

## Лабораторная работа №5. ООП.

Выполнил лабораторную работу:

Гордеев Максим Дмитриевич

Группа: 6204-010302D

**Цель лабораторной работы:** Расширить возможности классов, связанных с табулированными функциями, переопределив в них методы, унаследованные из класса **Object**.

### Ход выполнения лабораторной работы

#### Задание 1:

Переопределим в классе **FunctionPoint** следующие методы:

- `toString()`
- `equals()`
- `hashCode()`
- `clone()`

```
49     @Override
50     public String toString(){
51         return String.format(format: "%.3f, %.3f", coorX, coorY);
52     }
53
54     @Override
55     public boolean equals(Object a){
56         if (this == a){
57             return true;
58         }
59
60         if (!(a instanceof FunctionPoint) || a == null){
61             return false;
62         }
63
64         FunctionPoint b = (FunctionPoint) a;
65
66         if (equal(coorX, b.getCoorX()) && equal(coorY, b.getCoorY())){
67             return true;
68         }else{
69             return false;
70         }
71     }
72 }
```

```
74     @Override
75     public int hashCode() {
76
77         long xBits = Double.doubleToLongBits(coorX);
78         long yBits = Double.doubleToLongBits(coorY);
79
80         int xHigh = (int) (xBits >>> 32);
81         int xLow = (int) (xBits & 0xFFFFFFFFL);
82
83         int yHigh = (int) (yBits >>> 32);
84         int yLow = (int) (yBits & 0xFFFFFFFFL);
85
86
87         return xHigh ^ xLow ^ yHigh ^ yLow;
88     }
89
90     @Override
91     public FunctionPoint clone(){
92         return new FunctionPoint(this);
93     }
94 }
```

## Задание 2:

Теперь переопределим эти же методы для класса **ArrayTabulatedFunction**:

```
243     @Override
244     public String toString(){
245         String result = "{" + points[0].toString();
246         for (int i = 1; i < size; i++){
247             result += (", " +points[i].toString());
248         }
249         result += "}";
250
251         return result;
252     }
253 }
```

```

254     @Override
255     public boolean equals(Object a) {
256         if (this == a) { // Сравнение ссылок
257             return true;
258         }
259         // Проверим является ли переданный объект табулированной функцией
260         if (a == null || !(a instanceof TabulatedFunction) ) {
261             return false;
262         }
263
264
265         TabulatedFunction b = (TabulatedFunction) a;
266         // Перед тем как сравнивать координаты точек табулированных функций
267         // Сравним их размер и область определения
268         if (this.size != b.getPointsCount()){
269             return false;
270         }
271
272         if (!(equal(this.leftDomainBorder, b.getLeftDomainBorder()) && equal(this.rightDomainBorder,
273             b.getRightDomainBorder()))){
274             return false;
275         }
276
277         for (int i = 0; i < size; i++){
278             if (!(equal(points[i].getCoorX(), b.getPointX(i)) && equal(points[i].getCoorY(), b.getPointY(i))){
279                 return false;
280             }
281         }
282         return true;
283     }

```

```

285     @Override
286     public int hashCode() {
287         int hash = size; // Начинаем с количества точек
288         // Добавляем хэш-коды всех точек через XOR
289         for (int i = 0; i < size; i++) {
290             hash ^= points[i].hashCode();
291         }
292         return hash;
293     }
294
295     @Override
296     public ArrayTabulatedFunction clone(){
297         FunctionPoint[] clone_points = new FunctionPoint[this.size];
298
299         for (int i = 0; i < size; i++){ // Сначала собрали массив
300             clone_points[i] = new FunctionPoint(points[i]);
301         }
302
303         return new ArrayTabulatedFunction(clone_points);
304     }

