UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

Mediu de programare grafica

Structura Sistemelor de Calcul

Autori: Bugnariu Dan
Grupa: 30234

Facultatea de Automatica
si Calculatoare

9 Ianuarie 2024

Cuprins

1 Introducere **2**
1.1 Tema 2
1.2 Specificatii 2
1.3 Obiective 2

2 Design si analiza **2**
2.1 Interfața Utilizator (UI) 2
2.2 Arhitectura Sistemului 2
2.3 Modul de Funcționare a Blocurilor 2

3 Implementare **3**
3.1 Structura de Directoare și Module 3
3.2 Exemplificări de Callback-uri 3
3.3 Gestionarea Erorilor și Depanarea 3
3.4 Modularitate proiectului 5
3.5 Interfata Grafica 7
3.5.1 Bara de unelte 7
3.5.2 Consola 7
3.5.3 Zona principala 7
3.5.4 Panoul de blocuri 7

4 Testare **8**

5 Studiu bibliografic **9**
5.1 Blockly 9
5.2 Java Swing Documentation 9

6 Concluzii **9**
6.1 Principale Realizari 9
6.2 Posibile Extensii 10

1 Introducere

1.1 Tema

Tema proiectului se axează pe dezvoltarea unui mediu de programare grafic, influențat de conceptul Blockly. Alegerea acestei teme este motivată de dorința de a simplifica procesul de creare a codului sursă, oferind utilizatorilor o interfață vizuală intuitivă pentru exprimarea logică a programării.

1.2 Specificatii

Mediu de programare grafic propus include un editor vizual cu funcționalități drag-and-drop, permitând utilizatorilor să creeze fluxuri logice de programare fără a fi nevoie de cod scris. Acesta oferă un set de blocuri predefinite pentru structuri de control, operații logice și altele. Interfața este dezvoltată folosind java swing.

1.3 Obiective

Obiectivele principale ale proiectului includ furnizarea unei interfețe pentru programarea grafică, facilitarea învățării programării prin intermediul unei abordări vizuale și generarea de cod eficient și funcțional. Ne propunem să creăm un mediu intuitiv și ușor de utilizat, care să încurajeze atât începătorii, cât și programatorii experimentați să exploreze și să implementeze logică de programare într-un mod grafic.

2 Design si analiza

2.1 Interfața Utilizator (UI)

Proiectul nostru se concentrează pe o interfață utilizator atrăgătoare și ușor de utilizat. Elementele cheie ale interfeței includ meniul de blocuri, zona de lucru principală și consola. Designul este simplu și intuitiv, cu culori bine alese pentru a facilita distingerea blocurilor și a structurii programului.

2.2 Arhitectura Sistemului

Arhitectura sistemului este modulară, cu componente distincte pentru manipularea blocurilor, interpretarea logicii de programare și executarea de cod. Modulele sunt interconectate printr-un sistem de callback-uri, asigurând o separare clară a responsabilităților și o ușurință în extindere.

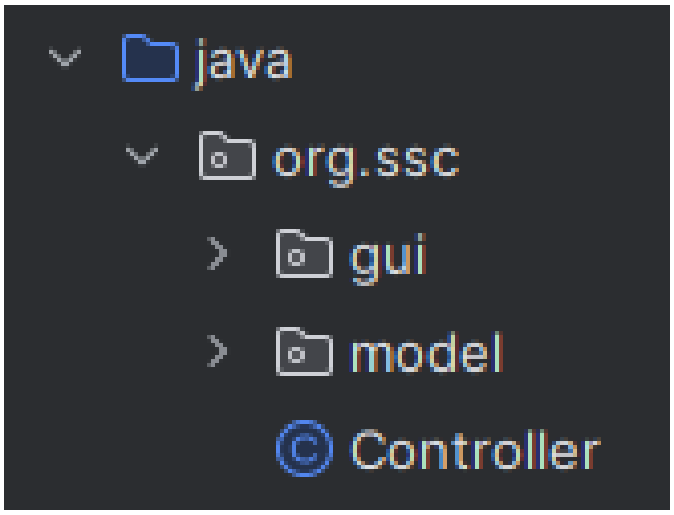
2.3 Modul de Funcționare a Blocurilor

Blocurile logice sunt create prin intermediul unui sistem drag-and-drop, iar conexiunile între acestea sunt gestionate printr-un model de conexiuni de intrare-iesire.

