

Лабораторная 5

Выполнил: Гонеев Роман

Группа: 6201-120303D

Задания:

1. Переопределил в классе FunctionPoint следующие методы:

- **String toString()** - возвращает текстовое описание точки:

```
@Override
public String toString(){
    return String.format("(%s; %s)", x, y);
}
```

- **boolean equals(Object o)** - возвращает true тогда и только тогда, когда переданный объект также является точкой и его координаты в точности совпадают с координатами объекта, у которого вызывается метод:

```
@Override
public boolean equals(Object o){
    if (this == o) return true; // Сравниваем адреса в памяти
    if (o == null || getClass() != o.getClass()) return false; // Если пришел null
    или другого класс, то false
```

```
    FunctionPoint that = (FunctionPoint) o; // Приводим к типу
    FunctionPoint
    // Копируем биты double в long
    return Double.doubleToLongBits(x) == Double.doubleToLongBits(that.x)
        && Double.doubleToLongBits(y) == Double.doubleToLongBits(that.y);
}
```

- **int hashCode()** - возвращает значение хэш-кода для объекта точки:

```
@Override
public int hashCode(){
    long xBits = Double.doubleToLongBits(x); // Копирование 64 битов
    double в long
    long yBits = Double.doubleToLongBits(y); // Копирование 64 битов
    double в long
```

```
    int x_hash = (int) (xBits ^ (xBits >>> 32));
    int y_hash = (int) (yBits ^ (yBits >>> 32));
```

```
    return (x_hash * 3) + y_hash;
```

```
}
```

- Метод clone():

@Override

```
public Object clone() throws CloneNotSupportedException{  
    return super.clone();  
}
```

2. Переопределил в классе ArrayTabulatedFunction следующие методы:

- **String toString()** - возвращает описание табулированной функции:

```
public String toString() {  
    StringBuilder sb = new StringBuilder("{}");  
    for (int i = 0; i < points.length; i++) {  
        sb.append(points[i].toString());  
        if (i < points.length - 1) sb.append(", ");  
    }  
    sb.append("{}");  
    return sb.toString();  
}
```

- **boolean equals(Object o)** - возвращает true тогда и только тогда, когда переданный объект также является табулированной функцией (реализует интерфейс TabulatedFunction) и её набор точек в точности совпадает с набором точек функции, у которой вызывается метод:

- @Override

```
public boolean equals(Object o) {  
    if (this == o) return true;  
    if (!(o instanceof TabulatedFunction)) return false; // Если o не  
    TabulatedFunction, то False  
  
    // Быстрая проверка для ArrayTabulatedFunction  
    if (o instanceof ArrayTabulatedFunction other) { // Если o не  
    ArrayTabulatedFunction, то False  
        if (points.length != other.points.length) return false; // Если разное  
        количество точек, то false  
        for (int i = 0; i < points.length; i++) {  
            if (!this.points[i].equals(other.points[i])) return false; // Если хотя бы  
            одна пара точек не равна, то false(функции не равны)  
        }  
        return true;  
    }  
}
```

```

    }

    // Медленная проверка для TabulatedFunction
    TabulatedFunction other = (TabulatedFunction) o;
    if (points.length != other.getPointsCount()) return false; // Если разное
    количество точек, то false
    for (int i = 0; i < points.length; i++){
        if (!points[i].equals(other.getPoint(i))) return false; // Если хотя бы
        одна пара точек не равна, то false(функции не равны)
    }
    return true;
}

```

- **hashCode()** - возвращает значение хэш-кода для объекта табулированной функции:

```

@Override
public int hashCode(){
    int result = getPointsCount(); // Количество точек
    for (int i = 0; i < getPointsCount(); i++) { // point — текущая точка
        result = result * 3 + points[i].hashCode(); // Хэш-код точки
    }
    return result;
}

```

- **Object clone()** - возвращает объект-копию для объекта табулированной функции:

```

public Object clone(){
    try {
        ArrayTabulatedFunction clone = (ArrayTabulatedFunction)
        super.clone(); // Поверхностное копирование
        clone.points = new FunctionPoint[points.length]; // Клонирование
        массива
        for (int i = 0; i < points.length; i++) {
            clone.points[i] = (FunctionPoint) points[i].clone(); // Клонирование
            каждой точки(глубокое копирование)
        }
        return clone;
    }
    catch (CloneNotSupportedException e){
        throw new RuntimeException();
    }
}

