

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.)

```
// equals
public boolean equals(Object o) { Nikitos100-15 *
    // проверка на тот же объект
    if (this == o) return true;
    // проверка на null
    if (o == null) return false;
    FunctionPoint that = (FunctionPoint) o;

    // Double.compare()
    return (Double.compare(that.x, x) == 0 && Double.compare(that.y, y) == 0);
}

// hashCode
public int hashCode() { Nikitos100-15 *
    int result = 17;
    //Double.hashCode()
    result = 31 * result + Double.hashCode(x);
    result = 31 * result + Double.hashCode(y);
    return result;
}

// метод clone():
public Object clone() throws CloneNotSupportedException { Nikitos100-15
    // 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() {
    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) {
    if (this == o) return true;
    // проверка на табулированную функцию
    if (!(o instanceof TabulatedFunction)) return false;
    TabulatedFunction that = (TabulatedFunction) o;
    // проверяем количество точек
    if (this.getPointsCount() != that.getPointsCount()) return false;

    // сравнение точек через equals - т.е. делегируем
    for (int i = 0; i < pointsCount; i++) {
        if (!this.getPoint(i).equals(that.getPoint(i))) {
            return false;
        }
    }
    return true;
}
```

Рис.2

```

public int hashCode() { new * 1 related problem
    int hash = pointsCount; // начинаем с количества точек
    for (int i = 0; i < pointsCount; i++) {
        // XOR с хэш-кодом каждой точки
        hash ^= points[i].hashCode();
    }
    return hash; }

public Object clone() throws CloneNotSupportedException { new * 1 related problem
    // создаём копии всех точек
    FunctionPoint[] pointsCopy = new FunctionPoint[pointsCount];
    for (int i = 0; i < pointsCount; i++) {
        pointsCopy[i] = new FunctionPoint(points[i].getX(), points[i].getY());
    }
    // создаём новую функцию
    return new ArrayTabulatedFunction(pointsCopy);
}
}

```

Рис.3

Задание 3

В третьем задании в классе `LinkedListTabulatedFunction` нужно переопределить 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)

```

// 5 лабораторная 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 current = head.getNext();
    while (current != head) {
        hash ^= current.getPoint().hashCode(); // иксор с хэшкой каждой точки
        current = current.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) { Nikitos100-15 *
    if (this == o) return true;
    // проверка на табулированную функцию
    if (!(o instanceof TabulatedFunction)) return false;

    TabulatedFunction that = (TabulatedFunction) o;
    // сравниваем точки
    if (this.getPointsCount() != that.getPointsCount()) return false;

    // сравнение точек через equals - делегируем
    for (int i = 0; i < pointsCount; i++) {
        if (!this.getPoint(i).equals(that.getPoint(i))) {
            return false;
        }
    }
    return true;
}

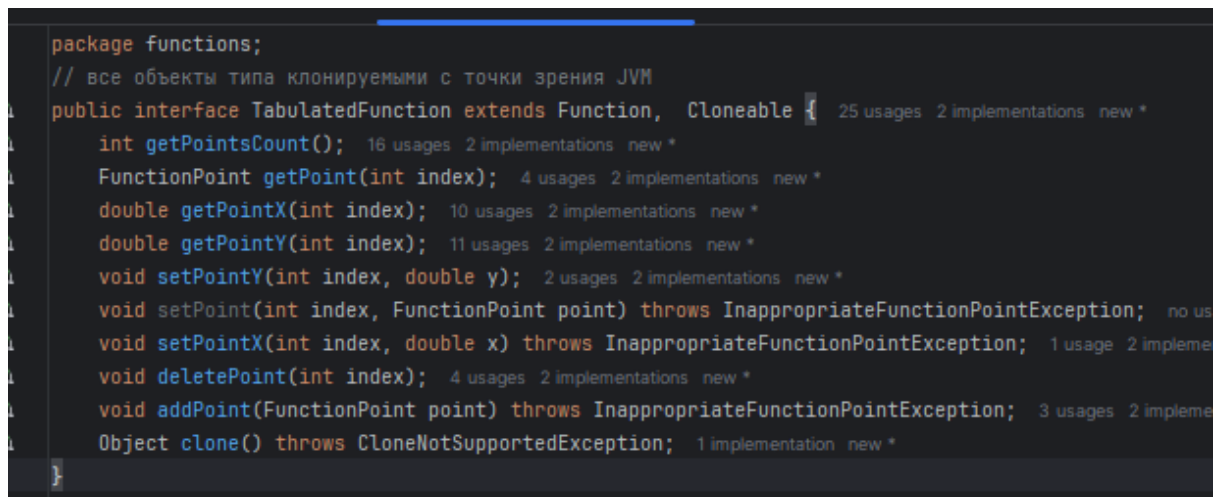
```

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 {
    int getPointsCount();
    FunctionPoint getPoint(int index);
    double getPointX(int index);
    double getPointY(int index);
    void setPointY(int index, double y);
    void setPoint(int index, FunctionPoint point) throws InappropriateFunctionPointException;
    void setPointX(int index, double x) throws InappropriateFunctionPointException;
    void deletePoint(int index);
    void addPoint(FunctionPoint point) throws InappropriateFunctionPointException;
    Object clone() throws CloneNotSupportedException;
}
```

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 лабораторная 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: -1103347758
Hash Array2: -1103347758
Hash List: -1103347758
equals/hashCode contract: true
Hash before: -1103347758
Hash after: -1400123888
Hash changed: true
Array clone unchanged: true
List clone unchanged: true
```

Рис.7