Министерство науки и высшего образования Российской Федерации

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

Институт информатики и кибернетики

Кафедра технической кибернетики

Отчет по лабораторной работе №5

Дисциплина: «ООП»

Тема «Расширить возможности классов, связанных с табулированными функциями, переопределив в них методы, унаследованные из класса Object.»

Выполнил: Зорин Дмитрий Сергеевич

Группа: 6201-120303D

Самара, 2025

Задание на лабораторную работу

Задание 1

В классе FunctionPoint переопределил методы: toString(), equals(Object o), hashCode(), clone(). Пример кода:

```java
@Override  Dima
public String toString() {
    return "(" + x + "; " + y + ")";
}

@Override  Dima *
public boolean equals(Object o) {
    if (this == o) return true;
    if (!(o instanceof FunctionPoint)) return false;

    FunctionPoint p = (FunctionPoint) o;

    return Math.abs(this.x - p.x) < EPS &&
           Math.abs(this.y - p.y) < EPS;
}

@Override  Dima
public int hashCode() {
    long bitsX = Double.doubleToLongBits(x);
    long bitsY = Double.doubleToLongBits(y);
    int hashX = (int)(bitsX ^ (bitsX >>> 32));
    int hashY = (int)(bitsY ^ (bitsY >>> 32));
    return hashX ^ hashY;
}

@Override  Dima
public Object clone() {
    return new FunctionPoint(this);
}
```

Создаем точки и ее копии и сравниваем через equals(). Проверяем через hashCode()

Задание 2

Здесь необходимо реализовать методы toString(), equals(), hashCode() и clone() для массива точек

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

@Override  Dima *
public boolean equals(Object o) {
    if (this == o) return true;
    if (!(o instanceof TabulatedFunction)) return false;

    TabulatedFunction other = (TabulatedFunction) o;
    if (this.getPointsCount() != other.getPointsCount()) return false;
    if (other instanceof ArrayTabulatedFunction arr) {
        FunctionPoint[] otherPoints = arr.getPoints();
        for (int i = 0; i < points.length; i++) {
            if (!points[i].equals(otherPoints[i]))
                return false;
        }
        return true;
    }
    for (int i = 0; i < points.length; i++) {
        if (!points[i].equals(other.getPoint(i)))
            return false;
    }
    return true;
}
```

```
@Override  Dima *
public int hashCode() {
    int hash = points.length;
    for (FunctionPoint p : points) {
        hash = 31 * hash + p.hashCode();
    }
    return hash;
}
@Override  Dima
public TabulatedFunction clone() {
    FunctionPoint[] newPoints = new FunctionPoint[points.length];
    for (int i = 0; i < points.length; i++) {
        newPoints[i] = new FunctionPoint(points[i]);
    }
    return new ArrayTabulatedFunction(newPoints);
}
```

Теперь все правильно отображается, сравнивается и клонируется

Задание 3

Задание 3 аналогично заданию 2, здесь код необходимо для linked, только глубокое клонирование реализовать через пересборку списка

Экспонента:

```java
@Override  Dima
public String toString() {
    StringBuilder sb = new StringBuilder("{");
    FunctionNode current = head.next;
    while (current != head) {
        sb.append("(").append(current.point.getX()).append("; ").append(current.point.getY()).append(")");
        if (current.next != head) sb.append(", ");
        current = current.next;
    }
    sb.append("}");
    return sb.toString();
}

@Override  Dima *
public boolean equals(Object o) {
    if (this == o) return true;
    if (!(o instanceof TabulatedFunction)) return false;
    TabulatedFunction other = (TabulatedFunction) o;
    if (this.size != other.getPointsCount()) return false;

    if (other instanceof LinkedListTabulatedFunction ll) {
        FunctionNode thisNode = this.head.next;
        FunctionNode otherNode = ll.head.next;

        while (thisNode != head) {
            if (!thisNode.point.equals(otherNode.point))
                return false;
            thisNode = thisNode.next;
            otherNode = otherNode.next;
        }
        return true;
    }
    }
    return true;
}


