Отчёт по лабораторной работе №5

Студент: Прокопенко А. П.

Группа: 6201-120303D

Задание 1: Работа с классом FunctionPoint

Реализованные методы:

Метод toString(): Возвращает текстовое описание точки в формате "(x; y)", где x и y - координаты точки.

```
public String toString() {
    return "(" + x + "; " + y + ")";
}
```

Метод equals(): Сравнивает две точки с учетом точности чисел с плавающей точкой. Возвращает true только если обе координаты совпадают.

```
public boolean equals(Object o) {
    if (this == o) return true;
    if (o == null || getClass() != o.getClass()) return false;
    FunctionPoint that = (FunctionPoint) o;
    return Double.compare(that.x, x) == 0 && Double.compare(that.y, y) == 0;
}
```

Метод hashCode(): Вычисляет хэш-код на основе координат точки, используя преобразование double в long и операцию XOR.

```
public int hashCode() {
    long xBits = Double.doubleToLongBits(x);
    long yBits = Double.doubleToLongBits(y);

    int xHash = (int)(xBits ^ (xBits >>> 32));
    int yHash = (int)(yBits ^ (yBits >>> 32));

    return xHash ^ yHash;
}
```

Метод clone(): Создает точную копию объекта точки.

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

Задание 2: Работа с классом ArrayTabulatedFunction

Реализованные методы:

Метод toString(): Формирует строковое представление табулированной функции в виде набора точек, заключенных в фигурные скобки.

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

Метод equals(): Сравнивает две табулированные функции. Оптимизирован для случаев, когда сравниваемые объекты принадлежат одному классу.

```java
public boolean equals(Object o) {
    if (this == o) return true;
    if (!(o instanceof TabulatedFunction)) return false;

    TabulatedFunction other = (TabulatedFunction) o;

    // Быстрая проверка для ArrayTabulatedFunction
    if (o instanceof ArrayTabulatedFunction) {
        ArrayTabulatedFunction otherArray = (ArrayTabulatedFunction) o;

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

        // Прямое сравнение массивов точек
        for (int i = 0; i < pointsCount; i++) {
            if (!this.points[i].equals(otherArray.points[i])) {
                return false;
            }
        }
        return true;
    }

    // Общий случай для любого TabulatedFunction
    if (this.getPointsCount() != other.getPointsCount()) return false;

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

        if (!thisPoint.equals(otherPoint)) {
            return false;
        }
    }
    return true;
}
```

Метод hashCode(): Вычисляет хэш-код на основе хэш-кодов всех точек и количества точек.

```
public int hashCode() {
    int hash = pointsCount; // Начинаем с количества точек

    for (int i = 0; i < pointsCount; i++) {
        // Комбинируем хэш текущей точки с общим хэшем через XOR
        hash ^= points[i].hashCode();
    }

    return hash;
}
```

Метод clone(): Реализует глубокое клонирование, создавая копии всех точек массива.

```
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();
        }

        // pointsCount примитивный тип - копируется по значению
        cloned.pointsCount = this.pointsCount;

        return cloned;
    } catch (CloneNotSupportedException e) {
        throw new AssertionError( message: "Клонирование не поддерживается", e);
    }
}
```

Задание 3: Работа с классом LinkedListTabulatedFunction

Реализованные методы:

Метод toString(): Формирует строковое представление связного списка точек.

```
public String toString() {
    StringBuilder sb = new StringBuilder();
    sb.append("{");

    FunctionNode current = head.getNext();
    for (int i = 0; i < size; i++) {
        FunctionPoint point = current.getPoint();
        sb.append("(").append(point.getX())
                .append("; ").append(point.getY()).append(")");

        if (i < size - 1) {
            sb.append(", ");
        }
        current = current.getNext();
    }

    sb.append("}");
    return sb.toString();
}
```

