Вкладка 1

Отчет о лабораторной работе №5

Выполнил:
Петров Евгений Станиславович
Группа: 6204-010302D

Вкладка 2

ВВЕДЕНИЕ

Цель работы —Расширить возможности классов, связанных с табулированными функциями, переопределив в них методы, унаследованные из класса Object

В рамках работы были поставлены следующие задачи: реализовать текстовое представление точек и функций; обеспечить точное сравнение объектов и корректное вычисление хэш-кодов; выполнить глубокое клонирование структур с вложенными объектами; проверить корректность работы на реализациях табулированных функций на массиве и на связном списке.

ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

Метод `toString()` используется для человекочитаемого вывода объекта. При его переопределении важно возвращать данные в явном и однозначном формате, отражающем структуру объекта.

Методы `equals()` и `hashCode()` подчиняются строгому контракту: если два объекта считаются равными, их хэш-коды также должны совпадать. Для чисел с плавающей точкой прямое сравнение может приводить к ошибкам, поэтому применяется сравнение битовых представлений с помощью метода `Double.doubleToLongBits`, что позволяет избежать потери точности.

Метод `clone()` используется для копирования объектов. Поверхностное копирование допустимо только для простых объектов без вложенных структур. Для контейнеров, таких как `ArrayTabulatedFunction` и `LinkedListTabulatedFunction`, требуется глубокое копирование, чтобы изменения исходного объекта не влияли на его клон.

Табулированная функция задаётся упорядоченным набором точек вида `(x, y)` и использует линейную интерполяцию между соседними точками. Область определения функции ограничена минимальным и максимальным значениями аргумента `x`.

МЕТОД И РЕАЛИЗАЦИЯ

Структура проекта

Проект состоит из следующих основных файлов и классов:

- `Main.java` — консольная программа для проверки работы всех реализованных методов.
- `FunctionPoint.java` — класс, описывающий точку табулированной функции.
- `ArrayTabulatedFunction.java` — реализация табулированной функции на основе массива точек.
- `LinkedListTabulatedFunction.java` — реализация табулированной функции на основе двусвязного списка.
- `TabulatedFunction.java` — интерфейс функции, расширяющий `Cloneable` и объявляющий метод `clone()`.

Ключевые решения реализации

1. *Метод* `toString()`
   Для класса `FunctionPoint` используется формат `(x; y)`.
   Для табулированных функций реализован вывод в виде списка точек в фигурных скобках: `{(x1; y1), (x2; y2), …}`.

2. *Метод* `equals()`
   В классе `FunctionPoint` сравнение выполняется по битовым представлениям координат `double`.
   В классах `ArrayTabulatedFunction` и `LinkedListTabulatedFunction` реализованы два пути сравнения: быстрый — для объектов одного класса, и универсальный — через интерфейс `TabulatedFunction`. Проверяется количество точек и побитовое равенство каждой точки.

3. *Метод* `hashCode()`
   Хэш-код точки вычисляется как XOR двух `int`, полученных из битового представления координат `double`.
   Хэш-код функции формируется как XOR всех хэш-кодов точек и

количества точек, что позволяет различать функции с одинаковыми значениями, но разной структурой.

4. *Метод* `clone()`
   Для `FunctionPoint` достаточно поверхностного клонирования.
   В `ArrayTabulatedFunction` создаётся новый массив и выполняется клонирование каждой точки.
   В `LinkedListTabulatedFunction` список пересобирается заново, создаются новые узлы и клон-копии точек; кеш индекса при этом сбрасывается.

5. *Обработка погрешностей*
   Для проверки порядка точек по оси X используется значение `EPS = Math.ulp(1.0)`.
   При вычислении значения функции вне области определения возвращается `NaN`.

Шаги запуска и проверки

В методе `main` создаются эквивалентные табулированные функции на массиве и на связном списке. Далее последовательно проверяются корректность работы `toString()`, симметричность `equals()` между разными реализациями, согласованность `hashCode()`, а также корректность глубокого клонирования и независимость клонов от оригиналов.

РЕЗУЛЬТАТЫ

Ожидаемый фрагмент консольного вывода:

```
=== toString() ===
Array:      {(0.0; 0.0), (1.0; 1.0), (2.0; 4.0)}
LinkedList: {(0.0; 0.0), (1.0; 1.0), (2.0; 4.0)}

=== equals() across implementations ===
Array vs LinkedList: true
LinkedList vs Array: true
Array vs new Array:  true
```

```
=== hashCode() consistency ===
Array hash:      1048579
LinkedList hash: 1048579
Tweaked hash (+0.003 to y1): -1169278603

=== clone() deep copy ===
Original array after change: {(0.0; 10.0), (1.0; 1.0),
(2.0; 4.0)}
Array clone preserved:       {(0.0; 0.0), (1.0; 1.0),
(2.0; 4.0)}
Original list after change:  {(0.0; 0.0), (1.0; 1.0),
(2.0; 8.0)}
List clone preserved:        {(0.0; 0.0), (1.0; 1.0),
(2.0; 4.0)}
```

Полученные результаты подтверждают корректность реализации: строковое представление формируется верно, метод `equals()` работает симметрично для разных реализаций, `hashCode()` изменяется при малейших изменениях данных, а клоны не зависят от модификаций исходных объектов.

ТЕСТИРОВАНИЕ И ПРОВЕРКА

Ручной запуск программы `Main` демонстрирует все сценарии, предусмотренные заданием: сравнение одинаковых и различных функций, вывод хэш-кодов и проверку глубокого клонирования.

Дополнительно могут быть проверены граничные случаи: попытка удаления точки при минимально допустимом количестве, добавление точки с уже существующим значением X (должно приводить к выбросу `InappropriateFunctionPointException`), а также вычисление значения функции за пределами области определения, при котором ожидается результат `NaN`.

ВЫВОДЫ

В ходе работы были корректно переопределены все требуемые методы класса `Object`. Реализация обеспечивает читаемое текстовое представление объектов, корректное сравнение и хэширование, а также глубокое клонирование контейнерных структур. Интерфейсная совместимость между различными реализациями табулированных функций сохранена, а поведение системы устойчиво к изменениям данных и ошибочным сценариям.

ИСТОЧНИКИ

1. Java Language Specification и документация Oracle по методам класса `Object` и интерфейсу `Cloneable`.

2. Документация по методу `Double.doubleToLongBits` для точного сравнения чисел с плавающей точкой.