

## **Лабораторная работа №5**

Валиневич Владислав Александрович

группа: 6204-010302D

Цель работы: расширить возможности классов, связанных с табулированными функциями, переопределив в них методы, унаследованные из класса `Object`.

### Задание 1:

Переопределим методы в классе `FunctionPoint`:

`toString()`-возвращает строку в формате `(x; y)`, где `x` и `y` — координаты точки.

`equals(Object o)`-сравнивает координаты точек с учётом погрешности.

`int hashCode()`-возвращает значение хэш-кода для объекта точки.

`Object clone()`-возвращает объект-копию для объекта точки.

```
// Возвращаем текстовое описание точки в формате (x; y)
@Override
public String toString() {
    return "(" + x + "; " + y + ")";
}

@Override
public boolean equals(Object o) {
    if (this == o) return true;
    if (o == null || getClass() != o.getClass()) return false;

    FunctionPoint that = (FunctionPoint) o;

    // Сравниваем координаты с учетом погрешности для чисел с плавающей точкой
    return Double.compare(that.x, x) == 0 && Double.compare(that.y, y) == 0;
}

// Возвращаем хэш-код точки на основе её координат
@Override
public int hashCode() {
    long xBits = Double.doubleToLongBits(x);
    long yBits = Double.doubleToLongBits(y);

    // Преобразуем double в два int и применяем XOR
    int xHash = (int) (xBits ^ (xBits >>> 32));
    int yHash = (int) (yBits ^ (yBits >>> 32));

    return xHash ^ yHash;
}

// Создаем и возвращаем копию текущей точки
@Override
public Object clone() {
    return new FunctionPoint(this.x, this.y);
}
```

### Задание 2:

Переопределим методы в классе `ArrayTabulatedFunction`:

`toString()`-формируем строку в формате `{(x1; y1), (x2; y2), ..., (xn; yn)}`

`equals(Object o)`-сравнивает количество точек и их координаты у табулированных функций.

`hashCode()`-комбинирует хэш-коды всех точек и количество точек через операцию XOR.

`clone()`-создает глубокую копию массива точек

```
//{(x1: y1), (x2: y2), ..., (xn: yn)}
@Override
public String toString() {
    StringBuilder sb = new StringBuilder();
    sb.append("{");
    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
public boolean equals(Object o) {
    if (this == o) return true; // Проверяем, не сравниваем ли объект сам с собой
    if (o == null) return false;

    // Если объект ArrayTabulatedFunction
    if (o instanceof ArrayTabulatedFunction) {
        ArrayTabulatedFunction other = (ArrayTabulatedFunction) o;
        // Если количество точек разное - функции точно не равны
        if (this.points.length != other.points.length) return false;
        // Сравниваем каждую точку попарно
        for (int i = 0; i < points.length; ++i) {
            if (!this.points[i].equals(other.points[i])) return false;
        }
        return true;
    }
    // Для любого TabulatedFunction
    if (o instanceof TabulatedFunction) {
        TabulatedFunction other = (TabulatedFunction) o;

        if (this.getPointsCount() != other.getPointsCount()) return false;

        for (int i = 0; i < points.length; ++i) {
            FunctionPoint thisPoint = this.points[i];
            FunctionPoint otherPoint = other.getPoint(i);
            if (!thisPoint.equals(otherPoint)) return false;
        }
        return true;
    }
    return false;
}

@Override
public int hashCode() {
    int hash = points.length;

    for (FunctionPoint point : points) {
        // Комбинируем хэш-код текущей точки с общим хэш-кодом через XOR
        hash ^= point.hashCode();
    }

    return hash;
}

@Override
public Object clone() {
    FunctionPoint[] clonedPoints = new FunctionPoint[points.length];
    for (int i = 0; i < points.length; ++i) {
        clonedPoints[i] = (FunctionPoint) points[i].clone();
    }

    ArrayTabulatedFunction cloned = new ArrayTabulatedFunction(clonedPoints);
    return cloned;
}
}
```

