Folder starter

25 printable files

(file list disabled)

starter\Main.java

```
import calculator.JCalculator;

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

starter\calculator\Add.java

```
package calculator;

class Add extends Operator {
    Add(State state) {
        super(state);
    }

    void execute() {
        if (state.cannotCalculate()) {
            return;
        }

        // Add together the current value and the last value of the stack
        state.currentValue = Double.parseDouble(state.currentValue) + Double.parseDouble(state.stack.pop()) + "";
        state.hasResult = true;

        if (state.isCurrentValueZero()) {
            state.currentValue = "0";
        }
    }
}
```

starter\calculator\Backspace.java

```
package calculator;

class Backspace extends Operator {
    Backspace(State state) {
        super(state);
    }

    void execute() {
        if (state.hasError) {
            state.clearCurrentValue();
            return;
        }

        // Remove the last character from the current value
        if (!state.currentValue.isEmpty()) {
            state.currentValue = state.currentValue.substring(0, state.currentValue.length() - 1);
        }
    }
}
```

starter\calculator\Calculator.java

```java
package calculator;

import java.util.Scanner;

public class Calculator {
    private final State state = new State();

    public static void main(String[] args) {
        Calculator calculator = new Calculator();
        Scanner scanner = new Scanner(System.in);
        String input;

        System.out.println("java Calculator");
        do {
            System.out.print("> ");
            input = scanner.nextLine();
            calculator.processInput(input);
        } while (!input.equals("exit"));
    }

    private void processInput(String input) {
        if (input.matches("-?\\d+(\\.\\d+)?")) {
            // If the input is a number, we set the current value to this number
            double number = Double.parseDouble(input);
            state.currentValue = Double.toString(number);
        } else {
            // If the input is not a number, we check if it is an operator
            switch (input) {
                case "+":
                    state.stack.pop();
                    new Add(state).execute();
                    break;
                case "-":
                    state.stack.pop();
                    new Subtract(state).execute();
                    break;
                case "*":
                    state.stack.pop();
                    new Multiply(state).execute();
                    break;
                case "/":
                    state.stack.pop();
                    new Divide(state).execute();
                    break;
                case "sqrt":
                    state.stack.pop();
                    new SquareRoot(state).execute();
                    break;
                case "square":
                    state.stack.pop();
                    new Square(state).execute();
                    break;
                case "oneover":
                    state.stack.pop();
                    new OneOver(state).execute();
                    break;
                case "negate":
                    state.stack.pop();
                    new Negate(state).execute();
                    break;
                case "store":
                    state.stack.pop();
                    new MemoryStore(state).execute();
                    // If the stack is empty, we don't print it
                    if (state.stack.isEmpty()) {
```

```
                        return;
                    }
                    break;
                case "recall":
                    new MemoryRecall(state).execute();
                    break;
                case "clear":
                    new Clear(state).execute();
                    return;
                case "exit":
                    return;
                default:
                    System.out.println("Erreur : Entrée non valide");
                    return;
            }
        }

        if (!state.isCurrentValueZero()) {
            // If the current value is not 0, we add it to the stack
            state.stack.push(state.currentValue);
            System.out.println(state.stack);
        } else if (!state.stack.isEmpty()) {
            // If the current value is 0 and the stack is not empty, we print the stack
            System.out.println(state.stack);
        }
    }
}
```

starter\calculator\Clear.java

```
package calculator;

class Clear extends Operator {
    Clear(State state) {
        super(state);
    }

    void execute() {
        // Set the current value to its default 0 and empty the stack
        state.clearCurrentValue();
        state.stack.clear();
    }
}
```

starter\calculator\ClearError.java

```
package calculator;

class ClearError extends Operator {
    ClearError(State state) {
        super(state);
    }

    void execute() {
        // Set the current value to its default 0
        state.clearCurrentValue();
    }
}
```

starter\calculator\Digit.java

```
package calculator;
```

```
import java.util.Objects;

class Digit extends Operator {

    String number;

    Digit(State state, int number) {
        super(state);
        this.number = Integer.toString(number);
    }

    void execute() {
        if (state.hasError) {
            return;
        }

        // If the current value is the result from the previous calculation, add it to the stack unless it is 0
        if (state.hasResult && !state.isCurrentValueZero()) {
            state.stack.push(state.currentValue);
            state.clearCurrentValue();
        }

        // Add a new digit to the current value if not default 0
        if (state.currentValue.equals("0")) {
            state.currentValue = number;
        } else {
            state.currentValue += number;
        }
    }
}
```

starter\calculator\Divide.java

```
package calculator;

class Divide extends Operator {
    Divide(State state) {
        super(state);
    }

    void execute() {
        if (state.cannotCalculate()) {
            return;
        }

        // Check if dividing by 0
        if (state.isCurrentValueZero()) {
            state.hasError = true;
            state.currentValue = "Cannot divide by zero";
            return;
        }

