Лабораторная работа №5

Скиба В.А.
Группа: 6204-010302D

Тема: Переопределение методов Object для классов, связанных с табулированными функциями

1. Цель работы

Целью лабораторной работы было расширить возможности классов, связанных с табулированными функциями, путём переопределения унаследованных от класса Object методов: *toString(), equals(Object), hashCode(), clone()*. Также требовалось добавить метод *clone()* в интерфейс *TabulatedFunction* и обеспечить глубокое копирование для реализаций.

2. Выполнение задания

Задание 1 — *FunctionPoint*
- Реализованы методы:
  - *toString()*: возвращает строку в формате (x; y). Пример: (1.1; -7.5).
  - *equals(Object o)*: сравнение с использованием Double.compare для корректной проверки double.
  - *hashCode()*: используется *Double.doubleToLongBits()* и *XOR* старших и младших 32 бит для получения int-хэша.
  - *clone*(): возвращает новый объект *FunctionPoint* с теми же координатами.

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

@Override
public boolean equals(Object o) {
    if (this == o) return true;
    if (!(o instanceof FunctionPoint)) return false;
    FunctionPoint p = (FunctionPoint) o;
    return MathUtil.equals(this.x, p.x) && MathUtil.equals(this.y, p.y);
}

@Override
public int hashCode() {
    long lx = Double.doubleToLongBits(x);
    long ly = Double.doubleToLongBits(y);
    int hx = (int) (lx ^ (lx >>> 32));
    int hy = (int) (ly ^ (ly >>> 32));
    return hx ^ hy;
}

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

Задание 2 — *ArrayTabulatedFunction*

- Реализованы методы:
  - *toString()*: возвращает набор точек в виде {(...), (...), ...}.
  - *equals(Object o):* корректно сравнивает с любым TabulatedFunction по количеству и координатам точек; добавлена логика для прямого сравнения с другим ArrayTabulatedFunction (при необходимости можно расширить доступность полей для более быстрой работы).
  - *hashCode()*: вычисляется как *XOR*-хэш количества точек и последовательных хэшей точек.
  - *clone()*: глубокая копия — создаётся новый массив FunctionPoint и клонируются все точки.

```java
@Override
public String toString() {
    StringBuilder sb = new StringBuilder();
    sb.append(str: "{");
    for (int i = 0; i < pointsCount; i++) {
        if (i > 0) sb.append(str: ", ");
        sb.append(points[i].toString());
    }
    sb.append(str: "}");
    return sb.toString();
}

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

    if (o instanceof ArrayTabulatedFunction) {
        ArrayTabulatedFunction other = (ArrayTabulatedFunction) o;
        for (int i = 0; i < pointsCount; i++) {
            if (!MathUtil.equals(this.points[i].getX(), other.points[i].getX())) return false;
            if (!MathUtil.equals(this.points[i].getY(), other.points[i].getY())) return false;
        }
        return true;
    }

    for (int i = 0; i < this.getPointsCount(); i++) {
        if (!this.points[i].equals(tf.getPoint(i))) return false;
    }
    return true;
}

@Override
public int hashCode() {
    int h = pointsCount;
    for (int i = 0; i < pointsCount; i++) {
        h ^= points[i].hashCode();
    }
    return h;
}

@Override
public Object clone() {
    FunctionPoint[] pts = new FunctionPoint[pointsCount];
    for (int i = 0; i < pointsCount; i++) {
        pts[i] = (FunctionPoint) points[i].clone();
    }
    return new ArrayTabulatedFunction(pts);
}
```

Задание 3 — *LinkedListTabulatedFunction*

- Реализованы методы:
  - *toString*(): обход списка и формирование строки {(...), ...}.
  - *equals*(*Object o*): корректное сравнение с любым *TabulatedFunction*; при сравнении с другим LinkedListTabulatedFunction возможна оптимизация путём параллельного прохода по узлам.
  - *hashCode*(): *XOR* по хэшам точек и включение количества точек.
  - *clone*(): глубокое копирование — точки клонируются, и новый список собирается по массиву клонов (без клонирования узлов через *Object*.*clone*).

```java
@Override
public String toString() {
    StringBuilder sb = new StringBuilder();
    sb.append(str: "{");
    FunctionNode current = head.next;
    for (int i = 0; i < pointsCount; i++) {
        if (i > 0) sb.append(str: ", ");
        sb.append(current.point.toString());
        current = current.next;
    }
    sb.append(str: "}");
    return sb.toString();
}

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

    if (o instanceof LinkedListTabulatedFunction) {
        LinkedListTabulatedFunction other = (LinkedListTabulatedFunction) o;
        FunctionNode curThis = this.head.next;
        FunctionNode curOther = other.head.next;
        for (int i = 0; i < pointsCount; i++) {
            if (!MathUtil.equals(curThis.point.getX(), curOther.point.getX())) return false;
            if (!MathUtil.equals(curThis.point.getY(), curOther.point.getY())) return false;
            curThis = curThis.next;
            curOther = curOther.next;
        }
        return true;
    }

    FunctionNode current = head.next;
    for (int i = 0; i < pointsCount; i++) {
        if (!current.point.equals(tf.getPoint(i))) return false;
        current = current.next;
    }
    return true;
}

@Override
public int hashCode() {
    int h = pointsCount;
    FunctionNode current = head.next;
    for (int i = 0; i < pointsCount; i++) {
        h ^= current.point.hashCode();
        current = current.next;
    }
    return h;
}

@Override
public Object clone() {
    FunctionPoint[] pts = new FunctionPoint[pointsCount];
    FunctionNode current = head.next;
    for (int i = 0; i < pointsCount; i++) {
        pts[i] = (FunctionPoint) current.point.clone();
        current = current.next;
    }
    return new LinkedListTabulatedFunction(pts);
}
```

