Лабораторная работа №5

Коновалов Сергей Сергеевич 6204-010302D

Содержание

Задание 1

Переопределил методы toString(), equals(), hashCode() и clone() в классе FunctionPoint. Обеспечил корректное сравнение чисел с плавающей точкой и глубокое клонирование.

```
@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;

    // Использование машинного эпсилона для сравнения double значений
    private static final double EPSILON = 1e-10;

    return Math.abs(that.x - x) < EPSILON &&
           Math.abs(that.y - y) < EPSILON;
}

@Override
public int hashCode() {
    return Objects.hash(x, y);
}

@Override
public Object clone() {
    try {
        return super.clone();
    } catch (CloneNotSupportedException e) {
        return new FunctionPoint(this);
    }
}
```

Задание 2

Переопределил методы в классе ArrayTabulatedFunction. Реализовал оптимизированное сравнение для объектов одного класса и глубокое клонирование.

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

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

    // Определяем машинный эпсилон для сравнения
    private static final double EPSILON = 1e-10;

    // Если объект является ArrayTabulatedFunction, используем оптимизированное сравнение
    if (o instanceof ArrayTabulatedFunction) {
        ArrayTabulatedFunction other = (ArrayTabulatedFunction) o;

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

        // Прямое сравнение массивов точек с использованием эпсилона
        for (int i = 0; i < pointsCount; i++) {
            FunctionPoint thisPoint = this.points[i];
            FunctionPoint otherPoint = other.points[i];

            // Сравнение координат с учетом погрешности
            if (Math.abs(thisPoint.getX() - otherPoint.getX()) >= EPSILON ||
                Math.abs(thisPoint.getY() - otherPoint.getY()) >= EPSILON) {
                return false;
            }
        }
        return true;
    }

    // Если объект реализует TabulatedFunction, но не ArrayTabulatedFunction
    if (o instanceof TabulatedFunction) {
        TabulatedFunction other = (TabulatedFunction) o;

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

```
        // Сравнение через методы интерфейса с использованием эпсилона
        for (int i = 0; i < pointsCount; i++) {
            FunctionPoint thisPoint = this.getPoint(i);
            FunctionPoint otherPoint = other.getPoint(i);

            // Сравнение координат с учетом погрешности
            if (Math.abs(thisPoint.getX() - otherPoint.getX()) >= EPSILON ||
                Math.abs(thisPoint.getY() - otherPoint.getY()) >= EPSILON) {
                return false;
            }
        }
        return true;
    }

    return false;
}
@Override
public int hashCode() {
    int hash = pointsCount; // Включаем количество точек в хэш

    // Комбинируем хэш-коды всех точек
    for (int i = 0; i < pointsCount; i++) {
        hash ^= points[i].hashCode();
    }

    return hash;
}

@Override
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();
        }
        cloned.pointsCount = this.pointsCount;

        return cloned;
    } catch (CloneNotSupportedException e) {
        // Этот случай не должен произойти, но на всякий случай
        // создаем копию через конструктор
        return new ArrayTabulatedFunction(this.points);
    }
}
```

