

Федеральное государственное автономное образовательное учреждение
высшего образования «Самарский национальный исследовательский
университет имени академика С.П. Королева»

Отчёт

по лабораторной работе №1

Выполнил:
студент группы
6301-030301D

Панин Ю. В.

Проверил:
Борисов Д. С.

Самара, 2025

Содержание

Содержание	1
Задание 1	2
Задание 2	3
Задание 3	4
Задание 4	6
Задание 5	10
Задание 6	12
Задание 7	13

Задание 1

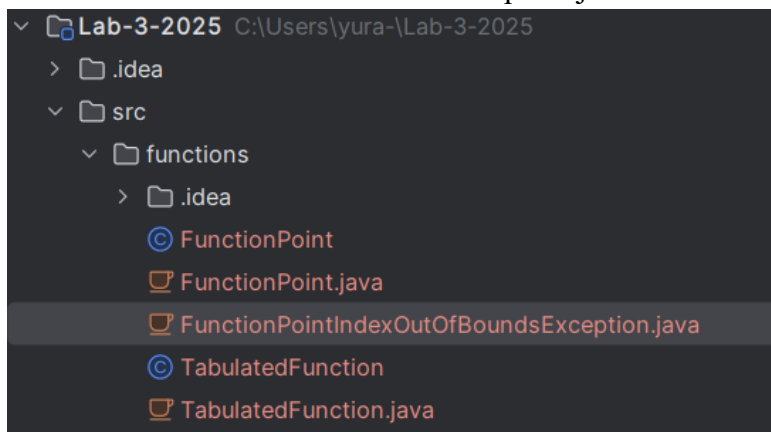
В ходе выполнения задания были изучены классы исключений:

- `java.lang.Exception` - базовый класс для всех исключений
- `java.lang.IndexOutOfBoundsException` - исключение при выходе за границы
- `java.lang.ArrayIndexOutOfBoundsException` - подкласс для массивов
- `java.lang.IllegalArgumentException` - недопустимый аргумент
- `java.lang.IllegalStateException` - недопустимое состояние объекта

Задание 2

При выполнении этого и всех последующих заданий были использованы специальные средства среды разработки - разработка программ проводилась в среде разработки IntelliJ IDEA Community Edition 2025.2.2.

В пакете `functions` был создан класс `FunctionPointIndexOutOfBoundsException.java`.



Данный класс содержит следующий код:

```
package functions;

public class FunctionPointIndexOutOfBoundsException extends IndexOutOfBoundsException { no usages
    public FunctionPointIndexOutOfBoundsException() { 6 usages
        super();
    }

    public FunctionPointIndexOutOfBoundsException(String message) { 6 usages
        super(message);
    }

    public FunctionPointIndexOutOfBoundsException(int index) { 6 usages
        super("Индекс точки выходит за границы: " + index);
    }
}
```

Далее, в пакете functions был создан класс InappropriateFunctionPointException.java, содержащий следующий код:

```
package functions;

public class InappropriateFunctionPointException extends Exception {
    public InappropriateFunctionPointException() {
        super();
    }

    public InappropriateFunctionPointException(String message) {
        super(message);
    }

    public InappropriateFunctionPointException(String message, Throwable cause) {
        super(message, cause);
    }
}
```

Задание 3

Далее, в класс TabulatedFunction были внесены изменения:

1. Согласно заданию в конструкторы были добавлены изменения, обеспечивающие выбрасывание исключений методами класса:

```
if (leftX >= rightX) {
    throw new IllegalArgumentException("Левая граница должна быть меньше правой");
}
if (pointsCount < 2) {
    throw new IllegalArgumentException("Количество точек должно быть не менее 2");
}
```

2. Внесены изменения в getPoint:

```

public FunctionPoint getPoint(int index) { no usages
    if (index < 0 || index >= pointsCount) {
        throw new FunctionPointIndexOutOfBoundsException(index);
    }
    return new FunctionPoint(points[index]);
}

```

3. SetPoint:

```

public void setPoint(int index, FunctionPoint point) throws InappropriateFunctionPointException {
    if (index < 0 || index >= pointsCount) {
        throw new FunctionPointIndexOutOfBoundsException(index);
    }
}

```

4. GetPointX:

```

public double getPointX(int index) { no usages
    if (index < 0 || index >= pointsCount) {
        throw new FunctionPointIndexOutOfBoundsException(index);
    }
    return points[index].getX();
}