        // Divide the current value with the last value of the stack
        state.currentValue = Double.parseDouble(state.stack.pop()) / Double.parseDouble(state.currentValue) + "";
        state.hasResult = true;
    }
}
```

starter\calculator\Dot.java

```
package calculator;

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

```
        void execute() {
            // Add a dot to the current value if there isn't already one in the value and there is a number
            if (state.currentValue.indexOf('.') == -1 && !state.hasError && !state.currentValue.isEmpty()) {
                state.currentValue += '.';
            }
        }
    }
```

starter\calculator\Enter.java

```
    package calculator;

    class Enter extends Operator {
        Enter(State state) {
            super(state);
        }

        void execute() {

            // Add the current value to the stack if not 0 or ending with '.'
            if (!state.isCurrentValueZero() && !state.hasError && !state.currentValue.endsWith(".")) {
                state.stack.push(state.currentValue);
                if (state.hasResult) {
                    state.hasResult = false;
                }
                state.currentValue = "0";
            }
        }
    }
```

starter\calculator\JButton.java

```
    package calculator;

    public class JButton {
    }
```

starter\calculator\JCalculator.java

```
    package calculator;

    import javax.swing.JButton;
    import javax.swing.*;
    import java.awt.*;

    //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();

      private final State state = new State();
```

```
// Mise a jour de l'interface apres une operation (jList et jStack)
private void update()
{
  // Modifier une zone de texte, JTextField.setText(string nom)
  jNumber.setText(state.currentValue);
  // Modifier un composant liste, JList.setListData(Object[] tableau
  if (state.stack.isEmpty()) {
    jStack.setListData(empty);
  } else {
    jStack.setListData(state.stack.getStack());
  }
}

// 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 MemoryRecall(state));

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

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

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

  // Comme CE + vide la pile
  addOperatorButton("C",  4, 1, Color.RED, new Clear(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 Digit(state,i));
  // Bouton 0
  addOperatorButton("0", 0, 5, Color.BLUE, new Zero(state));
```

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

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

        // Operateurs arithmetiques a deux operandes: /, *, -, +
        addOperatorButton("/", 3, 2, Color.RED, new Divide(state));
        addOperatorButton("*", 3, 3, Color.RED, new Multiply(state));
        addOperatorButton("-", 3, 4, Color.RED, new Subtract(state));
        addOperatorButton("+", 3, 5, Color.RED, new Add(state));

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

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

starter\calculator\MemoryRecall.java

```
    package calculator;

    class MemoryRecall extends Operator {
        MemoryRecall(State state) {
            super(state);
        }

        void execute() {
            // Recall the last stored value
            if (!state.memory.equals("0") && !state.hasError) {
                // Add the current value to the stack if it is a result
                if (state.hasResult) {
                    state.stack.push(state.currentValue);
                }
                state.currentValue = state.memory;
            }
        }
    }
```

starter\calculator\MemoryStore.java

```
package calculator;

class MemoryStore extends Operator {
    MemoryStore(State state) {
        super(state);
    }

    void execute() {
        // Store the current value
        if (!state.currentValue.equals("0") && !state.hasError) {
            // If we stored a result, we no longer have a result
            if (state.hasResult) {
                state.hasResult = false;
            }
            state.memory = state.currentValue;
            state.currentValue = "0";
        }
    }
}
```

starter\calculator\Multiply.java

```
package calculator;

class Multiply extends Operator {
    Multiply(State state) {
        super(state);
    }

    void execute() {
        if (state.cannotCalculate()) {
            return;
        }

        // Multiply together the current value and the last value of the stack
        state.currentValue = Double.parseDouble(state.currentValue) * Double.parseDouble(state.stack.pop()) + "";
        state.hasResult = true;

        if (state.isCurrentValueZero()) {
            state.currentValue = "0";
        }
    }
}
```

starter\calculator\Negate.java

```
package calculator;

class Negate extends Operator {
    Negate(State state) {
        super(state);
    }

    void execute() {
        if (state.isCurrentValueZero() || state.hasError) {
            return;
        }

        // Invert the sign of the current value
        if (state.currentValue.charAt(0) == '-') {
            state.currentValue = state.currentValue.substring(1);
        } else {
            state.currentValue = "-" + state.currentValue;
        }
```

```
    }
}
```

starter\calculator\OneOver.java

```
package calculator;

class OneOver extends Operator {
    OneOver(State state) {
        super(state);
    }

    void execute() {
        if (state.hasError) {
            return;
        }

        if (state.isCurrentValueZero()) {
            state.hasError = true;
            state.currentValue = "Cannot divide by zero";
            return;
        }

        // One divided by the current value
        state.currentValue = Double.toString(1 / Double.parseDouble(state.currentValue));
        state.hasResult = true;
    }
}
```

