# ЛАБОРАТОРНАЯ РАБОТА №5

## Табулированные функции: переопределение методов унаследованных из Object

**по курсу**
**Объектно-ориентированное программирование**

**Группа 6204-010302D**

**Студент: Е.Д.Васильев.**

**Преподаватель: Борисов Дмитрий Сергеевич.**

# Задание 1

## Переопределение методов в классе

В рамках первого задания в классе **FunctionPoint** были переопределены четыре метода базового класса Object: **toString()**, **equals(Object o)**, **hashCode()** и **clone()** - для корректного описания, сравнения, хеширования и клонирования объектов типа FunctionPoint, представляющих точку табулированной функции.

Переопределенные методы имеют следующий вид:

**toString():**

```
public String toString() {

    // Форматирование до одной цифры после запятой для краткости
    return String.format("(%.1f; %.1f)", x, y);
}
```

**equals(Object o):**

```
public boolean equals(Object o) {
    if (this == o) return true;

    // Проверяем, что объект не null и имеет тот же класс
    if (o == null || getClass() != o.getClass()) return false;

    FunctionPoint that = (FunctionPoint) o;

    return Math.abs(this.x - that.x) < EPS && Math.abs(this.y - that.y) <
EPS;
}
```

**hashCode():**

```
public int hashCode() {
    // Получаем 64-битное представление X и Y
    long bitsX = Double.doubleToLongBits(x);
    long bitsY = Double.doubleToLongBits(y);

    // Объединяем 32-битные части x и y с помощью XOR, (bits >>> 32) берет
старшие 32 бита, (int)bits берет младшие 32 бита
    int hashX = (int) (bitsX ^ (bitsX >>> 32));
    int hashY = (int) (bitsY ^ (bitsY >>> 32));

    // Объединяем хэши x и y
    return hashX ^ hashY;
}
```

**clone():**

```
    public Object clone() {

        return new FunctionPoint(this.x, this.y);
    }

}
```

# Задание 2

## Переопределение методов в классе ArrayTabulatedFunction

В рамках второго задания в классе **ArrayTabulatedFunction** были переопределены методы **toString()**, **equals()**, **hashCode()** и **clone()**, унаследованные от класса Object. Было обеспечено корректное строковое представление табулированной функции, сравнение объектов, вычисление хэш-кода и глубокое клонирование.

Переопределенные методы имеют следующий вид:

### toString():

```
public String toString() {
    if (pointsCount == 0) {
        return "{}";
    }

    // StringBuilder - для эффективной работы со строками в цикле
    StringBuilder sb = new StringBuilder();
    sb.append('{');

    for (int i = 0; i < pointsCount; i++) {
        // Добавляем строковое представление каждой точки
        sb.append(points[i].toString());

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

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

### equals(Object o):

```
public boolean equals(Object o) {
    if (this == o) return true;

    // Проверяем, что объект реализует интерфейс TabulatedFunction
    if (!(o instanceof TabulatedFunction)) return false;

    TabulatedFunction that = (TabulatedFunction) o;

    // Проверяем количество точек
    if (this.getPointsCount() != that.getPointsCount()) return false;

    // Если это ArrayTabulatedFunction, используем прямой доступ
    if (o instanceof ArrayTabulatedFunction) {
        ArrayTabulatedFunction thatArray = (ArrayTabulatedFunction) o;

        // Прямое сравнение массивов, используя FunctionPoint.equals()
        for (int i = 0; i < pointsCount; i++) {
            // Сравниваем точки через FunctionPoint.equals()
            if (!this.points[i].equals(thatArray.points[i])) {
                return false;
```

```
            }
        }
        return true;
    }

    // Если это LinkedList
    else {
        // Последовательно сравниваем точки через геттеры
        for (int i = 0; i < pointsCount; i++) {
            double x1 = this.getPointX(i);
            double y1 = this.getPointY(i);
            double x2 = that.getPointX(i);
            double y2 = that.getPointY(i);

            // Сравниваем координаты с учетом EPS
            if (Math.abs(x1 - x2) >= EPS || Math.abs(y1 - y2) >= EPS) {
                return false;
            }
        }
        return true;
    }
}
```

**hashCode():**

