

Лабораторная работа №5

Манякин Степан 6204-010302D

Задание 1

Необходимо переопределить в классе FunctionPoint следующие методы:

- **String toString():** Должен возвращать текстовое описание точки.
- **boolean equals(Object o):** Должен возвращать true тогда и только тогда, когда переданный объект также является точкой и его координаты в точности совпадают с координатами объекта, у которого вызывается метод
- **int hashCode():** Должен возвращать значение хэш-кода для объекта точки.
- **Object clone():** Должен возвращать объект-копию для объекта точки.

```
package functions;

import java.io.Serializable;

public class FunctionPoint implements Serializable {
    private double x; //координата по оси x
    private double y; //координата по оси y

    public FunctionPoint(double x, double y) { //конструктор с двумя
параметрами
        this.x = x;
        this.y = y;
    }

    public FunctionPoint(FunctionPoint point) { //конструктор копирования
        this.x = point.x;
        this.y = point.y;
    }

    public FunctionPoint() { //конструктор по умолчанию (0;0)
        this.x = 0;
        this.y = 0;
    }

    public double getX() { //геттер для x
        return x;
    }

    public void setX(double x) { //сеттер для x
        this.x = x;
    }

    public double getY() { //геттер для y
        return y;
    }

    public void setY(double y) { //сеттер для y
        this.y = y;
    }

    // Переопределение метода toString()
    @Override
    public String toString() {
        return "(" + x + ";" + y + ")";
    }
}
```

```

// Переопределение метода equals()
@Override
public boolean equals(Object o) {
    if (this == o) return true;
    if (o == null || getClass() != o.getClass()) return false;

    FunctionPoint that = (FunctionPoint) o;

    // Сравнение с учетом точности чисел с плавающей точкой
    return Double.compare(that.x, x) == 0 &&
           Double.compare(that.y, y) == 0;
}

// Переопределение метода hashCode()
@Override
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;
}

// Переопределение метода clone()
@Override
public Object clone() {
    return new FunctionPoint(this.x, this.y);
}
}

```

вот весь код класса FunctionPoint

Задание 2

Нам необходимо в классе ArrayTabulatedFunction следующие методы:

- **String toString():** Должен возвращать описание табулированной функции.
- **boolean equals(Object o):** Должен возвращать true тогда и только тогда, когда переданный объект также является табулированной функцией (реализует интерфейс TabulatedFunction) и её набор точек в точности совпадает с набором точек функции, у которой вызывается метод.
- **int hashCode():** Должен возвращать значение хэш-кода для объекта табулированной функции.
- **Object clone():** Должен возвращать объект-копию для объекта табулированной функции, клонирование должно быть глубоким.

```

@Override
public String toString() {
    StringBuilder sb = new StringBuilder();
    sb.append("{");
    for (int i = 0; i < pointsCount; i++) {
        sb.append(points[i].toString()); // используем toString() точки
    }
    sb.append("}");
    return sb.toString();
}

```

```

        if (i < pointsCount - 1) {
            sb.append(", ");
        }
    }
    sb.append("}");
    return sb.toString();
}

@Override
public boolean equals(Object o) {
    if (this == o) return true;
    if (o == null || !(o instanceof TabulatedFunction)) return false;

    TabulatedFunction that = (TabulatedFunction) o;

    // Проверка количества точек
    if (this.getPointsCount() != that.getPointsCount()) return false;

    // Оптимизация для ArrayTabulatedFunction
    if (o instanceof ArrayTabulatedFunction) {
        ArrayTabulatedFunction other = (ArrayTabulatedFunction) o;
        for (int i = 0; i < pointsCount; i++) {
            // Используем getPoint() вместо прямого доступа к points[i]
            if (!this.getPoint(i).equals(other.getPoint(i))) {
                return false;
            }
        }
    }
    // Оптимизация для LinkedListTabulatedFunction
    else if (o instanceof LinkedListTabulatedFunction) {
        LinkedListTabulatedFunction other = (LinkedListTabulatedFunction)
o;
        for (int i = 0; i < pointsCount; i++) {
            // Используем getPoint() для обеих функций
            if (!this.getPoint(i).equals(other.getPoint(i))) {
                return false;
            }
        }
    } else {
        // Общий случай для любого TabulatedFunction
        for (int i = 0; i < pointsCount; i++) {
            FunctionPoint thisPoint = this.getPoint(i);
            FunctionPoint thatPoint = that.getPoint(i);
            if (!thisPoint.equals(thatPoint)) {
                return false;
            }
        }
    }
}

return true;
}

// Переопределение метода hashCode()
@Override
public int hashCode() {
    int hash = pointsCount; // включаем количество точек в хэш

    for (int i = 0; i < pointsCount; i++) {
        hash ^= points[i].hashCode(); // XOR с хэш-кодом каждой точки
    }

    return hash;
}

