

Programação Orientada Pelos Objectos

Excepções



2

Excepções?

Como acrescenta um elemento v a um vector a ?

3

```
a[count++] = v;
```

- Na maioria das vezes, funciona sem problema
- Contudo, se o vector estiver cheio, gera um erro de execução!
 - O vector está cheio e não cabe mais nenhum elemento

Problema resolvido com uma *abordagem preventiva*

4

```
if (count < N)
    a[count++] = v;
else
    // lidar com o array cheio
    // por exemplo, fazer resize
```

- Basta actuar de forma preventiva, com a ajuda de uma instrução condicional
 - É o que temos feito até agora!

Problemas levantados pela *abordagem preventiva*

5

- Código poluído por testes relativos a situações raras
 - Esses testes prejudicam a compreensão do que se faz nas situações normais
- A biblioteca da linguagem pode não disponibilizar meios para testar preventivamente determinadas situações
 - Por exemplo, a falta de direitos de escrita de um ficheiro, ou o disco estar cheio.

Como funciona o tratamento de exceções?

6

- Em algumas situações, preferimos deixar o erro de execução ocorrer efectivamente
 - Lidamos com esse erro *à posteriori*, com a ajuda do mecanismo de tratamento de exceções

Podemos fazer bem melhor

7

- Há muito que a comunidade das linguagens de programação conhece uma forma de resolver este tipo de problemas bastante melhor:
 - Um **mecanismo de tratamento de exceções**

Não podemos programar métodos que nos devolvam um determinado código de erro?

8

- Poder, podemos
- Mas o código fica, normalmente, com o mesmo problema de legibilidade do código construído com a abordagem preventiva

O que ganhamos com um **mecanismo de tratamento de exceções**?

9

- Podemos escrever o código das situações normais “ignorando” as situações especiais
 - Podemos tratar as situações especiais numa zona separada do código, escolhida de acordo com as nossas conveniências
 - O ganho de legibilidade, em certos programas, é substancial

Qual é o preço a pagar pelo tratamento à posteriori?

10

- Necessitamos de uma linguagem que suporte o tratamento de exceções
 - O Java suporta-o
- Este mecanismo tem uma implementação complexa e envolve novos conceitos de tempo de execução, tais como:
 - Lançar exceção
 - Propagar exceção
 - Capturar exceção

O que é que um mecanismo de excepções nos oferece?

11

- Separação do código afecto à gestão e processamento de erros do restante código
 - Programador escreve o fluxo normal de código e remete o tratamento de casos excepcionais para outra localização
 - O trabalho de detectar, reportar e controlar erros fica organizado de uma forma mais clara e efectiva
- Propagação de erros até à pilha de chamadas de métodos que controla a execução do programa
 - Permite que os erros sejam propagados até ao método que está "interessado" no seu processamento, e que foi concebido para o efeito
- Agrupamento e diferenciação de diferentes tipos de erros

12

O que é uma exceção

Excepção

13

- Uma excepção é um evento pouco frequente, normalmente associado a um erro ou situação anormal, que é detectado por hardware ou software e necessita de algum tipo de processamento especial
 - Exemplos: divisão por zero, fim de ficheiro não esperado, dados inválidos, abertura de um ficheiro que não existe, falta de memória, acesso a um vector para além dos seus limites, etc.
- A ocorrência de uma excepção altera o fluxo de controlo normal do programa