```
public int hashCode() {
    int result = pointsCount;
    // Последовательно объединяем хэш-коды всех точек через XOR
    for (int i = 0; i < pointsCount; i++) {
        // points[i].hashCode() вызывает переопределенный метод из
FunctionPoint
        result ^= points[i].hashCode();
    }

    return result;
}
```

**clone():**

```
public Object clone() {
    try {
        // Поверхностное клонирование самого объекта
        ArrayTabulatedFunction clone = (ArrayTabulatedFunction)
super.clone();

        // Глубокое клонирование: создаем новый массив
        clone.points = new FunctionPoint[this.points.length];

        // Глубокое клонирование: копируем точки одну за другой
        for (int i = 0; i < this.pointsCount; i++) {
            clone.points[i] = (FunctionPoint) this.points[i].clone();
        }

        return clone;
    } catch (CloneNotSupportedException e) {
        throw new InternalError(e);
    }
}
```

# Задание 3

## Переопределение методов в классе LinkedListTabulatedFunction

В рамках третьего задания в классе **LinkedListTabulatedFunction** были переопределены методы toString(), equals(), hashCode() и clone(), унаследованные от класса Object. Было обеспечено корректное строковое представление табулированной функции, сравнение объектов, вычисление хэш-кода и глубокое клонирование.

**toString():**

```
public String toString() {
    if (pointsCount == 0) {
        return "{}";
    }

    StringBuilder sb = new StringBuilder();
    sb.append('{');

    FunctionNode current = head.next;

    for (int i = 0; i < pointsCount; i++) {
        // Убедимся, что узел существует, прежде чем обращаться к нему
        if (current == null) {
            throw new IllegalStateException("Corrupted list state: list ended
unexpectedly.");
        }

        sb.append(current.point.toString());

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

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

**equals(Object o):**

```
public boolean equals(Object o) {
    if (this == o) return true;

    // Проверяем, что объект реализует интерфейс TabulatedFunction
    if (!(o instanceof TabulatedFunction)) return false;

    TabulatedFunction that = (TabulatedFunction) o;

    // Проверяем количество точек
    if (this.pointsCount != that.getPointsCount()) return false;

    //Если это LinkedListTabulatedFunction, используем прямой обход узлов
    if (o instanceof LinkedListTabulatedFunction) {
        LinkedListTabulatedFunction thatList = (LinkedListTabulatedFunction)
o;

        FunctionNode currentThis = this.head.next;
        FunctionNode currentThat = thatList.head.next;

        for (int i = 0; i < pointsCount; i++) {
            // Используем FunctionPoint.equals() для сравнения координат
            if (!currentThis.point.equals(currentThat.point)) {
                return false;
```

```
            }
            currentThis = currentThis.next;
            currentThat = currentThat.next;
        }
        return true;
    }

    // Если это ArrayTabulatedFunction
    else {
        // Используем публичные методы getPointX/Y (менее эффективно, но
универсально)
        for (int i = 0; i < pointsCount; i++) {
            double x1 = this.getPointX(i);
            double y1 = this.getPointY(i);
            double x2 = that.getPointX(i);
            double y2 = that.getPointY(i);

            // Сравниваем координаты с учетом EPS
            if (Math.abs(x1 - x2) >= EPS || Math.abs(y1 - y2) >= EPS) {
                return false;
            }
        }
        return true;
    }
}
```

**hashCode():**

```
public int hashCode() {
    int result = pointsCount;

    FunctionNode current = head.next;

    // Обходим список и последовательно объединяем хэш-коды всех точек через
XOR
    for (int i = 0; i < pointsCount; i++) {
        // points[i].hashCode() вызывает переопределенный метод из
FunctionPoint
        result ^= current.point.hashCode();
        current = current.next;
    }

    return result;
}
```

