Міністерство освіти і науки України

Харківський національний університет імені В.Н. Каразіна

Факультет комп’ютерних наук

# Розрахунково-графічна робота

з дисципліни «Крос-платформне програмування»

На тему «Згадати все… (Основні принципи ООП)»

Виконав студент 2 курсу групи КС-21

Безрук Юрій Русланович

Перевірив: доц. Споров О.Є. ...............

Харків – 2020

ЗМІСТ

[Розрахунково-графічна робота 1](#_Toc41557906)

[ВСТУП 3](#_Toc41557907)

[Стислий глосарій 3](#_Toc41557908)

[РОЗДІЛ 1 РОЗРОБКА ТА ТЕСТУВАННЯ ОСНОВНОЇ СИСТЕМИ КЛАСІВ 5](#_Toc41557909)

[1.1 Постановка задачі 5](#_Toc41557910)

[1.2 Початок реалізації. Перша функція. 6](#_Toc41557911)

[1.3 Реалізація табличної функції. Інтерполяція 7](#_Toc41557912)

[1.4 Реалізація умовно заданої функції 12](#_Toc41557913)

[1.5 Тестування і налагодження системи 17](#_Toc41557914)

[РОЗДІЛ 2 РОЗШИРЕННЯ СИСТЕМИ 22](#_Toc41557915)

[2.1 Постановка задачі 22](#_Toc41557917)

[2.2 Реалізація табличної функції за допомогою TreeSet 22](#_Toc41557918)

[2.3 Реалізація табличної функції за допомогою TreeMap 24](#_Toc41557919)

[2.4 Реалізація функції, заданої рядком 25](#_Toc41557920)

[2.5 Графічний інтерфейс 31](#_Toc41557921)

[ВИСНОВКИ 33](#_Toc41557922)

[ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ 34](#_Toc41557923)

# ВСТУП

В ході виконання розрахунково-графічної роботи необхідно згадати основні відомості, вивчені в рамках курсу «Об’єктно-орієнтоване програмування»: основи об’єктно-орієнтованого підходу, особливості застосування абстрактних класів та інтерфейсів, основи роботи з колекціями, основи роботи з файлами (операції читання / запису), а також ознайомитися з основами численних методів та їх реалізацією на основі інтерфейсів.

Отже, необхідно написати програму, моделюючу поняття функції та різні способи її визначення та продумати процес взаємодії програми з користувачем.

На практиці це завдання реалізується як набір класів для спорідненого представлення функції однієї змінної, що задана різними способами: аналітичним (формула, що визначає функцію, вказана у тексті програми), табличним (*інтерполяція* за збереженими значеннями аргументу і функції) та словесним (система класів, пов’язана умовою усно). Після чого необхідно чисельно продиференціювати з вказаною точністю задані таким різним чином функції однієї змінної.

Після тестування і налагодження системи, її необхідно розширити, не «зламавши», додавши ще декілька інтерпретацій функції однієї змінної згідно завдання та створивши графічний інтерфейс взаємодії з користувачем.

# Стислий глосарій

*Інтерполяція –* це знаходження проміжних значень функції на підставі деяких відомих її значень [1].

*Трансцендентне рівняння –* рівняння, яке містить [аналітичну функці](https://uk.wikipedia.org/wiki/%D0%90%D0%BD%D0%B0%D0%BB%D1%96%D1%82%D0%B8%D1%87%D0%BD%D0%B0_%D1%84%D1%83%D0%BD%D0%BA%D1%86%D1%96%D1%8F)ю, що не є [алгебраїчною](https://uk.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%B5%D0%B1%D1%80%D0%B0%D1%97%D1%87%D0%BD%D0%B0_%D1%84%D1%83%D0%BD%D0%BA%D1%86%D1%96%D1%8F) [2].

*Апроксимація –* це наближене вираження одних математичних об'єктів іншими, простішими, наприклад, кривих ліній — ламаними, ірраціональних чисел — раціональними, неперервних функцій — многочленами [3].

*Парсер –* це програма або частина програми (об’єкт, функція, тощо), яка виділяє певні частини інформації з масиву даних для подальшого використання [4].

# **РОЗДІЛ 1 РОЗРОБКА ТА ТЕСТУВАННЯ ОСНОВНОЇ СИСТЕМИ КЛАСІВ**

## Постановка задачі

Згідно завдання, необхідно створити наступні декілька функцій и протестувати їх на зазначених діапазонах значень.

Перша функція повинна бути задана як:

*,*

(1.1)

де: – аргумент функції ;

– експоненційна функція.

Друга функція додатково використовує у своєму визначенні параметр:

*,*

(1.2)

де: – параметр заданої величини.

Згідно умові задачі, тестування слід провести при значеннях параметру *a* (*a*=0.5, *a*=1.0). Слід розуміти, що функція 1.2 при значенні параметру *а=*1 приймає вид функції 1.1, тому зрозуміло, що для їх реалізації достатньо одного класу.

Третя функція задається таблично – значення незалежної змінної та функції (згідно завданню, sin (*x*)), збережені в текстовому файлі даних. Таблиця значень функцій повинна зберігатися в відповідному класі у вигляді змінної типу ArrayList.