Excepção

14

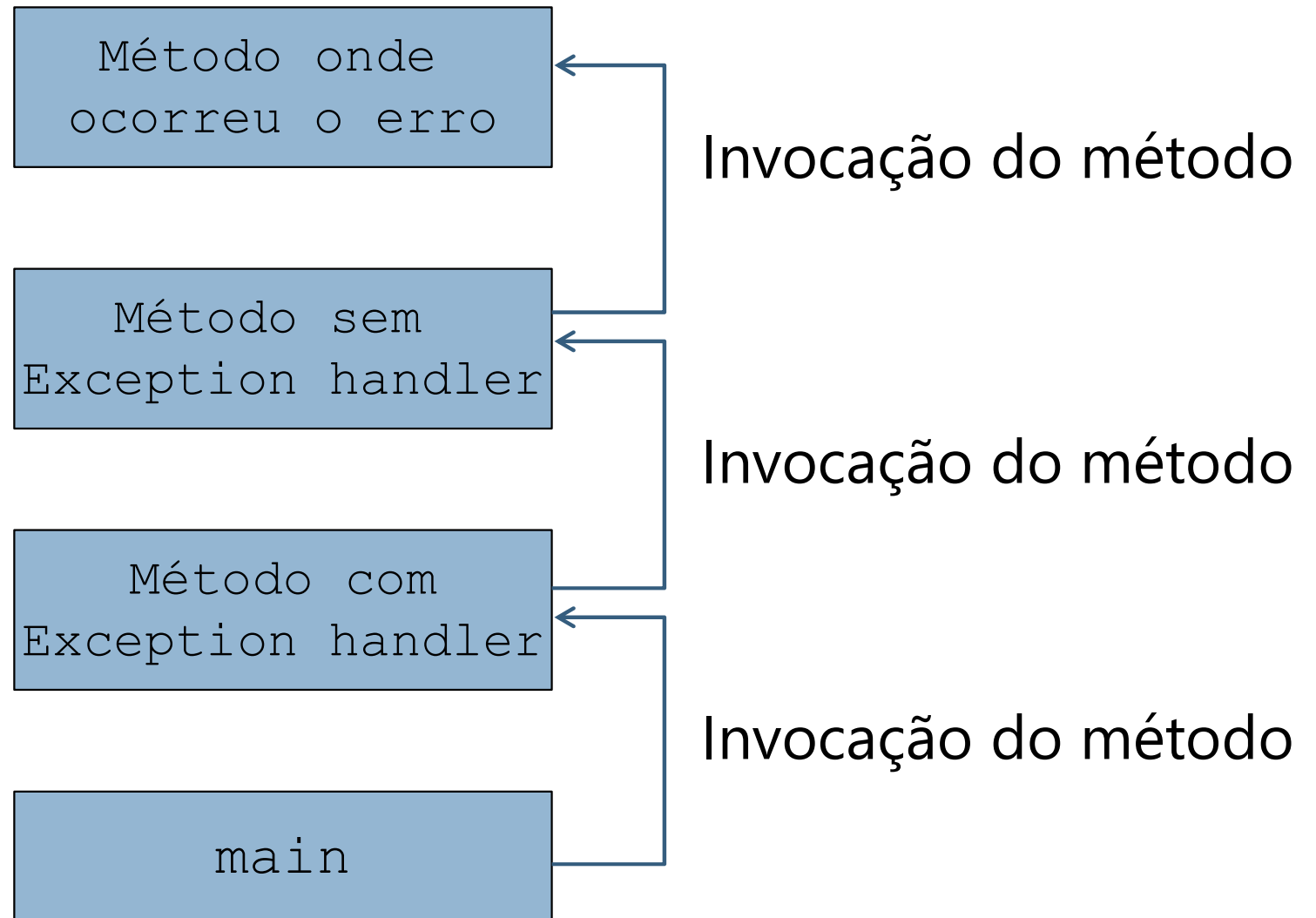
- Uma excepção em Java é um objecto com variáveis e métodos associados
 - Permite avaliar e processar a situação em causa
- Operações associadas a excepções
 - Criação da excepção
 - Lançamento da excepção
 - Programa notifica a ocorrência de um erro
 - Captura e processamento da excepção
 - Programa direcciona o controlo para código de análise e processamento de erros