**clone():**

```
public Object clone() {
    try {

        //Поверхностное клонирование
        LinkedListTabulatedFunction clone = (LinkedListTabulatedFunction)
super.clone();

        clone.head = new FunctionNode(null);
        clone.head.next = clone.head;
        clone.head.prev = clone.head;
        clone.pointsCount = 0;

        if (this.pointsCount == 0) {
            return clone;
        }

        FunctionNode currentOriginal = this.head.next;
```

```
        for (int i = 0; i < this.pointsCount; i++) {

            // Глубокое клонирование точки
            FunctionPoint clonedPoint = (FunctionPoint)
currentOriginal.point.clone();

            clone.addNodeToTail(clonedPoint);

            currentOriginal = currentOriginal.next;
        }

        return clone;
    } catch (CloneNotSupportedException e) {
        throw new InternalError(e);
    }
}
```

# Задание 4

# Создание функций-комбинаторов в пакете functions.meta

В рамках четвёртого задания был дополнен интерфейс **TabulatedFunction**, для соответствия стандартам **JVM** и обеспечения полиморфизма, были внесены следующие изменения в контракт интерфейса:

1. Наследование **Cloneable**: Интерфейс public interface TabulatedFunction был расширен наследованием от маркерного интерфейса **java.lang.Cloneable**. Это гарантирует, что любой класс, реализующий данный интерфейс, может быть клонирован с использованием нативного механизма **JVM** без возникновения ошибки **CloneNotSupportedException**.
2. Контракт **clone():** В интерфейс **TabulatedFunction** добавлен публичный метод **Object clone()**, что гарантирует, что контракт на клонирование является публичным и обязательным для всех классов-реализаций.

```
public interface TabulatedFunction extends Function, Cloneable {
Object clone() throws CloneNotSupportedException;

}
```