```

### Задание 3:

Теперь переопределим методы для **LinkedListTabulatedFunction**:

```
506     @Override
507     public String toString() {
508         FunctionNode currNode = head;
509         String result = "{" + currNode.getPoint().toString();
510         currNode = currNode.getNext();
511         for (int i = 1; i < size; i++) {
512             result += ", " + currNode.getPoint().toString();
513             currNode = currNode.getNext();
514         }
515         result += "}";
516
517     return result;
518 }
519
```

```
520     @Override
521     public boolean equals(Object a){
522         if (this == a) { // Сравнение ссылок
523             return true;
524         }
525         // Проверим является ли переданный объект табулированной функцией
526         if (a == null || !(a instanceof TabulatedFunction) ) {
527             return false;
528         }
529
530
531         TabulatedFunction b = (TabulatedFunction) a;
532         // Перед тем как сравнивать координаты точек табулированных функций
533         // Сравним их размер и область определения
534         if (this.size != b.getPointsCount()){
535             return false;
536         }
537
538         if (!(equal(this.leftDomainBorder, b.getLeftDomainBorder()) && equal(this.rightDomainBorder,
539                         b.getRightDomainBorder())){
540             return false;
541         }
542
543         for (int i = 0; i < size; i++){
544             if (!(equal(this.getPointX(i), b.getPointX(i)) && equal(this.getPointY(i), b.getPointY(i))){
545                 return false;
546             }
547         }
548         return true;
549     }
550 }
551
```

```

552     @Override
553     public int hashCode() {
554         FunctionNode currNode = head;
555
556         int hash = size; // Начинаем с количества точек
557         // Добавляем хэш-коды всех точек через XOR
558         for (int i = 0; i < size; i++) {
559             hash ^= currNode.getPoint().hashCode();
560             currNode = currNode.getNext();
561         }
562         return hash;
563     }
564
565     @Override
566     public LinkedListTabulatedFunction clone() {
567         LinkedListTabulatedFunction cloned = new LinkedListTabulatedFunction();
568
569         if (size == 0) {
570             return cloned; // возвращаем пустой список
571         }
572
573         for (int i = 0; i < size; i++) {
574             try {
575                 FunctionPoint point = this.getPoint(i);
576                 cloned.addPoint(new FunctionPoint(point));
577             } catch (InappropriateFunctionPointException e) {
578                 throw new IllegalArgumentException(message: "Ошибка во время клонирования", e);
579             }
580         }
581
582         return cloned;
583     }
584
585 }

```

#### Задание 4:

Сделаем так, чтобы все объекты типа **TabulatedFunction** были клонируемыми с точки зрения JVM и внесем метод **clone()** в этот интерфейс.

```

7  public class ArrayTabulatedFunction implements TabulatedFunction, Serializable, Cloneable{
8
9  public class LinkedListTabulatedFunction implements TabulatedFunction, Externalizable, Cloneable{
10
11 package functions;
12
13 public interface TabulatedFunction extends Function{
14
15     public int getPointsCount();
16
17     public FunctionPoint getPoint(int index) throws FunctionPointIndexOutOfBoundsException, IllegalStateException;
18
19     public void setPoint(int index, FunctionPoint point) throws FunctionPointIndexOutOfBoundsException, InappropriateFunctionPointException;
20
21     public double getPointX(int index) throws FunctionPointIndexOutOfBoundsException, IllegalStateException;
22
23     public double getPointY(int index) throws FunctionPointIndexOutOfBoundsException, IllegalStateException;
24
25     public void setPointX(int index, double x) throws FunctionPointIndexOutOfBoundsException, InappropriateFunctionPointException, IllegalStateException;
26
27     public void setPointY(int index, double y) throws FunctionPointIndexOutOfBoundsException;
28
29     public void deletePoint(int index) throws FunctionPointIndexOutOfBoundsException, IllegalStateException;
30
31     public void addPoint(FunctionPoint point) throws InappropriateFunctionPointException;
32
33     public TabulatedFunction clone(); ←
34 }