3 Implementare

3.1 Structura de Directoare și Module

Pentru a organiza eficient proiectul, am adoptat un model View-Controller (MVC), împărțind componentele în module distincte. Directoarele "Model", "View" și "Controller" reflectă clar separarea acestor responsabilități, facilitând dezvoltarea, testarea și întreținerea codului.



3.2 Exemplificări de Callback-uri

În cadrul MVC, Controller-ul reprezintă puntea dintre Model și View. Callback-urile sunt implementate pentru a gestiona interacțiunile utilizatorului și a actualiza modelul și vizualizarea corespunzătoare. Mai jos este un exemplu simplificat:

```
public static void main(String[] args) {
    MainWindow mainWindow = new MainWindow();
    mainWindow.addRunActionListener(e -> run(start, mainWindow));
}
```

In acest exemplu se adauca un callback pentru butonul de run, care va rula codul.

3.3 Gestionarea Erorilor și Depanarea

Implementarea gestionării erorilor și depanării se bazează pe mecanismele standard Java Swing. Erorile pot fi afișate prin ferestre de dialog sau în consolă, iar logging-ul poate fi utilizat pentru a înregistra informații relevante pentru depanare.

De asemeneas am introdus exceptii noi pentru simplificarea identificarii erorilor si depanarea lor.

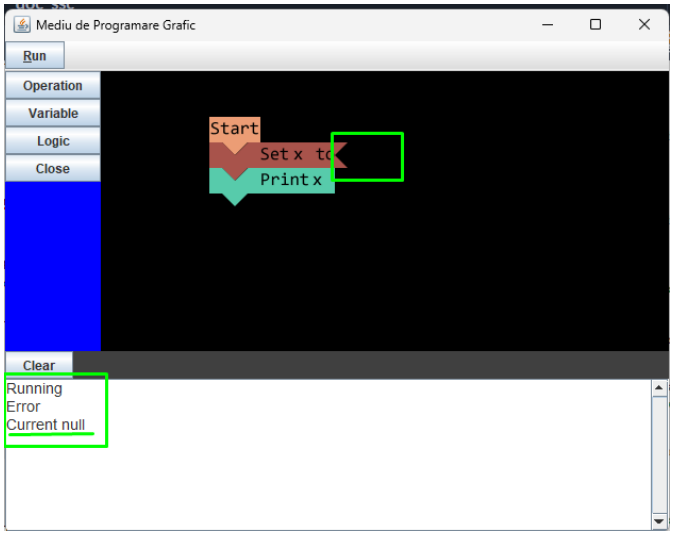
```
package org.ssc.model.math;

public class ComputeException extends Exception {
    private final String message;

    public ComputeException(String message) {
        super();
        this.message = message;
```

```
9       }
10
11      public String getMessageString() {
12          return message;
13      }
14  }
```

Aceasta exceptie este folosita cand un rezultat trebuie calculat si apare o eroare, spre exemplu:

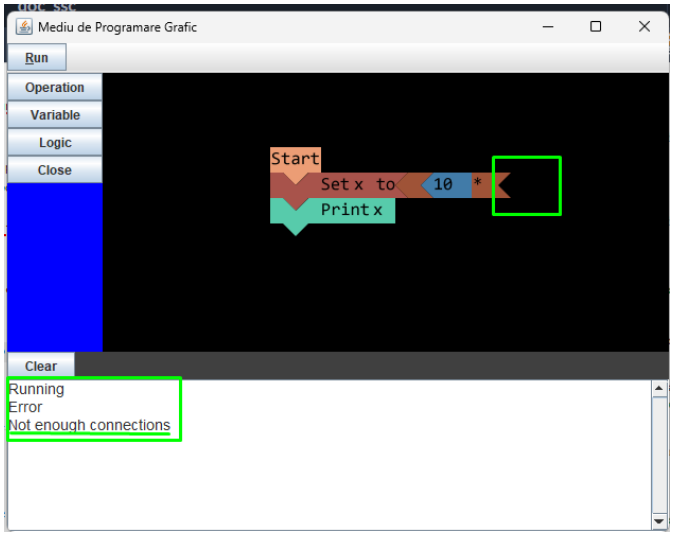
```
1  if (current == null) throw new ComputeException("Current null");
```

Cand un bloc nu are conectat un alt bloc de pe care sa ia o valoare.




```
1  if (current.getConnection(0) == null) throw new ComputeException("No connections");
2  if (current.getConnection(1) == null) throw new ComputeException("Not enough connections");
```

Cand un bloc de operatii nu are nici o conexiune/nu are suficiente conexiuni (doua).



