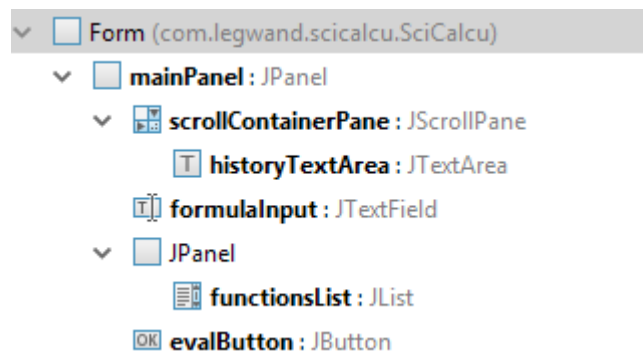
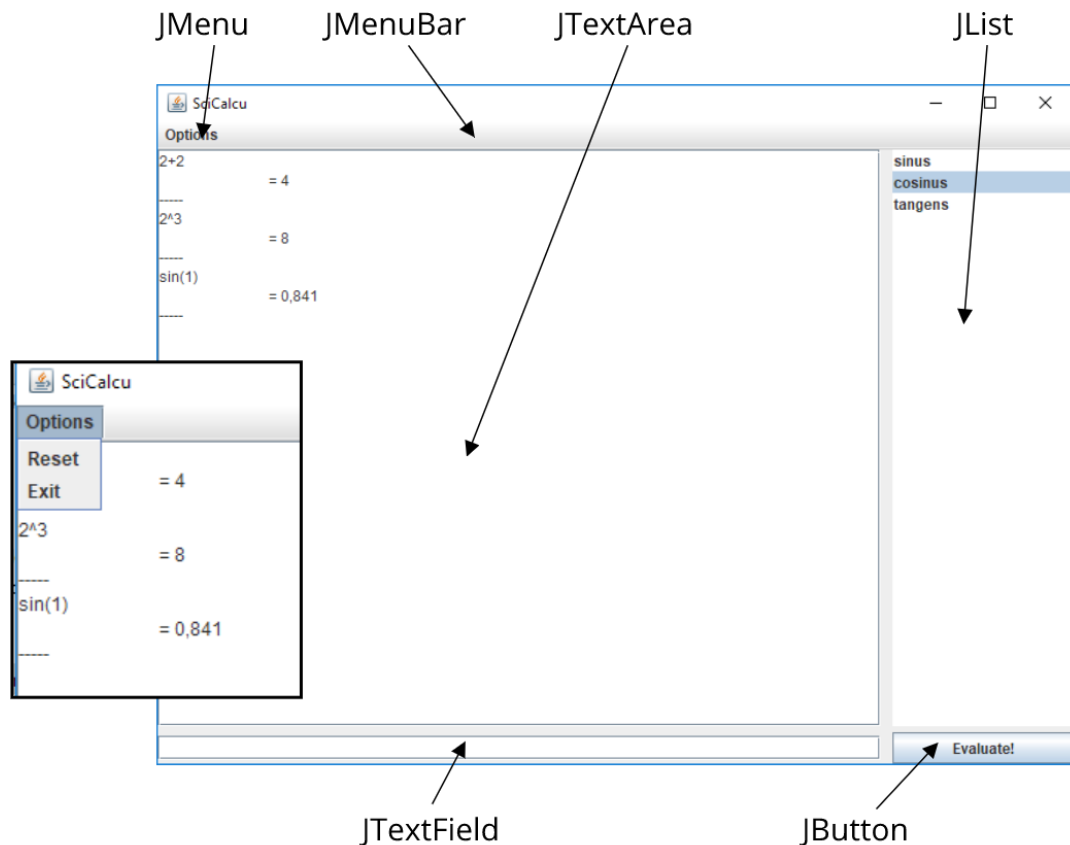


Programowanie w JAVA

Lab. 4 – Swing

1. **Cel zadania:** Implementacja kalkulatora naukowego z rozpoznawaniem działań w oparciu o bibliotekę mXparser!
2. Twoim zadaniem jest stworzenie kalkulatora z "zaawansowanym" interfejsem graficznym. Zastosuj się do poniższego przykładowego wyglądu. Nie musisz trzymać się go idealnie – pod warunkiem, że żądana funkcjonalność będzie spełniona.