```

```

// Переопределение метода clone()
@Override
public Object clone() {
    // Глубокое клонирование
    FunctionPoint[] clonedPoints = new FunctionPoint[pointsCount];
    for (int i = 0; i < pointsCount; i++) {
        clonedPoints[i] = (FunctionPoint) points[i].clone(); // клонируем
    }

    try {
        ArrayTabulatedFunction clone = (ArrayTabulatedFunction)
super.clone();
        clone.points = clonedPoints;
        clone.pointsCount = this.pointsCount;
        // transient поля сбрасываем
        clone.lastAccessedIndex = -1;
        clone.lastAccessedPoint = null;
        return clone;
    } catch (CloneNotSupportedException e) {
        // fallback - создаем через конструктор
        return new ArrayTabulatedFunction(clonedPoints);
    }
}
}

```

Задание 3

Аналогично, необходимо переопределить методы `toString()`, `equals()`, `hashCode()` и `clone()` в классе `LinkedListTabulatedFunction`.

```

// Переопределение метода toString()
@Override
public String toString() {
    StringBuilder sb = new StringBuilder();
    sb.append("{");
    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();
}

@Override
public boolean equals(Object o) {
    if (this == o) return true;
    if (o == null || !(o instanceof TabulatedFunction)) return false;

    TabulatedFunction that = (TabulatedFunction) o;

    // Проверка количества точек
    if (this.getPointsCount() != that.getPointsCount()) return false;
}

```

```

// Оптимизация для ArrayTabulatedFunction
if (o instanceof ArrayTabulatedFunction) {
    ArrayTabulatedFunction other = (ArrayTabulatedFunction) o;
    for (int i = 0; i < pointsCount; i++) {
        // Используем getPoint() для обеих функций
        if (!this.getPoint(i).equals(other.getPoint(i))) {
            return false;
        }
    }
}
// Оптимизация для LinkedListTabulatedFunction
else if (o instanceof LinkedListTabulatedFunction) {
    LinkedListTabulatedFunction other = (LinkedListTabulatedFunction)
o;
    // Используем прямое сравнение узлов через приватный метод
getNodeByIndex()
    for (int i = 0; i < pointsCount; i++) {
        if
(!this.getNodeByIndex(i).point.equals(other.getNodeByIndex(i).point)) {
            return false;
        }
    }
} else {
    // Общий случай для любого TabulatedFunction
    for (int i = 0; i < pointsCount; i++) {
        FunctionPoint thisPoint = this.getPoint(i);
        FunctionPoint thatPoint = that.getPoint(i);
        if (!thisPoint.equals(thatPoint)) {
            return false;
        }
    }
}

return true;
}
// Переопределение метода hashCode()
@Override
public int hashCode() {
    int hash = pointsCount; // включаем количество точек в хэш

    FunctionNode current = head.next;
    while (current != head) {
        hash ^= current.point.hashCode(); // XOR с хэш-кодом каждой точки
        current = current.next;
    }

    return hash;
}

// Переопределение метода clone()
@Override
public Object clone() {
    // "Пересборка" нового списка без использования методов добавления
    FunctionPoint[] pointsArray = new FunctionPoint[pointsCount];

    // Собираем массив точек из текущего списка
    FunctionNode current = head.next;
    int index = 0;
    while (current != head) {
        pointsArray[index++] = (FunctionPoint) current.point.clone(); // //
клонируем точку
        current = current.next;
    }
}

```

```
// Создаем новый список через конструктор
    return new LinkedListTabulatedFunction(pointsArray);
}
}
```

