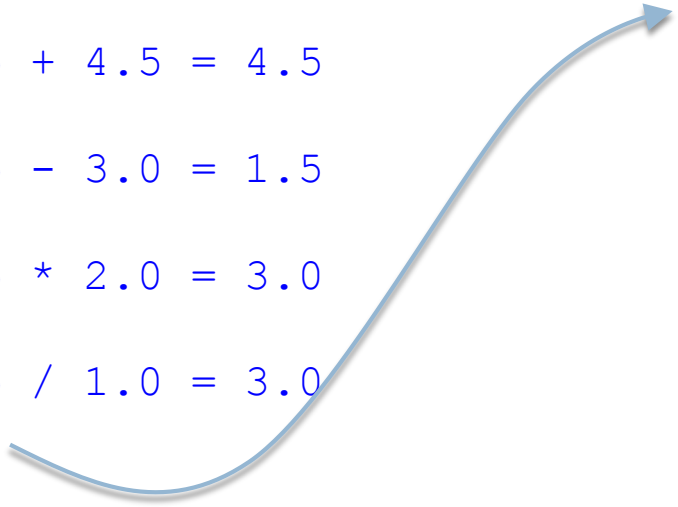
Uma calculadora simples

Calculadora simples

- Implementar calculadora simples mas robusta de modo a que eventuais erros na introdução de dados por parte do utilizador não impliquem a interrupção abrupta do programa
- Operações básicas
  - Soma
  - Subtracção
  - Multiplicação
  - Divisão

Exemplos de traço do programa


```
Formato de entrada: operador numero
Q para terminar o programa
Resultado = 0.0
+ 4.5
Resultado + 4.5 = 4.5
- 3
Resultado - 3.0 = 1.5
* 2
Resultado * 2.0 = 3.0
/ 1
Resultado / 1.0 = 3.0
```



```
q 2
Operador desconhecido: q
Tente mais uma vez ...

Formato de entrada: operador numero
Q para terminar o programa
Resultado = 0.0
+ 4
Resultado + 4.0 = 4.0
q 2
Operador desconhecido: q
Ja chega! Tente noutra altura.
Fim do programa.
```

Exemplos de traço do programa

```
Formato de entrada: operador numero
Q para terminar o programa
Resultado = 0.0
* 2
Resultado * 2.0 = 0.0
+ 45
Resultado + 45.0 = 45.0
/ 0.00001
Divisao por zero.
Fim do programa.
```

```
Formato de entrada: operador numero
Q para terminar o programa
Resultado = 0.0
+ 45.2
Resultado + 45.2 = 45.2
Q
Resultado final e 45.2
Fim do programa.
```

