Лабораторная работа №5

Выполнил: Магера Никита Алексеевич

Студент группы 6203-010302D

Ход выполнения

Задание 1

В 1 задании необходимо в классе FunctionPoint переопределить четыре метода. Метод toString() должен возвращать строку в формате "(x; y)", где x и y — координаты точки. Метод equals(Object o) должен корректно сравнивать точки: проверить, что объект не null и принадлежит тому же классу, после чего привести Object к FunctionPoint и сравнить координаты с допустимой погрешностью (т.е. Эпсилон). Метод hashCode() должен вычислять хэш-код на основе битового представления координат: преобразовать каждую координату double в long с помощью Double.doubleToLongBits(), затем смешать старшие и младшие 32 бита каждого long через XOR и объединить результаты для x и y. Метод clone() должен возвращать копию точки: для этого класс должен реализовывать интерфейс Cloneable, а метод должен вызывать super.clone().(Рис.1.)

```
// 5 лабораторная
// 1 задание переопределение методов
public String toString() {  new *
    return "(" + x +"," + y +")"; // возвращает текстовое описание точки
}
public boolean equals(Object o) {  new *
    // проверяем, чтобы не был передан null
    if (o == null) return false;
    // приводим Object к FunctionPoint
    FunctionPoint var = (FunctionPoint) o;
    double epsilon = 1e-10;
    return Math.abs(x - var.x) < epsilon && Math.abs(y - var.y) < epsilon;}
public int hashCode() {  new *
    // приводим в 64-битное представление
    long x_dbits = Double.doubleToLongBits(x);
    long y_dbits= Double.doubleToLongBits(y);

    int  x_hash= (int)(x_dbits ^ (x_dbits >>> 32));
    int y_hash = (int)(y_dbits ^ (y_dbits >>> 32));
    //  с помощью ^ смешивает по 32 бита с каждого элемента
    return x_hash ^ y_hash; }
// метод clone():
public Object clone() throws CloneNotSupportedException {  new *
    // возвращаем объект-копию для объекта точки
    return super.clone();
}
}
```

Рис. 1

Задание 2

Во втором задании в классе ArrayTabulatedFunction также нужно переопределить те же четыре метода. toString() должен формировать строку вида "{(x1; y1), (x2; y2), ...}", перебирая все точки массива. equals(Object o) должен проверять, является ли переданный объект реализацией TabulatedFunction, сравнивать количество точек, а затем — каждую точку с погрешностью. Если объект тоже ArrayTabulatedFunction, можно напрямую обращаться к его массиву points для ускорения. hashCode() должен вычисляться как XOR между количеством точек и хэш-кодами всех точек. clone() должен выполнять глубокое копирование: создать новый массив FunctionPoint и клонировать каждую точку в него, затем создать новый объект ArrayTabulatedFunction с этим массивом.(Рис.2-3)

```
// 5 задание 2 пункт
public String toString() {  new *
    String result = "{";
    // перебираем все точки массива
    for (int i = 0; i < pointsCount; i++) {
        // добавляем строковое представление текущей точки
        result += points[i]; // в этом случае уже автоматически вызывает points[i].toString()
        if (i != pointsCount - 1) result += ", ";  // если это не последняя точка, добавляем запятую и пробел
    }
    //записываем последнюю скобку
    result += "}";
    return result; // возвращаем итоговую строку
}
public boolean equals(Object o) {  new *  1 related problem
    if (o == null) return false;
    // проверка на табулированную функцию
    if (!(o instanceof TabulatedFunction))
        return false;
    TabulatedFunction o1 = (TabulatedFunction) o;
    // проверяем количество точек
    if (this.getPointsCount() != o1.getPointsCount()) {
        return false;
    }
    // сравниваем точки
    final double EPSILON = 1e-10;
    if (o instanceof ArrayTabulatedFunction) {
        ArrayTabulatedFunction arr = (ArrayTabulatedFunction) o;
        for (int i = 0; i < pointsCount; i++) {
            if (Math.abs(points[i].getX() - arr.points[i].getX()) >= EPSILON || Math.abs(points[i].getY() - arr.points[i].getY()) >= EPSILON) {
                return false;
            }
        }
    }
    else {
        for (int i = 0; i < pointsCount; i++) {
            FunctionPoint point = this.getPoint(i);
            FunctionPoint otherPoint = o1.getPoint(i);
            if (Math.abs(point.getX() - otherPoint.getX()) >= EPSILON || Math.abs(point.getY() - otherPoint.getY()) >= EPSILON) {
                return false;
            }
        }
    }
```