3. Dodawanie biblioteki mXparser do projektu:

a. Dla projektów typu Maven

Do pliku pom.xml pod tagiem <version> dodaj:

```
<dependencies>
  <dependency>
    <groupId>org.mariuszgromada.math</groupId>
    <artifactId>MathParser.org-mXparser</artifactId>
    <version>4.3.3</version>
  </dependency>
</dependencies>
```

b. Dla projektów typu Ant:

File -> Project Structure -> Libraries -> [+] -> From Maven

Wpisz w pole wyszukiwarki org.mariuszgromada.math, wybierz najnowszą wersję, potwierdź OK

4. Szczegóły funkcjonalności:

- a. Zaprojektuj interfejs użytkownik tak, aby lista funkcji (JList) miała stałą szerokość i była "przyklejona" do prawej strony okna aplikacji. Okno tekstowe (JTextArea) wypełniało całą resztę przestrzeni okna. Taki stan powinien utrzymać się nawet po zmianie wielkości okna. W projekcie wykorzystaj wybrany układ (layout)
- b. **Reset (przycisk w menu):** Czyści zawartość JTextField i JTextArea
- c. **Exit (przycisk w menu):** Zamyka aplikację
- d. **JTextArea:** Pole tekstowe, w którym wyświetlana jest historia operacji: działanie i wynik.
 - i. Do wyświetlenia pojedynczego wpisu zaprojektuj szablon i wykorzystaj klasę MessageFormat
 - ii. Komponent JTextArea powinien być read-only
 - iii. W razie potrzeby suwak powinien umożliwić przesuwanie zawartości okna
- e. **JTextField:** Służy do wpisywania działań matematycznych:
 - i. Jeśli użytkownik wciśnie na klawiaturze klawisz [↑] to w polu tekstowym powinno się pojawić ostatnie wpisane działanie.
- f. **JList:** Zawiera listę funkcji matematycznych
 - i. Zaimplementuj 5 wybranych funkcji z:
<http://mathparser.org/mxparser-math-collection/unary-functions/>
 - ii. Zaimplementuj 3 wybrane funkcje z:<http://mathparser.org/mxparser-math-collection/constants/>
 - iii. Zaimplementuj 3 wybrane funkcje z:<http://mathparser.org/mxparser-math-collection/operators/>
 - iv. Last result – wklejający ostatni obliczony wynik
- g. **Lista funkcji powinna zostać zaimplementowana przy użyciu DefaultListModel** (<http://www.codejava.net/java-se/swing/jlist-custom-renderer-example>), który będzie zawierał pełne nazwy funkcji (czytelne dla użytkownika) i ich odpowiedniki akceptowalne przez parser.
 - i. Lista musi wyświetlać czytelne nazwy funkcji
 - ii. Po podwójnym kliknięciu wybrana funkcja powinna się pojawić w polu JTextField w formie akceptowalnej dla parsera (np. po kliknięciu sinus w polu tekstowym powinno pojawić sin())

- iii. Jeśli funkcja zawiera nawiasy, kursor powinien automatycznie ustawić się między nawiasami
- iv. Element "last result" powinien wklejać do pola JTextField ostatni wynik
- v. Po wciśnięciu klawisza Enter lub kliknięciu Evaluate działanie powinno zniknąć z JTextField, zostać sparsowane i pojawiać się razem z wynikiem w historii (jeśli jest poprawne)
- vi. Jeśli składnia działania jest niepoprawna powinien się pojawić popup (AlertDialog) z opisem błędu
- h. **Przycisk Evaluate:** działa analogicznie jak wciśnięcie Enter w JTextField
- i. **Postaraj się aby implementacja funkcjonalności była maksymalnie niezależna od prezentacji i pobierania danych!**
- j. Możesz skorzystać z designerów dostępnych w Twoim środowisku.

Korzystanie z biblioteki mXparser (<http://mathparser.org/>)

```
Expression expression = new Expression("2+2");
if (expression.checkSyntax()) {
    Double result = expression.calculate();
}
else {
    String errorMessage = expression.getErrorMessage()
    //należy rzucić wyjątek
}
```

Wyświetlanie okienka popup:

```
JOptionPane.showMessageDialog(null, "Message, "Title",
                                JOptionPane.ERROR_MESSAGE);
```

Korzystanie z MessageFormat

```
String result = MessageFormat.format(
    "At {1,time} on {1,date}, there was {2} on planet {0,number,integer}.",
    planet, new Date(), event);
```

<https://docs.oracle.com/javase/7/docs/api/java/text/MessageFormat.html>

5. Teoria:

- a. Maven i Ant. Czym są, dlaczego i jak ich używamy?
- b. Swing vs AWT
- c. Layouty w Swing – po co je używać, jakie problemy rozwiązują? Jakie znasz layouty i do czego służą.
- d. Obsługa zdarzeń Komponentów w Swing – ActionListener
- e. SwingUtilities.invokeLater
- f. Po co używamy Modeli w Swingu? Na przykładzie ListModel.
- g. Idea architektury MVC. Jak wygląda MVC w Swing?
- h. <https://examples.javacodegeeks.com/desktop-java/swing/java-swing-application-example/>
- i. Jak możesz uniezależnić implementację GUI od logiki aplikacji? Podaj przykład.

Po uzyskaniu zaliczenia na zajęciach, prześlij źródła w archiwum **zgodnie z konwencją nazewnictwa** (patrz Lab0.pdf) do chmury na adres:

<https://cloud.kisim.eu.org/s/igSLKzRDWoxgmCp>

najpóźniej do następnych zajęć.