Лабораторная работа №3

Коновалов Сергей Сергеевич 6204-010302D

Содержание

Задание 1

Изучил классы исключений Java API:

- java.lang.Exception - базовый класс для проверяемых исключений

- java.lang.IndexOutOfBoundsException - выход за границы индекса

- java.lang.ArrayIndexOutOfBoundsException - выход за границы массива

- java.lang.IllegalArgumentException - недопустимый аргумент

- java.lang.IllegalStateException - недопустимое состояние объекта

Задание 2

Создал классы исключений FunctionPointIndexOutOfBoundsException и InappropriateFunctionPointException в пакете functions:

```
public class FunctionPointIndexOutOfBoundsException extends
IndexOutOfBoundsException {
    public FunctionPointIndexOutOfBoundsException() {
        super();
    }

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

    public FunctionPointIndexOutOfBoundsException(String message, Throwable cause)
{
        super(message);
        this.initCause(cause);
    }
}
public class InappropriateFunctionPointException extends Exception {

    public InappropriateFunctionPointException() {
        super();
    }

    public InappropriateFunctionPointException(String message) {
```

```
        super(message);
    }

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

    public InappropriateFunctionPointException(Throwable cause) {
        super(cause);
    }
}
```

Задание 3

В класс TabulatedFunction добавил проверки и исключения согласно требованиям:

```
public TabulatedFunction(double leftX, double rightX, int pointsCount) {
        if (leftX >= rightX) {
            throw new IllegalArgumentException("Левая граница области определения не может быть больше или равна правой");
        }
        if (pointsCount < 2) {
            throw new IllegalArgumentException("Количество точек не может быть меньше двух");
        }

public TabulatedFunction(double leftX, double rightX, double[] values) {
        if (leftX >= rightX) {
            throw new IllegalArgumentException("Левая граница области определения не может быть больше или равна правой");
        }
        if (values.length < 2) {
            throw new IllegalArgumentException("Количество точек не может быть меньше двух");
        }
```

```
 if (index < 0 || index >= pointsCount) {
            throw new FunctionPointIndexOutOfBoundsException(
                "Индекс " + index + " вне границ [0, " + (pointsCount - 1) + "]"
            );
        }

        if (index == 0) {
                if (newX >= points[1].getX()) {
                    throw new InappropriateFunctionPointException(
                        "Новая координата x=" + newX + " должна быть меньше " +
        points[1].getX()
                    );
                }
            }
            else if (index == pointsCount - 1) {
                if (newX <= points[pointsCount - 2].getX()) {
                    throw new InappropriateFunctionPointException(
                        "Новая координата x=" + newX + " должна быть больше " +
        points[pointsCount - 2].getX()
                    );
                }
            }
            else {
                if (newX <= points[index - 1].getX() || newX >= points[index + 1].getX()) {
                    throw new InappropriateFunctionPointException(
                        "Новая координата x=" + newX + " должна быть в интервале (" +
                        points[index - 1].getX() + ", " + points[index + 1].getX() + ")"
                    );
                }
    }

        public void deletePoint(int index) {
            if (index < 0 || index >= pointsCount) {
                throw new FunctionPointIndexOutOfBoundsException(
```

```
                "Индекс " + index + " вне границ [0, " + (pointsCount - 1) + "]"
            );
        }

        if (pointsCount < 3) {
            throw new IllegalStateException("Невозможно удалить точку: количество
    точек не может быть меньше двух");
}

    public void addPoint(FunctionPoint point) throws
    InappropriateFunctionPointException {
        for (int i = 0; i < pointsCount; i++) {
            if (point.getX() == points[i].getX()) {
                throw new InappropriateFunctionPointException(
                    "Точка с x=" + point.getX() + " уже существует"
                );
            }
}
```