```

3. Аналогично, переопределил методы toString(), equals(), hashCode() и clone() в классе LinkedListTabulatedFunction:

- Метод equals() :

```

public boolean equals(Object obj) {
    if (this == obj) return true;
    if (!(obj instanceof TabulatedFunction)) return false; // Если obj не
    TabulatedFunction, то False

    TabulatedFunction other = (TabulatedFunction) obj;
    if (obj instanceof LinkedListTabulatedFunction) { // Если obj не
    LinkedListTabulatedFunction, то False
        LinkedListTabulatedFunction otherLL =
        (LinkedListTabulatedFunction) obj;
        if (count != otherLL.count) return false; // Если разное количество
        точек, то false

        FunctionNode thisCurrent = head.next; // Указатель на голову
        первого списка
        FunctionNode otherCurrent = otherLL.head.next; // Указатель на
        голову второго списка
        while (thisCurrent != head) { // Пока не вернулись к голове
            if (!thisCurrent.point.equals(otherCurrent.point)) { // Попарно
            сравниваем
                return false; // Если точки не равны, то false
            }
            thisCurrent = thisCurrent.next; // Переход к следующей точке
            otherCurrent = otherCurrent.next;
        }
    }
    else {
        if (count != other.getPointsCount()) return false; // Проверка
        количества точек
        if (getLeftDomainBorder() != other.getLeftDomainBorder()) return
        false; // Проверка левой границы
        if (getRightDomainBorder() != other.getRightDomainBorder()) return
        false; // Правой границы

        // Обход списка
        for (int i = 0; i < getPointsCount(); i++) {
            if (!this.getPoint(i).equals(other.getPoint(i))) { // Если точки не
            равны
                return false;
            }
        }
    }
}

```

```

    }
    return true;
}

```

- **Метод toString():**
@Override
 // Текстовое представление функции
public String **toString**() {
 StringBuilder sb = **new** StringBuilder(); // Буфер для сборки строки
 sb.append("LinkedListTabulatedFunction: {"); // Заголовок класса
 sb.append("Левая граница = ").append(getLeftDomainBorder()); //
 Левая граница
 sb.append(", Правая граница = ").append(getRightDomainBorder());
 // Правая граница
 sb.append(", количество точек = ").append(getPointsCount()); //
 Колчество точек
 sb.append("\n ["); // Переход на новую строку

 // Обход списка
 FunctionNode current = **head.next**; // Старт с первой точки
 int index = 0; // Счетчик индексов
 while (current != **head**) { // Пока не вернулись к head
 sb.append("\n ").append(index).append(current.**point**);
 if (current.**next** != **head**) { // Если это не последняя точка, то
 ставим запятую
 sb.append(",");
 }
 current = current.**next**; // Переход к следующему узлу
 index++;
 }

 sb.append("\n]"); // Закрытие массива
 sb.append("\n}"); // и объекта
 return sb.toString();
 }

- **Метод hashCode():**
@Override
public **int** **hashCode**() {
 int result = **count**; //
 FunctionNode current = **head.next**;
 while (current != **head**) {
 result = (3 * result) + current.getPoint().hashCode(); //
 current = current.**next**;
 }
 }

```

    }
    return result;
}

```

- **Метод clone():**

```

public LinkedListTabulatedFunction clone(){
    LinkedListTabulatedFunction clone = new
LinkedListTabulatedFunction(); // Копия класса

    FunctionNode current = head.next; // Старт с первой реальной
точки(пропуская head)
    FunctionNode newHead = new FunctionNode(); // Новый узел
    newHead.prev = newHead; //
    newHead.next = newHead;
    clone.count = 0;

    if (count > 0) {
        FunctionNode newCurrent = new FunctionNode(); // Создает новый
узел
        newCurrent.point = new FunctionPoint(current.point); // Глубокая
копия первой точки
        // Вставляю между prevNew и newHead
        newCurrent.prev = newHead;
        newCurrent.next = newHead;
        newHead.prev = newCurrent;
        newHead.next = newCurrent;
        clone.count++;

        current = current.next; // Переходит к следующей точке
        FunctionNode prevNew = newCurrent;

        while (current != head) {
            newCurrent = new FunctionNode(); // Создание нового узла для
следующей точки
            newCurrent.point = new FunctionPoint(current.point); //
Глубокое копирование текущей точки
            newCurrent.prev = prevNew;
            newCurrent.next = newHead;
            prevNew.next = newCurrent;
            newHead.prev = newCurrent;

            prevNew = newCurrent;
            current = current.next;
        }
    }
}