### Задание 3:

Переопределим методы в классе `LinkedListTabulatedFunction`, аналогично с `ArrayTabulatedFunction`:

```
@Override
public String toString() {
    StringBuilder sb = new StringBuilder();

    sb.append("{");
    // Начинаем с первой реальной точки (после head)
    FunctionNode current = head.next;

    // Проходим по всем узлам списка до возврата к head
    while (current != head) {
        // Добавляем точку в формате (x; y)
        sb.append("(").append(current.point.getX()).append("; ").append(current.point.getY()).append(" ");

        // Переходим к следующему узлу
        current = current.next;
        // Если это не последняя точка, добавляем запятую и пробел
        if (current != head) {
            sb.append(", ");
        }
    }
    sb.append("}");
    return sb.toString();
}

@Override
public boolean equals(Object o) {
    if (this == o) return true;
    if (o == null) return false;
    if (o instanceof LinkedListTabulatedFunction) {
        LinkedListTabulatedFunction other = (LinkedListTabulatedFunction) o;

        if (this.pointsCount != other.pointsCount) return false;

        FunctionNode currentThis = this.head.next;
        FunctionNode currentOther = other.head.next;

        while (currentThis != this.head) {
            if (!currentThis.point.equals(currentOther.point)) {
                return false;
            }
            currentThis = currentThis.next;
            currentOther = currentOther.next;
        }
        return true;
    }
    if (o instanceof TabulatedFunction) {
        TabulatedFunction other = (TabulatedFunction) o;
        if (this.getPointsCount() != other.getPointsCount()) return false;
        FunctionNode current = this.head.next;
        int index = 0;
        while (current != this.head) {
            FunctionPoint otherPoint = other.getPoint(index);
```

```

    }
    return true;
}
}
if (o instanceof TabulatedFunction) {
    TabulatedFunction other = (TabulatedFunction) o;
    if (this.getPointsCount() != other.getPointsCount()) return false;
    FunctionNode current = this.head.next;
    int index = 0;
    while (current != this.head) {
        FunctionPoint otherPoint = other.getPoint(index);
        if (!current.point.equals(otherPoint)) {
            return false;
        }
        current = current.next;
        index++;
    }
    return true;
}
return false;
}

@Override
public int hashCode() {
    int hash = pointsCount;
    FunctionNode current = head.next;

    // Проходим по всем узлам списка
    while (current != head) {
        // Комбинируем хэш-код текущей точки с общим хэш-кодом через XOR
        hash ^= current.point.hashCode();
        // Переходим к следующему узлу
        current = current.next;
    }
    return hash;
}

@Override
public Object clone() {
    FunctionPoint[] clonePoints = new FunctionPoint[pointsCount];
    FunctionNode current = head.next;
    for (int i = 0; i < pointsCount; i++) {
        clonePoints[i] = (FunctionPoint) current.point.clone();
        current = current.next;
    }
    return new LinkedListTabulatedFunction(clonePoints);
}
}

```

## Задание 4:

Добавим метод clone() в интерфейс TabulatedFunction

```

package functions;

public interface TabulatedFunction extends Function, Cloneable {

    // Методы getLeftDomainBorder(), getRightDomainBorder() и getFunctionValue(double x)
    // теперь наследуются от интерфейса Function

    // Работа с точками
    int getPointsCount();
    FunctionPoint getPoint(int index);
    void setPoint(int index, FunctionPoint point) throws InappropriateFunctionPointException;
    double getPointX(int index);
    void setPointX(int index, double x) throws FunctionPointIndexOutOfBoundsException, InappropriateFunctionPointException;
    double getPointY(int index);
    void setPointY(int index, double y);
    void deletePoint(int index);
    void addPoint(FunctionPoint point) throws InappropriateFunctionPointException;

    // Метод клонирования
    Object clone();
    String toString();
}

