Folder src

27 printable files

(file list disabled)

src/Calculator.java

```java
/**
 * @Author :  Maxime Lestiboudois
 * @Author :  Nathan Parisod
 */

import calculator.*;
import util.*;
import java.util.Scanner;
public class Calculator {

    private final State state;
    private final Scanner scanner;

    public Calculator() {
        this.state = new State();
        this.scanner = new Scanner(System.in);
    }

    public void start(){
        boolean stay = true;
        while(stay){
            System.out.print("> ");
            String input = scanner.nextLine().trim();

            if(input.equals("exit")){
                break;
            }

            stay = processInput(input);

            displayState();
        }
    }

    private boolean processInput(String input){
        if(isNumber(input)){
            new NumberOperator(Integer.parseInt(input), state).execute();
            new EnterOperator(state).execute();
        }
        else {
            // Si l'entrée est un opérateur, trouver et exécuter l'opération correspondante
            switch (input) {
                case "+":
                    new OperandOperator(new Addition(),state).execute();
                    break;
                case "-":
                    new OperandOperator(new Subtraction(), state).execute();
                    break;
                case "*":
                    new OperandOperator(new Multiplication(), state).execute();
                    break;
                case "/":
                    new OperandOperator(new Division(), state).execute();
                    break;
                case "x^2":
                    new SquareOperator(state).execute();
                    break;
                case "sqrt":
                    new SqrtOperator(state).execute();
                    break;
                case "1/x":
                    new FractionnalOperator(state).execute();
                    break;
                case "c":
                    new COperator(state).execute();
                    break;
                case "exit":
                    return false;
                default:
                    System.out.println("Opérateur inconnu");
                    return false;
            }
        }
        return true;
    }

    private void displayState(){
        if
        (state.hasError()){
            System.out.println("Error");
        }
        else{
            System.out.println(state.getStack().toString());
        }
    }
    private boolean isNumber(String input){
        try {
            Integer.parseInt(input);
            return true;
        } catch (NumberFormatException e) {
            return false;
        }
    }

    public static void main(String[] args){
        new Calculator().start();
    }








}
```

src/Main.java

```java
/**
 * @Author :  Maxime Lestiboudois
 * @Author :  Nathan Parisod
 */

import calculator.JCalculator;

public class Main
{
    public static void main(String ... args) {
        new JCalculator();
    }
}
```

src/calculator/Addition.java

```java
/**
 * @Author :  Maxime Lestiboudois
 * @Author :  Nathan Parisod
 */

package calculator;

public class Addition extends Operation<Double>{
    @Override
    public Double apply(Double a, Double b) {
        return a+b;
    }
}
```

src/calculator/BackspaceOperator.java

```java
/**
 * @Author :  Maxime Lestiboudois
 * @Author :  Nathan Parisod
 */

package calculator;

public class BackspaceOperator extends Operator {
    public BackspaceOperator(State state) {
        super(state);
    }

    @Override
    public void execute() {
       state.backspace();
    }
}
```

src/calculator/CEOperator.java

```java
/**
 * @Author :  Maxime Lestiboudois
 * @Author :  Nathan Parisod
 */

package calculator;


public class CEOperator extends Operator {
    public CEOperator(State state) {
        super(state);
    }

    @Override
    public void execute() {
        if(state.hasError()){
            state.resetError();
        }
        state.clearCurrentValue();
    }
}
```

src/calculator/COperator.java

```java
/**
 * @Author :  Maxime Lestiboudois
 * @Author :  Nathan Parisod
 */

package calculator;

public class COperator extends CEOperator {

    public COperator(State state) {
        super(state);
    }
    @Override
    public void execute() {
        super.execute();
        state.clearStack();
    }
}
```

src/calculator/Division.java

```
/**
 * @Author :  Maxime Lestiboudois
 * @Author :  Nathan Parisod
 */

package calculator;

public class Division extends Operation<Double>{
    @Override
    public Double apply(Double a, Double b) {
        if(b!=0) {
            return a / b;
        }
        else {
            throw new ArithmeticException();
        }
    }
}
```