```

5. SetPointX:

```

public void setPointX(int index, double x) throws InappropriateFunctionPointException {
    if (index < 0 || index >= pointsCount) {
        throw new FunctionPointIndexOutOfBoundsException(index);
    }
}

```

6. GetPointY:

```

public double getPointY(int index) { no usages
    if (index < 0 || index >= pointsCount) {
        throw new FunctionPointIndexOutOfBoundsException(index);
    }
    return points[index].getY();
}

```

7. SetPointY

```

public void setPointY(int index, double y) { no usages
    if (index < 0 || index >= pointsCount) {
        throw new FunctionPointIndexOutOfBoundsException(index);
    }
    points[index].setY(y);
}

```

8. DeletePoint:

```

public void deletePoint(int index) { no usages
    if (index < 0 || index >= pointsCount) {
        throw new FunctionPointIndexOutOfBoundsException(index);
    }
    if (pointsCount < 3) {
        throw new IllegalStateException("Нельзя удалить точку: должно остаться минимум 2 точки");
    }

    System.arraycopy(points, index + 1, points, index, pointsCount - index - 1);
    pointsCount--;
    points[pointsCount] = null;
}

```

Задание 4

В пакете functions был создан файл LinkedListTabulatedFunction, содержащий программу, выполняющую поставленные задачи.

1. Описан класс элементов списка FunctionNode:

```

// Внутренний класс для узла списка
private static class FunctionNode { 29 usages
    FunctionPoint point; 29 usages
    FunctionNode prev; 14 usages
    FunctionNode next; 27 usages

    FunctionNode(FunctionPoint point) { 3 usages
        this.point = point;
        this.prev = null;
        this.next = null;
    }
}

```

2. Описан класс LinkedListTabulatedFunction объектов списка, содержащий поле ссылки на объект головы, а также иные вспомогательные поля:

```

public LinkedListTabulatedFunction(double leftX, double rightX, int pointsCount) { 23 usages
    if (leftX >= rightX) {
        throw new IllegalArgumentException("Левая граница должна быть меньше правой");
    }
    if (pointsCount < 2) {
        throw new IllegalArgumentException("Количество точек должно быть не менее 2");
    }

    // Создаем голову списка
    head = new FunctionNode(null);
    head.next = head;
    head.prev = head;
    this.pointsCount = 0;

    double step = (rightX - leftX) / (pointsCount - 1);
    for (int i = 0; i < pointsCount; i++) {
        double x = leftX + i * step;
        addNodeToTail(new FunctionPoint(x, 0));
    }
}

```

3. В классе `LinkedListTabulatedFunction` реализован метод `FunctionNode` `getNodeByIndex(int index)`, возвращающий ссылку на объект элемента списка по его номеру. Нумерация значащих элементов (голова списка в данном случае к ним не относится) начинается с 0. Метод обеспечивает оптимизацию доступа к элементам списка:

```

private FunctionNode getNodeByIndex(int index) { 14 usages
    if (index < 0 || index >= pointsCount) {
        throw new FunctionPointIndexOutOfBoundsException(index);
    }

    // Оптимизация: если обращаемся к тому же или соседнему узлу
    if (lastAccessedNode != null && lastAccessedIndex != -1) {
        if (index == lastAccessedIndex) {
            return lastAccessedNode;
        } else if (index == lastAccessedIndex + 1 && index < pointsCount) {
            lastAccessedNode = lastAccessedNode.next;
            lastAccessedIndex++;
            return lastAccessedNode;
        } else if (index == lastAccessedIndex - 1 && index >= 0) {
            lastAccessedNode = lastAccessedNode.prev;
            lastAccessedIndex--;
            return lastAccessedNode;
        }
    }
}

```

```

// Линейный поиск
FunctionNode current;
if (index < pointsCount / 2) {
    // Идем от начала
    current = head.next;
    for (int i = 0; i < index; i++) {
        current = current.next;
    }
} else {
    // Идем от конца
    current = head.prev;
    for (int i = pointsCount - 1; i > index; i--) {
        current = current.prev;
    }
}

lastAccessedNode = current;
lastAccessedIndex = index;
return current;
}