15

Gestão de uma exceção

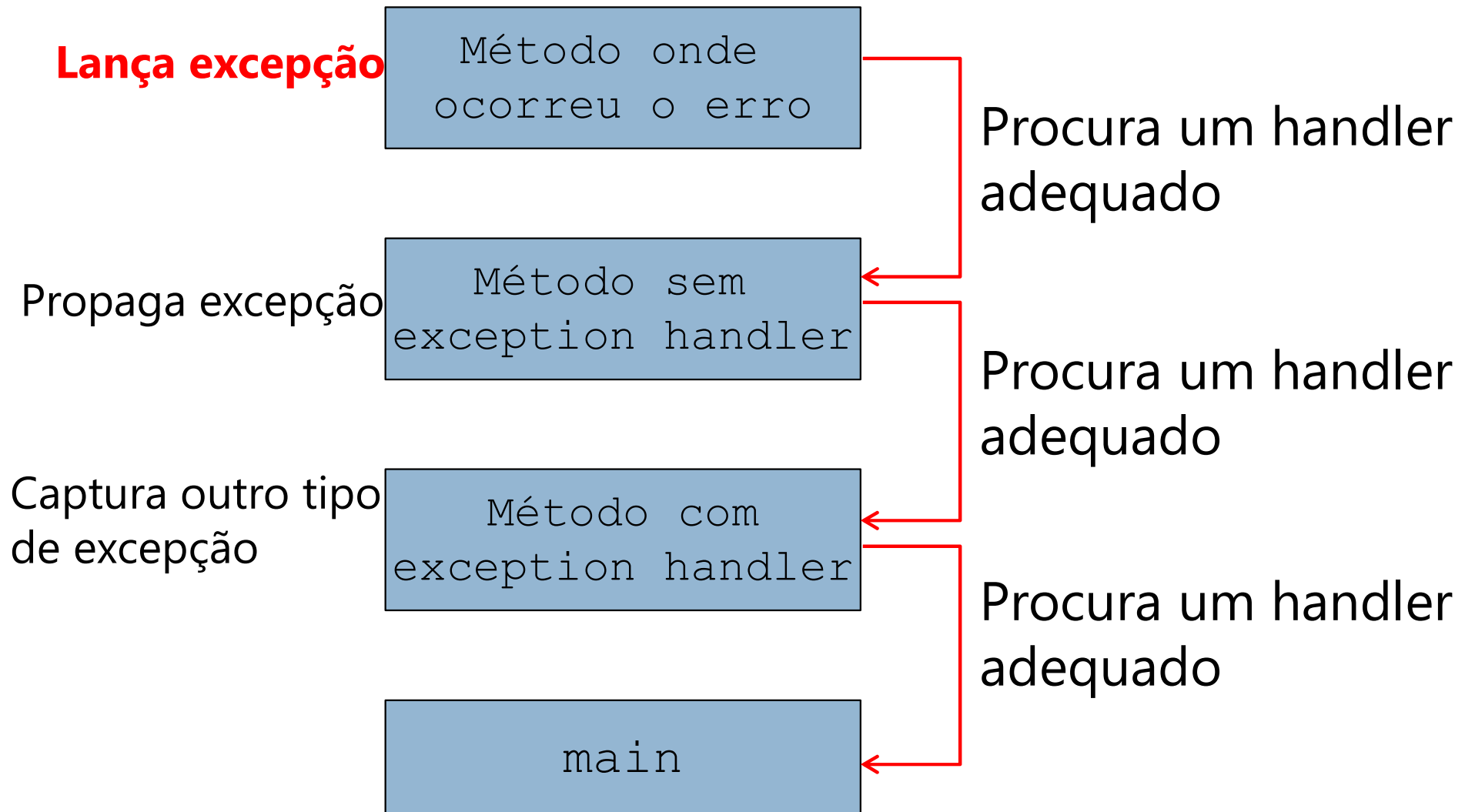
A pilha de chamadas

16



Procurando o gestor da exceção

17



Lançamento e processamento de exceções

18

- Lançamento de uma exceção em Java
 - Ocorrência de uma exceção
 - É criado um objecto de exceção que é passado para o sistema *runtime*
 - Exemplo

```
throw new NullPointerException();
```
 - Se o método que lançou a exceção não a capturar, o método termina
- Após o lançamento da exceção a execução continua no gestor de exceções
- Processamento da exceção lançada
 - Execução de código específico para o efeito
 - O objecto de exceção contém informação sobre o erro, incluindo o seu tipo e estado do programa quando este ocorreu