Четверта функція задається усною умовою: значення незалежної змінної – це значення параметру *а*, що належить відрізку *а* [1.0, 7.0] і змінюється з

кроком 0.1; значення функції – розв’язок рівняння

*,*

(1.3)

для заданого значення *а.*

Функції, які необхідно продиференціювати, визначені різним шляхом: за допомогою формули, таблиці значень тощо. У них є єдина спільна властивість: вказуючи значення незалежної змінної *х* можна отримати значення функції . Таким чином, для того, щоб можна було диференціювати будь-які функції однієї змінної зручно оголосити інтерфейс, що визначає можливість розрахувати результат, отримавши один аргумент. Усе, що буде реалізувати заданий інтерфейс можна диференціювати, не зважаючи на спосіб реалізації.

## Початок реалізації. Перша функція.

Усі класи початкової версії будемо розміщувати у пакеті consoleTasks. Для початку потрібно створити інтерфейс Evaluatable, який буде гарантувати, що об’єкт класу, який реалізує цей інтерфейс, може обчислювати значення по одному аргументу. Додаємо до нього метод обчислення значення функції, який потрібно буде перевизначити в усіх класах, що його реалізують. Остаточно інтерфейс матиме вигляд:

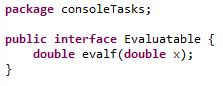


Рисунок 1. - інтерфейс Evaluatable

Тобто всі класи, що реалізують даний інтерфейс, будуть визначати, яким чином метод evalf(double x) вираховує значення функції.

Далі створюємо клас, що реалізує даний інтерфейс та призначений для обчислення перших двох функцій завдання, FFunction. Резервуємо приватне поле для параметру *а*, та створюємо два конструктори – для функції з заданими параметром (друга функція завдання) і без нього (конструктор за замовчуванням) – у цьому разі параметр дорівнює одиниці. Додаємо також відповідні геттер і сеттер для параметру і, відповідно, перевизначаємо метод обчислення. У результаті маємо такий клас:

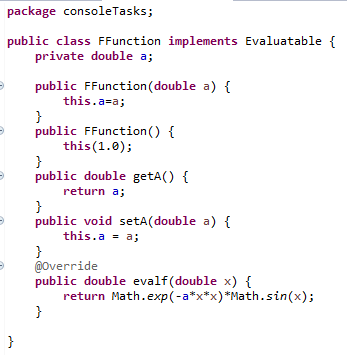


Рисунок 1.2 - клас FFunction

Таким чином створено клас, що представляє собою перші дві функції згідно завдання.

## Реалізація табличної функції. Інтерполяція

Далі необхідно створити набір класів, призначений для створення функції, заданої у вигляді набору точок, збережених у файлі. Для цього необхідні класи для зберігання пар значень () та для обчислення значення функції у проміжках між заданими точками, тобто для виконання інтерполяції.

Для представлення точок () створимо спочатку абстрактний клас Point, який буде містити у собі базові можливості по представленню точки у *n*-вимірному просторі. Він буде вміщувати в собі поле-масив координат *n*-вимірної точки, один конструктор і два методи доступу, а також метод toString(), для відображення рядкового представлення точки у вигляді (*х*, *у*). Остаточно клас матиме вигляд:

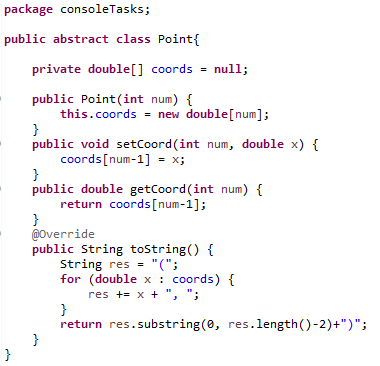


Рисунок 1. - абстрактний клас Point

Для комфортної роботи у двовимірному просторі створюємо спадкоємця класу Point – клас Point2D. Додаємо йому також інструкцію про те, що він реалізує інтерфейс Comparable, для можливості сортування об’єктів класу – точок. Далі додаємо у створений клас два конструктори (за замовчуванням, створює точку (0; 0), і за вказаною парою координат), зручні методи доступу і реалізацію методу compareTo для порівняння точок. Дві точки порівнюємо за значеннями *х* (розташування точок вздовж абсцис на координатній площині). У результаті маємо клас:

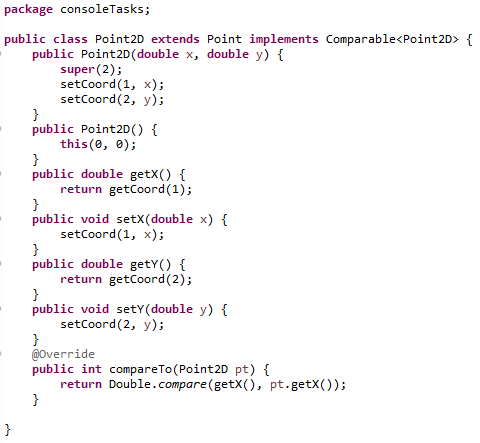


Рисунок 1. - клас Point2D

Далі, з використанням класів – точок потрібно написати класи, призначені для виконання інтерполяції і обчислення таблично заданої функції для заданого аргументу. Для виконання інтерполяції, тобто отримання значення функції у проміжку між її відомими значеннями будемо використовувати інтерполяційний поліном Лагранжа, що проходить через N+1 точку (), (), … , (), ступені не більшої, ніж N, та записується таким чином:

(1.4)

де – коефіцієнти поліному Лагранжа, основаного на цих вузлах;

N – ступінь поліному.