Задание 4 — интерфейс *TabulatedFunction*

- В `*src/functions/TabulatedFunction.java*` добавлен `*Object clone*()` и интерфейс теперь наследует `*Cloneable*`.

```java
package functions;

public interface TabulatedFunction extends Function, Cloneable {
    int getPointsCount();
    FunctionPoint getPoint(int index);
    void setPoint(int index, FunctionPoint point);
    double getPointX(int index);
    void setPointX(int index, double x);
    double getPointY(int index);
    void setPointY(int index, double y);

    void deletePoint(int index);
    void addPoint(FunctionPoint point);
    Object clone();
}
```

Задание 5 — проверка

- В `*src/Main.java*` оставлен минимальный демонстрационный код `demoLab5()`:
  - Создание `*ArrayTabulatedFunction*` и `*LinkedListTabulatedFunction*` с одинаковым набором точек.
  - Вывод `*toString*()` для обеих реализаций.
  - Проверка `*equals*()` между объектами разных реализаций и обратная проверка.
  - Печать `*hashCode*()` для обоих объектов.
  - Клонирование обоих объектов, изменение исходных и проверка, что клоны не изменились (глубокое копирование).

```java
import functions.*;
import java.util.Locale;

public class Main {
    Run main | Debug main | Run | Debug
    public static void main(String[] args) {
        Locale.setDefault(Locale.US);
        try {
            demoLab5();
        } catch (Exception e) {
            System.out.println("Ошибка демонстрации lab5: " + e.getMessage());
            e.printStackTrace();
        }
    }

    private static void demoLab5() {
        System.out.println(x: "\n=== Демонстрация лабораторной работы №5 ===");

        FunctionPoint[] points = { new FunctionPoint(x: 0.0, y: 1.2), new FunctionPoint(x: 1.0, y: 3.8), new FunctionPoint(x: 2.0, y: 15.2) };
        ArrayTabulatedFunction a = new ArrayTabulatedFunction(points);
        LinkedListTabulatedFunction l = new LinkedListTabulatedFunction(points);

        System.out.println("toString Array: " + a.toString());
        System.out.println("toString Linked: " + l.toString());

        ArrayTabulatedFunction aClone = (ArrayTabulatedFunction) a.clone();
        LinkedListTabulatedFunction lClone = (LinkedListTabulatedFunction) l.clone();

        System.out.println("equals(Array,ArrayClone): " + a.equals(aClone));
        System.out.println("equals(Linked,LinkedClone): " + l.equals(lClone));
        System.out.println("equals(Array,Linked): " + a.equals(l));
        System.out.println("equals(Linked,Array): " + l.equals(a));

        int hashA_before = a.hashCode();
        int hashL_before = l.hashCode();
        System.out.println("hashCode Array (before): " + hashA_before);
        System.out.println("hashCode Linked (before): " + hashL_before);

        System.out.println(x: "Клонирование произведено.");
        System.out.println(x: "Изменим исходные — увеличим у первой точки на 0.005");
        a.setPointY(index: 0, a.getPointY(index: 0) + 0.005);
        l.setPointY(index: 0, l.getPointY(index: 0) + 0.005);

        System.out.println(x: "После изменения исходных:");
        System.out.println("Original Array: " + a);
        System.out.println("Clone Array:    " + aClone);
        System.out.println("Original Linked: " + l);
        System.out.println("Clone Linked:    " + lClone);

        int hashA_after = a.hashCode();
        int hashL_after = l.hashCode();
        System.out.println("hashCode Array (after): " + hashA_after + " (diff: " + (hashA_after - hashA_before) + ")");
        System.out.println("hashCode Linked (after): " + hashL_after + " (diff: " + (hashL_after - hashL_before) + ")");
    }
}
```

3. Вывод и замечания

- Функционально все требования задания выполнены: методы переопределены и продемонстрированы в `*Main*`.
- Рекомендации:
  - При необходимости можно улучшить оптимизацию `*equals()*` для `*ArrayTabulatedFunction*` и `*LinkedListTabulatedFunction*`, добавив прямой доступ к внутренним полям передаваемых объектов того же класса (потребует изменения видимости полей или добавления *package-private* геттеров).
  - Формат вывода double в `*toString*()` можно зафиксировать (например, до 3 знаков), если преподаватель ожидает конкретное форматирование.
  - Можно объявить `*clone()*` с `throws *CloneNotSupportedException*` в интерфейсе, если требуется соответствие *Java*-конвенциям; текущая реализация возвращает объект и не бросает исключение.

=== Демонстрация лабораторной работы №5 ===
toString Array: {(0.0; 1.2), (1.0; 3.8), (2.0; 15.2)}
toString Linked: {(0.0; 1.2), (1.0; 3.8), (2.0; 15.2)}
equals(Array,ArrayClone): true
equals(Linked,LinkedClone): true
equals(Array,Linked): true
equals(Linked,Array): true
hashCode Array (before): 1930428419
hashCode Linked (before): 1930428419
Клонирование произведено.
Изменим исходные ? увеличим у первой точки на 0.005
После изменения исходных:
Original Array: {(0.0; 1.2049999999999998), (1.0; 3.8), (2.0; 15.2)}
Clone Array: {(0.0; 1.2), (1.0; 3.8), (2.0; 15.2)}
Original Linked: {(0.0; 1.2049999999999998), (1.0; 3.8), (2.0; 15.2)}
Clone Linked: {(0.0; 1.2), (1.0; 3.8), (2.0; 15.2)}
hashCode Array (after): 1415161578 (diff: -515266841)
hashCode Linked (after): 1415161578 (diff: -515266841)