

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

Отчёт

по лабораторной работе №5

Выполнил:

студент группы

6301-030301D

Панин Ю. В.

Проверил:

Борисов Д. С.

Самара, 2025

Содержание

Содержание	2
Задание 1	3
Задание 2	4
Задание 3	5
Задание 4	7
Задание 5	8

Задание 1

В класс `FunctionPoint` были добавлены методы согласно заданию:

- `toString()` возвращает координаты в формате (x; y).
- `equals()` сравнивает точки с точностью до машинного эпсилона.
- `hashCode()` использует битовое представление `double` для вычисления хэша.
- `clone()` создаёт поверхностную копию (поля `x` и `y` копируются автоматически).

```
// 1. Метод toString()
@Override no usages
public String toString() {
    return "(" + x + "; " + y + ")";
}

// 2. Метод equals() с учётом машинного эпсилона
@Override no usages
public boolean equals(Object o) {
    if (this == o) return true;
    if (o == null || getClass() != o.getClass()) return false;
    FunctionPoint that = (FunctionPoint) o;
    return Math.abs(x - that.x) < 1e-10 && Math.abs(y - that.y) < 1e-10;
}

// 3. Метод hashCode() на основе битового представления double
@Override no usages
public int hashCode() {
    long xBits = Double.doubleToLongBits(x);
    long yBits = Double.doubleToLongBits(y);
    int x1 = (int)(xBits & 0xFFFFFFFF);
    int x2 = (int)(xBits >>> 32);
    int y1 = (int)(yBits & 0xFFFFFFFF);
    int y2 = (int)(yBits >>> 32);
    return x1 ^ x2 ^ y1 ^ y2;
}

// 4. Метод clone()
@Override no usages
public Object clone() {
    try {
        return super.clone(); // Простое клонирование (нет ссылок на другие объекты)
    } catch (CloneNotSupportedException e) {
        throw new AssertionError("Клонирование не поддерживается", e);
    }
}
```

Задание 2

Согласно заданию в классе `ArrayTabulatedFunction` были переопределены следующие методы:

- `toString()` формирует строку вида `{(x1; y1), (x2; y2), ...}`.
- `equals()` оптимизирован для сравнения с объектами того же класса.
- `hashCode()` учитывает хэши всех точек и их количество.
- `clone()` выполняет глубокое копирование массива точек.

```
// 1. Метод toString()
@Override no usages
public String toString() {
    StringBuilder sb = new StringBuilder("{}");
    for (int i = 0; i < pointsCount; i++) {
        sb.append(points[i].toString());
        if (i < pointsCount - 1) {
            sb.append(", ");
        }
    }
    sb.append("}");
    return sb.toString();
}

// 2. Метод equals()
@Override no usages
public boolean equals(Object o) {
    if (this == o) return true;
    if (!(o instanceof TabulatedFunction)) return false;
    TabulatedFunction that = (TabulatedFunction) o;

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

    // Оптимизация для одинаковых типов
    if (o instanceof ArrayTabulatedFunction) {
        ArrayTabulatedFunction other = (ArrayTabulatedFunction) o;
        for (int i = 0; i < pointsCount; i++) {
            if (!points[i].equals(other.points[i])) return false;
        }
    } else {
        // Общий случай
        for (int i = 0; i < pointsCount; i++) {
            if (!this.getPoint(i).equals(that.getPoint(i))) return false;
        }
    }
    return true;
}
```

```

// 3. Метод hashCode()
@Override no usages
public int hashCode() {
    int result = 0;
    for (int i = 0; i < pointsCount; i++) {
        result ^= points[i].hashCode();
    }
    result ^= pointsCount; // Учитываем количество точек
    return result;
}

// 4. Метод clone() с глубоким копированием
@Override no usages
public Object clone() {
    try {
        ArrayTabulatedFunction cloned = (ArrayTabulatedFunction) super.clone();
        // Глубокое копирование массива точек
        cloned.points = new FunctionPoint[this.points.length];
        for (int i = 0; i < this.pointsCount; i++) {
            cloned.points[i] = (FunctionPoint) this.points[i].clone();
        }
        return cloned;
    } catch (CloneNotSupportedException e) {
        throw new AssertionError("Клонирование не поддерживается", e);
    }
}