```



## Задание 5:

Протестируем все созданные методы:

```
=== ТЕСТИРОВАНИЕ МЕТОДОВ TabulatedFunction ===
```

```
ТЕСТИРОВАНИЕ toString():
```

```
ArrayTabulatedFunction 1: {(0.0; 1.0), (1.0; 3.0), (2.0; 5.0), (3.0; 7.0)}
```

```
ArrayTabulatedFunction 2: {(0.0; 1.0), (1.0; 3.0), (2.0; 5.0), (3.0; 7.0)}
```

```
ArrayTabulatedFunction 3: {(0.0; 0.0), (1.0; 2.0), (2.0; 4.0), (3.0; 6.0)}
```

```
LinkedListTabulatedFunction 1: {(0.0; 1.0), (1.0; 3.0), (2.0; 5.0), (3.0; 7.0)}
```

```
LinkedListTabulatedFunction 2: {(0.0; 1.0), (1.0; 3.0), (2.0; 5.0), (3.0; 7.0)}
```

```
LinkedListTabulatedFunction 3: {(0.0; 1.0), (1.0; 3.0), (2.0; 5.0)}
```

```
ТЕСТИРОВАНИЕ equals():
```

```
arrayFunc1.equals(arrayFunc2) [одинаковые Array]: true
```

```
linkedListFunc1.equals(linkedListFunc2) [одинаковые LinkedList]: true
```

```
arrayFunc1.equals(arrayFunc3) [разные Array]: false
```

```
linkedListFunc1.equals(linkedListFunc3) [разные LinkedList]: false
```

```
arrayFunc1.equals(linkedListFunc1) [Array vs LinkedList, одинаковые точки]: true
```

```
linkedListFunc1.equals(arrayFunc1) [LinkedList vs Array, одинаковые точки]: true
```

```
arrayFunc1.equals(null) [с null]: false
```

```
arrayFunc1.equals(arrayFunc1) [с самим собой]: true
```

```
ТЕСТИРОВАНИЕ hashCode():
```

```
arrayFunc1.hashCode(): 1074266116
```

```
arrayFunc2.hashCode(): 1074266116
```

```
arrayFunc3.hashCode(): 2146435076
```

```
linkedListFunc1.hashCode(): 1074266116
```

```
linkedListFunc2.hashCode(): 1074266116
```

```
linkedListFunc3.hashCode(): 1075576835
```

```
Проверка согласованности equals() и hashCode():
```

```
arrayFunc1.equals(arrayFunc2) && arrayFunc1.hashCode() == arrayFunc2.hashCode(): true
```

```
linkedListFunc1.equals(linkedListFunc2) && linkedListFunc1.hashCode() == linkedListFunc2.hashCode(): true
```

```
arrayFunc1.equals(linkedListFunc1) && arrayFunc1.hashCode() == linkedListFunc1.hashCode(): true
```

```
Тестирование изменения при небольшом изменении координат:
```

```
Исходный arrayFunc1.hashCode(): 1074266116
```

```
После изменения Y[1] на +0.001, arrayFunc1.hashCode(): 162683965
```

```
После возврата исходного значения, arrayFunc1.hashCode(): 1074266116
```

```
ТЕСТИРОВАНИЕ clone():
```

```
arrayFunc1.clone().equals(arrayFunc1): true
```

```
linkedListFunc1.clone().equals(linkedListFunc1): true
```

```
ПРОВЕРКА ГЛУБОКОГО КЛОНИРОВАНИЯ:
```

```
До изменений:
```

```
arrayFunc1 Y[0]: 1.0, arrayClone Y[0]: 1.0
```

```
linkedListFunc1 Y[0]: 1.0, linkedListClone Y[0]: 1.0
```

```
После изменения оригиналов:
```

```
arrayFunc1 Y[0]: 11.0, arrayClone Y[0]: 1.0
```

```
linkedListFunc1 Y[0]: 21.0, linkedListClone Y[0]: 1.0
```

```

import functions.basic.*;
import functions.*;
import functions.meta.*;

import java.io.*;

public class main {
    public static void main(String[] args) {
        try {
            System.out.println("=== ТЕСТИРОВАНИЕ МЕТОДОВ TabulatedFunction ===\n");

            // Создаем тестовые данные
            FunctionPoint[] points1 = {
                new FunctionPoint(0.0, 1.0),
                new FunctionPoint(1.0, 3.0),
                new FunctionPoint(2.0, 5.0),
                new FunctionPoint(3.0, 7.0)
            };

            FunctionPoint[] points2 = {
                new FunctionPoint(0.0, 0.0),
                new FunctionPoint(1.0, 2.0),
                new FunctionPoint(2.0, 4.0),
                new FunctionPoint(3.0, 6.0)
            };

            FunctionPoint[] points3 = {
                new FunctionPoint(0.0, 1.0),
                new FunctionPoint(1.0, 3.0),
                new FunctionPoint(2.0, 5.0)
            };

            // Создаем объекты для тестирования
            ArrayTabulatedFunction arrayFunc1 = new ArrayTabulatedFunction(points1);
            ArrayTabulatedFunction arrayFunc2 = new ArrayTabulatedFunction(points1); // такой же как arrayFunc1
            ArrayTabulatedFunction arrayFunc3 = new ArrayTabulatedFunction(points2); // другой

            LinkedListTabulatedFunction linkedListFunc1 = new LinkedListTabulatedFunction(points1);
            LinkedListTabulatedFunction linkedListFunc2 = new LinkedListTabulatedFunction(points1); // такой же как
                                                                                               // linkedListFunc1
            LinkedListTabulatedFunction linkedListFunc3 = new LinkedListTabulatedFunction(points3); // другой

            // =====
            // ТЕСТИРОВАНИЕ toString()
            //
            System.out.println("ТЕСТИРОВАНИЕ toString():");
            System.out.println("ArrayTabulatedFunction 1: " + arrayFunc1.toString());
            System.out.println("ArrayTabulatedFunction 2: " + arrayFunc2.toString());
            System.out.println("ArrayTabulatedFunction 3: " + arrayFunc3.toString());
            System.out.println("LinkedListTabulatedFunction 1: " + linkedListFunc1.toString());
            System.out.println("LinkedListTabulatedFunction 2: " + linkedListFunc2.toString());
            System.out.println("LinkedListTabulatedFunction 3: " + linkedListFunc3.toString());
            System.out.println();
        }
    }
}