```

4. В классе `LinkedListTabulatedFunction` реализован метод `FunctionNode addNodeToTail()`, добавляющий новый элемент в конец списка и возвращающий ссылку на объект этого элемента:

```
private FunctionNode addNodeToTail(FunctionPoint point) { 2 usages
    FunctionNode newNode = new FunctionNode(new FunctionPoint(point));
    FunctionNode tail = head.prev;

    newNode.prev = tail;
    newNode.next = head;
    tail.next = newNode;
    head.prev = newNode;

    pointsCount++;
    lastAccessedNode = newNode;
    lastAccessedIndex = pointsCount - 1;

    return newNode;
}
```

5. В классе `LinkedListTabulatedFunction` реализован метод `FunctionNode addNodeByIndex(int index)`, добавляющий новый элемент в указанную позицию списка и возвращающий ссылку на объект этого элемента:

```
private FunctionNode addNodeByIndex(int index, FunctionPoint point) throws InappropriateFunctionPointException {
    if (index < 0 || index > pointsCount) {
        throw new FunctionPointIndexOutOfBoundsException(index);
    }
}
```

6. В классе `LinkedListTabulatedFunction` реализован метод `FunctionNode deleteNodeByIndex(int index)`, удаляющий элемент списка по номеру и возвращающий ссылку на объект удаленного элемента:

```
private FunctionNode deleteNodeByIndex(int index) { 1 usage
    if (index < 0 || index >= pointsCount) {
        throw new FunctionPointIndexOutOfBoundsException(index);
    }
    if (pointsCount < 3) {
        throw new IllegalStateException("Нельзя удалить точку: должно остаться минимум 2 точки");
    }

    FunctionNode nodeToDelete = getNodeByIndex(index);
    FunctionNode prevNode = nodeToDelete.prev;
    FunctionNode nextNode = nodeToDelete.next;

    prevNode.next = nextNode;
    nextNode.prev = prevNode;

    pointsCount--;

    // Сбрасываем кэш, если удалили его элемент
    if (lastAccessedNode == nodeToDelete) {
        lastAccessedNode = null;
        lastAccessedIndex = -1;
    } else if (lastAccessedIndex > index) {
        lastAccessedIndex--;
    }

    return nodeToDelete;
}
```

Класс `FunctionNode` должен быть реализован как внутренний приватный статический класс внутри `LinkedListTabulatedFunction` так как, во-первых, `FunctionNode` является неотъемлемой частью реализации связанного списка и не имеет смысла вне контекста `LinkedListTabulatedFunction`, во-вторых, пользователям пакета `functions` не нужно знать о внутреннем устройстве списка и, в-третьих, только `LinkedListTabulatedFunction` должен иметь доступ к узлам списка.

Обоснование выбора уровня инкапсуляции: поля `FunctionNode` не `private` с геттерами/сеттерами, так как прямой доступ к полям быстрее вызова методов, меньше повторяющегося кода, реализуется контроль доступа: класс `private`, поэтому доступ имеют только методы `LinkedListTabulatedFunction`. Кроме того не реализуются `public/protected` поля так как внешний код мог бы изменять связи между узлами, изменение связей извне могло бы нарушить структуру списка, а также при ошибках трудно отследить, кто изменил связи. Поэтому данная реализация инкапсуляции вполне оправдана.

Задание 5

Были определены методы, требующие оптимизации под связанный список, и оптимизированы:

1. `GetPoint`:

```
public FunctionPoint getPoint(int index) { no usages
    FunctionNode node = getNodeByIndex(index);
    return new FunctionPoint(node.point);
}
```

2. `SetPoint`:

```
public void setPoint(int index, FunctionPoint point) throws InappropriateFunctionPointException {
    FunctionNode node = getNodeByIndex(index);

    double newX = point.getX();
    if ((index > 0 && newX <= getNodeByIndex(index - 1).point.getX()) ||
        (index < pointsCount - 1 && newX >= getNodeByIndex(index + 1).point.getX())) {
        throw new InappropriateFunctionPointException("Нарушение порядка точек по X");
    }

    node.point = new FunctionPoint(point);
}
```

3. `GetPointX`:

```
public double getPointX(int index) { no usages
    return getNodeByIndex(index).point.getX();
}
```