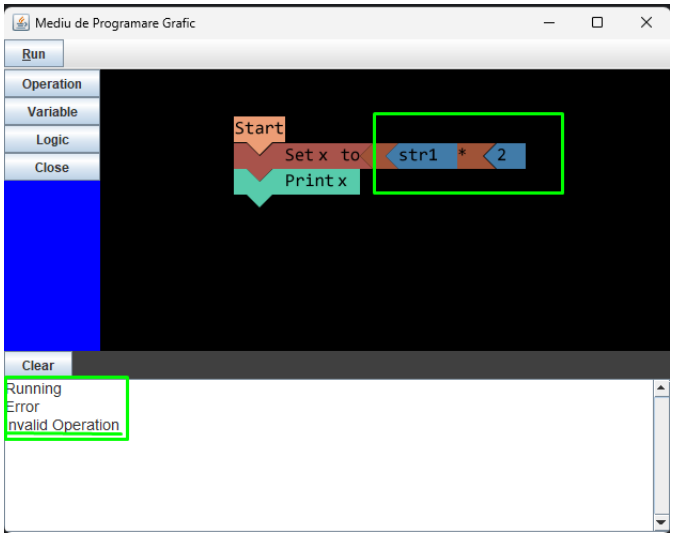

```
1  package org.ssc.model.math;
2
```

```
3  public class InvalidOperation extends RuntimeException {
4  }
```

Aceasta exceptie este folosita cand o anumita operatie nu e definita.

```
1  @Override
2  public Variable<ArrayList<T>> mul(Object value) throws TypeMismatchException, InvalidOperati
3      throw new InvalidOperation();
4  }
```

Spre exemplu operatia de inmultie pe un sir de caractere.




```
1  package org.ssc.model;
2
3  public class TypeMismatchException extends RuntimeException {
4  }
```

Aceasta exceptie este folostia cand o variabila ar urma sa execute o operatia cu o variabila de alt tip, pentru care operatia nu e definita.

```
1  @Override
2  public Variable<Character> add(Object value) throws TypeMismatchException, InvalidOperation
3      if (!(value instanceof VInt)) throw new TypeMismatchException();
4      VChar result = new VChar();
5      result.value = (char) (this.value + ((VInt) value).getValue());
6      return null;
7  }
```

Valoare unui caracter poate fi modificata doar de un intreg, ce ii creste sau scade valoarea in ascii, iar pentru alte tipuri arunca exceptia.

3.4 Modularitate proiectului

Toate blocurile au o clasa de baza numita Block.

```
1  package org.ssc.model;
2  import org.ssc.gui.BlockPanel;
```

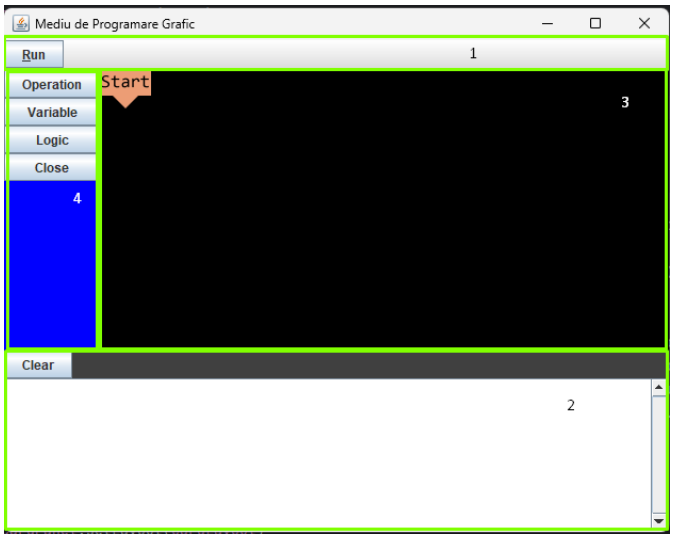
```java
import java.util.ArrayList;

public class Block {
    protected Block previous;
    protected Block next;
    protected ArrayList<Block> connections;
    protected String blockName;

    public Block() {...}
    public Block(Block previous, Block next) {...}
    public BlockPanel getBlockPanel() {...}
    public void setBlockPanel(BlockPanel blockPanel) {...}
    public String getBlockName() {...}
    public void setBlockName(String blockName) {...}
    public Block getPrevious() {...}
    public void setPrevious(Block previous) {...}
    public Block getNext() {...}
    public void setNext(Block next) {...}
    public void removeConnection(Block connection) {...}
    public void addConnection(Block connection) {...}
    public void setConnection(Block connection, int id) {...}
    public int getNumberOfConnections() {...}
    public Block getConnection(int index) {...}
    public Block[] getConnections() {...}
    public String getName() {...}
    public boolean setName(String name) {...}
}
```