Задание 4

```
public class LinkedListTabulatedFunction implements TabulatedFunction {
    private FunctionNode head;
    private int pointsCount;
    private FunctionNode lastAccessedNode;
    private int lastAccessedIndex;
    protected static class FunctionNode {
        private FunctionPoint point;
        private FunctionNode prev;
        private FunctionNode next;
        public FunctionNode(FunctionPoint point, FunctionNode prev, FunctionNode
next) {
            this.point = point;
            this.prev = prev;
            this.next = next;
        }
        public FunctionNode(FunctionNode node) {
            this.point = new FunctionPoint(node.point);
            this.prev = node.prev;
            this.next = node.next;
        }
        FunctionPoint getPoint() {
            return point;
        }
        void setPoint(FunctionPoint point) {
            this.point = point;
        }
        FunctionNode getPrev() {
            return prev;
        }
        void setPrev(FunctionNode prev) {
            this.prev = prev;
```

```
        }
        FunctionNode getNext() {
            return next;
        }
        void setNext(FunctionNode next) {
            this.next = next;
        }
    }
    private void initializeList() {
        head = new FunctionNode(null, null, null);
        head.setPrev(head);
        head.setNext(head);
        pointsCount = 0;
        lastAccessedNode = head;
        lastAccessedIndex = -1;
    }
    private FunctionNode addNodeToTail() {
        FunctionNode newNode = new FunctionNode(null, head.getPrev(), head);
        head.getPrev().setNext(newNode);
        head.setPrev(newNode);
        pointsCount++;
        lastAccessedIndex = -1;
        return newNode;
    }
    public LinkedListTabulatedFunction(double leftX, double rightX, int pointsCount)
{
        if (leftX >= rightX) {
            throw new IllegalArgumentException("Левая граница области
определения не может быть больше или равна правой");
        }
        if (pointsCount < 2) {
            throw new IllegalArgumentException("Количество точек не может быть
меньше двух");
        }
```

```
        initializeList();
        double step = (rightX - leftX) / (pointsCount - 1);
        for (int i = 0; i < pointsCount; i++) {
            double x = leftX + i * step;
            addNodeToTail().setPoint(new FunctionPoint(x, 0.0));
        }
    }
    public LinkedListTabulatedFunction(double leftX, double rightX, double[] values)
{
        if (leftX >= rightX) {
            throw new IllegalArgumentException("Левая граница области
определения не может быть больше или равна правой");
        }
        if (values.length < 2) {
            throw new IllegalArgumentException("Количество точек не может быть
меньше двух");
        }
        initializeList();
        double step = (rightX - leftX) / (values.length - 1);
        for (int i = 0; i < values.length; i++) {
            double x = leftX + i * step;
            addNodeToTail().setPoint(new FunctionPoint(x, values[i]));
        }
    }
    private FunctionNode getNodeByIndex(int index) {
        if (index < 0 || index >= pointsCount) {
            throw new FunctionPointIndexOutOfBoundsException(
                "Индекс " + index + " вне границ [0, " + (pointsCount - 1) + "]"
            );
        }
        if (lastAccessedIndex != -1) {
            int diff = index - lastAccessedIndex;
            if (Math.abs(diff) == 1) {
```

```
            lastAccessedNode = (diff > 0) ? lastAccessedNode.getNext() :
lastAccessedNode.getPrev();
            lastAccessedIndex = index;
            return lastAccessedNode;
        } else if (Math.abs(diff) < index && Math.abs(diff) < pointsCount - index - 1) {
            FunctionNode currentNode = lastAccessedNode;
            int currentIndex = lastAccessedIndex;
            while (currentIndex != index) {
                currentNode = (index > currentIndex) ? currentNode.getNext() :
currentNode.getPrev();
                currentIndex += (index > currentIndex) ? 1 : -1;
            }
            lastAccessedNode = currentNode;
            lastAccessedIndex = index;
            return currentNode;
        }
    }
    FunctionNode currentNode;
    int currentIndex;
    if (index < pointsCount - index) {
        currentNode = head.getNext();
        currentIndex = 0;
        while (currentIndex < index) {
            currentNode = currentNode.getNext();
            currentIndex++;
        }
    } else {
        currentNode = head.getPrev();
        currentIndex = pointsCount - 1;
        while (currentIndex > index) {
            currentNode = currentNode.getPrev();
            currentIndex--;
        }
    }
```

```
        lastAccessedNode = currentNode;
        lastAccessedIndex = index;
        return currentNode;
    }
    private FunctionNode addNodeByIndex(int index) {
        if (index < 0 || index > pointsCount) {
            throw new FunctionPointIndexOutOfBoundsException(
                "Индекс " + index + " вне границ [0, " + pointsCount + "]"
            );
        }
        if (index == pointsCount) {
            return addNodeToTail();
        }
        FunctionNode nextNode = getNodeByIndex(index);
        FunctionNode prevNode = nextNode.getPrev();
        FunctionNode newNode = new FunctionNode(null, prevNode, nextNode);
        prevNode.setNext(newNode);
        nextNode.setPrev(newNode);
        pointsCount++;
        lastAccessedIndex = -1;
        return newNode;
    }
    private FunctionNode deleteNodeByIndex(int index) {
        if (index < 0 || index >= pointsCount) {
            throw new FunctionPointIndexOutOfBoundsException(
                "Индекс " + index + " вне границ [0, " + (pointsCount - 1) + "]"
            );
        }
        if (pointsCount < 3) {
            throw new IllegalStateException("Невозможно удалить точку: количество
точек не может быть меньше двух");
        }
```

```
    FunctionNode nodeToDelete = getNodeByIndex(index);
    FunctionNode prevNode = nodeToDelete.getPrev();
    FunctionNode nextNode = nodeToDelete.getNext();
    prevNode.setNext(nextNode);
    nextNode.setPrev(prevNode);
    pointsCount--;
    if (lastAccessedIndex == index) {
        lastAccessedIndex = -1;
        lastAccessedNode = head;
    } else if (lastAccessedIndex > index) {
        lastAccessedIndex--;
    }
    return nodeToDelete;
}
```

