

ЛАБОРАТОРНАЯ РАБОТА №3

**Табулированные функции: реализация на массивах
и связанных списках**

**по курсу
Объектно-ориентированное программирование**

Группа 6204-010302D

Студент: С.О.Куропаткин.

**Преподаватель: Борисов Дмитрий
Сергеевич.**

Задание 1

Изучение классов исключений

Были изучены следующие классы исключений Java:

- **Exception** - базовый класс для всех проверяемых исключений
- **IndexOutOfBoundsException** - выход за границы коллекции
- **ArrayIndexOutOfBoundsException** - выход за границы массива
- **IllegalArgumentException** - неверный аргумент метода
- **IllegalStateException** - недопустимое состояние объекта

Задание 2

Создание пользовательских исключений

Созданы два класса исключений в пакете functions:

FunctionPointIndexOutOfBoundsException.java:

```
package functions;  
  
public class FunctionPointIndexOutOfBoundsException extends  
IndexOutOfBoundsException {  
}
```

Наследуется от `IndexOutOfBoundsException`, используется при обращении к несуществующему индексу точки.

InappropriateFunctionPointException.java:

```
package functions;  
  
public class InappropriateFunctionPointException extends Exception {  
}
```

Наследуется от `Exception`, используется при нарушении порядка точек или дублировании абсцисс.

Задание 3

Модификация `ArrayTabulatedFunction`

Класс `ArrayTabulatedFunction` реализует табулированную функцию с использованием массива.

Конструкторы:

1. Конструктор с равномерным распределением:

```
public ArrayTabulatedFunction(double leftX, double rightX, int pointsCount)
```

- Создает массив с запасом места (`pointsCount * 2`)

- Равномерно распределяет точки по области определения
- Выбрасывает `IllegalArgumentException` при неверных параметрах

2. Конструктор с заданными значениями:

```
public ArrayTabulatedFunction(double leftX, double rightX, double[] values)
```

- Использует переданные значения Y
- Равномерно распределяет X-координаты

Ключевые методы:

getFunctionValue() - линейная интерполяция:

```
public double getFunctionValue(double x)
```

addPoint() - добавление точки с сохранением порядка:

- Проверяет наличие точки с такой же абсциссой
- Определяет позицию для вставки
- Расширяет массив при необходимости (в 2 раза)
- Сохраняет упорядоченность точек по X

deletePoint() - удаление точки:

Обработка исключений:

- Методы доступа: `FunctionPointIndexOutOfBoundsException`
- Изменение точек: `InappropriateFunctionPointException`

Задание 4

Реализация связного списка

Класс `LinkedListTabulatedFunction` использует двусвязный циклический список с выделенной головой.

Внутренний класс `FunctionNode`:

```
private static class FunctionNode{
    private FunctionPoint point;
    private FunctionNode prev, next;
    private FunctionNode(FunctionPoint point){
        this.point = point;
    }
}
```

- **static** - не требует доступа к внешнему классу
- **private** - инкапсуляция внутри внешнего класса
- Содержит данные точки и ссылки на соседние узлы

Структура списка:

- Голова (head) не хранит данных, только связи
- Список всегда циклический, даже когда пустой

Вспомогательные методы:

getNodeByIndex() - оптимизированный доступ:

addNodeToTail() - добавление в конец:

Задание 5

Реализация **LinkedListTabulatedFunction**

Класс реализует тот же интерфейс, что и `ArrayTabulatedFunction`, но с использованием связанного списка.

Особенности реализации:

Конструкторы создают список через вызовы `addNodeToTail()`.

Методы доступа используют `getNodeByIndex()` для эффективного доступа к элементам.

addPoint() - эффективное добавление:

- Проверяет граничные случаи (добавление в начало/конец)
- Находит позицию за один проход по списку
- Использует `addNodeByIndex()` для вставки

getFunctionValue() - линейный поиск отрезка:

```
public double getFunctionValue(double x) throws
InappropriateFunctionPointException {
    if (x < getLeftDomainBorder() || x > getRightDomainBorder()) {
        throw new InappropriateFunctionPointException();
    }
    FunctionNode current = head.next;
    while (current.next != head) {
        double x1 = current.point.getX();
        double y1 = current.point.getY();
        double x2 = current.next.point.getX();
        double y2 = current.next.point.getY();

        if (x >= x1 && x <= x2) {
            return y1 + (x - x1) * (y2 - y1) / (x2 - x1);
        }
        current = current.next;
    }
    throw new InappropriateFunctionPointException();
}
```

