Graphing Calculator Application

MŰEGYETEM 1782

Faculty of Electrical Engineering and Informatics

Basics of Programming 3

Graphing Calculator Application

Specification

Student

Klevis Imeri T4XGKO

Budapest, October, 2023

Description of the task

Graphing calculators are essential tools in the everyday life of engineers, providing a deeper understanding of how functions behave. The aim of this project is to design a graphical calculator that can graph functions, much like the popular tool Desmos. The user will have the ability to input expressions in algebraic form and utilize general mathematical functions, such as sine (sin), cosine (cos), logarithms (log), and exponentials (exp), to explore a wide range of mathematical concepts.

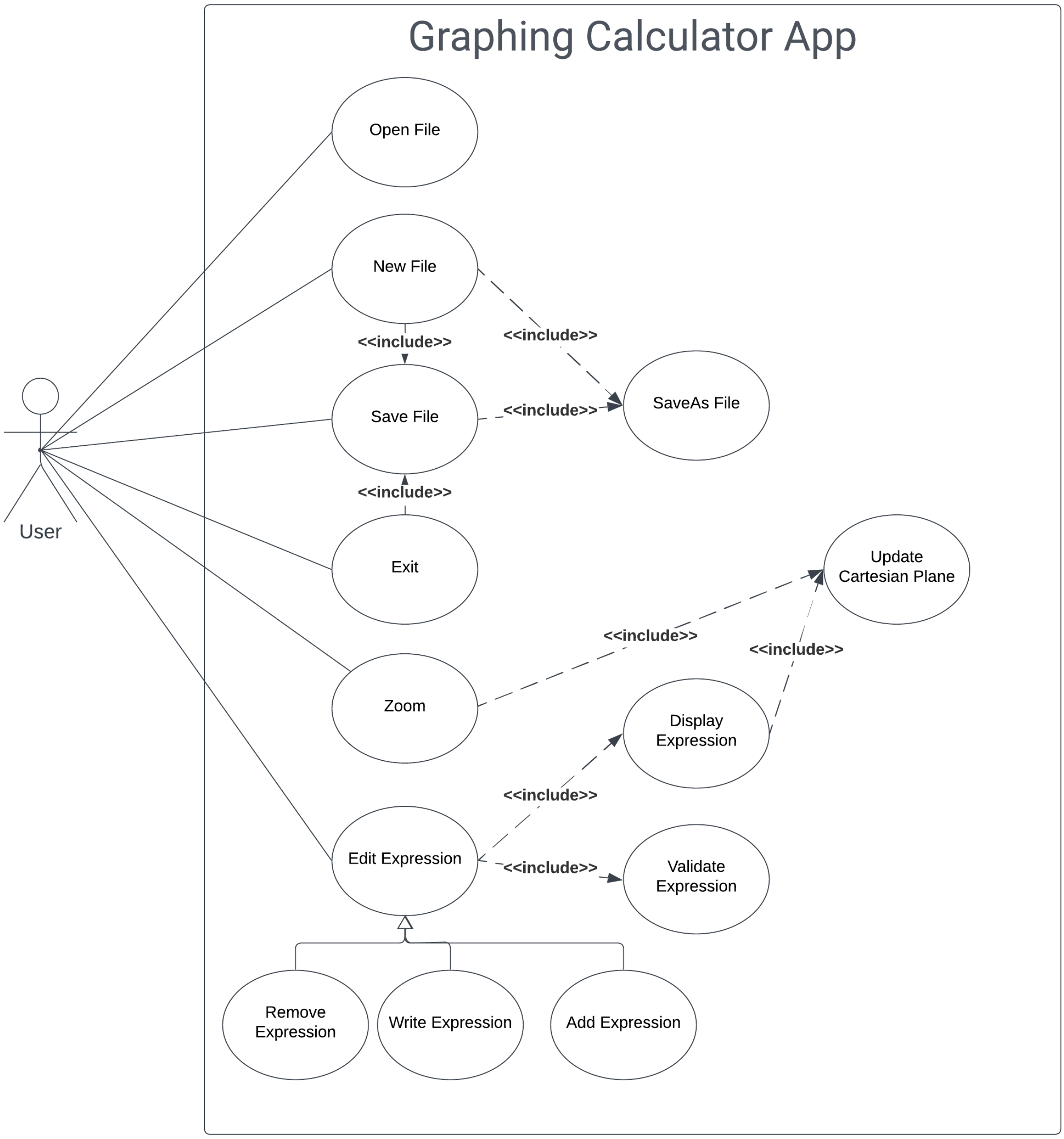
Furthermore, the program will facilitate project management, allowing users to open, edit, and save both new and existing projects. This feature ensures that users can organize their work efficiently and maintain a record of their mathematical explorations. Users will be able to create multiple expressions, edit them in real-time, and delete them as needed.