```

```

// =====
// ТЕСТИРОВАНИЕ equals()
// =====
System.out.println("ТЕСТИРОВАНИЕ equals():");

// Сравнение одинаковых объектов одного класса
System.out.println("arrayFunc1.equals(arrayFunc2) [одинаковые Array]: " + arrayFunc1.equals(arrayFunc2));
System.out.println("linkedListFunc1.equals(linkedListFunc2) [одинаковые LinkedList]: "
    + linkedListFunc1.equals(linkedListFunc2));

// Сравнение разных объектов одного класса
System.out.println("arrayFunc1.equals(arrayFunc3) [разные Array]: " + arrayFunc1.equals(arrayFunc3));
System.out.println("linkedListFunc1.equals(linkedListFunc3) [разные LinkedList]: "
    + linkedListFunc1.equals(linkedListFunc3));

// Сравнение объектов разных классов с одинаковыми точками
System.out.println("arrayFunc1.equals(linkedListFunc1) [Array vs LinkedList, одинаковые точки]: "
    + arrayFunc1.equals(linkedListFunc1));
System.out.println("linkedListFunc1.equals(arrayFunc1) [LinkedList vs Array, одинаковые точки]: "
    + linkedListFunc1.equals(arrayFunc1));

// Сравнение с null
System.out.println("arrayFunc1.equals(null) [с null]: " + arrayFunc1.equals(null));

// Сравнение с самим собой
System.out.println("arrayFunc1.equals(arrayFunc1) [с самим собой]: " + arrayFunc1.equals(arrayFunc1));
System.out.println();

// =====
// ТЕСТИРОВАНИЕ hashCode()
// =====
System.out.println("ТЕСТИРОВАНИЕ hashCode():");
System.out.println("arrayFunc1.hashCode(): " + arrayFunc1.hashCode());
System.out.println("arrayFunc2.hashCode(): " + arrayFunc2.hashCode());
System.out.println("arrayFunc3.hashCode(): " + arrayFunc3.hashCode());
System.out.println("linkedListFunc1.hashCode(): " + linkedListFunc1.hashCode());
System.out.println("linkedListFunc2.hashCode(): " + linkedListFunc2.hashCode());
System.out.println("linkedListFunc3.hashCode(): " + linkedListFunc3.hashCode());

// Проверка согласованности equals() и hashCode()
System.out.println("\nПроверка согласованности equals() и hashCode():");
System.out.println("arrayFunc1.equals(arrayFunc2) && arrayFunc1.hashCode() == arrayFunc2.hashCode(): " +
    (arrayFunc1.equals(arrayFunc2) && arrayFunc1.hashCode() == arrayFunc2.hashCode()));
System.out.println(
    "linkedListFunc1.equals(linkedListFunc2) && linkedListFunc1.hashCode() == linkedListFunc2.hashCode(): "
    +
    (linkedListFunc1.equals(linkedListFunc2)
        && linkedListFunc1.hashCode() == linkedListFunc2.hashCode()));
System.out.println(
    "arrayFunc1.equals(linkedListFunc1) && arrayFunc1.hashCode() == linkedListFunc1.hashCode(): " +
    (arrayFunc1.equals(linkedListFunc1)
        && arrayFunc1.hashCode() == linkedListFunc1.hashCode()));