Toate tipurile de variabile pe langa extinderea clasei Block, implementeaza interfata Variable.

```java
package org.ssc.model.variable;
import org.ssc.model.TypeMismatchException;
import org.ssc.model.math.InvalidOperation;

public interface Variable<T> {
    T getValue();
    void setValue(Object value) throws TypeMismatchException;
    void changeValue(Object value) throws TypeMismatchException;
    Variable<T> add(Object value) throws TypeMismatchException, InvalidOperation;
    Variable<T> sub(Object value) throws TypeMismatchException, InvalidOperation;
    Variable<T> mul(Object value) throws TypeMismatchException, InvalidOperation;
    Variable<T> div(Object value) throws TypeMismatchException, InvalidOperation;
    Variable<T> mod(Object value) throws TypeMismatchException, InvalidOperation;
    String getPrint();
    Class<?> getType();
    Variable<T> cloneVariable();
}
```

3.5 Interfata Grafica

Interfrata Grafica este compusa din 4 parti principale



3.5.1 Bara de unelte

Aici se gaste butonul ce ruleaza programul (Run).

3.5.2 Consola

Aici apar erorile din programul generat, si afisarea rezultatelor rulari, aceasta poate fi curatata cu ajutorul butonului Clear.

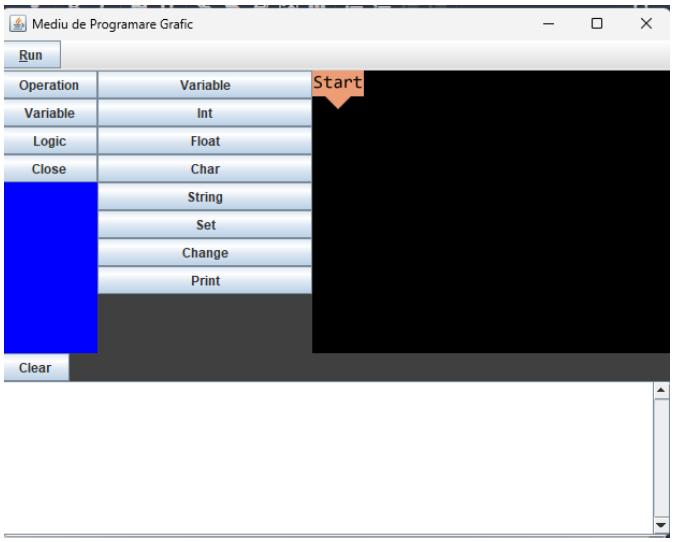
3.5.3 Zona principala

Aceasta este zona principala de lucru, unde se regaseste blocul de start si unde se va construi programul.

Aici blocurile pot fi mutate, se pot conecta, deconecta, modifica valoarea lor, pentru generarea programului dorit.

3.5.4 Panoul de blocuri

De aici se alege tipul de blocuri dorite, care va extinde panoul, ce poate fi inchis cu close.