```

Задание 3

Аналогичные преобразования были произведены для класса `LinkedListTabulatedFunction`:

- `toString()` обходит связный список для формирования строки.
- `equals()` оптимизирован для сравнения с другими связными списками.
- `hashCode()` вычисляется на основе всех точек в списке.
- `clone()` пересобирает список для избежания проблем с клонированием узлов.

```

// 1. Метод toString()
@Override no usages
public String toString() {
    StringBuilder sb = new StringBuilder("{}");
    FunctionNode current = head.next;
    while (current != head) {
        sb.append(current.point.toString());
        if (current.next != head) {
            sb.append(", ");
        }
        current = current.next;
    }
    sb.append("{}");
    return sb.toString();
}

```

```

// 2. Метод equals()
@Override no usages
public boolean equals(Object o) {
    if (this == o) return true;
    if (!(o instanceof TabulatedFunction)) return false;
    TabulatedFunction that = (TabulatedFunction) o;

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

    // Оптимизация для одинаковых типов
    if (o instanceof LinkedListTabulatedFunction) {
        LinkedListTabulatedFunction other = (LinkedListTabulatedFunction) o;
        FunctionNode thisNode = this.head.next;
        FunctionNode otherNode = other.head.next;

        while (thisNode != this.head) {
            if (!thisNode.point.equals(otherNode.point)) return false;
            thisNode = thisNode.next;
            otherNode = otherNode.next;
        }
    } else {
        // Общий случай
        for (int i = 0; i < pointsCount; i++) {
            if (!this.getPoint(i).equals(that.getPoint(i))) return false;
        }
    }
    return true;
}

```

```

// 3. Метод hashCode()
@Override no usages
public int hashCode() {
    int result = 0;
    FunctionNode current = head.next;
    while (current != head) {
        result ^= current.point.hashCode();
        current = current.next;
    }
    result ^= pointsCount;
    return result;
}

```

```

public Object clone() {
    LinkedListTabulatedFunction cloned = new LinkedListTabulatedFunction();
    cloned.head = new FunctionNode(null);
    cloned.head.next = cloned.head;
    cloned.head.prev = cloned.head;
    cloned.pointsCount = 0;
    cloned.lastAccessedNode = null;
    cloned.lastAccessedIndex = -1;

    FunctionNode current = this.head.next;
    FunctionNode lastNode = cloned.head;

    while (current != this.head) {
        FunctionNode newNode = new FunctionNode((FunctionPoint) current.point.clone());

        newNode.prev = lastNode;
        newNode.next = cloned.head;
        lastNode.next = newNode;
        cloned.head.prev = newNode;

        lastNode = newNode;
        cloned.pointsCount++;
        current = current.next;
    }

    return cloned;
}

```

Задание 4

В интерфейс TabulatedFunction были внесены изменения согласно заданию и теперь все классы, реализующие TabulatedFunction, должны предоставить реализацию clone():

```

package functions;

public interface TabulatedFunction extends Function, Cloneable {
    // Методы для работы с точками табулированной функции
    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 InappropriateFunctionPointException;
    double getPointY(int index);
    void setPointY(int index, double y);
    void deletePoint(int index);
    void addPoint(FunctionPoint point) throws InappropriateFunctionPointException;

    // Метод clone() добавлен в интерфейс
    Object clone();
}

```

Задание 5

Проверили работу написанных методов:

- Метод toString, как и ожидается, выводит координаты в виде текста.
- Проверили работу метода equals() - он работает согласно заданию.
- Проверили работу метода hashCode, затем координаты были незначительно изменены и были сверены полученные hash-коды.
- Был проверен метод clone - было подтверждено, что произведено именно глубокое клонирование.

```
PS C:\Users\yura-\Lab-5-2025\src> javac functions/meta/*.java
PS C:\Users\yura-\Lab-5-2025\src> javac functions/basic/*.java
PS C:\Users\yura-\Lab-5-2025\src> javac functions/*.java
PS C:\Users\yura-\Lab-5-2025\src> javac Main.java
PS C:\Users\yura-\Lab-5-2025\src> java Main
```

1. Тестирование FunctionPoint:

```
p1 = (1.5; 2.5)
p2 = (1.5; 2.5)
p3 = (1.5000001; 2.5)
p1.equals(p2) = true
p1.equals(p3) = false
p1.hashCode() = 2147221504
p2.hashCode() = 2147221504
p3.hashCode() = 1697378971
p4 = clone(p1) = (1.5; 2.5)
p1.equals(p4) = true
```

2. Тестирование ArrayTabulatedFunction:

```
func1 = {(0.0; 0.0), (1.0; 1.0), (2.0; 4.0)}
func2 = {(0.0; 0.0), (1.0; 1.0), (2.0; 4.0)}
func3 = {(0.0; 0.0), (1.0; 0.0), (2.0; 0.0)}
func1.equals(func2) = true
func1.equals(func3) = false
func1.hashCode() = 1048579
func2.hashCode() = 1048579
func3.hashCode() = 2146435075
cloned = {(0.0; 0.0), (1.0; 1.0), (2.0; 4.0)}
func1.equals(cloned) = true
После изменения func1[0].y = 100:
func1 = {(0.0; 100.0), (1.0; 1.0), (2.0; 4.0)}
cloned = {(0.0; 0.0), (1.0; 1.0), (2.0; 4.0)}
func1.equals(cloned) = false
```

3. Тестирование LinkedListTabulatedFunction:

```
func1 = {(0.0; 0.0), (1.0; 1.0), (2.0; 4.0)}  
func2 = {(0.0; 0.0), (1.0; 1.0), (2.0; 4.0)}  
func3 = {(0.0; 0.0), (1.0; 0.0), (2.0; 0.0)}  
func1.equals(func2) = true  
func1.equals(func3) = false  
func1.hashCode() = 1048579  
func2.hashCode() = 1048579  
func3.hashCode() = 2146435075  
cloned = {(0.0; 0.0), (1.0; 1.0), (2.0; 4.0)}  
func1.equals(cloned) = true  
После изменения func1[0].y = 100:  
func1 = {(0.0; 100.0), (1.0; 1.0), (2.0; 4.0)}  
cloned = {(0.0; 0.0), (1.0; 1.0), (2.0; 4.0)}  
func1.equals(cloned) = false
```

4. Сравнение Array и LinkedList реализаций:

```
ArrayTabulatedFunction: {(0.0; 0.0), (1.0; 1.0), (2.0; 4.0)}  
LinkedListTabulatedFunction: {(0.0; 0.0), (1.0; 1.0), (2.0; 4.0)}  
arrayFunc.equals(listFunc) = true  
listFunc.equals(arrayFunc) = true
```

5. Тестирование клонирования через интерфейс:

Исходные функции равны своим клонам:

```
func1.equals(clone1) = true  
func2.equals(clone2) = true
```

После модификации исходных функций:

```
func1.equals(clone1) = false  
func2.equals(clone2) = false
```

6. Проверка согласованности equals() и hashCode():

Проверка согласованности для FunctionPoint:

```
p1.equals(p2) = true  
p1.hashCode() == p2.hashCode() = true  
p1.equals(p3) = false  
p1.hashCode() == p3.hashCode() = false
```

Проверка согласованности для функций:

```
func1.equals(func2) = true  
func1.hashCode() == func2.hashCode() = true  
func1.equals(func3) = false  
func1.hashCode() == func3.hashCode() = false
```