4. SetPointX:

```
public void setPointX(int index, double x) throws InappropriateFunctionPointException {
    FunctionNode node = getNodeByIndex(index);

    if ((index > 0 && x <= getNodeByIndex(index - 1).point.getX()) ||
        (index < pointsCount - 1 && x >= getNodeByIndex(index + 1).point.getX())) {
        throw new InappropriateFunctionPointException("Нарушение порядка точек по X");
    }

    node.point.setX(x);
}
```

5. GetPointY:

```
public double getPointY(int index) { no usages
    return getNodeByIndex(index).point.getY();
}
```

6. SetPointY:

```
public void setPointY(int index, double y) {
    getNodeByIndex(index).point.setY(y);
}
```

7. DeletePoint:

```
public void deletePoint(int index) {
    deleteNodeByIndex(index);
}
```

Таким образом, оптимизация состоит в следующем:

1. Метод getNodeByIndex(int index) - основная оптимизация:

- 1.1.Кэширование последнего доступного узла (lastAccessedNode и lastAccessedIndex).
- 1.2.Быстрый доступ к соседним элементам ($O(1)$ вместо $O(n)$).
- 1.3.Выбор направления обхода (с начала или с конца) для минимизации количества шагов.
2. Метод `getFunctionValue(double x)` - используется прямое перемещение по списку без вызова `getNodeByIndex`, так как поиск интервала требует последовательного прохода.
3. Метод `addPoint(FunctionPoint point)` - объединение поиска позиции и проверки на дублирование в один проход по списку.
4. Оптимизация добавления/удаления узлов: методы `addNodeToTail`, `addNodeByIndex`, `deleteNodeByIndex` - используют прямые операции со ссылками $O(1)$ для вставки/удаления. Обновляют кэш (lastAccessedNode, lastAccessedIndex) при изменении структуры списка.
5. Вспомогательный метод `doubleEquals` - единый метод для сравнения `double` с учетом машинного эпсилона вместо повторения проверок.

Задание 6

Класс `TabulatedFunction` переименован в класс `ArrayTabulatedFunction`.

Создан интерфейс `TabulatedFunction`, содержащий объявления общих методов классов `ArrayTabulatedFunction` и `LinkedListTabulatedFunction`:

```
package functions;

public interface TabulatedFunction {
    // Получение границ области определения
    double getLeftDomainBorder();
    double getRightDomainBorder();

    // Получение значения функции
    double getFunctionValue(double x);

    // Работа с точками
    int getPointsCount();
    FunctionPoint getPoint(int index);
    void setPoint(int index, FunctionPoint point) throws InappropriateFunctionPointException;
    double getPointX(int index);
    void setPointX(int index, double x) throws InappropriateFunctionPointException;
    double getPointY(int index);
    void setPointY(int index, double y);
    void deletePoint(int index);
    void addPoint(FunctionPoint point) throws InappropriateFunctionPointException;
}
```

Также изменен **FunctionPoint**: добавлен конструктор копирования:

```
public FunctionPoint(double x, double y) {  
    this.x = x;  
    this.y = y;  
}
```

Задание 7

В созданный ранее класс Main добавлена проверка для случаев, в которых объект табулированной функции должен выбрасывать исключения:

```
private static void testArrayFunction() {
    try {
        // 1.1 Некорректное создание функции
        System.out.println(" 1.1 Попытка создания с некорректными параметрами:");
        try {
            TabulatedFunction func = new ArrayTabulatedFunction(5, 3, 4);
        } catch (IllegalArgumentException e) {
            System.out.println("    Поймано исключение: " + e.getMessage());
        }

        try {
            TabulatedFunction func = new ArrayTabulatedFunction(0, 5, 1);
        } catch (IllegalArgumentException e) {
            System.out.println("    Поймано исключение: " + e.getMessage());
        }

        // 1.2 Корректное создание
        System.out.println("\n 1.2 Корректное создание функции:");
        TabulatedFunction func = new ArrayTabulatedFunction(0, 10, 5);
        System.out.println("    Функция создана. Точек: " + func.getPointsCount());

        // 1.3 Выход за границы индексов
        System.out.println("\n 1.3 Тест выхода за границы индексов:");
        try {
            func.getPoint(10);
        } catch (FunctionPointIndexOutOfBoundsException e) {
            System.out.println("    Поймано исключение: " + e.getMessage());
        }
    }
}
```

```

// 1.4 Некорректная установка точки
System.out.println("\n 1.4 Тест некорректной установки точки:");
try {
    func.setPoint(2, new FunctionPoint(8, 5));
} catch (InappropriateFunctionPointException e) {
    System.out.println("    Поймано исключение: " + e.getMessage());
}

// 1.5 Некорректное добавление точки
System.out.println("\n 1.5 Тест некорректного добавления точки:");
try {
    func.addPoint(new FunctionPoint(2.5, 3));
    func.addPoint(new FunctionPoint(2.5, 4)); // Дублирование X
} catch (InappropriateFunctionPointException e) {
    System.out.println("    Поймано исключение: " + e.getMessage());
}

// 1.6 Некорректное удаление точки
System.out.println("\n 1.6 Тест некорректного удаления:");
// Сначала добавим точки
try {
    func.deletePoint(0);
    func.deletePoint(0);
    func.deletePoint(0); // Попытка удалить третью точку (останется 2)
} catch (IllegalStateException e) {
    System.out.println("    Поймано исключение: " + e.getMessage());
}

} catch (Exception e) {
    System.out.println("Неожиданное исключение: " + e);
    e.printStackTrace();
}
}