```

```

        clone.count++;
    }
}
return clone; // Возвращает готовый клон
}

```

4. Сделал так, чтобы все объекты типа TabulatedFunction были клонируемыми с точки зрения JVM и внесите метод clone() в этот интерфейс:

```

public interface TabulatedFunction extends Function, java.io.Serializable,
Cloneable

```

```

@Override
TabulatedFunction clone() throws CloneNotSupportedException;

```

5. Main:

- Создал 3 набора точек:

```

FunctionPoint[] points1 = {
    new FunctionPoint(1.0, 1.0),
    new FunctionPoint(2.0, 4.0),
    new FunctionPoint(3.0, 9.0),
    new FunctionPoint(4.0, 16.0)
};

```

```

FunctionPoint[] points2 = {
    new FunctionPoint(1.0, 1.0),
    new FunctionPoint(2.0, 4.0),
    new FunctionPoint(3.0, 9.0),
    new FunctionPoint(4.0, 16.0)
};

```

```

FunctionPoint[] points3 = {
    new FunctionPoint(1.0, 6.0),
    new FunctionPoint(2.0, 9.0),
    new FunctionPoint(3.0, 12.0),
    new FunctionPoint(4.0, 15.0)
};

```

- Тест по toString():

Показать, что toString() выводит содержимое функции:

```

ArrayTabulatedFunction arrayFunc1 = new
ArrayTabulatedFunction(points1);
LinkedListTabulatedFunction listFunc1 = new

```

```
LinkedListTabulatedFunction(points1);
```

```
System.out.println("Array: " + arrayFunc1);
```

```
System.out.println("List: " + listFunc1)
```

Результат:

ТЕСТ 1. toString()

Array: {(1.0; 1.0), (2.0; 4.0), (3.0; 9.0), (4.0; 16.0)}

List: LinkedListTabulatedFunction: {Левая граница = 1.0, Правая граница = 4.0, количество точек = 4

```
[  
0(1.0; 1.0),  
1(2.0; 4.0),  
2(3.0; 9.0),  
3(4.0; 16.0)  
]  
}
```

- Тест по equals():

Проверяет 3 случая: Одинаковые данные, одинаковый класс -

True, Одинаковые данные, разные классы - True, Разные данные -

False:

```
ArrayTabulatedFunction arrayFunc2 = new
```

```
ArrayTabulatedFunction(points2);
```

```
LinkedListTabulatedFunction listFunc2 = new
```

```
LinkedListTabulatedFunction(points2);
```

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

```
System.out.println("array1 == list1: " + arrayFunc1.equals(listFunc1));
```

```
System.out.println("array1 != array3: " + arrayFunc1.equals(new  
ArrayTabulatedFunction(points3)));
```

Результат:

ТЕСТ 2. equals()

array1 == array2: true

array1 == list1: true

array1 != array3: false

- Тест по hashCode():

Доказывает что если равные объекты, то одинаковый hashCode();

Изменённый объект, то новый hashCode(); hashCode() согласован с equals():

```
System.out.println("hash array1: " + arrayFunc1.hashCode());
```



```
System.out.println("hash array2: " + arrayFunc2.hashCode());
System.out.println("LinkedList1: " + listFunc1.hashCode());
arrayFunc1.setPointY(0, 1.001);
System.out.println("hash array1 (изменён): " + arrayFunc1.hashCode());
System.out.println();
```

Вывод:

```
ТЕСТ 3. hashCode()
hash array1: -86113980
hash array2: -86113980
LinkedList1: -86113980
hash array1 (изменён): 1854764362
```

- **Тест по clone():**

```
ArrayTabulatedFunction arrayClone = (ArrayTabulatedFunction)
arrayFunc2.clone();
System.out.println("До: " + arrayClone.getPointY(0));
arrayFunc2.setPointY(0, 999.0);
System.out.println("После: " + arrayClone.getPointY(0));
```

Вывод:

До: 1.0

После: 1.0