Exemplo de lançamento de uma exceção

19

```
public class OtherAccount {  
    ...  
  
    public void withdraw(double amount) throws NegativeAmountException,  
                                                SaldoInsuficienteException{  
        if (amount < 0)  
            throw new NegativeAmountException();  
        if (amount > balance)  
            throw new SaldoInsuficienteException();  
  
        balance -= amount;  
    }  
}
```

Controlo e processamento de exceções

20

```
try {  
    // Código que pode lançar exceções  
    // seja com uma instrução throw ou  
    // a invocação de um método que pode lançar exceções  
} catch ( <ExceptionType1> <Obj1> ) {  
    // Trata exceções do tipo <ExceptionType1>  
} catch ( <ExceptionType2> <Obj2> ) {  
    // Trata exceções do tipo <ExceptionType2>  
} . . .  
} finally {  
    // Código a ser executado no final do bloco try  
}
```

Recapitulando

21

- Executam-se as instruções que estão no bloco `try`
- Se não ocorrer nenhuma exceção, as cláusulas que constam na lista de `catch` não são executadas
- Se ocorrer uma exceção com um dos tipos indicados em `catch`, então a execução vai para a respectiva cláusula `catch`
- Se ocorrer uma exceção de outro tipo, essa exceção é lançada até que seja capturada, eventualmente por outro bloco `try`
 - No limite, será detectada pelo gestor de exceções por defeito do próprio sistema

Exemplo de um bloco try/catch

22

```
try {  
    String filename = ... ;  
    FileReader reader = new FileReader(filename);  
    Scanner in = new Scanner(reader);  
    String input = in.next();  
    int value = Integer.parseInt(input);  
    ...  
}
```

```
catch (NumberFormatException exception) {  
    System.out.println("Input was not a number");  
}
```

```
catch (IOException exception) {  
    exception.printStackTrace();  
}
```

Gestão de exceções com cláusula `finally`

23

- Quando uma exceção termina um método, há o risco de ser omitida a execução de operações importantes
- Exemplo
 - A instrução `reader.close()` deve ser executada mesmo que seja lançada uma exceção. Nestas situações, deve-se utilizar uma cláusula `finally`

```
. . .  
reader = new FileReader("ficheiroTeste.txt");  
Scanner in = new Scanner(reader);  
readData(in);  
reader.close(); // pode não chegar aqui !!!
```

Execução da cláusula `finally`

24

- Uma cláusula `finally` no bloco `try` é sempre executada, de acordo com um dos seguintes cenários:
 - depois da última instrução do bloco `try`
 - depois da última instrução da cláusula `catch` que capturou a exceção
 - quando no bloco `try` é lançada uma exceção e não é capturada por nenhuma cláusula `catch`
 - ...

Propagação de erros na pilha de chamada

25

- Quando é lançada uma exceção, o sistema de execução do Java faz uma pesquisa em sentido inverso na pilha de chamadas com o objectivo de encontrar métodos ou blocos que estejam associados à gestão ou tratamento dessa exceção

Excepção não detectada pelo código

26

- O gestor de exceções do java por omissão
 - Escreve a descrição da exceção
 - Escreve o traço da pilha, indicando a hierarquia de métodos onde ocorreu a exceção
 - Termina o programa

```
Formato de entrada: operador numero
Q para terminar o programa
Resultado = 0.0
+ palavra
Exception in thread "main" java.util.InputMismatchException
    at java.util.Scanner.throwFor(Scanner.java:909)
    at java.util.Scanner.next(Scanner.java:1530)
    at java.util.Scanner.nextDouble(Scanner.java:2456)
    at Main.doCalculation(Main.java:51)
    at Main.main(Main.java:18)
```

Implementação de excepções

27

- O Java permite criar excepções:
 - Criar uma classe que especializa a classe `Exception` ou `RuntimeException`
 - Redesenhar a classe criada, adicionando variáveis de classe e construtores
- Na nova classe podemos
 - Invocar um dos métodos standards `e.getMessage()` e `e.printStackTrace()`
 - Escrever uma mensagem com informação própria da classe
 - A invocação da classe de excepção segue os mesmos princípios de outras classes de excepção
- Note-se que pode ser conveniente tomar alguma acção correctiva em função da excepção gerada