```

Ссылочная переменная для работы с объектом функции объявлена типа `TabulatedFunction`, а при создании объекта указан реальный класс:

```

private static void testLinkedListFunction() {
    try {
        // 2.1 Корректное создание
        System.out.println(" 2.1 Корректное создание функции:");
        TabulatedFunction func = new LinkedListTabulatedFunction(-5, 5, 6);
        System.out.println("    Функция создана. Точек: " + func.getPointsCount());

        // Массив функций для тестирования
        TabulatedFunction[] functions = new TabulatedFunction[2];

        // Создаем функции разных типов
        functions[0] = new ArrayTabulatedFunction(0, Math.PI, new double[]{0, 1, 0, -1, 0});
        functions[1] = new LinkedListTabulatedFunction(-2, 2, new double[]{4, 1, 0, 1, 4});
    }
}

```

После этого была запущена на проверку полученная программа:

```

PS C:\Users\yura-\Lab-3-2025> cd src
PS C:\Users\yura-\Lab-3-2025\src> javac functions/*.java
PS C:\Users\yura-\Lab-3-2025\src> javac Main.java
PS C:\Users\yura-\Lab-3-2025\src> java Main

```

Которая выполняет поставленные задачи:

=== Тестирование лабораторной работы №3 ===

1. Тестирование ArrayTabulatedFunction:

1.1 Попытка создания с некорректными параметрами:

Поймано исключение: Левая граница должна быть меньше правой

Поймано исключение: Количество точек должно быть не менее 2

1.2 Корректное создание функции:

Функция создана. Точек: 5

1.3 Тест выхода за границы индексов:

Поймано исключение: Индекс точки выходит за границы: 10

1.4 Тест некорректной установки точки:

Поймано исключение: Нарушение порядка точек по X

1.5 Тест некорректного добавления точки:

Поймано исключение: Точка с таким X уже существует

1.6 Тест некорректного удаления:

=====

=====

2. Тестирование LinkedListTabulatedFunction:

2.1 Корректное создание функции:

Функция создана. Точек: 6

2.2 Работа с точками:

Значения в точках:

$f(-5,0) = 0,0$

$f(-3,0) = 1,0$

$f(-1,0) = 4,0$

$f(1,0) = 9,0$

$f(3,0) = 16,0$

$f(5,0) = 25,0$

2.3 Добавление и удаление точек:

Точка (2.5, 6.25) добавлена

Точка с индексом 3 удалена

2.4 Тест интерполяции:

$f(0.5) = 4,96$

$f(10) = \text{NaN}$

=====

3. Полиморфное использование:

Работа через интерфейс `TabulatedFunction`:

Функция $\sin(x)$ (`ArrayTabulatedFunction`):

Область определения: `[0.0, 3.141592653589793]`

Количество точек: 5

$f(0,5) = 0,637$

$f(1,0) = 0,727$

$f(1,5) = 0,090$

Функция x^2 (`LinkedListTabulatedFunction`):

Область определения: `[-2.0, 2.0]`

Количество точек: 5

$f(0,5) = 0,500$

$f(1,0) = 1,000$

$f(1,5) = 2,500$

=== Тестирование завершено ===