Вирази для коефіцієнтів можуть бути записані у вигляді:

(1.5)

Для кожного фіксованого *k* коефіцієнти поліному Лагранжа мають наступні властивості:

(1.6)

Створюємо абстрактний клас Interpolator, що реалізує інтерфейс Evaluatable, який буде оголошувати основний абстрактний функціонал для третьої функції і перевизначати метод evalf() для підрахування значення функції відносно аргументу. В методі виконуємо інтерполяцію за допомогою формул (1.4) – (1.6), при цьому використовуємо абстракцію у зверненнях.

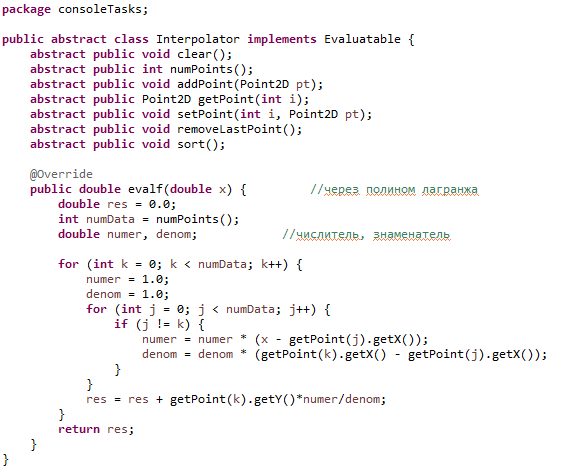


Рисунок 1. - абстрактний клас Interpolator

Таким чином, Interpolator абстрактно реалізує інтерполюючу функцію, без конкретного представлення структури збереження точок. Наслідуючи цей клас, ми можемо реалізувати збереження у будь-якому форматі.

Для такої реалізації створимо наслідника класу Interpolator – клас ListInterpolation, який буде зберігати точки, по яким будується інтерполяційний поліном у переліковому масиві. У класі описуємо приватне поле-перелік точок на площині, три конструктори (за замовчуванням, з переліку точок і з масиву точок) і реалізуємо абстрактні методи суперкласу.

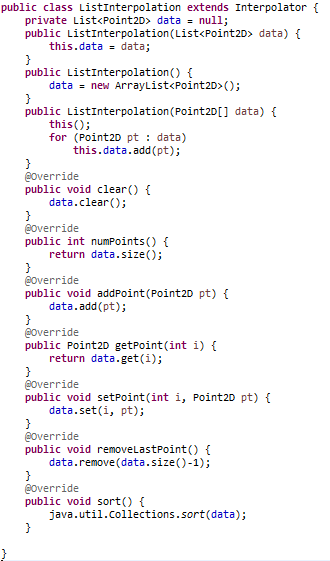


Рисунок 1. - клас ListInterpolation

Даний клас вже являє собою реалізацію функції, заданої таблично. Але для виконання умови завдання необхідно брати значення з файлу. Тому створюємо наслідника цього класу – FileListInterpolation, який додає функціонал читання та запису в файл до функції.

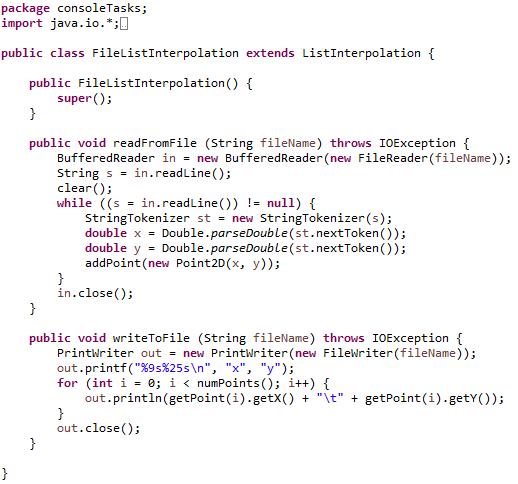


Рисунок 1. – FileListInterpolation

Таким чином, третя функція завдання повністю визначена і готова до роботи.

## Реалізація умовно заданої функції

Усна умова для функції задається таким чином: значення функції – це корені рівняння (1.3), при значеннях параметру рівному незалежній змінній. Отже, для опису цієї функції необхідно додатково створити клас, інкапсулюючий у собі поведінку заданого *трансцендентного рівняння*. Назвемо його LeftHand, і опишем основний функціонал. Він також буде реалізувати інтерфейс Evaluatable, тобто являтиме собою таку ж функцію, що й попередні класи. За реалізацією він схожий на другу функцію, відрізняється лише формулою для обчислення значення функції від відомого аргументу.