Diagrama de classes

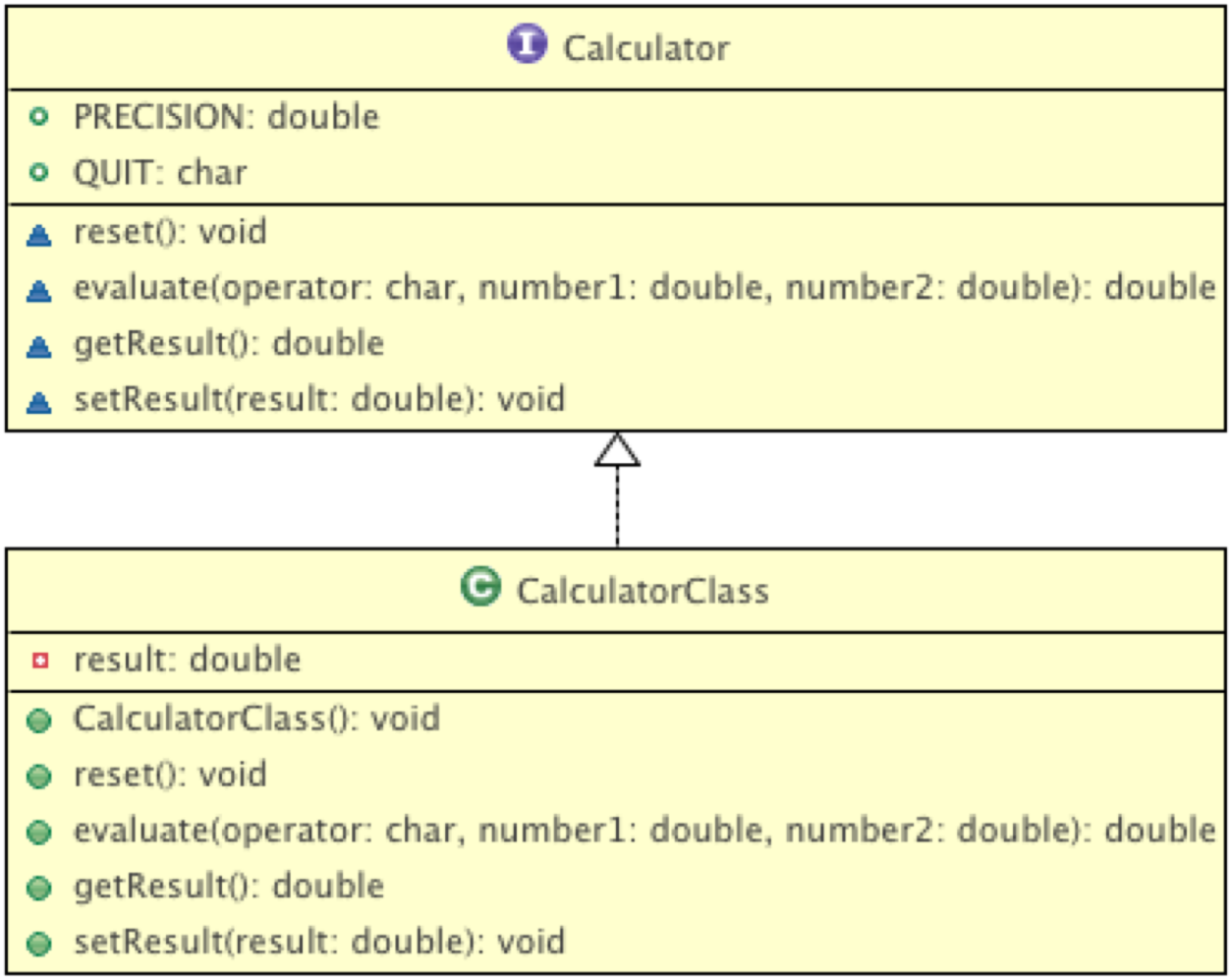
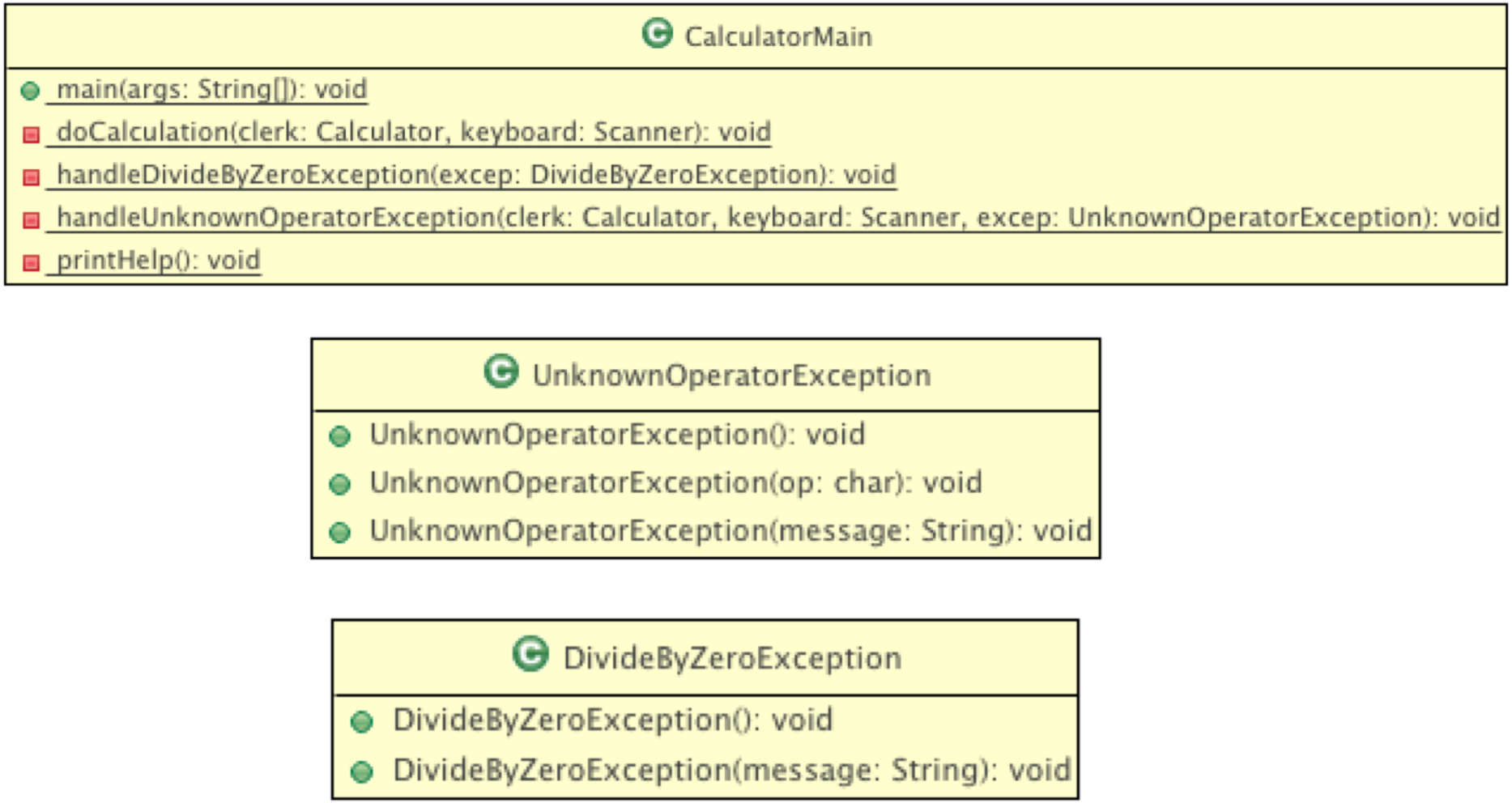