```

## Задание 5:

Проверим работу добавленных методов, Main.java:

```
1 import functions.*;
2
3 public class Main {
4     Run|Debug
5     public static void main(String[] args) {
6         try {
7             System.out.println(x: "==== ТЕСТИРОВАНИЕ МЕТОДОВ toString(), equals(), hashCode(), clone() ===\n");
8
9             // 1. Создаем тестовые данные
10            System.out.println(x: "1. СОЗДАНИЕ ТЕСТОВЫХ ФУНКЦИЙ:");
11
12            // ArrayTabulatedFunction
13            FunctionPoint[] points1 = {
14                new FunctionPoint(x: 0, y: 0),
15                new FunctionPoint(x: 1, y: 1),
16                new FunctionPoint(x: 2, y: 4),
17                new FunctionPoint(x: 3, y: 9)
18            };
19            ArrayTabulatedFunction arrayFunc1 = new ArrayTabulatedFunction(points1);
20            System.out.println(x: "    ArrayTabulatedFunction1 создан");
21
22            // LinkedListTabulatedFunction с такими же точками
23            LinkedListTabulatedFunction linkedFunc1 = new LinkedListTabulatedFunction(points1);
24            System.out.println(x: "    LinkedListTabulatedFunction1 создан");
25
26            // Функции с другими точками
27            FunctionPoint[] points2 = {
28                new FunctionPoint(x: 0, y: 0),
29                new FunctionPoint(x: 1, y: 2),
30                new FunctionPoint(x: 2, y: 5)
31            };
32            ArrayTabulatedFunction arrayFunc2 = new ArrayTabulatedFunction(points2);
33            System.out.println(x: "    ArrayTabulatedFunction2 создан");
34
35            // 2. Тестирование toString()
36            System.out.println(x: "\n2. ТЕСТИРОВАНИЕ toString():");
37            System.out.println("    arrayFunc1: " + arrayFunc1.toString());
38            System.out.println("    LinkedFunc1: " + linkedFunc1.toString());
39            System.out.println("    arrayFunc2: " + arrayFunc2.toString());
40
41            // 3. Тестирование equals()
42            System.out.println(x: "\n3. ТЕСТИРОВАНИЕ equals():");
43            System.out.println("    arrayFunc1.equals(linkedFunc1): " + arrayFunc1.equals(linkedFunc1));
44            System.out.println("    arrayFunc1.equals(arrayFunc2): " + arrayFunc1.equals(arrayFunc2));
45            System.out.println("    arrayFunc1.equals(arrayFunc1): " + arrayFunc1.equals(arrayFunc1));
46            System.out.println("    arrayFunc1.equals(null): " + arrayFunc1.equals(a: null));
47
48            // 4. Тестирование hashCode()
49            System.out.println(x: "\n4. ТЕСТИРОВАНИЕ hashCode():");
50            System.out.println("    arrayFunc1.hashCode(): " + arrayFunc1.hashCode());
51            System.out.println("    linkedFunc1.hashCode(): " + linkedFunc1.hashCode());
52            System.out.println("    arrayFunc2.hashCode(): " + arrayFunc2.hashCode());
53
54            // Проверка согласованности equals() и hashCode()
55            System.out.println(x: "\n    СОГЛАСОВАННОСТЬ equals() и hashCode():");
56            System.out.println("    arrayFunc1.equals(linkedFunc1): " + arrayFunc1.equals(linkedFunc1));
57            System.out.println("    arrayFunc1.hashCode() == linkedFunc1.hashCode(): " +
58                (arrayFunc1.hashCode() == linkedFunc1.hashCode()));
59
60            // Изменяем точку и проверяем изменение хэш-кода
61            ArrayTabulatedFunction arrayFunc1Modified = new ArrayTabulatedFunction(points1);
62            arrayFunc1Modified.setPointY(index: 1, y: 1.001); // Незначительное изменение
63            System.out.println(x: "\n    После изменения точки Y[1] с 1.0 на 1.001:");
64            System.out.println("    Старый hashCode: " + arrayFunc1.hashCode());
65            System.out.println("    Новый hashCode: " + arrayFunc1Modified.hashCode());
66            System.out.println("    Хэши различны: " + (arrayFunc1.hashCode() != arrayFunc1Modified.hashCode()));  
Активи

