode = otherNode.next;
    }
}
else { // если объект другой реализации TabulatedFunction
    if (this.getPointsCount() != other.getPointsCount()) { // сравниваем количество точек
        return false; // если разное количество точек возвращаем false
    }
    FunctionNode currentNode = this.head.next;
    int index = 0; //индекс для доступа к точкам другой функции
    while (currentNode != this.head) { // проходим по всем точкам
        FunctionPoint thisPoint = this.getPoint(index); // получаем точку
        FunctionPoint otherPoint = other.getPoint(index); // получаем точку
        if (!thisPoint.equals(otherPoint)) { // сравниваем точки
            return false; // если точки не равны возвращаем false
        }
        currentNode = currentNode.next;
        index

```

```

@Override
public int hashCode() {
    // начинаем с количества точек, чтобы отличать функции с разным
    int hash = count;
    // начинаем обход списка с первой реальной точки (после головы)
    FunctionNode current = head.next;
    // проходим по всем точкам
    while (current != head) {
        // вычисляем хэш для каждой точки и объединяем через операци
        hash ^= current.point.hashCode();
        // переходим к следующей точке в списке
        current = current.next;
    }
    // возвращаем итоговый хэш-код
    return hash;
}

@Override
public Object clone(){
    // создаем временный массив для хранения копий всех точек
    FunctionPoint[] copiedPoints = new FunctionPoint[count];
    // начинаем обход списка с первой реальной точки
    FunctionNode current = head.next;
    int index = 0;
    // проходим по всем точкам исходного списка
    while (current != head) {
        // создаем глубокую копию каждой точки через конструктор ко
        copiedPoints[index] = new FunctionPoint(current.point);
        // переходим к следующей точке
        current = current.next;
        index++;
    }
    // создаем новый объект функции через конструктор, передавая ск
    return new LinkedListTabulatedFunction(copiedPoints);
}

```

Задание №4

В четвертом задании я сделала интерфейс TabulatedFunction клонируемым добавив в него метод clone и унаследовав от Cloneable чтобы все реализации могли создавать копии через единый интерфейс независимо от их внутреннего устройства.

```
/* расширяем базовый интерфейс Function дополнительными методами */
public interface TabulatedFunction extends Function, Cloneable { 13 usages 2 implementations

    // возвращаем количество точек табуляции
    int getPointsCount(); 8 usages 2 implementations

    // получаем точку по указанному индексу
    FunctionPoint getPoint(int index); 4 usages 2 implementations

    // устанавливаем новую точку по указанному индексу
    void setPoint(int index, FunctionPoint point) throws InappropriateFunctionPointException

    // получаем координату x точки по указанному индексу
    double getPointX(int index); 3 usages 2 implementations

    // устанавливаем координату x точки по указанному индексу
    void setPointX(int index, double x) throws InappropriateFunctionPointException

    // получаем координату y точки по указанному индексу
    double getPointY(int index); 8 usages 2 implementations

    // устанавливаем координату y точки по указанному индексу
    void setPointY(int index, double y); 4 usages 2 implementations

    // удаляем точку по указанному индексу
    void deletePoint(int index); no usages 2 implementations

    // добавляем новую точку в табулированную функцию
    void addPoint(FunctionPoint point) throws InappropriateFunctionPointException

    Object clone(); 2 implementations
```

Задание №5

Проверка работы написанных методов. Результаты прикрепляю ниже:

```
==== Тестирование методов ArrayTabulatedFunction и LinkedListTabulatedFunction ===

1. Метод toString():
ArrayTabulatedFunction: {(0.0; 1.0), (1.0; 3.0), (2.0; 5.0), (3.0; 7.0)}
LinkedListTabulatedFunction: {(0.0; 1.0), (1.0; 3.0), (2.0; 5.0), (3.0; 7.0)}

2. Метод equals():
arrayFunc.equals(arrayFunc2): true
listFunc.equals(listFunc2): true
arrayFunc.equals(listFunc): true
arrayFunc.equals(differentArrayFunc): false

3. Метод hashCode():
arrayFunc.hashCode(): 1074266116
arrayFunc2.hashCode(): 1074266116
listFunc.hashCode(): 1074266116
listFunc2.hashCode(): 1074266116
differentArrayFunc.hashCode(): 1075052547
arrayFunc.equals(arrayFunc2) && arrayFunc.hashCode() == arrayFunc2.hashCode(): true

4. Изменение объекта и хэш-кода:
Исходный hashCode arrayFunc: 1074266116
После изменения Y[1] на 3.001: 162683965

5. Метод clone():
arrayFunc == arrayClone: false
arrayFunc.equals(arrayClone): true
listFunc == listClone: false
listFunc.equals(listClone): true
6. Проверка глубокого клонирования:
После изменения оригиналов:
arrayFunc.getPointY(0): 100.0
arrayClone.getPointY(0): 1.0
listFunc.getPointY(0): 200.0
listClone.getPointY(0): 1.0
Клоны не изменились: true
```