Задание 4

Необходимо сделать так, чтобы все объекты типа TabulatedFunction были клонируемыми с точки зрения JVM и внести метод clone() в этот интерфейс.

```
package functions;

public interface TabulatedFunction extends Function, Cloneable {
    //возвращает количество точек в функции
    int getPointsCount();

    //возвращает точку по указанному индексу
    //выбрасывает исключение, если индекс выходит за границы
    FunctionPoint getPoint(int index) throws
FunctionPointIndexOutOfBoundsException;

    //заменяет точку по указанному индексу
    //выбрасывает исключения при недопустимом индексе или нарушении порядка
точек
    void setPoint(int index, FunctionPoint point) throws
FunctionPointIndexOutOfBoundsException, InappropriateFunctionPointException;

    //возвращает координату x точки по указанному индексу
    double getPointX(int index) throws
FunctionPointIndexOutOfBoundsException;

    //устанавливает координату x точки по указанному индексу
    //проверяет сохранение упорядоченности точек по x
    void setPointX(int index, double x) throws
FunctionPointIndexOutOfBoundsException, InappropriateFunctionPointException;

    //возвращает координату y точки по указанному индексу
    double getPointY(int index) throws
FunctionPointIndexOutOfBoundsException;

    //устанавливает координату y точки по указанному индексу
    void setPointY(int index, double y) throws
FunctionPointIndexOutOfBoundsException;

    //добавляет новую точку в функцию
    //проверяет отсутствие дублирования координат x и сохраняет
упорядоченность
    void addPoint(FunctionPoint point) throws
InappropriateFunctionPointException;

    //удаляет точку по указанному индексу
    //требует наличия минимум 2 точек после удаления
    void deletePoint(int index) throws
FunctionPointIndexOutOfBoundsException, IllegalStateException;

    //возвращает копию объекта табулированной функции
    Object clone();
}
```

Задание 5

Реализуем main(), проверяя работу переопределенных нами методов.

Проверяем работу метода `toString()` для объектов типов `ArrayTabulatedFunction` и `LinkedListTabulatedFunction`, выведя строковое представление объектов в консоль.

```
// 2) Тестирование toString()
System.out.println("\n2) Testing toString() method:");
System.out.println("ArrayTabulatedFunction 1: " + arrayFunc1.toString());
System.out.println("LinkedListTabulatedFunction 1: " +
linkedFunc1.toString());
System.out.println("ArrayTabulatedFunction 2: " + arrayFunc2.toString());
System.out.println("LinkedListTabulatedFunction 2: " +
linkedFunc2.toString());
```

Проверяем работу метода `equals()`, вызывая его для одинаковых и различающихся объектов одинаковых и различающихся классов.

```
// 3) Тестирование equals()
System.out.println("\n3) Testing equals() method:");

// Сравнение одинаковых объектов одного класса
System.out.println("arrayFunc1.equals(arrayFunc1): " +
arrayFunc1.equals(arrayFunc1));
System.out.println("linkedFunc1.equals(linkedFunc1): " +
linkedFunc1.equals(linkedFunc1));

// Сравнение одинаковых объектов разных классов (одинаковые точки)
System.out.println("arrayFunc1.equals(linkedFunc1): " +
arrayFunc1.equals(linkedFunc1));
System.out.println("linkedFunc1.equals(arrayFunc1): " +
linkedFunc1.equals(arrayFunc1));

// Сравнение разных объектов
System.out.println("arrayFunc1.equals(arrayFunc2): " +
arrayFunc1.equals(arrayFunc2));
System.out.println("arrayFunc1.equals(linkedFunc2): " +
arrayFunc1.equals(linkedFunc2));
System.out.println("linkedFunc1.equals(arrayFunc2): " +
linkedFunc1.equals(arrayFunc2));

// Сравнение с null и другим типом
System.out.println("arrayFunc1.equals(null): " + arrayFunc1.equals(null));
System.out.println("arrayFunc1.equals(\"string\"): " +
arrayFunc1.equals("string"));
```

