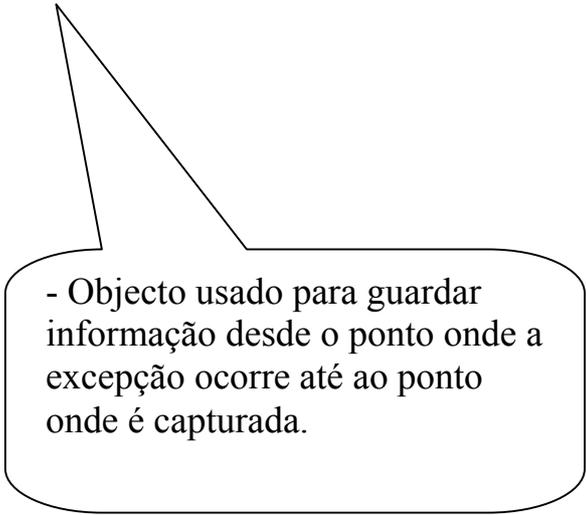


7 – Excepções

Quando um programa viola as restrições semânticas da linguagem, a JVM assinala um erro ao programa, sob a forma de excepção.

Uma excepção é um erro recuperável

- O controlo da execução do programa é transferido do ponto onde ocorre a excepção para um ponto que pode ser especificado pelo utilizador.
- Uma excepção diz-se lançada (thrown) no ponto onde ocorre e diz-se capturada (caught) no ponto para onde o controlo de execução é transferido
- Cada excepção é representada por uma instância da classe Throwable ou de uma das suas subclasses.



- Objecto usado para guardar informação desde o ponto onde a excepção ocorre até ao ponto onde é capturada.

O tratamento de uma excepção é definido pela cláusula **catch** de uma instrução

try:

try {

- bloco de instruções, nas quais queremos ter a possibilidade de detectar a ocorrência de erros recuperáveis.

} catch (<classe da excepção> <instância da excepção gerada>) {

- instruções para tratamento da excepção considerada

[... outras cláusulas catch]*

} [finally {

*- instruções que serão executadas, **ocorra ou não** uma excepção no bloco try*

}]*

*[opcional]

Uma excepção pode ser lançada porque:

1) A JVM detecta uma violação da semântica da linguagem

Exemplo - divisão por zero

- o limite de uma dado recurso foi ultrapassado

2) Foi executada uma instrução **throw**

throw – geração explícita de uma excepção definida ou não pelo utilizador

throw new < construtor da classe exception ou de uma sua subclasse >

Existem dois tipos de excepções:

A - Excepções verificáveis pelo compilador

O compilador verifica se o programa trata as excepções que poderão ocorrer no código.

Para cada excepção verificável, o método onde essa excepção pode ocorrer deve:

- prever o tratamento da excepção (try – catch)

ou

- lançar a excepção, através da cláusula throws, para que seja tratada no método invocador ou noutra mais externo

Ex.

```
public void metodoExemplo () throws <nome da classe da excepção> {  
    ....  
    public static void main (String [] args ) throws IOException {
```