src/calculator/EnterOperator.java

```
/**
 * @Author :  Maxime Lestiboudois
 * @Author :  Nathan Parisod
 */

package calculator;

public class EnterOperator extends Operator {
   public EnterOperator(State state) {
       super(state);
   }

    @Override
    public void execute() {
        state.pushCurrentValue();
    }
}
```

src/calculator/FractionnalOperator.java

```
/**
 * @Author :  Maxime Lestiboudois
 * @Author :  Nathan Parisod
 */
```

```java
package calculator;

public class FractionnalOperator extends Operator {
    public FractionnalOperator(State state) {
        super(state);
    }
    @Override
    public void execute() {
        if (state.hasError()) return;
        if(!state.getCurrentValue().equals("0")) {
            state.pushCurrentValue();
        }
        Double a = state.popFromStack();
        if(a != null){
            Double b = 1/a;
            state.setCurrentValue(b.toString());
            state.pushCurrentValue();
        }
        else {
            state.setError("Erreur d'addition");
        }
    }
}
```

src/calculator/JCalculator.java

```java
/**
 * @Author :  Maxime Lestiboudois
 * @Author :  Nathan Parisod
 */

package calculator;

import java.awt.Color;
import java.awt.Font;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.Insets;
import java.util.ArrayList;
import java.util.List;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JList;
import javax.swing.JScrollPane;
import javax.swing.JTextField;

//import java.awt.event.*;

public class JCalculator extends JFrame {
    // Tableau representant une pile vide
    private static final String[] empty = {"< empty stack >"};

    // Zone de texte contenant la valeur introduite ou resultat courant
    private final JTextField jNumber = new JTextField("0");

    // Composant liste representant le contenu de la pile
    private final JList jStack = new JList(empty);

    // Contraintes pour le placement des composants graphiques
    private final GridBagConstraints constraints = new GridBagConstraints();

    // Instance de l'état de la calculatrice
    private final State state = new State();
```

```java
// Mise a jour de l'interface apres une operation (jList et jStack)
private void update() {
    // Modifier une zone de texte, JTextField.setText(string nom)
    // Modifier un composant liste, JList.setListData(Object[] tableau)
  System.out.println(state.getCurrentValue());
    jNumber.setText(state.getCurrentValue());
    Object[] stackData = state.getStack().toArray();
    if (stackData.length == 0) {
        jStack.setListData(empty);
    } else {
        jStack.setListData(stackData);
    }
}

// Ajout d'un bouton dans l'interface et de l'operation associee,
// instance de la classe Operation, possedeant une methode execute()
private void addOperatorButton(String name, int x, int y, Color color,
                               final Operator operator) {
    JButton b = new JButton(name);
    b.setForeground(color);
    constraints.gridx = x;
    constraints.gridy = y;
    getContentPane().add(b, constraints);
    b.addActionListener(e -> {
        operator.execute();
        update();
    });
}

public JCalculator() {
    super("JCalculator");
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    getContentPane().setLayout(new GridBagLayout());

    // Contraintes des composants graphiques
    constraints.insets = new Insets(3, 3, 3, 3);
    constraints.fill = GridBagConstraints.HORIZONTAL;

    // Nombre courant
    jNumber.setEditable(false);
    jNumber.setBackground(Color.WHITE);
    jNumber.setHorizontalAlignment(JTextField.RIGHT);
    constraints.gridx = 0;
    constraints.gridy = 0;
    constraints.gridwidth = 5;
    getContentPane().add(jNumber, constraints);
    constraints.gridwidth = 1; // reset width

    // Rappel de la valeur en memoire
    addOperatorButton("MR", 0, 1, Color.RED, new MROperator(state));

    // Stockage d'une valeur en memoire
    addOperatorButton("MS", 1, 1, Color.RED, new MSOperator(state));

    // Backspace
    addOperatorButton("<=", 2, 1, Color.RED, new BackspaceOperator(state));

    // Mise a zero de la valeur courante + suppression des erreurs
    addOperatorButton("CE", 3, 1, Color.RED, new CEOperator(state));

    // Comme CE + vide la pile
    addOperatorButton("C", 4, 1, Color.RED, new COperator(state));
```

```
        // Boutons 1-9
        for (int i = 1; i < 10; i++)
            addOperatorButton(String.valueOf(i), (i - 1) % 3, 4 - (i - 1) / 3,
                    Color.BLUE, new NumberOperator(i, state));
        // Bouton 0
        addOperatorButton("0", 0, 5, Color.BLUE, new NumberOperator(0, state));

        // Changement de signe de la valeur courante
        addOperatorButton("+/-", 1, 5, Color.BLUE, new PositiveNegativeOperator(state));

        // Operateur point (chiffres apres la virgule ensuite)
        addOperatorButton(".", 2, 5, Color.BLUE, new PointOperator(state));

        // Operateurs arithmetiques a deux operandes: /, *, -, +
        addOperatorButton("/", 3, 2, Color.RED, new OperandOperator(new Division(), state));
        addOperatorButton("*", 3, 3, Color.RED, new OperandOperator(new Multiplication(), state));
        addOperatorButton("-", 3, 4, Color.RED, new OperandOperator(new Subtraction(),state));
        addOperatorButton("+", 3, 5, Color.RED, new OperandOperator(new Addition(),state));

        // Operateurs arithmetiques a un operande: 1/x, x^2, Sqrt
        addOperatorButton("1/x", 4, 2, Color.RED, new FractionnalOperator(state));
        addOperatorButton("x^2", 4, 3, Color.RED, new SquareOperator(state));
        addOperatorButton("Sqrt", 4, 4, Color.RED, new SqrtOperator(state));

        // Entree: met la valeur courante sur le sommet de la pile
        addOperatorButton("Ent", 4, 5, Color.RED, new EnterOperator(state));

        // Affichage de la pile
        JLabel jLabel = new JLabel("Stack");
        jLabel.setFont(new Font("Dialog", 0, 12));
        jLabel.setHorizontalAlignment(JLabel.CENTER);
        constraints.gridx = 5;
        constraints.gridy = 0;
        getContentPane().add(jLabel, constraints);

        jStack.setFont(new Font("Dialog", 0, 12));
        jStack.setVisibleRowCount(8);
        JScrollPane scrollPane = new JScrollPane(jStack);
        constraints.gridx = 5;
        constraints.gridy = 1;
        constraints.gridheight = 5;
        getContentPane().add(scrollPane, constraints);
        constraints.gridheight = 1; // reset height

        setResizable(false);
        pack();
        setVisible(true);
    }
}
```