Задание 5

Сделал методы табулированной функции для LinkedListTabulatedFunction:

```
public double getLeftDomainBorder() {
    if (pointsCount == 0) {
        throw new IllegalStateException("Функция не содержит точек");
    }
    return head.getNext().getPoint().getX();
}

public double getRightDomainBorder() {
    if (pointsCount == 0) {
        throw new IllegalStateException("Функция не содержит точек");
    }
    return head.getPrev().getPoint().getX();
}

private int findNodeIndex(FunctionNode node) {
FunctionNode current = head.getNext();
for (int i = 0; i < pointsCount; i++) {
    if (current == node) {
        return i;
    }
    current = current.getNext();
}
return -1;
}

public double getFunctionValue(double x) {
    if (pointsCount == 0) return Double.NaN;

    FunctionNode startNode = (lastAccessedIndex != -1) ? lastAccessedNode :
head.getNext();
```

```
int startIndex = (lastAccessedIndex != -1) ? lastAccessedIndex : 0;

FunctionNode currentNode = startNode;
int checkedNodes = 0;

while (checkedNodes < pointsCount) {
    FunctionPoint point1 = currentNode.getPoint();
    FunctionPoint point2 = currentNode.getNext().getPoint();

    if (point1 == null || point2 == null) {
        currentNode = currentNode.getNext();
        checkedNodes++;
        continue;
    }

    double x1 = point1.getX();
    double x2 = point2.getX();

    if (x >= x1 && x <= x2) {
        double y1 = point1.getY();
        double y2 = point2.getY();

        lastAccessedNode = currentNode;
        lastAccessedIndex = findNodeIndex(currentNode);

        return y1 + (y2 - y1) * (x - x1) / (x2 - x1);
    }

    currentNode = currentNode.getNext();
    checkedNodes++;

    if (currentNode == head) {
        currentNode = head.getNext();
```

```
            }
        }

        return Double.NaN;
    }

    public int getPointsCount() {
        return pointsCount;
    }

    public FunctionPoint getPoint(int index) {
        return new FunctionPoint(getNodeByIndex(index).getPoint());
    }

    public void setPoint(int index, FunctionPoint point) throws
InappropriateFunctionPointException {
        FunctionNode node = getNodeByIndex(index);
        double newX = point.getX();

        FunctionNode prevNode = node.getPrev();
        FunctionNode nextNode = node.getNext();

        if (prevNode == head) {
            if (newX >= nextNode.getPoint().getX()) {
                throw new InappropriateFunctionPointException(
                    "Новая координата x=" + newX + " должна быть меньше " +
nextNode.getPoint().getX()
                );
            }
        } else if (nextNode == head) {
            if (newX <= prevNode.getPoint().getX()) {
                throw new InappropriateFunctionPointException(
                    "Новая координата x=" + newX + " должна быть больше " +
prevNode.getPoint().getX()
```

```
                );
            }
        } else {
            if (newX <= prevNode.getPoint().getX() || newX >=
nextNode.getPoint().getX()) {
                throw new InappropriateFunctionPointException(
                    "Новая координата x=" + newX + " должна быть в интервале (" +
                    prevNode.getPoint().getX() + ", " + nextNode.getPoint().getX() + ")"
                );
            }
        }

        node.setPoint(new FunctionPoint(point));
    }

    public double getPointX(int index) {
        return getNodeByIndex(index).getPoint().getX();
    }

    public double getPointY(int index) {
        return getNodeByIndex(index).getPoint().getY();
    }

    public void setPointX(int index, double x) throws
InappropriateFunctionPointException {
        FunctionPoint currentPoint = getNodeByIndex(index).getPoint();
        setPoint(index, new FunctionPoint(x, currentPoint.getY()));
    }

    public void setPointY(int index, double y) {
        FunctionNode node = getNodeByIndex(index);
        FunctionPoint currentPoint = node.getPoint();
        node.setPoint(new FunctionPoint(currentPoint.getX(), y));
    }
```

