

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САМАРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
ИМЕНИ АКАДЕМИКА С.П. КОРОЛЕВА»

Лабораторная работа №2
Разработка классов для работы с табличными функциями одной
переменной

Выполнила:

Иванова Елизавета Вадимовна,
студентка группы 6301-030301D

Самара 2025

Задание 1

Создан пакет `functions` в корне проекта. Все последующие классы (`FunctionPoint`, `TabulatedFunction`) размещены внутри этого пакета, что обеспечивает логическую группировку и соответствие требованиям задания.

Задание 2

Реализован класс `FunctionPoint`, инкапсулирующий координаты точки (x , y). Реализованы три конструктора:

- `FunctionPoint(double x, double y)` — инициализация заданными значениями,
- `FunctionPoint(FunctionPoint point)` — копирующий конструктор,
- `FunctionPoint()` — конструктор по умолчанию (точка в начале координат).

Поля x и y объявлены как `private`, доступ к ним осуществляется через геттеры и сеттеры, что гарантирует соблюдение принципа инкапсуляции.

Задание 3

Создан класс `TabulatedFunction`, хранящий массив объектов `FunctionPoint`.

Реализованы два конструктора:

- `TabulatedFunction(double leftX, double rightX, int pointsCount)` — создаёт равномерную сетку по оси X , значения $Y = 0$.
- `TabulatedFunction(double leftX, double rightX, double[] values)` — аналогично, но использует переданный массив значений Y .

В обоих случаях точки автоматически упорядочены по возрастанию X , так как генерируются с фиксированным шагом.

Задание 4

Реализованы методы:

- `getLeftDomainBorder()` и `getRightDomainBorder()` — возвращают X первой и последней точки.
- `getFunctionValue(double x)` — выполняет линейную интерполяцию между соседними точками. Если x вне диапазона, возвращает `Double.NaN`.

Для интерполяции используется формула прямой через две точки:

$$y = y_1 + \frac{(y_2 - y_1)(x - x_1)}{x_2 - x_1}$$

Задание 5

Добавлены методы для безопасной работы с отдельными точками:

- `getPoint(int index)` возвращает копию точки (инкапсуляция соблюдена)
- `setPoint(int index, FunctionPoint point)` заменяет точку только если её X лежит между X соседей (или на границе)

- аналогичные проверки реализованы в `setPointX()`
- `getPointX()`, `getPointY()`, `setPointY()` работают без ограничений на `Y`

Методы `setPoint` и `setPointX` включают проверку корректности нового значения `X`: при попытке нарушить упорядоченность по `X` (например, задать `X` вне интервала соседних точек) замена не выполняется. Особый случай — единственная точка в функции: в этом случае её `X` может быть изменён без ограничений.

Задание 6

Реализованы методы динамического изменения количества точек:

- `deletePoint(int index)` — удаляет точку, сдвигая оставшиеся элементы (использован `System.arraycopy`),
- `addPoint(FunctionPoint point)` — вставляет новую точку в нужную позицию, сохраняя упорядоченность по `X`.

Внутренний массив имеет запас длины (ёмкость > количества точек), и расширяется только при необходимости (например, при добавлении в полный массив). Это повышает эффективность.

Задание 7

Для проверки корректности реализации классов `FunctionPoint` и `TabulatedFunction` был создан класс `Main` вне пакета `functions`. В методе `main()` выполнены следующие действия:

1) Создание табличной функции: инициализирован объект `TabulatedFunction` с областью определения `[2;9]` и массивом значений: `double[] values = { 1, 3, 5, 2, 6, 7, 26, 8 };`

Это задаёт 8 точек с равномерным шагом по оси `x` :

`(2,1),(3,3),(4,5),(5,2),(6,6),(7,7),(8,26),(9,8)` .

Сразу после создания вызван метод `printTabFun()`, подтвердивший, что первая точка имеет `x=2`, а последняя — `x=9`, как и ожидалось.

2) Тестирование до изменений: вызван метод `getFunctionValue()` для следующих аргументов:

- `x=3.5` — точка внутри интервала `[3;4]` , результат получается линейной интерполяцией между `(3,3)` и `(4,5)` ;
- `x=1` и `x=10` — точки вне области определения, метод корректно возвращает `Double.NaN`.

3) Модификация функции:

- Удалена точка с индексом 1 (то есть `(3,3)`) с помощью `deletePoint(1)`;
- Добавлена новая точка `(0,-1)` с помощью `addPoint()`, что расширило область определения до `[0;9]` .

4) Тестирование после изменений:

Повторный вызов `getFunctionValue()` показал:

- $f(1)=0.0$ — теперь значение внутри новой области $[0;2]$, вычислено интерполяцией между $(0,-1)$ и $(2,1)$;
- $f(3)=3.0$ — интерполяция между $(2,1)$ и $(4,5)$ (после удаления точки $x=3$);
- $f(-1)=\text{NaN}$ — по-прежнему вне области (левая граница теперь $x=0$).

5) Визуальная проверка: вызван метод `printTabFun()`, который вывел итоговый набор точек, подтвердив корректность операций добавления и удаления, а также сохранение упорядоченности по x .

ТОЧКИ ДО ИЗМЕНЕНИЙ

№1 x:2.0 y:1.0

№2 x:3.0 y:3.0

№3 x:4.0 y:5.0

№4 x:5.0 y:2.0

№5 x:6.0 y:6.0

№6 x:7.0 y:7.0

№7 x:8.0 y:26.0

№8 x:9.0 y:8.0

ДО ИЗМЕНЕНИЙ

$f(3.5) = 4.0$

$f(1) = \text{NaN}$

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

ПОСЛЕ ИЗМЕНЕНИЙ

$f(1) = 0.0$

$f(3) = 3.0$

$f(-1) = \text{NaN}$

Точки:

№1 x:0.0 y:-1.0

№2 x:2.0 y:1.0

№3 x:4.0 y:5.0

№4 x:5.0 y:2.0

№5 x:6.0 y:6.0

№6 x:7.0 y:7.0

№7 x:8.0 y:26.0

№8 x:9.0 y:8.0

Process finished with exit code 0