Задание 3

Переопределил методы в классе LinkedListTabulatedFunction. Использовал стратегию "пересборки" для клонирования вместо классического глубокого клонирования.

```
@Override
public String toString() {
    StringBuilder sb = new StringBuilder();
    sb.append("{");
    FunctionNode currentNode = head.getNext();
    for (int i = 0; i < pointsCount; i++) {
        if (i > 0) sb.append(", ");
        FunctionPoint point = currentNode.getPoint();
        sb.append("(").append(point.getX())
          .append("; ").append(point.getY()).append(")");
        currentNode = currentNode.getNext();
    }
    sb.append("}");
    return sb.toString();
}
```

```java
@Override
public boolean equals(Object o) {
    if (this == o) return true;
    if (o == null) return false;

    // Определяем машинный эпсилон для сравнения
    private static final double EPSILON = 1e-10;

    // Если объект является LinkedListTabulatedFunction, используем оптимизированное сравнение
    if (o instanceof LinkedListTabulatedFunction) {
        LinkedListTabulatedFunction other = (LinkedListTabulatedFunction) o;

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

        // Прямое сравнение узлов списка с использованием эпсилона
        FunctionNode thisNode = this.head.getNext();
        FunctionNode otherNode = other.head.getNext();

        for (int i = 0; i < pointsCount; i++) {
            FunctionPoint thisPoint = thisNode.getPoint();
            FunctionPoint otherPoint = otherNode.getPoint();

            // Сравнение координат с учетом погрешности
            if (Math.abs(thisPoint.getX() - otherPoint.getX()) >= EPSILON ||
                Math.abs(thisPoint.getY() - otherPoint.getY()) >= EPSILON) {
                return false;
            }
            thisNode = thisNode.getNext();
            otherNode = otherNode.getNext();
        }
        return true;
    }

    // Если объект реализует TabulatedFunction, но не LinkedListTabulatedFunction
    if (o instanceof TabulatedFunction) {
        TabulatedFunction other = (TabulatedFunction) o;

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

        // Сравнение через методы интерфейса с использованием эпсилона
        for (int i = 0; i < pointsCount; i++) {
            FunctionPoint thisPoint = this.getPoint(i);
            FunctionPoint otherPoint = other.getPoint(i);

            // Сравнение координат с учетом погрешности
            if (Math.abs(thisPoint.getX() - otherPoint.getX()) >= EPSILON ||
                Math.abs(thisPoint.getY() - otherPoint.getY()) >= EPSILON) {
                return false;
            }
        }
        return true;
    }

    return false;
}
@Override
public int hashCode() {
    int hash = pointsCount; // Включаем количество точек в хэш

    // Комбинируем хэш-коды всех точек
    FunctionNode currentNode = head.getNext();
    for (int i = 0; i < pointsCount; i++) {
        hash ^= currentNode.getPoint().hashCode();
        currentNode = currentNode.getNext();
    }

    return hash;
}
```

```java
@Override
public Object clone() {
    try {
        LinkedListTabulatedFunction cloned = (LinkedListTabulatedFunction) super.clone();

        // Создаем новую голову для клонированного списка
        cloned.head = new FunctionNode(null, null, null);
        cloned.head.setPrev(cloned.head);
        cloned.head.setNext(cloned.head);
        cloned.pointsCount = 0;
        cloned.lastAccessedNode = cloned.head;
        cloned.lastAccessedIndex = -1;

        // Пересобираем список, создавая копии точек
        FunctionNode currentNode = this.head.getNext();
        for (int i = 0; i < this.pointsCount; i++) {
            FunctionPoint pointCopy = (FunctionPoint) currentNode.getPoint().clone();
            FunctionNode newNode = cloned.addNodeToTail();
            newNode.setPoint(pointCopy);
            currentNode = currentNode.getNext();
        }

        return cloned;
    } catch (CloneNotSupportedException e) {
        // Этот случай не должен произойти, но на всякий случай
        // создаем копию через конструктор с массивом точек
        FunctionPoint[] pointsArray = new FunctionPoint[pointsCount];
        FunctionNode currentNode = head.getNext();
        for (int i = 0; i < pointsCount; i++) {
            pointsArray[i] = (FunctionPoint) currentNode.getPoint().clone();
            currentNode = currentNode.getNext();
        }
        return new LinkedListTabulatedFunction(pointsArray);
    }
}
```

Задание 4

Добавил метод clone() в интерфейс TabulatedFunction и наследование от Cloneable для обеспечения клонируемости всех объектов.

```java
Object clone();
```

Задание 5

Написал комплексные тесты для проверки работы всех переопределенных методов. Проверил:

- Корректность форматирования в toString()
- Правильность сравнения в equals() для одинаковых и разных объектов
- Согласованность equals() и hashCode()
- Глубокое клонирование в clone()

```
================================================================================
ЗАДАНИЕ 5: ТЕСТИРОВАНИЕ toString(), equals(), hashCode(), clone()
================================================================================

1. ТЕСТИРОВАНИЕ toString()
----------------------------------------
ArrayTabulatedFunction.toString():
  {(0.0; 1.0), (1.0; 3.0), (2.0; 5.0), (3.0; 7.0)}
LinkedListTabulatedFunction.toString():
  {(0.0; 1.0), (1.0; 3.0), (2.0; 5.0), (3.0; 7.0)}
Короткая функция (3 точки):
  {(0.0; 0.0), (0.5; 0.25), (1.0; 1.0)}
```

```
2. ТЕСТИРОВАНИЕ equals()
--------------------------------------
Сравнение одинаковых ArrayTabulatedFunction:
  array1.equals(array2): true
  array2.equals(array1): true
Сравнение одинаковых LinkedListTabulatedFunction:
  list1.equals(list2): true
  list2.equals(list1): true
Сравнение разных ArrayTabulatedFunction:
  array1.equals(arrayDifferent): false
Сравнение ArrayTabulatedFunction и LinkedListTabulatedFunction:
  array1.equals(list1): true
  list1.equals(array1): true
Сравнение с разным количеством точек:
  array1.equals(shortArray): false
Сравнение с null:
  array1.equals(null): false
Сравнение с другим типом объекта:
  array1.equals("строка"): false

3. ТЕСТИРОВАНИЕ hashCode()
--------------------------------------
Хэш-коды одинаковых ArrayTabulatedFunction:
  array1.hashCode(): -16121852
  array2.hashCode(): -16121852
  Совпадают: true
Хэш-коды одинаковых LinkedListTabulatedFunction:
  list1.hashCode(): -16121852
  list2.hashCode(): -16121852
  Совпадают: true
Хэш-коды Array и LinkedList с одинаковыми точками:
  array1.hashCode(): -16121852
  list1.hashCode(): -16121852
  Совпадают: true

Тестирование изменения объекта:
  Исходный hashCode: -16121852
  После изменения Y[1] на 0.001: 1805565942
  Хэш-код изменился: true
  Разница: 1821687794

Тестирование с разным количеством точек:
  func3points.hashCode(): -32504894
  func2points.hashCode(): -33554430
  Совпадают: false
```

```
4. ТЕСТИРОВАНИЕ clone()
--------------------------------------
ArrayTabulatedFunction.clone():
  Исходная функция: {(0.0; 0.0), (1.0; 1.0), (2.0; 4.0), (3.0; 9.0)}
  Клонированная функция: {(0.0; 0.0), (1.0; 1.0), (2.0; 4.0), (3.0; 9.0)}
  equals(): true
  == : false
  После изменения исходной функции:
    Исходная: {(0.0; 0.0), (1.0; 999.0), (2.5; 4.0), (3.0; 9.0)}
    Клон: {(0.0; 0.0), (1.0; 1.0), (2.0; 4.0), (3.0; 9.0)}
    Клон не изменился: true

LinkedListTabulatedFunction.clone():
  Исходная функция: {(0.0; 0.0), (1.0; 1.0), (2.0; 4.0), (3.0; 9.0)}
  Клонированная функция: {(0.0; 0.0), (1.0; 1.0), (2.0; 4.0), (3.0; 9.0)}
  equals(): true
  == : false
  После изменения исходной функции:
    Исходная: {(0.0; 0.0), (1.0; 888.0), (3.0; 9.0)}
    Клон: {(0.0; 0.0), (1.0; 1.0), (2.0; 4.0), (3.0; 9.0)}
    Клон не изменился: true

Клонирование через интерфейс TabulatedFunction:
  ArrayTabulatedFunction.clone() тип: ArrayTabulatedFunction
  LinkedListTabulatedFunction.clone() тип: LinkedListTabulatedFunction
  Array clone equals original: true
  List clone equals original: true
  После изменения исходных функций:
    Array clone Y[0]: 10.0 (должно быть 10.0)
    List clone Y[0]: 10.0 (должно быть 10.0)
    Глубокое клонирование работает: true
PS D:\Other\Study\GitHubLab5>
```