Метод equals(): Сравнивает две функции, оптимизируя сравнение для объектов одного класса.

```java
public boolean equals(Object o) {
    if (this == o) return true;
    if (!(o instanceof TabulatedFunction)) return false;

    TabulatedFunction other = (TabulatedFunction) o;

    // Быстрая проверка для LinkedListTabulatedFunction
    if (o instanceof LinkedListTabulatedFunction) {
        LinkedListTabulatedFunction otherList = (LinkedListTabulatedFunction) o;

        if (this.size != otherList.size) return false;

        // Прямое сравнение узлов списка
        FunctionNode thisCurrent = this.head.getNext();
        FunctionNode otherCurrent = otherList.head.getNext();

        for (int i = 0; i < size; i++) {
            if (!thisCurrent.getPoint().equals(otherCurrent.getPoint())) {
                return false;
            }
            thisCurrent = thisCurrent.getNext();
            otherCurrent = otherCurrent.getNext();
        }
        return true;
    }

    // Общий случай для любого TabulatedFunction
    if (this.getPointsCount() != other.getPointsCount()) return false;

    for (int i = 0; i < size; i++) {
        FunctionPoint thisPoint = this.getPoint(i);
        FunctionPoint otherPoint = other.getPoint(i);

        if (!thisPoint.equals(otherPoint)) {
            return false;
        }
    }
    return true;
}
```

Метод hashCode(): Вычисляет хэш-код путем комбинирования хэш-кодов всех узлов списка.

```
public int hashCode() {
    int hash = size; // Начинаем с количества точек

    FunctionNode current = head.getNext();
    for (int i = 0; i < size; i++) {
        // Комбинируем хэш текущей точки с общим хэшем через XOR
        hash ^= current.getPoint().hashCode();
        current = current.getNext();
    }

    return hash;
}
```

Метод clone(): Реализует глубокое клонирование через пересборку списка.

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

        // Инициализируем новый пустой список
        cloned.initializeList();

        if (size > 0) {
            // Создаем массив точек из исходного списка
            FunctionPoint[] pointsArray = new FunctionPoint[size];
            FunctionNode current = head.getNext();

            for (int i = 0; i < size; i++) {
                pointsArray[i] = new FunctionPoint(current.getPoint());
                current = current.getNext();
            }

            // "Пересобираем" новый список вручную
            FunctionNode firstNode = new FunctionNode(pointsArray[0]);
            cloned.head.setNext(firstNode);
            cloned.head.setPrev(firstNode);
            firstNode.setPrev(cloned.head);
            firstNode.setNext(cloned.head);
            cloned.size = 1;
```

```
            // Добавляем остальные узлы
            FunctionNode lastNode = firstNode;
            for (int i = 1; i < pointsArray.length; i++) {
                FunctionNode newNode = new FunctionNode(pointsArray[i]);

                // Вставляем в конец
                lastNode.setNext(newNode);
                newNode.setPrev(lastNode);
                newNode.setNext(cloned.head);
                cloned.head.setPrev(newNode);

                lastNode = newNode;
                cloned.size++;
            }
        }

        // Сбрасываем кэш доступа
        cloned.lastAccessedNode = cloned.head;
        cloned.lastAccessedIndex = -1;

        return cloned;

    } catch (CloneNotSupportedException e) {
        throw new AssertionError( message: "Клонирование не поддерживается", e);
    }
}
```

Задание 4: Модификация интерфейса

Интерфейс TabulatedFunction был расширен методом clone(), что делает все его реализации клонируемыми с точки зрения JVM.

Результаты тестирования

```
1. СОЗДАНИЕ ТЕСТОВЫХ ФУНКЦИЙ (x^2 - x + 41)
Созданы функции с точками:
  (0,0; 41,0)  (2,0; 43,0)  (4,0; 53,0)  (6,0; 71,0)  (8,0; 97,0)

2. ТЕСТИРОВАНИЕ toString()
ArrayTabulatedFunction: {(0.0; 41.0), (2.0; 43.0), (4.0; 53.0), (6.0; 71.0), (8.0; 97.0)}
LinkedListTabulatedFunction: {(0.0; 41.0), (2.0; 43.0), (4.0; 53.0), (6.0; 71.0), (8.0; 97.0)}