```



```

// Тестирование изменения хэш-кода при небольшом изменении объекта
System.out.println("\nТестирование изменения при небольшом изменении координат:");
double originalY = arrayFunc1.getPointY(1);
System.out.println("Исходный arrayFunc1.hashCode(): " + arrayFunc1.hashCode());

// Небольшое изменение (на 0.001)
arrayFunc1.setPointY(1, originalY + 0.001);
System.out.println("После изменения Y[1] на +0.001, arrayFunc1.hashCode(): " + arrayFunc1.hashCode());

// Возвращаем обратно
arrayFunc1.setPointY(1, originalY);
System.out.println("После возврата исходного значения, arrayFunc1.hashCode(): " + arrayFunc1.hashCode());
System.out.println();

// =====
// ТЕСТИРОВАНИЕ clone()
// =====
System.out.println("ТЕСТИРОВАНИЕ clone():");

// Клонирование ArrayTabulatedFunction
ArrayTabulatedFunction arrayClone = (ArrayTabulatedFunction) arrayFunc1.clone();
System.out.println("arrayFunc1.clone().equals(arrayFunc1): " + arrayClone.equals(arrayFunc1));

// Клонирование LinkedListTabulatedFunction
LinkedListTabulatedFunction linkedListClone = (LinkedListTabulatedFunction) linkedListFunc1.clone();
System.out.println(
    "linkedListFunc1.clone().equals(linkedListFunc1): " + linkedListClone.equals(linkedListFunc1));

// =====
// ПРОВЕРКА ГЛУБОКОГО КЛОНИРОВАНИЯ
// =====
System.out.println("\nПРОВЕРКА ГЛУБОКОГО КЛОНИРОВАНИЯ:");

// Запоминаем исходные значения
double originalArrayY = arrayFunc1.getPointY(0);
double originalLinkedListY = linkedListFunc1.getPointY(0);

System.out.println("До изменений:");
System.out.println(
    "arrayFunc1 Y[0]: " + arrayFunc1.getPointY(0) + ", arrayClone Y[0]: " + arrayClone.getPointY(0));
System.out.println("linkedListFunc1 Y[0]: " + linkedListFunc1.getPointY(0) + ", linkedListClone Y[0]: "
    + linkedListClone.getPointY(0));

// Изменяем оригинальные объекты
arrayFunc1.setPointY(0, originalArrayY + 10.0);
linkedListFunc1.setPointY(0, originalLinkedListY + 20.0);

System.out.println("\nПосле изменения оригиналов:");
System.out.println(
    "arrayFunc1 Y[0]: " + arrayFunc1.getPointY(0) + ", arrayClone Y[0]: " + arrayClone.getPointY(0));
System.out.println("linkedListFunc1 Y[0]: " + linkedListFunc1.getPointY(0) + ", linkedListClone Y[0]: "
    + linkedListClone.getPointY(0));

    } catch (Exception e) {
        System.err.println("Ошибка при тестировании: " + e.getMessage());
        e.printStackTrace();
    }
}

```

Вывод: Я расширил возможности классов, связанных с табулированными функциями, переопределив в них методы, унаследованные из класса Object.