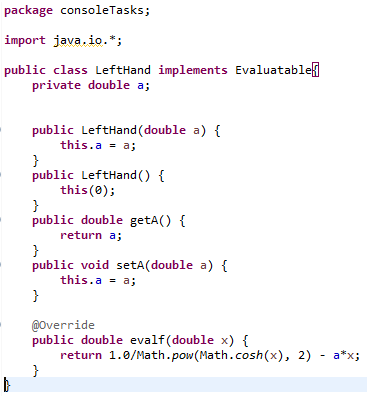


Рисунок 1.8 - клас LeftHand

Далі потрібно описати процес знаходження кореню до заданої функції. Пошук таких коренів виконується у два етапи. На першому етапі виконується локалізація кореню, тобто визначається такий відрізок [a,b], на якому досліджувана функція має рівно один корінь. Далі виконуємо уточнення знаходження кореню за допомогою методу січних. Цей метод є модифікацією методу Ньютона, у якому похідна замінена кінцево-різничним наближенням. У даному методі нове наближення до кореню обчислюється за формулою:

*,*

(1.7)

де: – нове наближення до кореню;

, – попередні наближення

При обчисленні кореню рівняння будемо вважати, що корінь знайдено з вказаною точністю, якщо виконується нерівність

*,*

(1.7)

де: – два послідовнихнаближення до кореню;

– задана точність розрахунків.

Для реалізації математичних методів, зокрема знаходження кореню створимо клас NumMethods. Статичний метод findRoot буде приймати в себе число для *апроксимації* кореню, точність обчислень та функцію, для якої необхідно знайти корінь. У самому ж методі наведемо реалізацію описаного вище методу. Для того, щоб неможливо було створити об’єктів цього класу, створимо йому приватний конструктор.

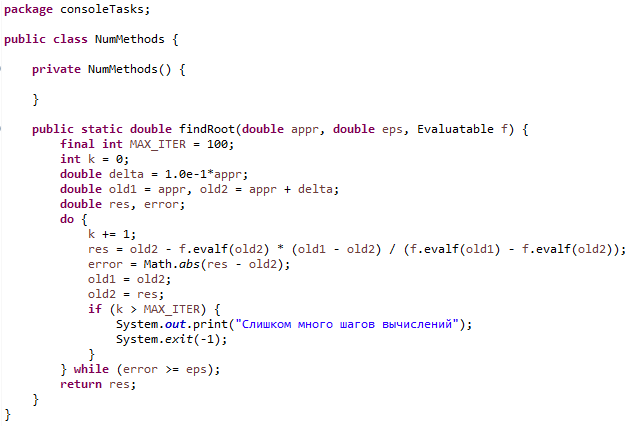


Рисунок 1. - клас NumMethods і метод findRoot

Для зручності, у цьому ж класі можна описати і обчислення похідної функції в точці. Для цього можна скористатися центральною кінцево-різничною формулою чисельного диференціювання за трьома точками:

(1.8)

де: – точка, у якій потрібно знайти похідну;

=;

*h* – крок, з яким обчислюється похідна (точність).

Для того, щоб оцінити те значення *h*, для якого похідна обчислена с заданою точністю , потрібно обчислити послідовність з значеннями кроку , що постійно зменшуються. Обчислення послідовності слід виконувати до тих пір, поки не буде виконана хоча б одна з нерівностей:

(1.9)

Реалізуємо даний алгоритм у класі NumMethods. Публічний статичний метод dif() буде обчислювати похідну в точці *x* з заданою точністю у функції викликаючи на кожному кроці обчислень приватний метод difWithH() для постійно зменшуваного значення *h*.

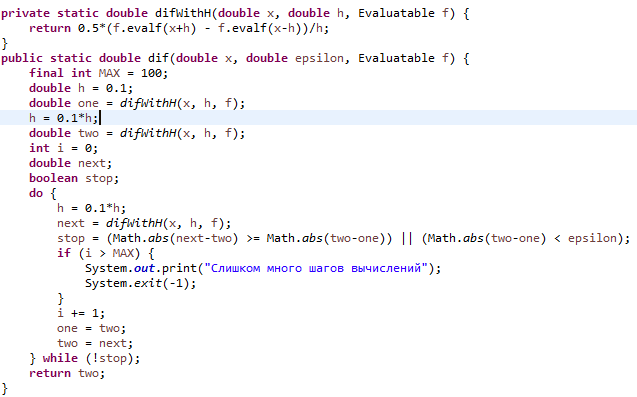


Рисунок 1. - методи dif та difWithH класу NumMethods

Тепер, коли маємо клас-рівняння і математичний метод його рішення, можемо створити клас, що описує четверту функцію завдання. Назвемо його SolveEqFunction. У собі він буде містити приватні поля – класу-рівняння (LeftHand), точності та апроксимованого кореню. Клас має реалізовувати інтерфейс Evaluatable, і у методі evalf() встановлювати аргумент як параметр у рівняння, знаходити і повертати його корінь як результат функції. Також додамо метод перевірки якості знайденого кореню, обчислюючий функцію класу LeftHand, методи-сеттери і конструктор за замовчуванням.



Рисунок 1. - клас SolveEqFunction

Тепер усі функції за умовою завдання створено.

## Тестування і налагодження системи

Для початку слід перевірити окремо роботу функцій і методів. У класах, що перевіряються, створюємо відповідно методи main(), у яких і тестуємо роботу класу. Зокрема:

У класі FFunction перевіряємо обчислення функції при різних значеннях незалежної змінної і параметру.