# Задание 5

# Тестирование

**Код Main:**

```
import functions.*;
import functions.basic.*;

public class Main {

    private static FunctionPoint[] toPoints(TabulatedFunction f) {
        FunctionPoint[] pts = new FunctionPoint[f.getPointsCount()];
        for (int i = 0; i < pts.length; i++) {
            pts[i] = new FunctionPoint(f.getPointX(i), f.getPointY(i));
```

```
        }
        return pts;
    }

    private static void printFunction(String title, TabulatedFunction f) {
        System.out.println(title);
        System.out.println("Количество точек: " + f.getPointsCount());
        for (int i = 0; i < f.getPointsCount(); i++) {
            System.out.printf("  Точка %d: (%.6f; %.6f)%n", i,
f.getPointX(i), f.getPointY(i));
        }
        System.out.println("Строковое представление: " + f);
        System.out.println("hashCode: " + f.hashCode());
        System.out.println();
    }

    public static void main(String[] args) {

        Function sin = new Sin();
        Function cos = new Cos();

        // ArrayTabulatedFunction
        TabulatedFunction arr1 = TabulatedFunctions.tabulate(sin, 0, Math.PI,
5);
        TabulatedFunction arr2 = TabulatedFunctions.tabulate(sin, 0, Math.PI,
5);
        TabulatedFunction arr3 = TabulatedFunctions.tabulate(sin, 0, Math.PI,
6);

        //  LinkedListTabulatedFunction
        TabulatedFunction list1 = new
LinkedListTabulatedFunction(toPoints(arr1));
        TabulatedFunction list2 = new
LinkedListTabulatedFunction(toPoints(arr1));
        TabulatedFunction list3 = new
LinkedListTabulatedFunction(toPoints(TabulatedFunctions.tabulate(cos, 0,
Math.PI, 5)));

        // FunctionPoint
        FunctionPoint p1 = new FunctionPoint(1.0, 2.0);
        FunctionPoint p2 = new FunctionPoint(1.0, 2.0);
        FunctionPoint p3 = new FunctionPoint(1.0, 2.0000001);

        // FunctionPoint: вывод + equals + hash

System.out.println("=======================================================");
        System.out.println("      ТЕСТЫ КЛАССА FunctionPoint");

System.out.println("=======================================================");

        System.out.println("p1 = " + p1 + ", hash = " + p1.hashCode());
        System.out.println("p2 = " + p2 + ", hash = " + p2.hashCode());
        System.out.println("p3 = " + p3 + ", hash = " + p3.hashCode());

        System.out.println("\nСравнение точек:");
        System.out.println("p1.equals(p2) → " + p1.equals(p2) + " (должно
быть true)");
        System.out.println("p1.equals(p3) → " + p1.equals(p3) + " (должно
быть false)");


        //  Табулированные функции: подробный вывод всех точек

System.out.println("\n=======================================================")
```

```
;
        System.out.println("      ТЕСТЫ КЛАССОВ ТАБУЛИРОВАННЫХ ФУНКЦИЙ");

System.out.println("=====================================================\n")
;

        printFunction("ARRAY arr1 (Sin, 5 точек):", arr1);
        printFunction("ARRAY arr2 (Sin, 5 точек — копия arr1):", arr2);
        printFunction("ARRAY arr3 (Sin, 6 точек — отличается от arr1):",
arr3);

        printFunction("LIST  list1 (Sin, 5 точек):", list1);
        printFunction("LIST  list2 (Sin, 5 точек — копия list1):", list2);
        printFunction("LIST  list3 (Cos, 5 точек — отличается от list1):",
list3);

        // equals() и hashCode() для табулированных функций

System.out.println("\n=====================================================")
;
        System.out.println("      СРАВНЕНИЕ equals() И hashCode()");

System.out.println("=====================================================\n")
;

        System.out.println("arr1.equals(arr2) → " + arr1.equals(arr2) + "
(ожидается: true)");
        System.out.println("arr1.equals(arr3) → " + arr1.equals(arr3) + "
(ожидается: false — разное число точек)");
        System.out.println("arr1.equals(list1) → " + arr1.equals(list1) + "
(ожидается: true — одинаковые точки)");
        System.out.println("list1.equals(list3) → " + list1.equals(list3) + "
(ожидается: false — другая функция cos)");

        // Проверка изменения hash
        TabulatedFunction arrMod = TabulatedFunctions.tabulate(sin, 0,
Math.PI, 5);
        int hashBefore = arrMod.hashCode();
        arrMod.setPointY(1, 999);
        int hashAfter = arrMod.hashCode();

        System.out.println("\nПроверка изменения hashCode при изменении
данных:");
        System.out.println("hash до изменения   = " + hashBefore);
        System.out.println("hash после изменения = " + hashAfter);

        // 4. clone()

System.out.println("\n=====================================================")
;
        System.out.println("      ТЕСТЫ ГЛУБОКОГО КЛОНИРОВАНИЯ");

System.out.println("=====================================================\n")
;

        System.out.println(" Клонирование ArrayTabulatedFunction:");
        try {
            TabulatedFunction arrClone = (TabulatedFunction) arr1.clone();

            System.out.println("До изменения:");
            System.out.println("  arr1[2] = " + arr1.getPointY(2));
            System.out.println("  clone[2] = " + arrClone.getPointY(2));

            arr1.setPointY(2, 500);
```

```
            System.out.println("После изменения arr1:");
            System.out.println("  arr1[2] = " + arr1.getPointY(2));
            System.out.println("  clone[2] = " + arrClone.getPointY(2) + "
(должно НЕ измениться)");

        } catch (Exception e) {
            System.out.println("Ошибка: клон функции создать нельзя!");
        }


        System.out.println("\n Клонирование LinkedListTabulatedFunction:");
        try {
            TabulatedFunction listClone = (TabulatedFunction) list1.clone();

            System.out.println("До изменения:");
            System.out.println("  list1[2] = " + list1.getPointY(2));
            System.out.println("  clone[2] = " + listClone.getPointY(2));

            list1.setPointY(2, -500);

            System.out.println("После изменения list1:");
            System.out.println("  list1[2] = " + list1.getPointY(2));
            System.out.println("  clone[2] = " + listClone.getPointY(2) + "
(должно НЕ измениться)");

        } catch (Exception e) {
            System.out.println("Ошибка: клон функции создать нельзя!");
        }
    }
}
```

**Вывод в консоль:**

==========

ТЕСТЫ КЛАССА FunctionPoint

==========

p1 = (1,0; 2,0), hash = 2146435072
p2 = (1,0; 2,0), hash = 2146435072
p3 = (1,0; 2,0), hash = 1922824525

Сравнение точек:
p1.equals(p2) → true (должно быть true)
p1.equals(p3) → false (должно быть false)