Рис.2

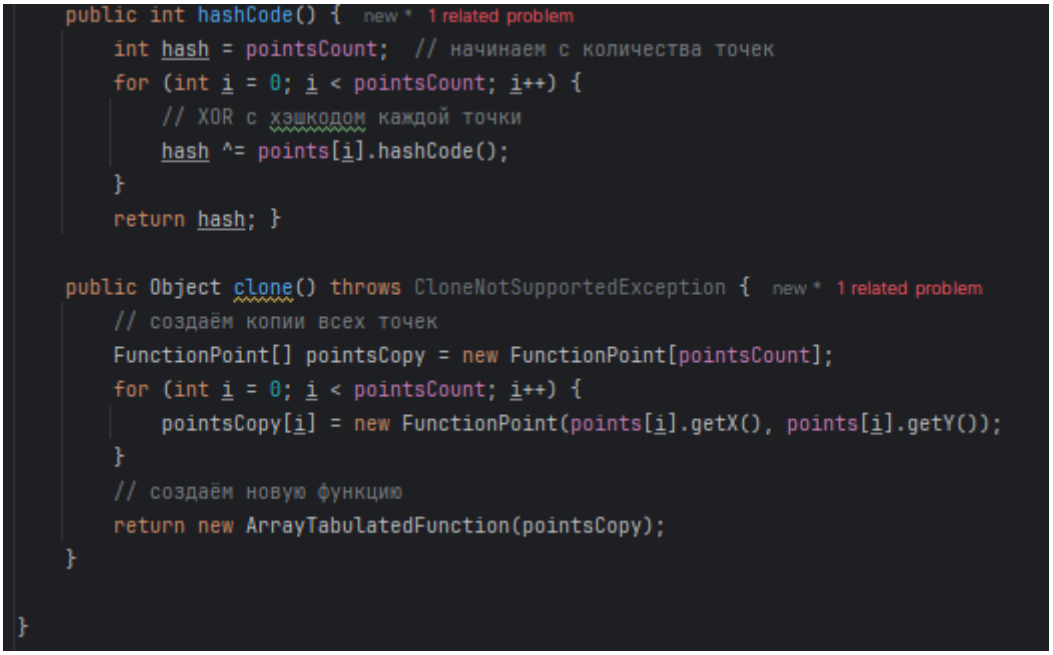


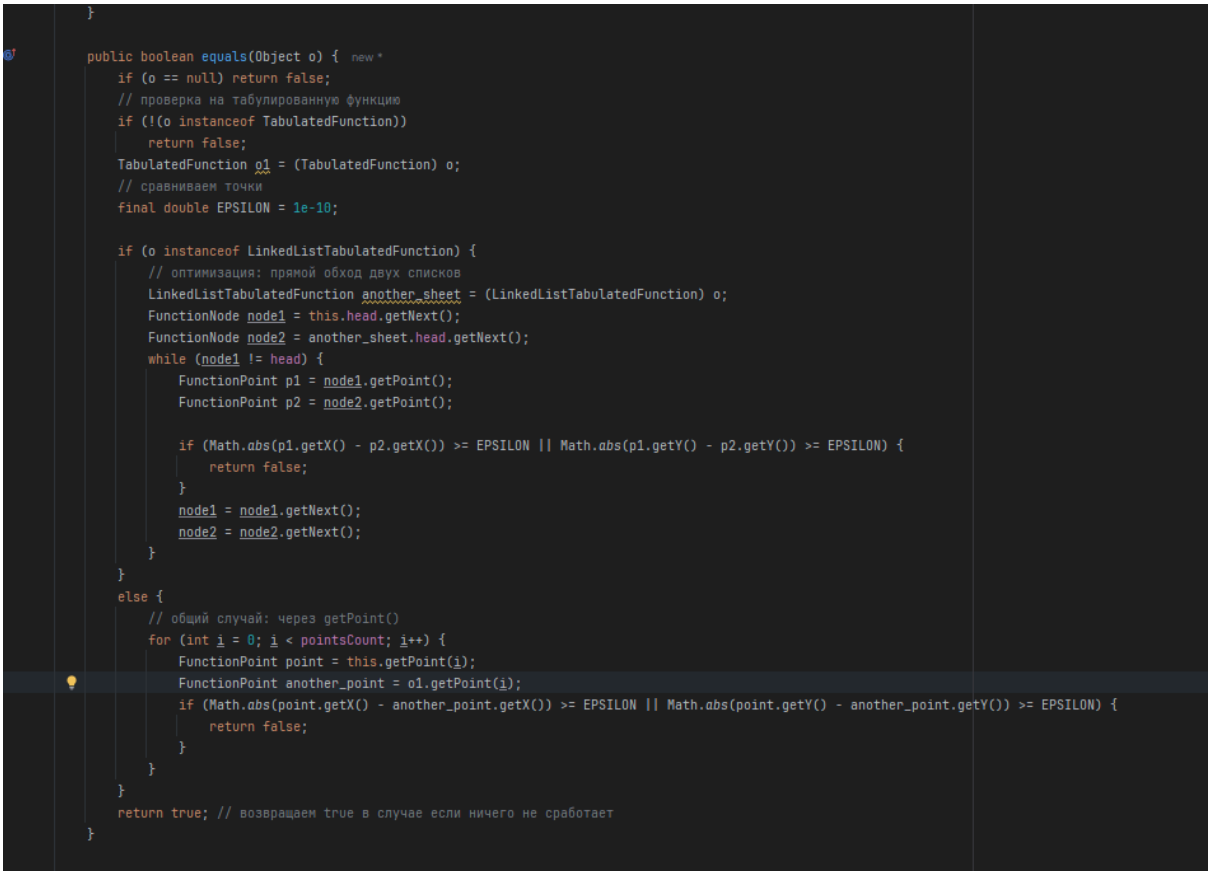
Рис.3

Задание 3

В третьем задании в классе LinkedListTabulatedFunction нужно переопределенить методы с учетом структуры связного списка. toString() обходит узлы списка от head.getNext() до возврата к head. equals(Object o) работает аналогично ArrayTabulatedFunction, но при сравнении с другим LinkedListTabulatedFunction выполняется прямой обход узлов обоих списков. hashCode() вычисляется обходом списка: начальное значение — pointsCount, затем XOR с хэш-кодом точки каждого узла. clone() выполняет глубокое копирование путем копирования всех точек в массив и передачи его в конструктор LinkedListTabulatedFunction, который строит новый список. (Рис 4-5)

```java
// 5 лаболаторная 3 пункт (переделанные для этого класса из Array)
public String toString() {  new *
    String result = "{";
    // начинаем с первого узла после головы
    FunctionNode current = head.getNext();
    int count = 0;
    // перебираем все узлы списка пока не вернёмся к голове
    while (current != head) {
        result += current.getPoint(); // автоматически вызывается toString()
        count++;
        // если это не последняя точка, добавляем запятую и пробел
        if (count < pointsCount)
            result += ", ";
        // переходим к следующему узлу
        current = current.getNext();
    }

    // записываем последнюю скобку
    result += "}";
    return result; // возвращаем итоговую строку
}

public int hashCode() {  new *
    int hash = pointsCount;  // начало с количества точек
    // обходим все узлы списка
    FunctionNode curent = head.getNext();
    while (curent != head) {
        hash ^= curent.getPoint().hashCode(); // иксор с хэшкодом каждой точки
        curent = curent.getNext();
    }
    return hash;
}
public Object clone() throws CloneNotSupportedException {  new *  1 related problem
    // создаём массив для копий точек
    FunctionPoint[] pointsCopy = new FunctionPoint[pointsCount];
    // копируем все точки из списка
    FunctionNode massiv = head.getNext();
    for (int i = 0; i < pointsCount; i++) {
        pointsCopy[i] = new FunctionPoint(massiv.getPoint().getX(), massiv.getPoint().getY());massiv = massiv.getNext();
    }
    // создаём новую функцию через конструктор с массивом
    return new LinkedListTabulatedFunction(pointsCopy);
}
```