Задание 6

Создание интерфейса

Создан интерфейс `TabulatedFunction`, объединяющий оба класса:

```

package functions;

public interface TabulatedFunction {

    //Левая/правая граница области определения функции
    double getLeftDomainBorder() throws InappropriateFunctionPointException;
    double getRightDomainBorder() throws InappropriateFunctionPointException;

    // Метод, возвращающий значение функции в точке x
    double getFunctionValue(double x) throws
InappropriateFunctionPointException;

    // Метод, возвращающий количество точек
    int getPointsCount();
    //Метод, возвращающий ссылку на объект по индексу
    FunctionPoint getPoint(int index);
    //Возвращает X точки с указанным индексом
    double getPointX(int index);
    // Возвращает Y точки с указанным индексом
    double getPointY(int index);

    // Метод, заменяющий точку по индексу на заданную
    void setPoint(int index, FunctionPoint point) throws
InappropriateFunctionPointException;
    // Метод, изменяющий абсциссу точки по индексу
    void setPointX(int index, double x) throws
InappropriateFunctionPointException;
    // Метод, изменяющий ординату точки по индексу
    void setPointY(int index, double y);

    // Удаление точки
    void deletePoint(int index);
    // Добавление точки
    void addPoint(FunctionPoint point) throws
InappropriateFunctionPointException;
}

```

Классы `ArrayTabulatedFunction` и `LinkedListTabulatedFunction` теперь реализуют этот интерфейс.

Задание 7

Тестирование

Создан класс `Main` для тестирования:

Исходные точки:

$x = -1.0, y = 0.0$

$x = -0.5, y = 0.0$

$x = 0.0, y = 0.0$

$x = 0.5, y = 0.0$

$x = 1.0, y = 0.0$

После замены точки:

$x = -1.0, y = 0.0$

$x = -0.5, y = 0.0$

$x = 0.0, y = 0.0$

$x = 0.55, y = 0.25$

$x = 1.0, y = 0.0$

После добавления точки:

$x = -1.0, y = 0.0$

$x = -0.5, y = 0.0$

$x = 0.0, y = 0.0$

$x = 0.55, y = 0.25$

$x = 0.7, y = 0.3$

$x = 1.0, y = 0.0$

После удаления точки:

$x = -1.0, y = 0.0$

$x = -0.5, y = 0.0$

$x = 0.0, y = 0.0$

$x = 0.7, y = 0.3$

$x = 1.0, y = 0.0$

Поймано исключение: `functions.FunctionPointIndexOutOfBoundsException`

Поймано исключение: `functions.InappropriateFunctionPointException`

Поймано исключение: `functions.InappropriateFunctionPointException`

Исходные точки:

$x = -1.0, y = 0.0$

$x = -0.5, y = 0.0$

$x = 0.0, y = 0.0$

$x = 0.5, y = 0.0$

$x = 1.0, y = 0.0$

После замены точки:

$x = -1.0, y = 0.0$

$x = -0.5, y = 0.0$

$x = 0.0, y = 0.0$

$x = 0.55, y = 0.25$

$x = 1.0, y = 0.0$

После добавления точки:

$x = -1.0, y = 0.0$

$x = -0.5, y = 0.0$

$x = 0.0, y = 0.0$

$x = 0.55, y = 0.25$

$x = 0.7, y = 0.3$

$x = 1.0, y = 0.0$

После удаления точки:

$x = -1.0, y = 0.0$

$x = -0.5, y = 0.0$

$x = 0.0, y = 0.0$

$x = 0.7, y = 0.3$

$x = 1.0, y = 0.0$

$f(-0.8) = 0.0$

$f(-0.2) = 0.0$

$f(0.2) = 0.08571428571428572$

$f(0.6) = 0.2571428571428572$

$f(1.5)$ Поймано исключение: `functions.InappropriateFunctionPointException`

Поймано исключение: `functions.FunctionPointIndexOutOfBoundsException`

Поймано исключение: `functions.InappropriateFunctionPointException`

Поймано исключение: `functions.InappropriateFunctionPointException`