```

```

68 // 5. Тестирование clone()
69 System.out.println(x: "\n5. ТЕСТИРОВАНИЕ clone():");
70
71 // Клонирование ArrayTabulatedFunction
72 ArrayTabulatedFunction arrayClone = arrayFunc1.clone();
73 System.out.println("  arrayFunc1.clone().equals(arrayFunc1): " + arrayClone.equals(arrayFunc1));
74 System.out.println("  arrayFunc1 == clone: " + (arrayFunc1 == arrayClone));
75
76 // Клонирование LinkedListTabulatedFunction
77 LinkedListTabulatedFunction linkedClone = linkedFunc1.clone();
78 System.out.println("  linkedFunc1.clone().equals(linkedFunc1): " + linkedClone.equals(linkedFunc1));
79 System.out.println("  linkedFunc1 == clone: " + (linkedFunc1 == linkedClone));
80
81 // 6. Проверка глубокого клонирования
82 System.out.println(x: "\n6. ПРОВЕРКА ГЛУБОКОГО КЛОНИРОВАНИЯ:");
83
84 // Изменяем оригинальные функции
85 arrayFunc1.setPointY(index: 0, y: 999);
86 linkedFunc1.setPointY(index: 0, y: 888);
87
88 System.out.println(x: "  После изменения оригиналов:");
89 System.out.println("  arrayClone.getPointY(0): " + arrayClone.getPointY(index: 0) + " (должно быть 0)");
90 System.out.println("  linkedClone.getPointY(0): " + linkedClone.getPointY(index: 0) + " (должно быть 0)");
91 System.out.println("  Клоны не изменились: " +
92 | | | (arrayClone.getPointY(index: 0) == 0 && linkedClone.getPointY(index: 0) == 0));
93
94 // 7. Дополнительные проверки
95 System.out.println(x: "\n7. ДОПОЛНИТЕЛЬНЫЕ ПРОВЕРКИ:");
96
97 // Проверка с пустой функцией
98 ArrayTabulatedFunction emptyArray = new ArrayTabulatedFunction(leftX: 0, rightX: 1, pointsCount: 2);
99 ArrayTabulatedFunction emptyArrayClone = emptyArray.clone();
100 System.out.println("  Пустая функция: " + emptyArray.toString());
101 System.out.println("  Клон пустой функции: " + emptyArrayClone.toString());
102 System.out.println("  Пустые функции равны: " + emptyArray.equals(emptyArrayClone));
103
104 System.out.println(x: "\n== ТЕСТИРОВАНИЕ ЗАВЕРШЕНО ==");
105
106 } catch (Exception e) {
107     System.err.println("Ошибка во время тестирования: " + e.getMessage());
108     e.printStackTrace();
109 }

```

Результат:

1. СОЗДАНИЕ ТЕСТОВЫХ ФУНКЦИЙ:  
ArrayTabulatedFunction1 создан  
LinkedListTabulatedFunction1 создан  
ArrayTabulatedFunction2 создан
  2. ТЕСТИРОВАНИЕ `toString()`:  
ArrayFunc1: {(0,000, 0,000), (1,000, 1,000), (2,000, 4,000), (3,000, 9,000)}  
LinkedFunc1: {(0,000, 0,000), (1,000, 1,000), (2,000, 4,000), (3,000, 9,000)}  
ArrayFunc2: {(0,000, 0,000), (1,000, 2,000), (2,000, 5,000)}
  3. ТЕСТИРОВАНИЕ `equals()`:  
arrayFunc1.equals(linkedFunc1): true  
arrayFunc1.equals(arrayFunc2): false  
arrayFunc1.equals(arrayFunc1): true  
arrayFunc1.equals(null): false
  4. ТЕСТИРОВАНИЕ `hashCode()`:  
arrayFunc1.hashCode(): 3801092  
linkedFunc1.hashCode(): 3801092  
arrayFunc2.hashCode(): 2145648643
- СОГЛАСОВАННОСТЬ `equals()` и `hashCode()`:
- arrayFunc1.equals(linkedFunc1): true  
arrayFunc1.hashCode() == linkedFunc1.hashCode(): true
- После изменения точки Y[1] с 1.0 на 1.001:
- Старый hashCode: 3801092  
Новый hashCode: -1823557514  
Хэши различны: true
5. ТЕСТИРОВАНИЕ `clone()`:  
arrayFunc1.clone().equals(arrayFunc1): true  
arrayFunc1 == clone: false  
linkedFunc1.clone().equals(linkedFunc1): true  
linkedFunc1 == clone: false
  6. ПРОВЕРКА ГЛУБОКОГО КЛОНИРОВАНИЯ:  
После изменения оригиналов:  
arrayClone.getPointY(0): 0.0 (должно быть 0)  
linkedClone.getPointY(0): 0.0 (должно быть 0)  
Клоны не изменились: true