Рис.4



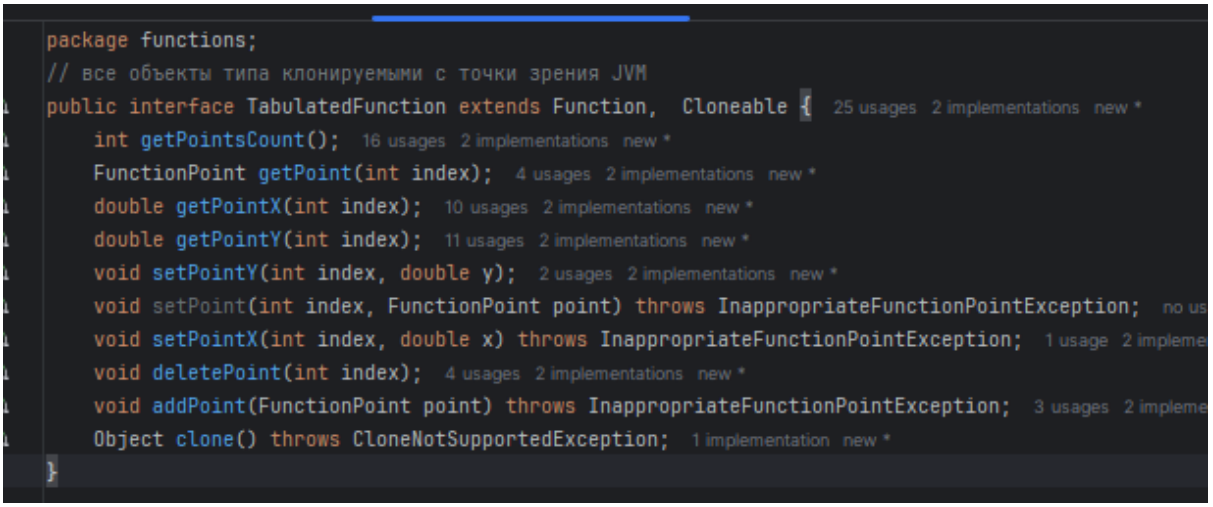
```
}

public boolean equals(Object o) {  new *
    if (o == null) return false;
    // проверка на табулированную функцию
    if (!(o instanceof TabulatedFunction))
        return false;
    TabulatedFunction o1 = (TabulatedFunction) o;
    // сравниваем точки
    final double EPSILON = 1e-10;

    if (o instanceof LinkedListTabulatedFunction) {
        // оптимизация: прямой обход двух списков
        LinkedListTabulatedFunction another_sheet = (LinkedListTabulatedFunction) o;
        FunctionNode node1 = this.head.getNext();
        FunctionNode node2 = another_sheet.head.getNext();
        while (node1 != head) {
            FunctionPoint p1 = node1.getPoint();
            FunctionPoint p2 = node2.getPoint();

            if (Math.abs(p1.getX() - p2.getX()) >= EPSILON || Math.abs(p1.getY() - p2.getY()) >= EPSILON) {
                return false;
            }
            node1 = node1.getNext();
            node2 = node2.getNext();
        }
    }
    else {
        // общий случай: через getPoint()
        for (int i = 0; i < pointsCount; i++) {
            FunctionPoint point = this.getPoint(i);
            FunctionPoint another_point = o1.getPoint(i);
            if (Math.abs(point.getX() - another_point.getX()) >= EPSILON || Math.abs(point.getY() - another_point.getY()) >= EPSILON) {
                return false;
            }
        }
    }
    return true; // возвращаем true в случае если ничего не сработает
}
```

Рис.5

Задание 4

В 4 задании необходимо, чтобы все объекты TabulatedFunction были клонируемыми с точки зрения JVM, в интерфейс TabulatedFunction. добавлено объявление метода clone() (с throws CloneNotSupportedException) и унаследован интерфейс Cloneable (объявлено "extends Function, Cloneable"). Это гарантирует, что все классы, реализующие TabulatedFunction, автоматически становятся Cloneable.

(Рис. 6)



```
package functions;
// все объекты типа клонируемыми с точки зрения JVM
public interface TabulatedFunction extends Function,  Cloneable {  25 usages  2 implementations  new *
    int getPointsCount();  16 usages  2 implementations  new *
    FunctionPoint getPoint(int index);  4 usages  2 implementations  new *
    double getPointX(int index);  10 usages  2 implementations  new *
    double getPointY(int index);  11 usages  2 implementations  new *
    void setPointY(int index, double y);  2 usages  2 implementations  new *
    void setPoint(int index, FunctionPoint point) throws InappropriateFunctionPointException;  no us
    void setPointX(int index, double x) throws InappropriateFunctionPointException;  1 usage  2 impleme
    void deletePoint(int index);  4 usages  2 implementations  new *
    void addPoint(FunctionPoint point) throws InappropriateFunctionPointException;  3 usages  2 impleme
    Object clone() throws CloneNotSupportedException;  1 implementation  new *
}
```

Рис(6

Задание 5

В 5 задании созданы объекты ArrayTabulatedFunction и LinkedListTabulatedFunction с одинаковыми наборами точек. Метод toString() выводит строковое представление объектов в требуемом формате. Метод equals() корректно работает для одинаковых и различающихся объектов как одного, так и разных классов. Метод hashCode() выводит значения для всех объектов; проверена согласованность с equals(): для равных объектов хэш-коды совпадают. При незначительном изменении координаты точки хэш-код изменяется. Метод clone() протестирован для обоих классов; после клонирования изменение исходного объекта не затрагивает клон, что подтверждает глубокое копирование. (Рис.7)

```
5 лабоаторная тест
Array: {(0.0,3.0), (1.0,4.0), (2.0,5.0)}
LinkedList: {(0.0,3.0), (1.0,4.0), (2.0,5.0)}
Array==Array: true
Array==List: true
Hash Array: 1073479683
Hash Array2: 1073479683
Hash List: 1073479683
equals/hashCode contract: true
Hash before: 1073479683
Hash after: 1366003265
Hash changed: true
Array clone unchanged: true
List clone unchanged: true
```

Рис.7