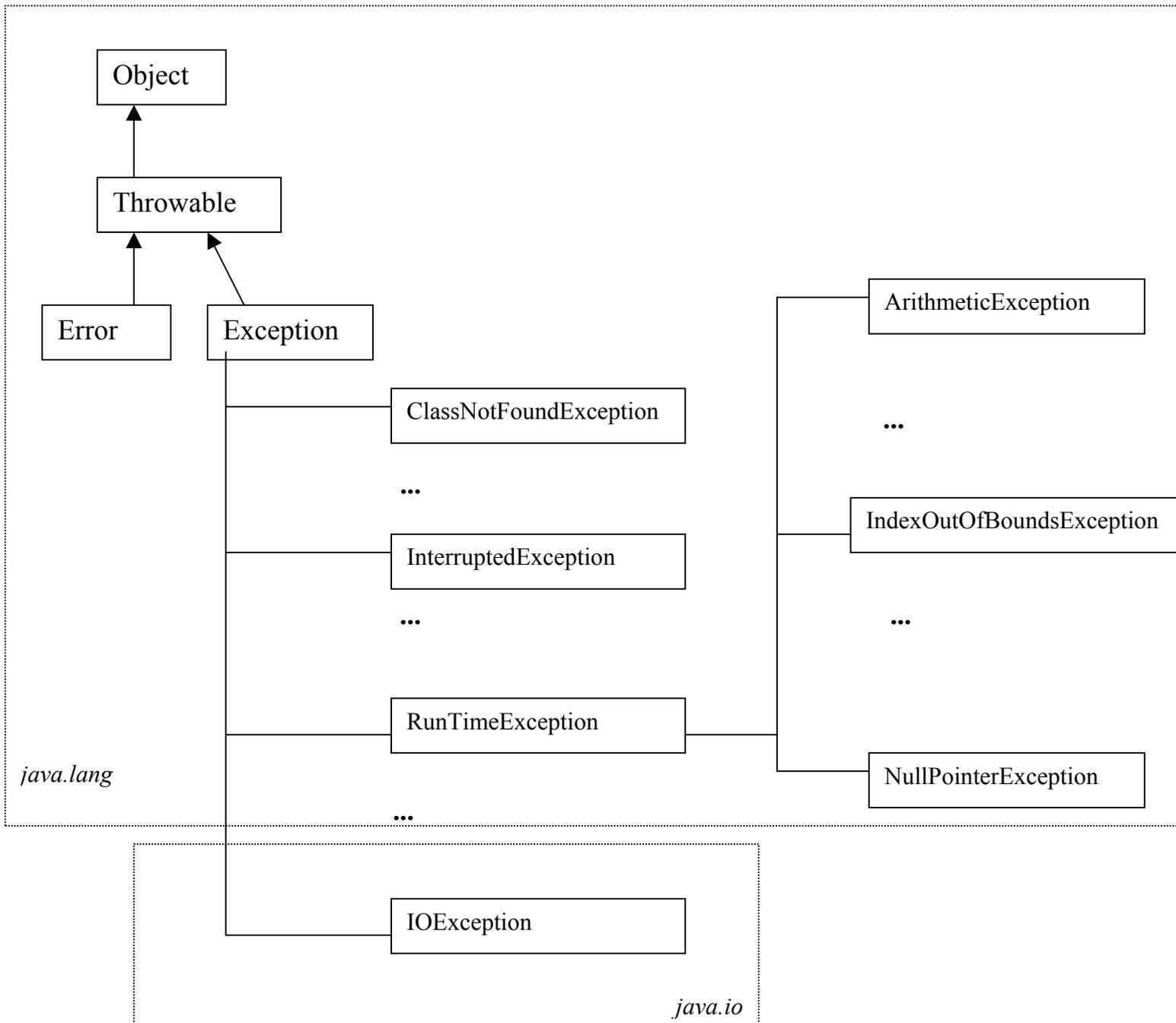
B - Excepções não verificáveis pelo compilador

são objectos das classes `RuntimeException`, `Error` e respectivas subclasses

`RuntimeException` – excepções cuja ocorrência é difícil de ser verificável pelo programador

`Error` – erros não recuperáveis

Hierarquia das classes de excepção em Java:



(ver Mário Martins pág. 234)

Exemplo 1: Definir uma excepção

Subclasse de Exception

```
public class Bomba extends Exception {  
  
    public Bomba () {  
  
        super ();  
  
    }  
  
    public Bomba ( String s ) {  
  
        super ( s );  
  
    }  
  
}  
  
public class Teste1 {  
  
    public static void explode () throws Bomba {  
  
        throw new Bomba ();  
  
    }  
  
    public static void main (String []args) {  
  
        try {  
  
            explode ();  
  
        }  
  
        catch ( RuntimeException e ) {  
  
            System.out.println ("RuntimeExplode"+ e.getMessage());  
  
        }  
  
    }  
  
}
```

```
catch ( Bomba b ) {  
    System.out.println ( "Bomba" );  
}  
  
} // main  
} // Teste1
```

Output do programa:

Bomba

A exceção que ocorre não é compatível com a classe RuntimeException mas é compatível com a classe Bomba.