The application's main functionality is the real-time display of graphs on a Cartesian plane. This dynamic graphing capability enables users to instantly visualize the behavior of various functions. Additionally, users will have the flexibility to zoom in or out on the Cartesian plane, allowing them to explore specific regions of the graph in greater detail or view a broader overview of the function's behavior.

While the core remains focused on these capabilities, the program will be implemented in Java to ensure cross-platform compatibility, making it accessible to a wide range of users. The user interface will be designed with simplicity and usability in mind, featuring intuitive tools for entering, editing, and managing expressions.

Moreover, the program will incorporate effective error handling to provide informative error messages to users in case of invalid expressions.

Use-case Diagram



After the user opens the program, they will be met with a `menu`. There will be a `plane` where they can enter `expressions` and a blank `Cartesian plane`. From here, the user can perform the following actions:

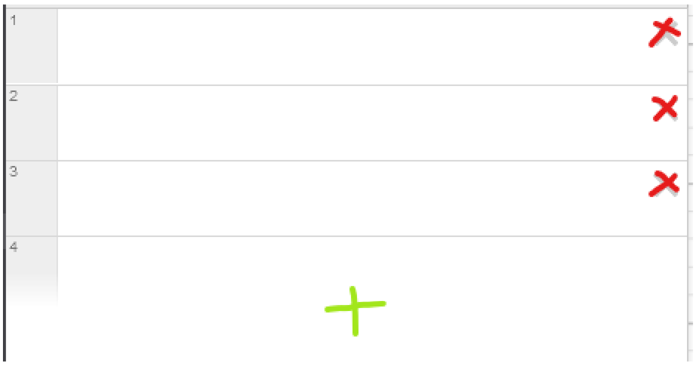
- **Open File**:
    - The user can access this feature from the `menu` by selecting `File` and then `Open File`.
    - A file browser window will open.
    - The files that can be opened will have the extension `.gcl` in our program. For example, it could be named `graph1.gcl`.
- **New File**:
    - The user can access this feature from the `menu` by selecting `File` and then `New File`.
    - The program will ask if they want to save the currently opened file with a prompt.
    - Then, the program will ask how the user wants to save the newly created file (via `Save As File`).
- **Save File**:
    - The user can access this feature from the `menu` by selecting `File` and then `Save File`.
    - This state is also invoked when the user exits the program to ensure they want to save their work.
    - If the file has been saved before:
        - The file will be saved in the same location as before.
    - If it hasn't been saved before:
        - The program will ask how the user wants to save the file (using `Save As File`).
- **Save As File**:
    - This is a program-related state that does not have direct interaction with the user. It's an internal state included within other states.
    - The `Save As` window will open, allowing the user to enter a name and choose a location.
    - After the user presses `OK`, the program will save the file with the specified name and location.
- **Exit**:
    - The user can initiate an exit by pressing the exit button on the program's main window.
    - When the exit is performed, the `Save File` state is initiated.