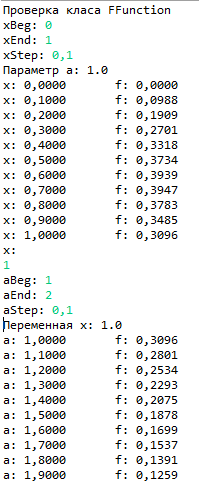


Рисунок 1. - тестування класу FFunction

У класі Point2D перевіряємо правильність вказання сортування точок (тобто правильність реалізації методу compareTo).

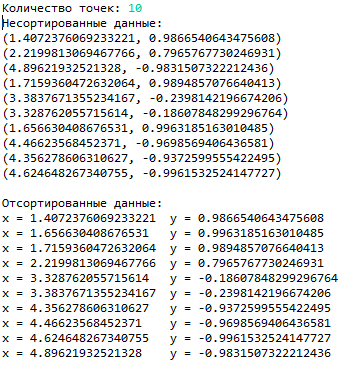


Рисунок 1. - тестування класу Point2D

У класі ListInterpolation тестуємо значення інтерполяції за заданою кількістю точок та перевіряємо абсолютну похибку.

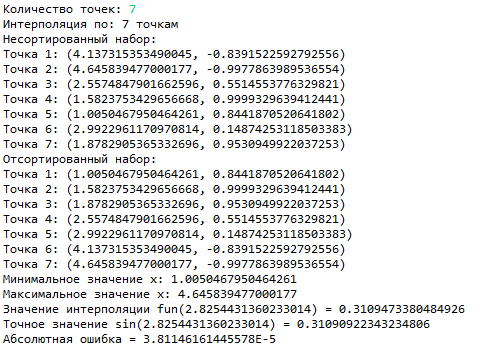


Рисунок 1. - тестування класу ListInterpolation

У класі FileListInterpolation тестуємо запис у файл, і підготовлюємо таблицю значень для третьої функції у файлі даних TblFunc.dat.

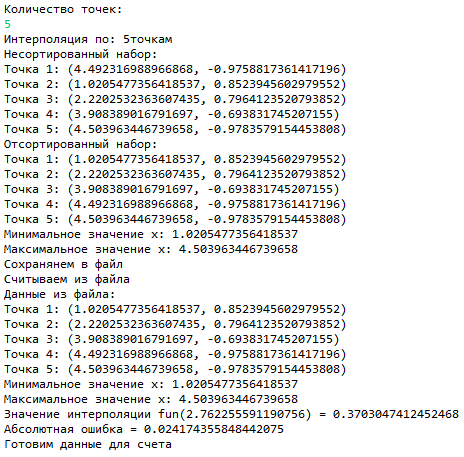


Рисунок 1. - тестування класу FileListInterpolation

У класі LeftHand за аналогією до FFunction тестуємо значення функції при значеннях при різних параметру *а*.

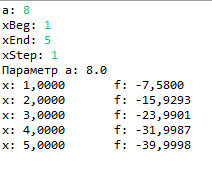


Рисунок 1. - тестування класу LeftHand

Для класу NumMethods перевіряємо процес знаходження коренів рівняння, а також абсолютну похибку при обчисленні похідної від функції в точці.

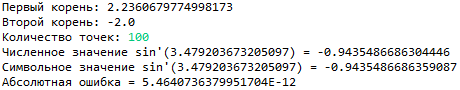


Рисунок 1. - тестування класу NumMethods

Для класу ж SolveEqFunction просто тестуємо роботу функції – знаходження кореню при значенні параметру рівним незалежній змінній і приблизній апроксимації попереднього кореню.

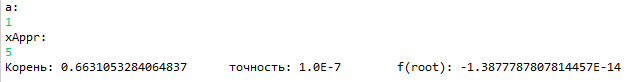


Рисунок 1.18 – тестування класу SolveEqFunction

Таким чином, основні класи системи створено й протестовано. Для тестування системи і порівняння значень функцій створимо клас DerivativeApplication. Там, у методі main() створюємо 4 функції згідно завданню. Для табличної функції завантажуємо дані з підготовленого файлу, для умовної – встановлюємо значення апроксимованого кореню. Далі виконуємо обчислення для кожної з функцій та її похідної у заданому діапазоні, виводимо результати на екран і зберігаємо у відповідних файлах даних.



Рисунок 1. – клас DerivativeApplication

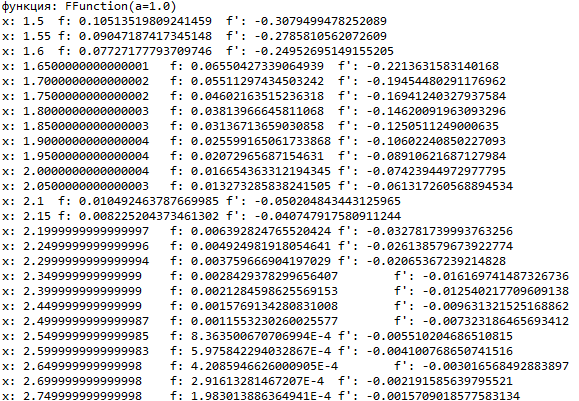


Рисунок 1. - фрагмент результатів виконання

# РОЗДІЛ 2 РОЗШИРЕННЯ СИСТЕМИ



## Постановка задачі

Тепер, згідно умови задачі, слід розширити систему, виконавши наступні умови.

Перша: дані для функції, заданої таблично зберігаються у структурі типу TreeSet.