Alguns conselhos

28

- Devemos usar o mecanismo das exceções com ponderação
- A ordem das cláusulas `catch` é importante
 - É executada a primeira cláusula que coincide com a exceção
 - Colocar a mais específica em primeiro lugar

```
catch (DivideByZeroException exception) {  
    . . .  
}
```

```
catch (Exception exception) {  
    . . .  
}
```

mais específica



Alguns conselhos

29

- Em geral, o código de lançamento e de captura da exceção estão em métodos distintos

```
public void methodB() {  
  
    try {  
        ... methodA() ...  
    }  
    catch ( MyException exception ) {  
        // Tratar exceção  
        ...  
    }  
    ...  
}  
  
public void methodA() throws MyException {  
    . . .  
    throw new MyException();  
    . . .  
}
```

30

Tipos de exceções

Tipos de exceções

31

- Exceções verificadas
 - O compilador verifica se são ignoradas pelo programa
 - Se forem ignoradas, origina um erro de compilação
 - Associadas a circunstâncias externas que o programador não pode prever
 - A maior parte destas exceções estão relacionadas com operações de entrada/saída
- Exceções não verificadas
 - Não são sujeitas à verificação de gestão de exceções por parte do compilador
 - São uma extensão da classe `RuntimeException` ou `Error`. Em princípio, resultam de erros de programação

Excepções verificadas

32

- As classes podem não ter capacidade de responder a todas as situações inesperadas
 - Exemplo
 - `Scanner.nextInt()` lança a exceção não verificada `InputMismatchException` quando o utilizador, incorretamente, fornece um valor não inteiro
- Devemos considerar exceções verificadas sobretudo quando se está a lidar com ficheiros
 - Exemplo
 - Na leitura de um ficheiro com a classe `Scanner` o construtor de `FileReader` pode lançar uma exceção `FileNotFoundException` se o ficheiro não existir !

```
. . .  
String fileName = "ficheiroTeste.txt";  
FileReader reader = new FileReader(fileName);  
Scanner in = new Scanner(reader);
```


Exceções verificadas

33

- Duas soluções possíveis:
 - Capturar a exceção
 - Propagar a exceção
 - Usar um especificador de lançamento para que o método possa lançar uma exceção verificada

```
public void read(String filename) throws IOException, ClassNotFoundException{  
    ...  
}  
  
public void read(String filename) {  
    try {  
        reader = new FileReader(filename);  
        Scanner in = new Scanner(reader);  
        ...  
    }  
    catch (ClassNotFoundException exception) { ... }  
    catch (IOException exception) { ... }  
}
```

Hierarquia de classes de exceções

34

Throwable (java.lang)

- **Error**

- LinkageError, ...
- VirtualMachineError, ...

- **Exception**

- ClassNotFoundException
- CloneNotSupportedException
- IllegalAccessException
- **IOException**
 - EOFException
 - FileNotFoundException
- ...

- **RuntimeException**

- ArithmeticException
- IllegalArgumentException
- IndexOutOfBoundsException
- NullPointerException
- ...
- ...

checked

unchecked

○ **Error**

- Para gerir erros ocorridos no ambiente de execução, fora do controlo dos utilizadores do programa
 - Por exemplo, erros de memória ou falha do disco rígido

○ **Exception**

- Para situações que os utilizadores podem gerir
 - Por exemplo, divisão por zero ou acesso fora dos limites de vectores

Agrupamento e diferenciação de erros

35

- Organizar o tratamento de erros segundo a hierarquia de classes de excepções
- Exemplo: `java.io.IOException` e descendentes
 - `IOException` é a classe mais genérica, associada aos erros que possam ocorrer relacionados com operações I/O
 - As classes descendentes representam erros mais específicos, como é o caso de `FileNotFoundException`
- Um método pode detectar uma excepção baseada no seu tipo ou em alguma das superclasses respectivas
 - Exemplo: `IOException` na cláusula `catch`, captura todas as excepções de I/O, incluindo `FileNotFoundException`, `EOFException`

