Stage 1 - Source Code (Before and After Refactoring)

# Singleton Pattern

## Before Singleton

public class Calculator {  
 private String currentOperand;  
 private String previousOperand;  
 private String operation;  
  
 public void compute() {  
 float computation;  
 float curr = Float.parseFloat(this.currentOperand);  
 float prev = Float.parseFloat(this.previousOperand);  
  
 switch (this.operation) {  
 case "+" -> computation = prev + curr;  
 case "-" -> computation = prev - curr;  
 case "×" -> computation = prev \* curr;  
 case "÷" -> computation = prev / curr;  
 default -> { return; }  
 }  
 this.currentOperand = Float.toString(computation);  
 this.previousOperand = "";  
 this.operation = "";  
 }  
}

## After Singleton

public class CalculatorEngine {  
 private static CalculatorEngine instance;  
 private String current;  
 private String previous;  
 private String opSymbol;  
  
 private CalculatorEngine() {}  
  
 public static CalculatorEngine getInstance() {  
 if (instance == null) {  
 instance = new CalculatorEngine();  
 }  
 return instance;  
 }  
  
 public void compute() {  
 float computation;  
 float curr = Float.parseFloat(this.current);  
 float prev = Float.parseFloat(this.previous);  
  
 switch (this.opSymbol) {  
 case "+" -> computation = prev + curr;  
 case "-" -> computation = prev - curr;  
 case "×" -> computation = prev \* curr;  
 case "÷" -> computation = prev / curr;  
 default -> { return; }  
 }  
 this.current = Float.toString(computation);  
 this.previous = "";  
 this.opSymbol = "";  
 }  
}

# Factory Method Pattern

## Before Factory Method

public class CalculatorEngine {  
 private String current;  
 private String previous;  
 private String opSymbol;  
  
 public void compute() {  
 float computation;  
 float curr = Float.parseFloat(this.current);  
 float prev = Float.parseFloat(this.previous);  
  
 switch (this.opSymbol) {  
 case "+" -> computation = prev + curr;  
 case "-" -> computation = prev - curr;  
 case "×" -> computation = prev \* curr;  
 case "÷" -> computation = prev / curr;  
 default -> { return; }  
 }  
 this.current = Float.toString(computation);  
 this.previous = "";  
 this.opSymbol = "";  
 }  
}

## After Factory Method

// Operation interface  
public interface Operation {  
 String symbol();  
 double apply(double prev, double curr);  
}  
  
// Concrete operations  
public class AddOperation implements Operation {  
 public String symbol() { return "+"; }  
 public double apply(double prev, double curr) { return prev + curr; }  
}  
public class SubOperation implements Operation {  
 public String symbol() { return "-"; }  
 public double apply(double prev, double curr) { return prev - curr; }  
}  
public class MulOperation implements Operation {  
 public String symbol() { return "×"; }  
 public double apply(double prev, double curr) { return prev \* curr; }  
}  
public class DivOperation implements Operation {  
 public String symbol() { return "÷"; }  
 public double apply(double prev, double curr) { return prev / curr; }  
}  
  
// Factory  
public class OperationFactory {  
 private Map<String, Operation> registry = new HashMap<>();  
  
 public OperationFactory() {  
 register(new AddOperation());  
 register(new SubOperation());  
 register(new MulOperation());  
 register(new DivOperation());  
 }  
  
 private void register(Operation op) {  
 registry.put(op.symbol(), op);  
 }  
  
 public Operation get(String symbol) {  
 return registry.get(symbol);  
 }  
}  
  
// Updated CalculatorEngine  
public class CalculatorEngine {  
 private String current;  
 private String previous;  
 private String opSymbol;  
 private OperationFactory factory = new OperationFactory();  
  
 public void compute() {  
 double curr = Double.parseDouble(this.current);  
 double prev = Double.parseDouble(this.previous);  
 Operation op = factory.get(this.opSymbol);  
 double result = op.apply(prev, curr);  
 this.current = Double.toString(result);  
 this.previous = "";  
 this.opSymbol = "";  
 }  
}