Проверяем работу метода `hashCode()`, выведя в консоль его значения для всех использованных объектов.

```
// 4) Тестирование hashCode()
System.out.println("\n4) Testing hashCode() method:");
System.out.println("arrayFunc1.hashCode(): " + arrayFunc1.hashCode());
System.out.println("linkedFunc1.hashCode(): " + linkedFunc1.hashCode());
System.out.println("arrayFunc2.hashCode(): " + arrayFunc2.hashCode());
System.out.println("linkedFunc2.hashCode(): " + linkedFunc2.hashCode());
```

```

// Проверка согласованности equals() и hashCode()
System.out.println("\nConsistency check (equal objects should have equal
hashCodes):");
System.out.println("arrayFunc1.equals(linkedFunc1): " +
arrayFunc1.equals(linkedFunc1));
System.out.println("arrayFunc1.hashCode() == linkedFunc1.hashCode(): " +
(arrayFunc1.hashCode() == linkedFunc1.hashCode()));

// Изменение объекта и проверка изменения hashCode
System.out.println("\nTesting hashCode change after modification:");
int originalHash = arrayFunc1.hashCode();
System.out.println("Original hashCode: " + originalHash);

// Незначительно изменяем точку
arrayFunc1.setPointY(2, 7.001); // изменяем на несколько тысячных
int modifiedHash = arrayFunc1.hashCode();
System.out.println("After modifying point Y: " + modifiedHash);
System.out.println("HashCode changed: " + (originalHash != modifiedHash));

// Восстанавливаем исходное значение для дальнейших тестов
arrayFunc1.setPointY(2, 7.0);

```

Проверяем работу метода `clone()` для объектов обоих классов табулированных функций.

```

// 5) Тестирование clone()
System.out.println("\n5) Testing clone() method:");

// Клонирование ArrayTabulatedFunction
ArrayTabulatedFunction arrayClone = (ArrayTabulatedFunction)
arrayFunc1.clone();
System.out.println("Array original: " + arrayFunc1.toString());
System.out.println("Array clone: " + arrayClone.toString());
System.out.println("Original == Clone: " + (arrayFunc1 == arrayClone));
System.out.println("Original.equals(Clone): " +
arrayFunc1.equals(arrayClone));

// Клонирование LinkedListTabulatedFunction
LinkedListTabulatedFunction linkedClone = (LinkedListTabulatedFunction)
linkedFunc1.clone();
System.out.println("Linked original: " + linkedFunc1.toString());
System.out.println("Linked clone: " + linkedClone.toString());
System.out.println("Original == Clone: " + (linkedFunc1 == linkedClone));
System.out.println("Original.equals(Clone): " +
linkedFunc1.equals(linkedClone));

```

Убеждаемся, что произведено именно глубокое клонирование: для этого после клонирования измените исходные объекты и проверьте, что объекты-клоны не изменились.

```

// 6) Проверка глубокого клонирования
System.out.println("\n6) Testing deep cloning:");

// Изменяем исходный Array
System.out.println("Before modification - Array:");
System.out.println("Original: " + arrayFunc1.toString());
System.out.println("Clone: " + arrayClone.toString());

```

```

arrayFunc1.setPointY(1, 999.0); // изменяем исходный объект
arrayFunc1.setPointX(2, 2.5);

System.out.println("After modifying original - Array:");
System.out.println("Original: " + arrayFunc1.toString());
System.out.println("Clone: " + arrayClone.toString());
System.out.println("Clone NOT affected: " +
(!arrayFunc1.equals(arrayClone)));

// Изменяем исходный LinkedList
System.out.println("\nBefore modification - LinkedList:");
System.out.println("Original: " + linkedFunc1.toString());
System.out.println("Clone: " + linkedClone.toString());

linkedFunc1.setPointY(0, 888.0); // изменяем исходный объект
linkedFunc1.deletePoint(1); // удаляем точку из оригинала

System.out.println("After modifying original - LinkedList:");
System.out.println("Original: " + linkedFunc1.toString());
System.out.println("Clone: " + linkedClone.toString());
System.out.println("Clone NOT affected: " +
(!linkedFunc1.equals(linkedClone)));

```

Вывод main() представлен ниже:

1) Creating test functions...