==========

ТЕСТЫ КЛАССОВ ТАБУЛИРОВАННЫХ ФУНКЦИЙ

==========

ARRAY arr1 (Sin, 5 точек):
Количество точек: 5
  Точка 0: (0,000000; 0,000000)
  Точка 1: (0,785398; 0,707107)

Точка 2: (1,570796; 1,000000)
Точка 3: (2,356194; 0,707107)
Точка 4: (3,141593; 0,000000)
Строковое представление: {(0,0; 0,0), (0,8; 0,7), (1,6; 1,0), (2,4; 0,7), (3,1; 0,0)}
hashCode: 455675496

ARRAY arr2 (Sin, 5 точек — копия arr1):
Количество точек: 5
Точка 0: (0,000000; 0,000000)
Точка 1: (0,785398; 0,707107)
Точка 2: (1,570796; 1,000000)
Точка 3: (2,356194; 0,707107)
Точка 4: (3,141593; 0,000000)
Строковое представление: {(0,0; 0,0), (0,8; 0,7), (1,6; 1,0), (2,4; 0,7), (3,1; 0,0)}
hashCode: 455675496

ARRAY arr3 (Sin, 6 точек — отличается от arr1):
Количество точек: 6
Точка 0: (0,000000; 0,000000)
Точка 1: (0,628319; 0,587785)
Точка 2: (1,256637; 0,951057)
Точка 3: (1,884956; 0,951057)
Точка 4: (2,513274; 0,587785)
Точка 5: (3,141593; 0,000000)
Строковое представление: {(0,0; 0,0), (0,6; 0,6), (1,3; 1,0), (1,9; 1,0), (2,5; 0,6), (3,1; 0,0)}
hashCode: 593845602

LIST list1 (Sin, 5 точек):
Количество точек: 5
Точка 0: (0,000000; 0,000000)
Точка 1: (0,785398; 0,707107)
Точка 2: (1,570796; 1,000000)
Точка 3: (2,356194; 0,707107)
Точка 4: (3,141593; 0,000000)
Строковое представление: {(0,0; 0,0), (0,8; 0,7), (1,6; 1,0), (2,4; 0,7), (3,1; 0,0)}
hashCode: 455675496

LIST list2 (Sin, 5 точек — копия list1):
Количество точек: 5

```
  Точка 0: (0,000000; 0,000000)
  Точка 1: (0,785398; 0,707107)
  Точка 2: (1,570796; 1,000000)
  Точка 3: (2,356194; 0,707107)
  Точка 4: (3,141593; 0,000000)
Строковое представление: {(0,0; 0,0), (0,8; 0,7), (1,6; 1,0), (2,4; 0,7), (3,1; 0,0)}
hashCode: 455675496

LIST  list3 (Cos, 5 точек — отличается от list1):
Количество точек: 5
  Точка 0: (0,000000; 1,000000)
  Точка 1: (0,785398; 0,707107)
  Точка 2: (1,570796; 0,000000)
  Точка 3: (2,356194; -0,707107)
  Точка 4: (3,141593; -1,000000)
Строковое представление: {(0,0; 1,0), (0,8; 0,7), (1,6; 0,0), (2,4; -0,7), (3,1; -
1,0)}
hashCode: 619253352


======================================================
     СРАВНЕНИЕ equals() И hashCode()
======================================================

arr1.equals(arr2) → true (ожидается: true)
arr1.equals(arr3) → false (ожидается: false — разное число точек)
arr1.equals(list1) → true (ожидается: true — одинаковые точки)
list1.equals(list3) → false (ожидается: false — другая функция cos)

Проверка изменения hashCode при изменении данных:
hash до изменения   = 455675496
hash после изменения = 37727546

======================================================
     ТЕСТЫ ГЛУБОКОГО КЛОНИРОВАНИЯ
======================================================

 Клонирование ArrayTabulatedFunction:
До изменения:
  arr1[2] = 1.0
```

clone[2] = 1.0
После изменения arr1:
arr1[2] = 500.0
clone[2] = 1.0 (должно НЕ измениться)

Клонирование LinkedListTabulatedFunction:
До изменения:
list1[2] = 1.0
clone[2] = 1.0
После изменения list1:
list1[2] = -500.0
clone[2] = 1.0 (должно НЕ измениться)