src/calculator/MROperator.java

```
/**
 * @Author :  Maxime Lestiboudois
 * @Author :  Nathan Parisod
 */

package calculator;

public class MROperator extends Operator {

    public MROperator(State state) {
        super(state);
    }
    @Override
    public void execute() {
        System.out.println("currentValue in MR" + state.getCurrentValue() + " memory="+ (state.getMemory() == null));
        if(state.getMemory() == null) {
            return;
        }
        state.setCurrentValue(state.getMemory().toString());
        System.out.println("currentValue in MR" + state.getCurrentValue());
        //est-ce qu'il faut reset la mémoire? non
        //est-ce que la valeur est directement push dans la stack? ???
    }
}
```

src/calculator/MSOperator.java

```
/**
 * @Author :  Maxime Lestiboudois
 * @Author :  Nathan Parisod
 */

package calculator;

public class MSOperator extends Operator {

    public MSOperator(State state) {
        super(state);
    }
    @Override
    public void execute() {
        state.setMemory(Double.parseDouble(state.getCurrentValue()));
        System.out.println("enregistré: "+ state.getMemory());
        //state.clearCurrentValue();

    }
}
```

src/calculator/Multiplication.java

```
/**
 * @Author :  Maxime Lestiboudois
 * @Author :  Nathan Parisod
 */

package calculator;

public class Multiplication  extends Operation<Double>{
    @Override
    public Double apply(Double a, Double b) {
        return a*b;
    }
}
```

src/calculator/NumberOperator.java

```
/**
 * @Author :  Maxime Lestiboudois
 * @Author :  Nathan Parisod
 */

package calculator;

import java.sql.SQLOutput;

public class NumberOperator extends Operator {
    private final int value;

    public NumberOperator(int value, State state) {
       super(state);
        this.value = value;
    }

    @Override
    public void execute() {
        state.appendToCurrentValue((char) (value + 48));    // 48 is the ASCII code for '0'
    }
}
```