@Override  Dima *
public int hashCode() {
    int hash = size;
    FunctionNode node = head.next;

    while (node != head) {
        hash = 31 * hash + node.point.hashCode();
        node = node.next;
    }

    return hash;
}


@Override  Dima
public TabulatedFunction clone() {
    LinkedListTabulatedFunction copy = new LinkedListTabulatedFunction();
    FunctionNode current = this.head.next;
    while (current != this.head) {
        copy.addNodeToTail(new FunctionPoint(current.point));
        current = current.next;
    }
    return copy;
}
```

Как итог методы корректно работают для связного списка

Задание 4

Все объекты типа TabulatedFunction должны быть клонируемыми с точки зрения JVM и добавить метод clone() в интерфейс

Пример кода:

```
public interface TabulatedFunction extends Function, Serializable, Cloneable {
```

любой объект, реализующий TabulatedFunction, можно клонировать через JVM

Задание 5

В этом задании мы проверяем работу написанных методов, для этого я создал отдельный класс Test:

```java
package functions;

public class Test {
    public static void main(String[] args) throws InappropriateFunctionPointException {
        double[] yValues = {1, 4, 9};
        ArrayTabulatedFunction arrayFunc = new ArrayTabulatedFunction( leftX: 0,  rightX: 2, yValues);

        FunctionPoint[] points = {
                new FunctionPoint( x: 0,  y: 1),
                new FunctionPoint( x: 1,  y: 4),
                new FunctionPoint( x: 2,  y: 9)
        };
        LinkedListTabulatedFunction listFunc = new LinkedListTabulatedFunction(points);

        //Проверка toString()
        System.out.println("ArrayTabulatedFunction: " + arrayFunc);
        System.out.println("LinkedListTabulatedFunction: " + listFunc);

        //Проверка equals()
        ArrayTabulatedFunction arrayCopy = (ArrayTabulatedFunction) arrayFunc.clone();
        LinkedListTabulatedFunction listCopy = (LinkedListTabulatedFunction) listFunc.clone();

        System.out.println("arrayFunc.equals(arrayCopy): " + arrayFunc.equals(arrayCopy)); // true
        System.out.println("listFunc.equals(listCopy): " + listFunc.equals(listCopy));       // true
        System.out.println("arrayFunc.equals(listFunc): " + arrayFunc.equals(listFunc));     // false

        //Проверка hashCode()
        System.out.println("arrayFunc.hashCode(): " + arrayFunc.hashCode());
        System.out.println("arrayCopy.hashCode(): " + arrayCopy.hashCode());
        System.out.println("listFunc.hashCode(): " + listFunc.hashCode());
        System.out.println("listCopy.hashCode(): " + listCopy.hashCode());
```

Пример работы кода:
ArrayTabulatedFunction: {(0.0; 1.0), (1.0; 4.0), (2.0; 9.0)}
LinkedListTabulatedFunction: {(0.0; 1.0), (1.0; 4.0), (2.0; 9.0)}
arrayFunc.equals(arrayCopy): true
listFunc.equals(listCopy): true
arrayFunc.equals(listFunc): true
arrayFunc.hashCode(): -2144117475
arrayCopy.hashCode(): -2144117475
listFunc.hashCode(): -2144117475
listCopy.hashCode(): -2144117475
После изменения arrayFunc.hashCode(): -1727756284
Исходный arrayFunc: {(0.0; 1.0), (1.0; 4.002), (2.0; 9.0)}
Клон arrayCopy: {(0.0; 10.0), (1.0; 4.0), (2.0; 9.0)}
Исходный listFunc: {(0.0; 1.0), (1.0; 4.0), (2.0; 9.0)}
Клон listCopy: {(0.0; 20.0), (1.0; 4.0), (2.0; 9.0)}