To support these features, a `menu` GUI bar should be provided with the following options:

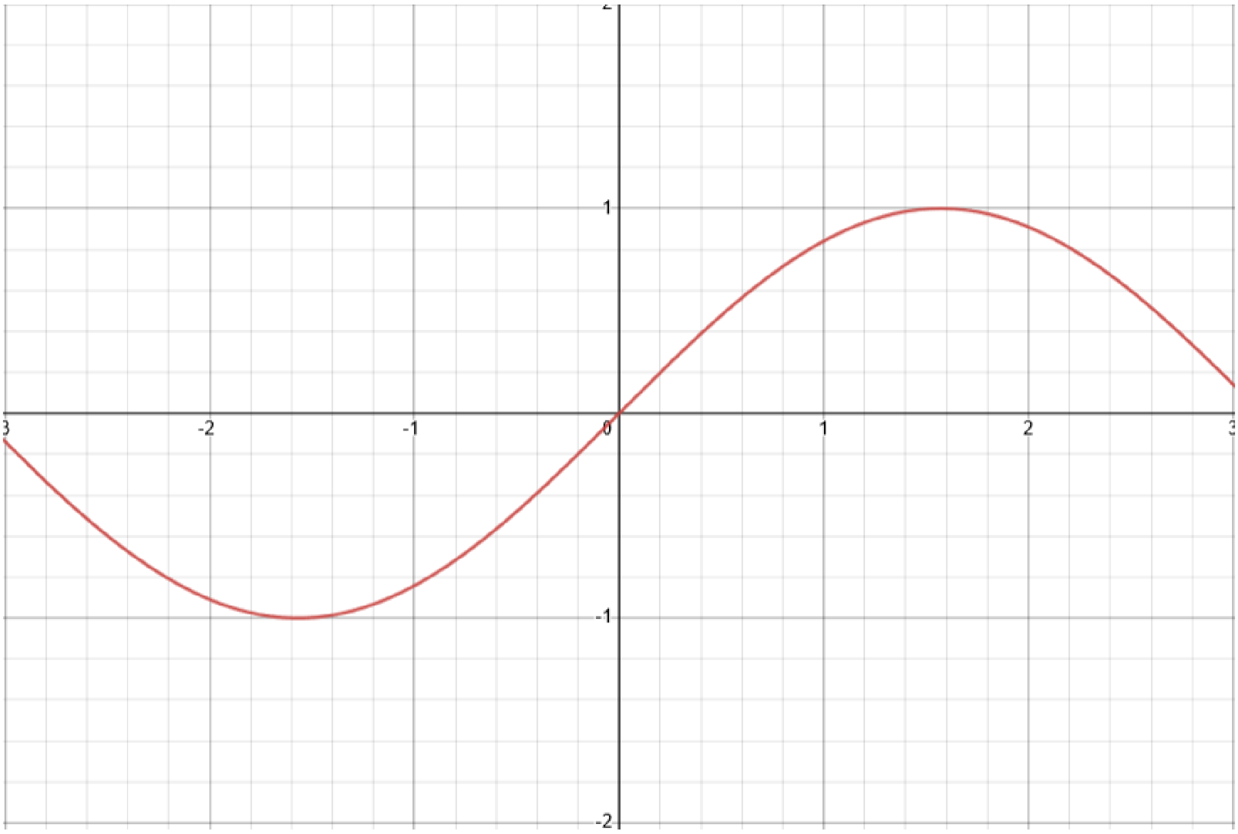
- Open File
- New File
- Save File
- Save As File

Now, moving on to the more graphing calculator-specific states:

- **Exit Expression**:
    - On the right, there will be a plane where the user can enter algebraic expressions.
    - This is somewhat abstract because the exact implementation is in `Remove Expression`, `Write Expression`, and `Add Expression`.
    - Here's how the space for entering expressions may look:

- 

- **Write Expression**:
  - Here, the user can input expressions in the `Text Fields`.
  - Examples of expressions include:
    - `x^(2/3) + 0.9(3.3 - x^2)^(1/2) * sin(10πx)`
    - `sin(2x) * cos(4x - 3) * log(10x, e)`
    - `1/(-x)`
    - `sin(2sin(2sin(2sin(x)))`
  - Every time a user enters a character, the `Validate Expression` state checks if the input is in the correct form.
  - If the input is not valid, the function will not proceed to display until the expression in the text field is valid. The user can be informed by displaying an error indicator (e.g., ⚠ in the text field.
  - When the expression is valid, it will be displayed using `Display Expression`.
- **Remove Expression**:
  - When the user presses ✕, the expression related to the text field, along with the text field itself, should be removed.
  - Then, the function will be removed from the display on the Cartesian Plane using `Display Expression`.
- **Add Expression**:
  - The user can instruct the program to add one more expression by clicking the + button.
  - A new text field should be added where the user can input their desired expression.
- **Verify Expression**:
  - An internal state.
  - Every time a user enters a new string in a text field, it is checked for validity.
  - The validation process ensures that the entered expression follows the correct syntax and can be parsed correctly.
  - If the input is not valid, an appropriate error message or indicator may be displayed to inform the user about the issue.
  - Validity checking is essential to prevent incorrect or incomplete expressions from causing errors and to provide real-time feedback to the user.
- **Zoom**:
  - The user should be able to zoom in the Cartesian plane with the mouse scroll.
  - This action triggers the `Update Cartesian Plane` to update the GUI displaying the Cartesian plane.
- **Display Expression**:
  - This is an internal state.

  - Every time the user enters a valid expression, this state displays it on the Cartesian plane by updating it (`Update Cartesian Plane`).
- **Update Cartesian Plane**:
  - This is an internal state.
  - It updates the Cartesian plane in accordance with the updated parameters.
  - Here’s an example of how the plane may look:
  - 

Basic Solution for the Task

- **Main Frame (`JFrame`):**
  - We will create a main window using `JFrame` to house the entire graphical calculator interface.
  - The frame will provide a user-friendly environment for working with algebraic expressions and visualizing graphs.
- **Menu (`JMenu`):**
  - The menu will include options like ”File” where users can open, save, and create new projects.
  - Users can access these features via the menu to manage their work effectively.
- **Panel with Text Fields (`JPanel`, `List`, `Text Fields`):**
  - The `JPanel` will hold the input components, making it easy for users to enter algebraic expressions.
  - A `List` will be employed to store multiple `Text Fields`, allowing users to work with several expressions simultaneously.
  - Each `Text Field` will be equipped with action listeners to respond in real-time as users input expressions.
- **Parser and Evaluator**:
  - The `Parser` will dissect the expressions, ensuring they are correctly formatted and understandable.
  - The `Evaluator` will calculate the value of the expressions for a certain `x`.
- **Displaying Cartesian Plane and Functions (`Graphics2D`):**
  - The `Graphics2D` component will be utilized to render the Cartesian plane and display the graphs of functions.
- **Parsing the Expression String (Our Own `Lexer`, `Parser`, and `Evaluator`):**
  - Our custom `Lexer` will break down user-entered expressions into meaningful parts.
  - The `Parser` will construct a structured representation of the expressions, making it possible to understand and manipulate them.
  - The `Evaluator` will perform real-time calculations and display the results on the Cartesian plane.
- **Tree (for Expression Representation)**:
  - We’ll use a structured data representation, possibly a tree, to store and manage expressions.
  - This data structure facilitates efficient parsing and evaluation of complex expressions, helping users work with mathematical functions effectively.
- **Handling I/O Operations (`JFileChooser`, `Serialization`, Additional Action Listeners)**:
  - `JFileChooser` will enable users to manage their projects, open existing ones, and save their work.
  - `Serialization` wil be used to save the state of the application, allowing users to resume their work later through projects.
  - Additional action listeners will be linked to buttons or menu items, streamlining tasks like saving files and opening projects.

> Please note that during the implementation phase, some of the solutions mentioned here may undergo revisions and updates.

```
Error parsing Mermaid diagram!

No diagram type detected matching given configuration for text:
```