src/calculator/OperandOperator.java

```
/**
 * @Author :  Maxime Lestiboudois
 * @Author :  Nathan Parisod
 */

package calculator;

public class OperandOperator extends Operator {
    private final Operation<Double> operand;
    public OperandOperator(Operation<Double> operand, State state) {
        super(state);
        this.operand = operand;
    }
    @Override
    public void execute() {
        if (state.hasError()) return;
        if(!state.getCurrentValue().equals("0")) {
            state.pushCurrentValue();
        }
        Double b = state.popFromStack();
        Double a = state.popFromStack();
        if (a != null && b != null) {
            state.setCurrentValue(operand.apply(a, b).toString());
            state.pushCurrentValue();
        } else {
            state.setError("Erreur");
        }
    }
}
```

src/calculator/Operation.java

```
/**
 * @Author :  Maxime Lestiboudois
 * @Author :  Nathan Parisod
 */

package calculator;

public abstract class Operation<T> {

    public abstract T apply(T a, T b);
}
```

src/calculator/Operator.java

```
/**
 * @Author :  Maxime Lestiboudois
 * @Author :  Nathan Parisod
```

```java
 */

package calculator;

abstract class Operator {
    protected State state;

    public Operator(State state) {
      this.state = state;
    }
    abstract void execute();
}
```

src/calculator/PointOperator.java

```java
/**
 * @Author :  Maxime Lestiboudois
 * @Author :  Nathan Parisod
 */

package calculator;

public class PointOperator extends Operator {

    public PointOperator(State state) {
        super(state);
    }
    @Override
    public void execute() {
        state.appendToCurrentValue('.');
    }
}
```

src/calculator/PositiveNegativeOperator.java

```java
/**
 * @Author :  Maxime Lestiboudois
 * @Author :  Nathan Parisod
 */

package calculator;

public class PositiveNegativeOperator extends Operator {

    public PositiveNegativeOperator(State state) {
        super(state);
    }
    @Override
    public void execute() {
        if(state.getCurrentValue().indexOf(0) == '-'){
            state.negativeToPositive();
        }
        else{
            state.positiveToNegative();
        }
    }
}
```

src/calculator/SqrtOperator.java

```java
/**
 * @Author :  Maxime Lestiboudois
 * @Author :  Nathan Parisod
 */
```

```java
package calculator;

public class SqrtOperator extends Operator {
    public SqrtOperator(State state) {
        super(state);
    }

    @Override
    public void execute() {
        if (state.hasError()) return;
        if(!state.getCurrentValue().equals("0")) {
            state.pushCurrentValue();
        }
        Double a = state.popFromStack();
        if(a != null){
            Double b = Math.sqrt(a);
            state.setCurrentValue(b.toString());
            state.pushCurrentValue();
        }
        else {
            state.setError("Erreur");
        }
    }
}
```

src/calculator/SquareOperator.java

```java
/**
 * @Author :  Maxime Lestiboudois
 * @Author :  Nathan Parisod
 */

package calculator;

public class SquareOperator extends Operator {
    public SquareOperator(State state) {
        super(state);
    }
    @Override
    public void execute() {
        if (state.hasError()) return;
        if(!state.getCurrentValue().equals("0")) {
            state.pushCurrentValue();
        }
        Double a = state.popFromStack();
        if(a != null){
            Double b = a*a;
            state.setCurrentValue(b.toString());
            state.pushCurrentValue();
        }
        else {
            state.setError("Erreur");
        }
    }

}
```