Друга умова схожа: дані для функції, заданої таблично зберігаються у структурі типу TreeMap.

Третя: аналітично задана функція може бути вказана у вигляді рядку в процесі роботи програми. Похідна у цьому разі повинна бути обчислена не тільки чисельно, а й у символьний (аналітичний) спосіб. Також слід передбачити можливість задання функції без параметрів і з одним параметром.

Четверта умова: створити додаток с графічним інтерфейсом користувача для відображення аналітично заданої функції та її похідної у вказаних границях.

Класи розширеної системи зберігатимемо у пакеті expantion.

## Реалізація табличної функції за допомогою TreeSet

Для того, щоб реалізувати табличну функцію за допомогою структури «множина», достатньо створити ще одного наслідника класу Interpolator і реалізувати його методи так, як це можна зробити з множиною. Отже клас TreeSetInterpolation матиме у собі приватне поле типу Set, три конструктори (за аналогією до ListInterpolation) і реалізовані методи абстрактного класу Interpolator. Методи clear(), numPoints() і addPoint() працюють так само, а ось реалізація інших методів буде відрізнятися. Зокрема, методи getPoint і removeLastPoint використовують інформацію про позицію елементу (перший повертає елемент за вказаною позицією, а другий – видаляє елемент на останній позиції), а у множині не зберігається інформація про позицію елементів, що входять до неї. Тому у цих методах спочатку виконується перетворення множини на масив, а вже потім дія з елементом за індексом. А ось методи setPoint() і sort() неможливо реалізувати вже через особливості розміщення елементів саме у TreeSet. Оскільки множина побудована на основі дерева, всі її елементи сортуються при додаванні будь-якого нового елементу. Тому метод setPoint() буде лише видаляти елемент за вказаним індексом, а новий елемент буде просто додавати у дерево. Його позиція залежатиме від його значення. Метод sort() взагалі безкорисний, оскільки TreeSet завжди відсортоване. Також слід розуміти, що оскільки точки зберігаються у структурі Set, вони не можуть повторюватися. Остаточно клас матиме вигляд:



Рисунок 2.1 – клас TreeSetInterpolation

Як і для випадку з переліком, створюємо для цього класу наслідник для роботи з файлами FileTreeSetInterpolation, що за функціоналом нічим не відрізняється від FileListInterpolation, крім того, що використовує для зберігання структуру типу TreeSet.

## Реалізація табличної функції за допомогою TreeMap

Асоціативний масив також створює деякі незручності при реалізації. Створюємо клас TreeMapInterpolation, додаємо йому приватне поле типу Map, три конструктори і реалізації методів суперкласу – Interpolator. Оскільки Map призначений для збереження пар ключ-значення, зручно буде зберігати у ньому не точки як суцільний об’єкт, а їх координати, де *x* – ключ, а *у* – відповідне йому значення. Тоді метод addPoint() буде виконувати «розпаковку» об’єкту-точки у ключ і значення для додавання у колекцію. Методи clear() і numPoints() не змінюються. Для того, щоб повернути значення точки за індексом, необхідно спочатку сформувати множину ключів, з неї – масив ключів, у якому знайти за індексом необхідний ключ, і далі сформувати з нього та відповідного йому значення *y* об’єкт класу Point2D. Методи setPoint() і sort() як і раніше не вдається реалізувати через особливості розташування елементів у дереві. Також можна додати метод для отримання координати *х* за індексом і координати *у* за *х*-вою. Клас має вигляд:

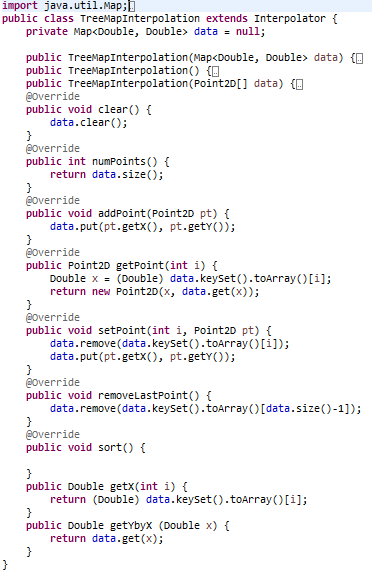


Рисунок 2.2 – клас TreeMapInterpolation

До нього відповідним чином створимо наслідника для роботи з файлами – FileTreeMapInterpolation.

## Реалізація функції, заданої рядком

Для опису такої функції скористаємося бібліотекою Java Components for Mathematics, яка може представляти математичні вирази та змінні у вигляді рядків. Тестування її можливостей проведемо окремо у пакеті test і класі JcmTest. Далі вже у пакеті expantion створюємо клас AnalyticFunction, який реалізує інтерфейс Evaluatable. Клас буде мати 4 приватних поля: вираз-функцію, *парсер* для його обробки, змінну та параметр, та 3 конструктори: за замовчуванням, за функцією та її змінною, та за функцією, її змінною та параметром. Процес створення аналітичної функції виконується таким чином: спочатку створюється об’єкт-парсер. Далі - об’єкти-змінна, і, за необхідності, параметр. Змінна та параметр додаються у парсер, після чого у нього викликається метод parse() з параметром – рядком функції, який повертає об’єкт-вираз функції.