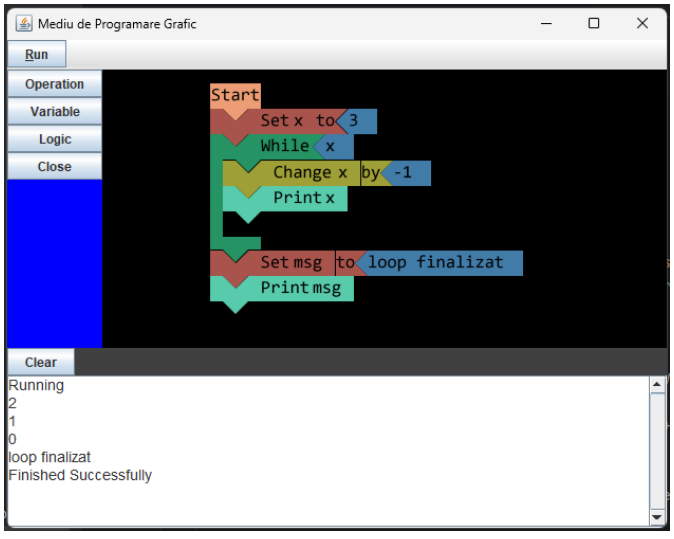


Spre exemplu, selectaria blocurilor ce au legatura cu variabile.

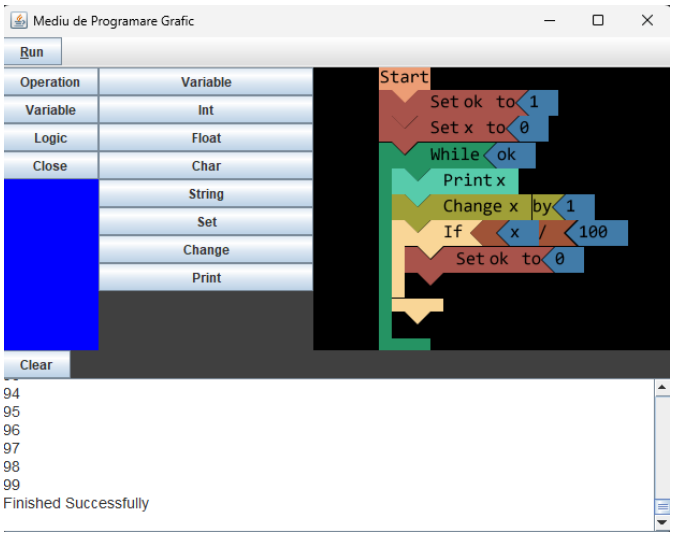
- Variable - insereaza un bloc ce acceseaza o variabila dupa nume
- Int - insereaza un bloc cu valoarea unui int
- Float - insereaza un bloc cu valoarea unui flaot
- Char - insereaza un bloc cu valoarea unui char
- String - insereaza un bloc cu valoarea unui string
- Set - seteaza valoare unei variabile
- Change - modifica valoare unei variabile (adauga valoarea la variabila)
- Print - afiseaza in consola valoarea variabilei

4 Testare

Testele au fost de acceptanta, ele au fost realizate prin generarea unor programe si rularea lor.



Un program care afiseaza numerele naturale mai mici ca 3 in ordine descrescatoare.



Un program care afiseaza numerele naturale mai mici ca 100 in ordine crescatoare.

5 Studiu bibliografic

În procesul de dezvoltare a proiectului, am consultat o serie de surse care au contribuit la înțelegerea și implementarea componentelor cheie ale proiectului, în special în ceea ce privește integrarea și adaptarea funcționalităților din Blockly și Java Swing.

5.1 Blockly

A Web-Based, Visual Programming Editor

Acesta fiind inspiratia principala a proiectului, am utilizat acest fremework pentru a vedea cum functioneaza, ce pot modifica sa se potriveasca proiectului meu si pentru a intelege cum arata un mediu grafic de programare.

5.2 Java Swing Documentation

Pentru generarea interfeti grafice am utilizat java swing, asadar am folosit documentatia pentru a crea butoane, panouri, zone de text, zona principala in care blocurile pot fi mutate individual sau grupate si crearea ferestrei.

6 Concluzii

Proiectul a reprezentat o aventura în lumea programării vizuale, folosindu-se de limbajul java si kit-ul de dezvoltare java swing. Acest demers a avut ca rezultat crearea unei platforme flexibile și ușor de utilizat pentru dezvoltarea de aplicații bazate pe programarea vizuală.

6.1 Principale Realizari

- Crearea unui proiect modular, usor de extins
- Dezvoltarea abilităților de programare în java
- Intelegerea kit-ului de dezvoltare a interfetei grafice, java swing

6.2 Posibile Extensii

- Functii, crearea si utilizarea lor
- Salvarea si incarcarea programelor
- Copierea de bucati de cod
- Stergerea blocurilor introduse din greseala, sau care nu mai sunt folosite