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

    public void addPoint(FunctionPoint point) throws
InappropriateFunctionPointException {
        FunctionNode currentNode = head.getNext();
        for (int i = 0; i < pointsCount; i++) {
            if (point.getX() == currentNode.getPoint().getX()) {
                throw new InappropriateFunctionPointException(
                    "Точка с x=" + point.getX() + " уже существует"
                );
            }
            currentNode = currentNode.getNext();
        }

        int insertIndex = 0;
        currentNode = head.getNext();
        while (insertIndex < pointsCount && point.getX() >
currentNode.getPoint().getX()) {
            currentNode = currentNode.getNext();
            insertIndex++;
        }

        FunctionNode newNode = addNodeByIndex(insertIndex);
        newNode.setPoint(new FunctionPoint(point));

        FunctionNode prevNode = newNode.getPrev();
        FunctionNode nextNode = newNode.getNext();

        if ((prevNode != head && newNode.getPoint().getX() <=
prevNode.getPoint().getX()) ||
            (nextNode != head && newNode.getPoint().getX() >=
```

```
            nextNode.getPoint().getX())) {
                deleteNodeByIndex(insertIndex);
                throw new InappropriateFunctionPointException(
                    "Новая точка нарушает упорядоченность функции"
                );
            }
    }
```

Задание 6

Класс TabulatedFunction переименовал в класс ArrayTabulatedFunction.

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

```
public interface TabulatedFunction {

    double getLeftDomainBorder();

    double getRightDomainBorder();

    double getFunctionValue(double x);

    int getPointsCount();

    FunctionPoint getPoint(int index);

    void setPoint(int index, FunctionPoint point) throws
InappropriateFunctionPointException;

    double getPointX(int index);

    double getPointY(int index);

    void setPointX(int index, double x) throws InappropriateFunctionPointException;

    void setPointY(int index, double y);

    void deletePoint(int index);

    void addPoint(FunctionPoint point) throws InappropriateFunctionPointException;
}
```

Задание 7

Создал Main.java для тестирования функций и проверок:

```
public class Main {

    private static void testFunctionExceptions(TabulatedFunction function) {
        System.out.println("\nТестирование исключений:");
```

```
        try {
            function.getPoint(-1);
            System.out.println("ОШИБКА: Ожидалось
FunctionPointIndexOutOfBoundsException");
        } catch (FunctionPointIndexOutOfBoundsException e) {
            System.out.println("FunctionPointIndexOutOfBoundsException: " +
e.getMessage());
        }

        try {
            function.getPoint(100);
            System.out.println("ОШИБКА: Ожидалось
FunctionPointIndexOutOfBoundsException");
        } catch (FunctionPointIndexOutOfBoundsException e) {
            System.out.println("FunctionPointIndexOutOfBoundsException: " +
e.getMessage());
        }

        try {
            function.setPointX(2, function.getPointX(0) - 1);
            System.out.println("ОШИБКА: Ожидалось
InappropriateFunctionPointException");
        } catch (InappropriateFunctionPointException e) {
            System.out.println("InappropriateFunctionPointException: " + e.getMessage());
        }

        try {
            function.addPoint(new FunctionPoint(function.getPointX(1), 10.0));
            System.out.println("ОШИБКА: Ожидалось
InappropriateFunctionPointException");
        } catch (InappropriateFunctionPointException e) {
            System.out.println("InappropriateFunctionPointException: " + e.getMessage());
        }

        TabulatedFunction tempFunction = new ArrayTabulatedFunction(0, 2, 2);
        try {
            tempFunction.deletePoint(0);
            System.out.println("ОШИБКА: Ожидалось IllegalStateException");
        } catch (IllegalStateException e) {
            System.out.println("IllegalStateException: " + e.getMessage());
        }

        try {
            function.addPoint(new FunctionPoint(2.5, 10.0));
            System.out.println("Точка успешно добавлена");

            function.setPointY(1, 15.0);
            System.out.println("Y-координата успешно изменена");

        } catch (InappropriateFunctionPointException e) {
            System.out.println("ОШИБКА: " + e.getMessage());
        }
    }

    private static void testTabulatedFunction(String functionName, TabulatedFunction
function) {
        System.out.println("Тестирование: " + functionName);
```

```
        System.out.println("Область определения: [" +
            function.getLeftDomainBorder() + ", " + function.getRightDomainBorder() + "]");
        System.out.println("Количество точек: " + function.getPointsCount());