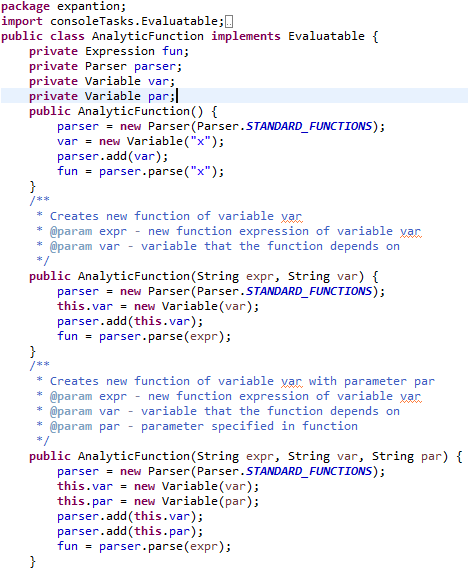


Рисунок 2.3 – поля та конструктори класу AnalyticFunction

Далі починаємо додавати методи. Створимо перевантажені методи setFunction() з різною кількістю аргументів, що будуть змінювати функцію за аналогією до конструкторів.

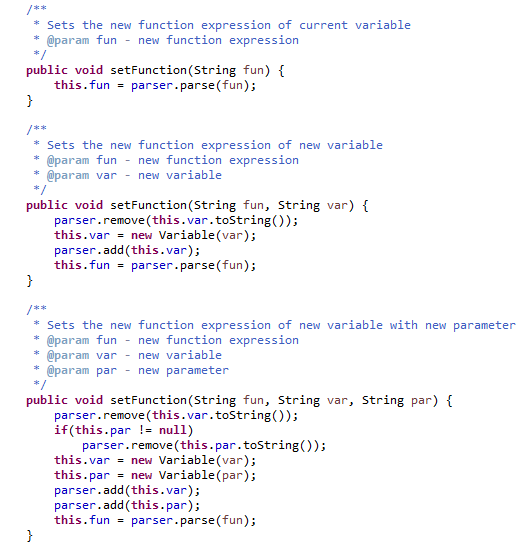


Рисунок 2.4 – перевантажені методи setFunction() у класі AnalyticFunction

Далі створюємо геттери і сеттери для змінної і параметру. При чому геттери двох форматів – повертають змінні і як рядок і як об’єкт.

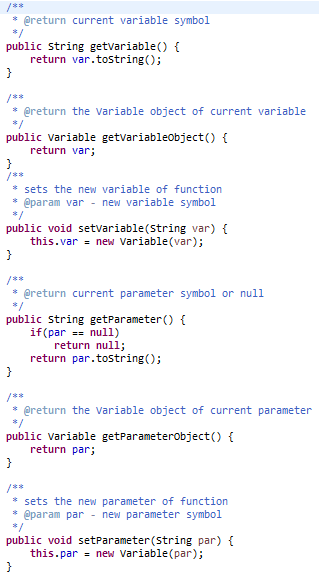


Рисунок 2.5 – методи доступу

Також додамо методи доступу до значення параметру, toString() та метод, що буде міняти місцями змінну і параметр. І ключовий метод, через який функція і реалізує свій основний функціонал – evalf().

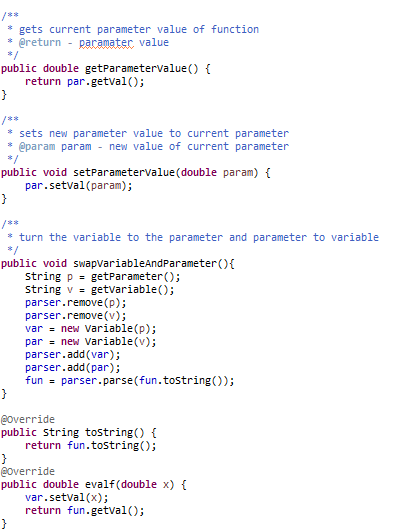


Рисунок 2.6 – додаткові методи

Таким чином, аналітично задана функція створена. Для тестування не вистачає тільки розширеного класу NumMethods, який би мав змогу символьно диференціювати функцію. Хоча відповідний метод і можна додати до самої функції, краще буде зробити це зовні, як і у першій версії програми. Створимо клас SymbolNumMethods, який матиме у собі методи аналогічні методу NumMethods (реалізовані через виклик методів саме початкових методів першої версії системи), а до них додамо ще два: метод символьного диференціювання функції та метод точного диференціювання функції в точці (шукає символьну похідну і повертає її значення в точці). Тому за допомогою цього класу похідну в точці можна буде обчислити як приблизно, з заданою точністю (для будь яких функцій, що реалізують Evaluatable), і точно (тільки для рядково заданої функції). Клас матиме вигляд:

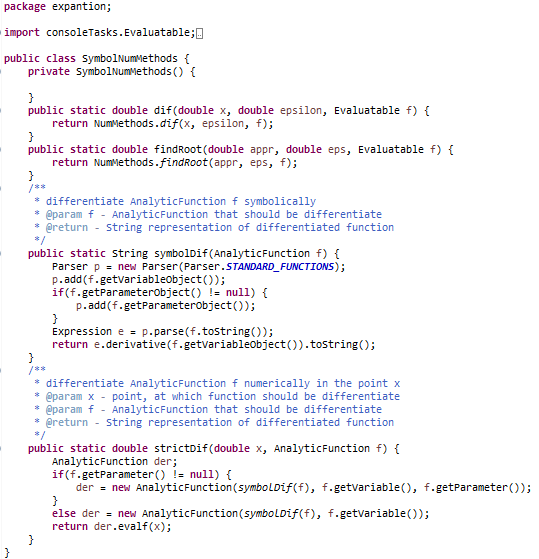


Рисунок 2.7 – клас SymbolNumMethods

Тепер все готово для тестування. Створімо клас TestAnalitycF з методом main(), де організуємо консольну взаємодію користувача з можливостями даної функції за допомогою меню з різними можливостями роботи з функціями. Фрагмент тестування:

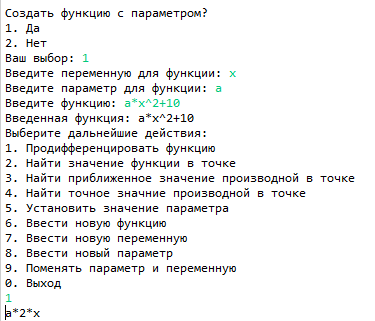


Рисунок 2.8 – фрагмент тестування можливостей аналітичної функції

## Графічний інтерфейс

Графічний інтерфейс користувача має представляти собою вікно з можливістю вводу функції, її змінної, параметру та його значення, а також кроком обчислень (точність графіку). У центрі ж вікна на системі координат повинен відображатися графік введеної функції та її похідної. Графік будемо будувати за допомогою бібліотеки JFreeChart. Її тестування наведемо у пакеті gui. А вже у головному пакеті expantion створимо клас FunctionFrame. У якому за допомогою можливостей вбудованих бібліотек awt, swing і підключеної JFreeChart створимо графічний інтерфейс. Додамо необхідні описані вище поля вводу значень, сам графік, а також кнопки «Plot» (побудова графіку) і «Exit» (вихід). Прив’яжемо до цих кнопок відповідні їм слухачі подій. Також, для зручності, напишемо вбудований клас-слухач подій клавіатури на кнопці Enter, і прив’яжемо його до всіх елементів взаємодії користувача, щоб при будь якому введенні на натисненні на цю клавішу графік перебудовувався и не потрібно було тягнутися до кнопки на фреймі. Створення та настройку графіку винесемо у окрему функцію createChart(). В кінцевому результаті отримаємо такий фрейм:

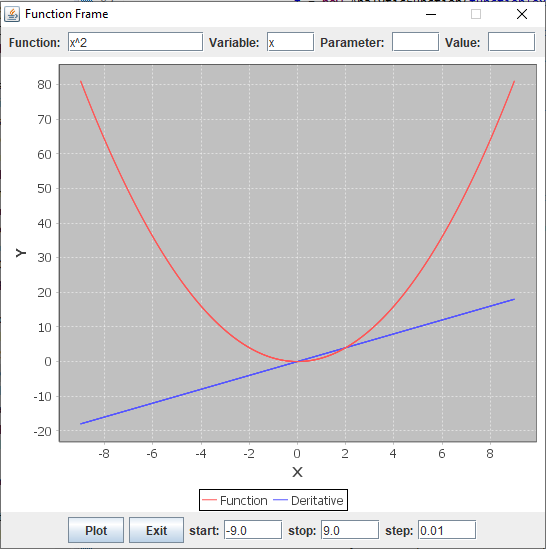


Рисунок 2.9 – графічний інтерфейс взаємодії з користувачем

Таким чином, усі пункти розширення системи були виконані.

# ВИСНОВКИ

Отже, в ході виконання розрахунково графічної роботи була створена система класів, що інкапсулює у собі математичні функції задані різноманітним чином: аналітично (в коді програми або ж під час виконання), таблично (у файлі та у програмі, де дані зберігалися у структурах типу «перелік на основі масиву» (ArrayList), «множина на основі дерева» (TreeSet) та «асоціативний масив на основі дерева» (TreeMap)) і за допомогою усного опису умови залежності значень функції від незалежної змінної.

У роботі використовувалися математичні методи та формули для інтерполяції функції, апроксимації кореню та обчисленню похідної у точці.

Під час виконання роботи були згадані та застосовані основні принципи і прийоми об’єктно-орієнтованого програмування, використаний широкий спектр можливостей об’єктно-орієнтованого підходу мови Java, зокрема наслідування, інкапсуляція, особливості застосування абстрактних класів та інтерфейсів, основи роботи з колекціями, основи роботи з файлами (операції читання / запису), консольна та графічна взаємодія з користувачем та ін.

Розроблена система повністю протестована і налагоджена і надається разом з пояснювальною запискою до роботи.

# ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Інтерполяція – Академічний тлумачний словник. // http://sum.in.ua/s/interpoljacija. Дата звернення: 26.05.2020.
2. Трансцендентне рівняння – Вікіпедія. // https://%20uk.wikipedia.org/ wiki/Трансцендентне\_рівняння. Дата звернення: 27.05.2020.
3. Апроксимація – Словотвір // https:// slovotvir.org.ua/words/ aproksymatsiia. Дата звернення: 27.05.2020.
4. Парсер – ActiveTraffic // https:// www.activetraffic.ru/wiki/parser. Дата звернення: 28.05.2020.