starter\calculator\Operator.java

```
package calculator;

abstract class Operator {
    State state;

    Operator(State state) {
        this.state = state;
    }

    abstract void execute();
}
```

starter\calculator\Square.java

```
package calculator;

class Square extends Operator {
    Square(State state) {
        super(state);
    }

    void execute() {
        if (state.hasError) {
            return;
        }

        // Current value to the power of 2
        state.currentValue = String.valueOf(Math.pow(Double.parseDouble(state.currentValue), 2));
        state.hasResult = true;
    }
}
```

starter\calculator\SquareRoot.java

```java
package calculator;

class SquareRoot extends Operator {
    SquareRoot(State state) {
        super(state);
    }

    void execute() {
        if (state.hasError) {
            return;
        }

        if (state.currentValue.charAt(0) == '-') {
            state.hasError = true;
            state.currentValue = "Cannot calculate the SquareRoot of a negative number";
            return;
        }

        // SquareRoot of the current value
        state.currentValue = Math.sqrt(Double.parseDouble(state.currentValue)) + "";
        state.hasResult = true;
    }
}
```

starter\calculator\State.java

```java
package calculator;

import util.Stack;

class State {
    final Stack<String> stack;

    String currentValue, memory;

    boolean hasResult, hasError;

    State() {
        stack = new Stack<>();
        currentValue = "0";
        memory = "0";
        hasResult = false;
        hasError = false;
    }

    /**
     * Check if the current value is 0
     * Since we are using doubles, we need to check if the value is 0.0
     * @return true if the current value is 0, false otherwise
     */
    boolean isCurrentValueZero() {
        return Double.parseDouble(currentValue) == 0;
    }

    /**
     * Check if the calculus cannot be done
     * @return true if the calculus cannot be done, false otherwise
     */
    boolean cannotCalculate() {
        return hasError || stack.isEmpty() || currentValue.isEmpty();
    }

    /**
     * Clear the current value and reset the state
     */
    void clearCurrentValue() {
```

```
            currentValue = "0";
            hasResult = false;
            hasError = false;
        }
    }
```

starter\calculator\Subtract.java

```
    package calculator;

    class Subtract extends Operator {
        Subtract(State state) {
            super(state);
        }

        void execute() {
            if (state.cannotCalculate()) {
                return;
            }

            // Subtract the current value with the last value of the stack
            state.currentValue = Double.parseDouble(state.currentValue) - Double.parseDouble(state.stack.pop()) + "";
            state.hasResult = true;

            if (state.isCurrentValueZero()) {
                state.currentValue = "0";
            }
        }
    }
```

starter\calculator\Zero.java

```
    package calculator;

    import java.util.Objects;

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

        void execute() {
            // Add a zero to the current value unless current value is 0
            if (!Objects.equals(state.currentValue, "0") && !state.hasError) {
                state.currentValue += "0";
            }
        }
    }
```

starter\util\Stack.java

```
    package util;

    import java.util.EmptyStackException;
    import java.util.Iterator;

    class Item<T> {
        T value;
        Item<T> next;

        Item(T value) {
            this.value = value;
            next = null;
```

```
    }
}

public class Stack<T> implements Iterable<T> {
    Item<T> top;
    int size;

    public Stack() {
        top = null;
        size = 0;
    }

    public void push(T value) {
        Item<T> newItem = new Item<>(value);
        newItem.next = top;
        top = newItem;
        ++size;
    }

    public T pop() {
        if (isEmpty()) {
            throw new EmptyStackException();
        }
        T value = top.value;
        top = top.next;
        --size;

        return value;
    }

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

    public Object[] getStack() {
        if (isEmpty()) {
            throw new EmptyStackException();
        }
        Object[] array = new Object[size];
        Item<T> item = top;

        for (int i = 0; i < array.length; ++i) {
            array[i] = item.value;
            item = item.next;
        }

        return array;
    }

    public void clear() {
        while (!isEmpty()) {
            pop();
        }
    }

    @Override
    public String toString() {
        StringBuilder sb = new StringBuilder();
        Item<T> item = top;

        while (item != null) {
            sb.append(item.value).append(" ");
            item = item.next;
        }

        return sb.toString();
    }

    @Override
    public Iterator<T> iterator() {
        return new StackIterator<>(this);
```

```
    }
}
```

starter\util\StackIterator.java

```
package util;

import java.util.Iterator;
import java.util.NoSuchElementException;

public class StackIterator<T> implements Iterator<T> {
    private Item<T> current;

    public StackIterator(Stack<T> stack) {
        current = stack.top;
    }

    @Override
    public boolean hasNext() {
        return current.next != null;
    }

    @Override
    public T next() {
        if (!hasNext()) {
            throw new NoSuchElementException();
        }
        current = current.next;

        return current.value;
    }
}
```