src/calculator/State.java

```java
/**
 * @Author :  Maxime Lestiboudois
 * @Author :  Nathan Parisod
 */

package calculator;
```

```java
import util.Stack;

public class State {
    private final Stack<Double> stack = new Stack<>();
    private String currentValue = "0";
    private boolean error = false;
    private Double memory;


    public State() {
    }

    public void appendToCurrentValue(char c) {

        if (error) {
            resetError();
        }
        if (currentValue.equals("0")) {
            currentValue = Character.toString(c);
        } else {
            currentValue += c;
        }
   }

    public void backspace() {
        if (!error && currentValue.length() > 1) {
            currentValue = currentValue.substring(0, currentValue.length() - 1);
        } else {
            currentValue = "0";
        }
    }

    public void clearCurrentValue() {
        currentValue = "0";
    }

    public void pushCurrentValue() {
        try {
            double value = Double.parseDouble(currentValue);
            stack.insert(value);
            clearCurrentValue();
        } catch (NumberFormatException e) {
            System.out.println("Error in pushCurrentValue");
        }
    }

    public void setError(String message) {
        error = true;
        currentValue = "Erreur";
    }

    public void resetError() {
        error = false;
        clearCurrentValue();
    }

    public Double popFromStack() {
        if (!stack.isEmpty()) {
            return stack.pop();
        } else {
            System.out.println("Error in popFromStack");
            return null;
        }
    }
    public void clearStack() {
        stack.clear();
    }

    public String getCurrentValue() {
        return currentValue;
    }

    public boolean hasError() {
        return error;
    }

    public void negativeToPositive() {
        currentValue = currentValue.substring(1, currentValue.length() - 1);
    }

    public void positiveToNegative() {
        currentValue = "-" + currentValue;
    }

    public void setCurrentValue(String currentValue) {
        this.currentValue = currentValue;
    }

    public Stack<Double> getStack() {
        return stack;
    }

    public Double getMemory() {
        return memory;
    }
    public void setMemory(Double memory) {
        this.memory = memory;
    }


}
```

src/calculator/Subtraction.java

```
/**
 * @Author :  Maxime Lestiboudois
 * @Author :  Nathan Parisod
 */

package calculator;

public class Subtraction  extends Operation<Double>{
    @Override
    public Double apply(Double a, Double b) {
        return a-b;
    }
}
```

src/util/It.java

```
/**
 * @Author :  Maxime Lestiboudois
 * @Author :  Nathan Parisod
 */

package util;

import java.util.NoSuchElementException;
```

```java
public class It<T> {
    Node<T> current;

    public It(Node<T> start){
        this.current = start;
    }

    public boolean hasNext() {
        return current != null;
    }

    public T next() {
        if(!hasNext()){
            throw new NoSuchElementException();
        }
        T value = current.data;
        current = current.next;
        return value;
    }
}
```

src/util/Liste.java

```java
/**
 * @Author :  Maxime Lestiboudois
 * @Author :  Nathan Parisod
 */

package util;

public class Liste<T> {
    Node<T> head;

    public Liste() {
        this.head = null;
    }

    public void add(T value) {
        Node<T> newNode = new Node<>(value);
        if (head == null) {
            head = newNode;
        } else {
            Node<T> temp = head;
            while (temp.next != null) {
                temp = temp.next;
            }
            temp.next = newNode;
        }
    }

    public It<T> iterator() {
        return new It<>(head);
    }
}
```

src/util/Node.java

```java
/**
 * @Author :  Maxime Lestiboudois
 * @Author :  Nathan Parisod
 */

package util;

class Node<T> {
    T data;
    Node<T> next;

    Node(T data) {
        this.data = data;
        this.next = null;
    }
}
```

src/util/Stack.java

```java
/**
 * @Author :  Maxime Lestiboudois
 * @Author :  Nathan Parisod
 */

package util;

import java.util.NoSuchElementException;

public class Stack<T> {
    private Node<T> head;

    public Stack() {
        this.head = null;
    }

    public void insert(T value) {
        Node<T> newNode = new Node<>(value);
        newNode.next = head;
        head = newNode;
    }

    public void clear() {
        head = null;
    }

    public T pop() {
        if (isEmpty()) {
            throw new NoSuchElementException("Stack is empty");
        }
        T value = head.data;
        head = head.next;
        return value;
    }

    public boolean isEmpty() {
        return head == null;
    }

    @Override
    public String toString() {
        String result = "";
        Node<T> current = head;
        while (current != null) {
            result += current.data;
            if (current.next != null) {
                result += " ";
            }
            current = current.next;
        }
        return result;
    }

    public Object[] toArray() {
        int size = 0;
        Node<T> current = head;
        while (current != null) {
            size++;
            current = current.next;
        }

        Object[] array = new Object[size];
        current = head;
        for (int i = 0; i < size; i++) {
            array[i] = current.data;
            current = current.next;
        }
        return array;
    }

    public It<T> iterator() {
        return new It<>(head);
    }

}
```