2) Testing toString() method:

```

ArrayTabulatedFunction 1: {(0.0; 1.0), (1.0; 3.0), (2.0; 7.0), (3.0; 13.0)}
LinkedListTabulatedFunction 1: {(0.0; 1.0), (1.0; 3.0), (2.0; 7.0), (3.0; 13.0)}
ArrayTabulatedFunction 2: {(0.0; 1.0), (1.0; 3.0), (2.0; 7.0), (3.0; 12.0)}
LinkedListTabulatedFunction 2: {(0.0; 1.0), (1.0; 3.0), (2.0; 7.0)}

```

3) Testing equals() method:

```

arrayFunc1.equals(arrayFunc1): true
linkedFunc1.equals(linkedFunc1): true
arrayFunc1.equals(linkedFunc1): true
linkedFunc1.equals(arrayFunc1): true
arrayFunc1.equals(arrayFunc2): false
arrayFunc1.equals(linkedFunc2): false
linkedFunc1.equals(arrayFunc2): false
arrayFunc1.equals(null): false
arrayFunc1.equals("string"): false

```

4) Testing hashCode() method:

```

arrayFunc1.hashCode(): 1077280772
linkedFunc1.hashCode(): 1077280772
arrayFunc2.hashCode(): 1077149700
linkedFunc2.hashCode(): 1075052547

```

Consistency check (equal objects should have equal hashCode):

arrayFunc1.equals(linkedFunc1): true

arrayFunc1.hashCode() == linkedFunc1.hashCode(): true

Testing hashCode change after modification:

Original hashCode: 1077280772

After modifying point Y: 1693134361

HashCode changed: true

5) Testing clone() method:

Array original: {(0.0; 1.0), (1.0; 3.0), (2.0; 7.0), (3.0; 13.0)}

Array clone: {(0.0; 1.0), (1.0; 3.0), (2.0; 7.0), (3.0; 13.0)}

Original == Clone: false

Original.equals(Clone): true

Linked original: {(0.0; 1.0), (1.0; 3.0), (2.0; 7.0), (3.0; 13.0)}

Linked clone: {(0.0; 1.0), (1.0; 3.0), (2.0; 7.0), (3.0; 13.0)}

Original == Clone: false

Original.equals(Clone): true

6) Testing deep cloning:

Before modification - Array:

Original: {(0.0; 1.0), (1.0; 3.0), (2.0; 7.0), (3.0; 13.0)}

Clone: {(0.0; 1.0), (1.0; 3.0), (2.0; 7.0), (3.0; 13.0)}

After modifying original - Array:

Original: {(0.0; 1.0), (1.0; 999.0), (2.5; 7.0), (3.0; 13.0)}

Clone: {(0.0; 1.0), (1.0; 3.0), (2.0; 7.0), (3.0; 13.0)}

Clone NOT affected: true

Before modification - LinkedList:

Original: {(0.0; 1.0), (1.0; 3.0), (2.0; 7.0), (3.0; 13.0)}

Clone: {(0.0; 1.0), (1.0; 3.0), (2.0; 7.0), (3.0; 13.0)}

After modifying original - LinkedList:

Original: {(0.0; 888.0), (2.0; 7.0), (3.0; 13.0)}

Clone: {(0.0; 1.0), (1.0; 3.0), (2.0; 7.0), (3.0; 13.0)}

Clone NOT affected: true

7) Testing FunctionPoint methods:

Point1: (1.5; 2.5)

Point2: (1.5; 2.5)

Point3: (1.5; 2.6)

Point1.equals(Point2): true

Point1.equals(Point3): false

Point1.equals(Point4): false

Point1.hashCode(): 2147221504

Point2.hashCode(): 2147221504

```
Point3.hashCode(): -1288699903
Point1 clone: (1.5; 2.5)
Point1 == Clone: false
Point1.equals(Clone): true
```

8) Testing with analytical functions:

```
Sin tabulated: {(0.0; 0.0), (0.7853981633974483; 0.7071067811865475),
(1.5707963267948966; 1.0), (2.356194490192345; 0.7071067811865476),
(3.141592653589793; 1.2246467991473532E-16)}
Cos tabulated: {(0.0; 1.0), (0.7853981633974483; 0.7071067811865476),
(1.5707963267948966; 6.123233995736766E-17), (2.356194490192345; -
0.7071067811865475), (3.141592653589793; -1.0)}
Sin.equals(Cos): false
Sin.hashCode(): 455675496
Cos.hashCode(): 619253352
Sin clone equals original: true
```

Process finished with exit code 0

Спасибо за внимание!