Exemplo 2: cláusula finally

```
public class Teste2 {  
    public static void explode () {  
        throw new NullPointerException();  
    }  
    public static void main (String []args) {  
        try {  
            explode();  
        }  
        catch ( Bomba e ) {  
            System.out.println ("Bomba");  
        }  
        finally {  
            System.out.println ("Excepção não capturada");  
        }  
    }  
}
```

Subclasse de RuntimeException, exceção não verificável, não é necessário lançar a exceção.

```
}// main
```

```
}//Teste2
```

Output do programa:

```
Excepção não capturada  
java.lang.NullPointerException  
at Teste2.Explode (Teste2.java:7)  
at Teste.main ( Teste2.java:11)
```

Herança e cláusula throws

- Um método que sobrepõe (“overrides”) outro não pode declarar lançar mais excepções do que o método que é sobreposto.

Exemplo:

```
public class C1 {  
    public void m2() throws Exception {  
        System.out.println ("Método 2 da classe C1 ");  
    }  
}  
public class C2 extends C1 {  
    public void m2() * throws InterruptedException,  
                        ClassNotFoundException {  
        System.out.println ("Método 2 da classe C2 ");  
    }  
}
```

Válido

* O método m2 da classe C2 sobrepõe m2 da classe C1

Herança e cláusula throws (cont ...)

- Cada excepção declarada em m2 da classe C2 tem que ser do mesmo tipo (classe) de uma excepção lançada em m2 de C1 ou de um seu subtipo (subclasse).
- No método m2 da classe C1 tem que ser lançada a mesma excepção que em m2 de C2, ou uma excepção de uma sua superclasse.

Exemplo 3: Qual o output do seguinte programa?

```
package excepcoes;
```

```
public class TesteException extends Exception {  
  
    public TesteException () {  
  
        super ();  
  
    }  
  
    public TesteException ( String s ) {  
  
        super ( s );  
  
    }  
  
}
```

```
public class Teste {  
    public static int atirador (String s)  
        throws TesteException {  
        try {  
            if ( s.equals ("dividir") ) {  
                int i= 0 ;  
                return i/i;  
            }  
            if ( s.equals( "null") {  
                s = null;  
                return s.length();  
            }  
            if ( s.equals ("teste") ) {  
                throw new TesteException ("Teste");  
                return 0;  
            }  
        }  
    }  
} //try  
    finally {  
        System.out.println ("*****");  
        System.out.println (" [Atirador(" + s +  
            ") executado]" );  
    }  
} //atirador
```

TesteException é
uma excepção
verificável

cria uma instância
de
TesteException

sempre executado, quer
ocorra uma excepção
quer não

```
public static void main (String []args) {  
  
    String [] txt = new String [4];  
  
    txt [0] = "dividir" ;  
  
    txt [1] = "null" ;  
  
    txt [2] = "não" ;  
  
    txt [3] = "teste" ;  
  
    for ( int i = 0; i< txt.length ; i++ ){  
  
        try {  
  
            atirador (txt [i] );  
  
            System.out.println ( "Teste(" + txt[i] + ")" +  
  
                "não lançou uma excepção");  
  
        }  
  
        catch (Exception e){  
  
            System.out.println ( "Teste(" + txt[i] +  
  
                ") lançou uma " + e.getClass() +  
  
                "\n com a mensagem " + e.getMessage() );  
  
        }  
    }  
} // main  
  
} // Teste
```

- O método main invoca o método atirador 4 vezes
- Cada excepção lançada pelo método atirador será capturada pelo main.

Output

```
****
[Atirador (dividir) executado]
Teste (dividir) lançou uma class java.lang.ArithmeticException
com a mensagem / by zero
****
[Atirador (null) executado]
Teste (null) lançou uma class java.lang.NullPointerException
com a mensagem null
****
[Atirador (não) executado]
Teste (não) não lançou uma excepção
****
[Atirador (teste) executado]
Teste (teste) lançou uma class excepcoes.TesteException
com a mensagem teste
```