3. ТЕСТИРОВАНИЕ equals()
arrayFunc.equals(arrayFunc2): true
linkedListFunc.equals(linkedListFunc2): true
arrayFunc.equals(linkedListFunc): true
arrayFunc.equals(differentArrayFunc): false
linkedListFunc.equals(differentLinkedListFunc): false
arrayFunc.equals(null): false
arrayFunc.equals("строка"): false

4. ТЕСТИРОВАНИЕ hashCode()
arrayFunc.hashCode(): 1080688645
arrayFunc2.hashCode(): 1080688645
linkedListFunc.hashCode(): 1080688645
linkedListFunc2.hashCode(): 1080688645
differentArrayFunc.hashCode(): 1080655877
differentLinkedListFunc.hashCode(): 1080655877

Проверка согласованности equals() и hashCode():
arrayFunc.equals(arrayFunc2) && arrayFunc.hashCode() == arrayFunc2.hashCode(): true
linkedListFunc.equals(linkedListFunc2) && linkedListFunc.hashCode() == linkedListFunc2.hashCode(): true

5. ТЕСТИРОВАНИЕ ИЗМЕНЕНИЯ ХЭШ-КОДА ПРИ ИЗМЕНЕНИИ ОБЪЕКТА
Исходный hashCode arrayFunc: 1080688645
hashCode после изменения Y на 0.001: -2064538170
hashCode после возврата исходного значения: 1080688645
```

```
5. ТЕСТИРОВАНИЕ ИЗМЕНЕНИЯ ХЭШ-КОДА ПРИ ИЗМЕНЕНИИ ОБЪЕКТА
Исходный hashCode arrayFunc: 1080688645
hashCode после изменения Y на 0.001: -2064538170
hashCode после возврата исходного значения: 1080688645

6. ТЕСТИРОВАНИЕ clone() И ГЛУБОКОГО КЛОНИРОВАНИЯ
Исходная arrayFunc: {(0.0; 41.0), (2.0; 43.0), (4.0; 53.0), (6.0; 71.0), (8.0; 97.0)}
Клон arrayClone: {(0.0; 41.0), (2.0; 43.0), (4.0; 53.0), (6.0; 71.0), (8.0; 97.0)}
Исходная linkedListFunc: {(0.0; 41.0), (2.0; 43.0), (4.0; 53.0), (6.0; 71.0), (8.0; 97.0)}
Клон linkedListClone: {(0.0; 41.0), (2.0; 43.0), (4.0; 53.0), (6.0; 71.0), (8.0; 97.0)}

arrayFunc.equals(arrayClone): true
linkedListFunc.equals(linkedListClone): true

7. ПРОВЕРКА ГЛУБОКОГО КЛОНИРОВАНИЯ
После изменения исходных функций:
Исходная arrayFunc: {(0.0; 41.0), (2.0; 100.0), (4.0; 53.0), (6.0; 71.0), (8.0; 97.0)}
Клон arrayClone: {(0.0; 41.0), (2.0; 43.0), (4.0; 53.0), (6.0; 71.0), (8.0; 97.0)}
Исходная linkedListFunc: {(0.0; 41.0), (2.0; 200.0), (4.0; 53.0), (6.0; 71.0), (8.0; 97.0)}
Клон linkedListClone: {(0.0; 41.0), (2.0; 43.0), (4.0; 53.0), (6.0; 71.0), (8.0; 97.0)}

arrayFunc.equals(arrayClone) после изменений: false
linkedListFunc.equals(linkedListClone) после изменений: false

Клон arrayClone не изменился: true
Клон linkedListClone не изменился: true

8. ДОПОЛНИТЕЛЬНЫЕ ТЕСТЫ
arrayFunc.equals(fewerPointsFunc) [разное количество точек]: false
arrayFunc.hashCode() == fewerPointsFunc.hashCode(): false
Глубокое клонирование FunctionPoint: original.x=15.0, clone.x=10.0

Process finished with exit code 0
```