36

Uma calculadora simples



Calculadora simples

37

- 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

38

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

39



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

/ 0.000000001

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

40

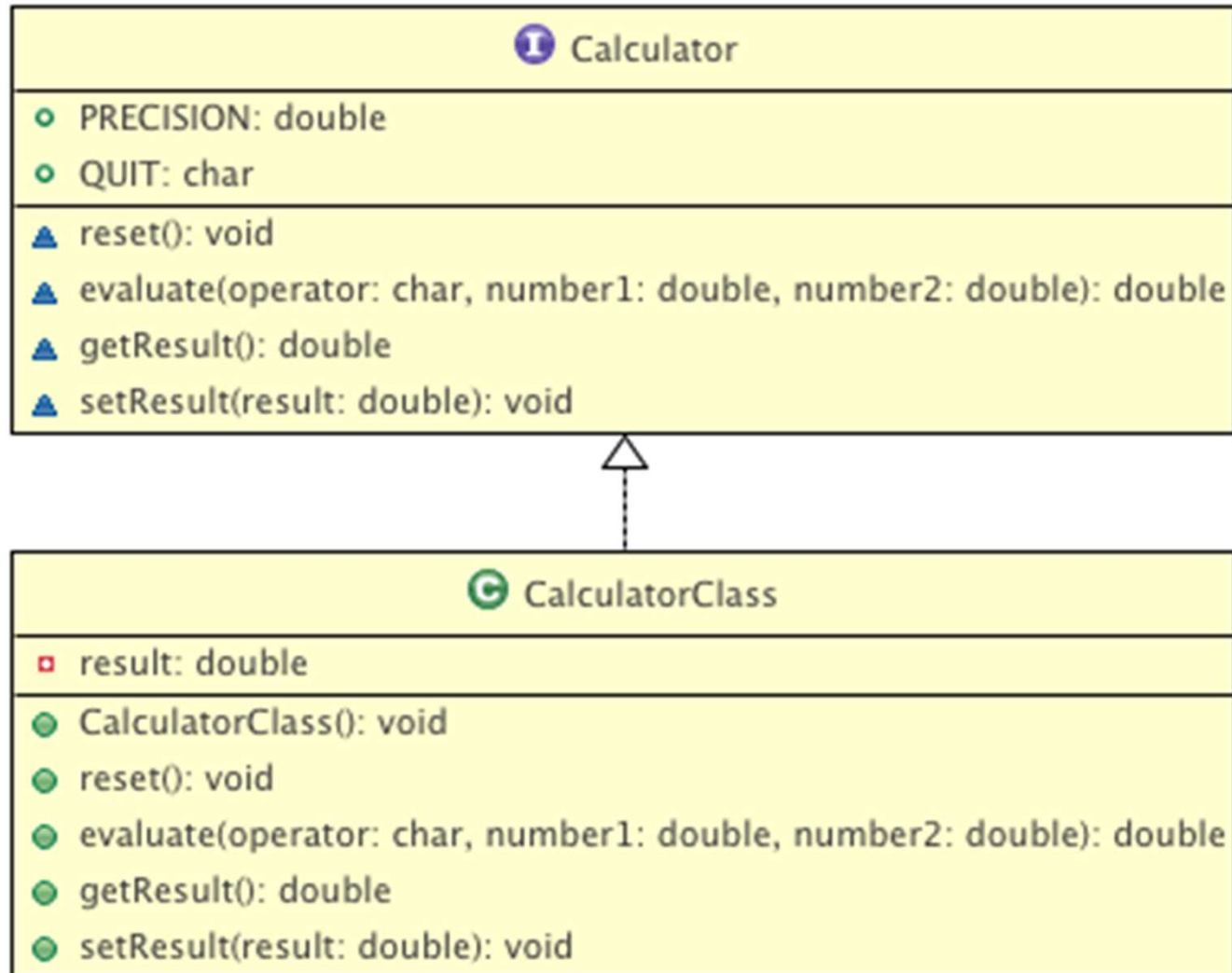


Diagrama de classes

41