        System.out.println("Точки функции:");
        for (int i = 0; i < function.getPointsCount(); i++) {
            FunctionPoint point = function.getPoint(i);
            System.out.printf("  [%d] (%.2f, %.2f)%n", i, point.getX(), point.getY());
        }

        System.out.println("Значения функции:");
        double left = function.getLeftDomainBorder();
        double right = function.getRightDomainBorder();
        for (double x = left; x <= right; x += (right - left) / 4) {
            double y = function.getFunctionValue(x);
            System.out.printf("  f(%.2f) = %.2f%n", x, y);
        }

        testFunctionExceptions(function);
    }

    private static void testConstructorExceptions() {
        System.out.println("Тестирование исключений в конструкторах:");

        try {
            new ArrayTabulatedFunction(10, 0, 5);
            System.out.println("ОШИБКА: Ожидалось IllegalArgumentException");
        } catch (IllegalArgumentException e) {
            System.out.println("IllegalArgumentException: " + e.getMessage());
        }

        try {
            new LinkedListTabulatedFunction(5, 5, 5);
            System.out.println("ОШИБКА: Ожидалось IllegalArgumentException");
        } catch (IllegalArgumentException e) {
            System.out.println("IllegalArgumentException: " + e.getMessage());
        }

        try {
            new ArrayTabulatedFunction(0, 10, 1);
            System.out.println("ОШИБКА: Ожидалось IllegalArgumentException");
        } catch (IllegalArgumentException e) {
            System.out.println("IllegalArgumentException: " + e.getMessage());
        }

        try {
            new LinkedListTabulatedFunction(0, 10, new double[]{1});
            System.out.println("ОШИБКА: Ожидалось IllegalArgumentException");
        } catch (IllegalArgumentException e) {
            System.out.println("IllegalArgumentException: " + e.getMessage());
        }

        try {
            TabulatedFunction func1 = new ArrayTabulatedFunction(0, 10, 3);
            System.out.println("ArrayTabulatedFunction создан успешно");

            TabulatedFunction func2 = new LinkedListTabulatedFunction(0, 10, new
```

```
double[]{1, 2, 3});
            System.out.println("LinkedListTabulatedFunction создан успешно");

        } catch (Exception e) {
            System.out.println("ОШИБКА: " + e.getMessage());
        }
    }

    private static void testPointOperations() {
        System.out.println("Тестирование операций с точками:");

        TabulatedFunction function = new ArrayTabulatedFunction(0, 4, 3);

        System.out.println("Исходная функция:");
        printFunctionPoints(function);

        try {
            System.out.println("\nДобавление новой точки:");
            function.addPoint(new FunctionPoint(2.5, 10.0));
            printFunctionPoints(function);

            System.out.println("\nДобавление новой точки:");
            function.addPoint(new FunctionPoint(1.5, 5.0));
            printFunctionPoints(function);

        } catch (InappropriateFunctionPointException e) {
            System.out.println("Ошибка при добавлении: " + e.getMessage());
        }

        try {
            System.out.println("\nУдаление точки:");
            function.deletePoint(2);
            printFunctionPoints(function);

        } catch (Exception e) {
            System.out.println("Ошибка при удалении: " + e.getMessage());
        }

        try {
            System.out.println("\nИзменение точки:");
            function.setPoint(1, new FunctionPoint(1.8, 8.0));
            printFunctionPoints(function);

        } catch (InappropriateFunctionPointException e) {
            System.out.println("Ошибка при изменении: " + e.getMessage());
        }
    }

    private static void printFunctionPoints(TabulatedFunction function) {
        for (int i = 0; i < function.getPointsCount(); i++) {
            FunctionPoint point = function.getPoint(i);
            System.out.printf("  [%d] (%.2f, %.2f)%n", i, point.getX(), point.getY());
        }
    }

    public static void main(String[] args) {
        System.out.println("||| Тестирование TabulatedFunction |||\n");
```

```
        testTabulatedFunction("ArrayTabulatedFunction",
            new ArrayTabulatedFunction(0, 10, 5));

        System.out.println("\n" + "=".repeat(50) + "\n");

        testTabulatedFunction("LinkedListTabulatedFunction",
            new LinkedListTabulatedFunction(0, 10, new double[]{1, 2, 3, 4, 5}));

        System.out.println("\n" + "=".repeat(50) + "\n");

        testConstructorExceptions();

        System.out.println("\n" + "=".repeat(50) + "\n");

        testPointOperations();
    }
```