Diagrama de classes



A interface Calculator


```
package calculator;

public interface Calculator {
   // Numbers this close are treated as if equal to zero
    final double PRECISION = 0.000001;
   final char QUIT = 'Q';

   void reset();
   double evaluate(char operator, double number1, double number2)
                throws DivideByZeroException, UnknownOperatorException;
   double getResult();
   void setResult(double result);
}
```

A classe CalculatorClass


```
package calculator;

public class CalculatorClass implements Calculator {
   private double result;

   public CalculatorClass() { result = 0; }

   public void reset() { result = 0; }
   public double evaluate(char operator, double number1, double number2)
                  throws  DivideByZeroException, UnknownOperatorException {
          . . .
   }
   public double getResult() { return result; }
   public void setResult(double result) { this.result = result; }
}
```

A classe CalculatorClass


```
public double evaluate(char operator, double number1, double number2)
        throws DivideByZeroException, UnknownOperatorException {

  double answer;
  switch (operator) {
        case '+': answer = number1 + number2; break;
        case '-': answer = number1 - number2; break;
        case '*': answer = number1 * number2; break;
        case '/': if ( (-Calculator.PRECISION < number2) &&
                        (number2 < Calculator.PRECISION) )
                                throw new DivideByZeroException();
                  answer = number1/number2;
                  break;
        default: throw new UnknownOperatorException(operator);
  }
  return answer;
}
```

A classe `DivideByZeroException`


```
package calculator;

public class DivideByZeroException extends RuntimeException {
    public DivideByZeroException( ) {
        super();
    }

    public DivideByZeroException(String message) {
        super(message);
    }
}
```

A classe **UnknownOperatorException**


```
package calculator;

public class UnknownOperatorException
                                        extends RuntimeException {
    public UnknownOperatorException( ) {
        super();
    }

    public UnknownOperatorException(String message) {
        super(message);
    }
}
```

A classe **MainCalculator**


```
public static void main(String[] args) {
  Calculator clerk = new CalculatorClass( );
  Scanner keyboard = new Scanner(System.in);
  try {
    printHelp();
    doCalculation(clerk, keyboard );
  }
  catch (UnknownOperatorException excep) {
    handleUnknownOperatorException(clerk, keyboard, excep);
  }
  catch (DivideByZeroException excep) {
    handleDivideByZeroException(excep);
  }
  System.out.println("O resultado final e" + clerk.getResult());
  System.out.println("Fim do programa.");
}
```

A classe **MainCalculator**


```
private static void doCalculation(Calculator clerk, Scanner keyboard)
        throws DivideByZeroException, UnknownOperatorException {
   char nextOp;
   double nextNumber;
   boolean done = false;
   clerk.setResult(0);
   double result = clerk.getResult();
   System.out.println("Resultado = " + result);
   while (!done) {
     System.out.print("> ");
     nextOp = (keyboard.next( )).charAt(0);
     if ((nextOp == Calculator.QUIT)) done = true;
     else {
         nextNumber = keyboard.nextDouble( );  // may launch an exception !!!
         result = clerk.evaluate(nextOp, result, nextNumber);
         clerk.setResult(result);
         System.out.println("Resultado" + nextOp + nextNumber + "=" + result);
     }
}}
```

A classe **MainCalculator**


```
private static void handleDivideByZeroException(DivideByZeroException excep) {
  System.out.println("Divisao por zero.");
  System.out.println("Fim do programa.");
}

private static void handleUnknownOperatorException(Calculator clerk,
                          Scanner keyboard, UnknownOperatorException excep) {
  System.out.println(excep.getMessage());
  System.out.println("Tente mais uma vez ... ");
  try {
      printHelp();
      doCalculation(clerk, keyboard);
  } catch(UnknownOperatorException excep2)          {
      System.out.println(excep2.getMessage());
      System.out.println("Ja chega! Tente noutra altura.");
      System.out.println("Fim do programa.");
  } catch(DivideByZeroException excep3) {
      handleDivideByZeroException(excep3);
  }
}
```

Excepções e pré-condições

- As excepções permitem-nos tratar situações inesperadas no código
  - podemos então tornar os nossos programas mais robustos e não dependentes do "mundo seguro"
- Nos comentários javadoc em vez de especificar pré-condições podemos especificar em que condições é que os métodos levantam excepções

```
/**
* ...
* @throws NoMoreElementsException if !hasNext()
**/
E next() throws NoMoreElementsException;
```

Mundo seguro vs. mundo inseguro

- Assumindo um "mundo seguro" podemos programar assumindo que o código é usado correctamente
  - por exemplo antes de chamar um método as respectivas pré-condições são validadas
  - **"Design by Contract"**
- Contudo muitas vezes temos de assumir um "mundo inseguro" e desenvolver código mais defensivo
  - desenvolvimento deve antecipar a existência de erros
  - **Programação Defensiva**