

Java Exceções

Programação III
José Luis Oliveira; Carlos Costa

Problema!

- Nem todos os erros são detectados na compilação.
- Estes são, geralmente, os que são mais menosprezados pelos programadores:
 - Compila sem erros → Funciona!!!
- Tratamento Clássico:
 - Se sabemos à partida que pode surgir uma situação de erro em determinada passagem podemos tratá-la nesse contexto (if...)

```
if ((i = doTheJob()) != -1) {  
    /* tratamento de erro */  
}
```

Excepção

- Uma excepção é gerada por algo imprevisto que não é possível controlar.
- Utilização de Excepções:

- Tratamento do erro no contexto local

```
try {  
    /* O que se pretende fazer */  
}  
catch (Errortype a) {  
}
```

- Delegação do erro - gerar um objecto excepção (throw) no qual se delega esse tratamento.

```
if (t == null)  
    throw new NullPointerException();  
// throw new NullPointerException("t null");
```

Controlo de Excepção

- A manipulação de exceções é feita através de um bloco especial - try.

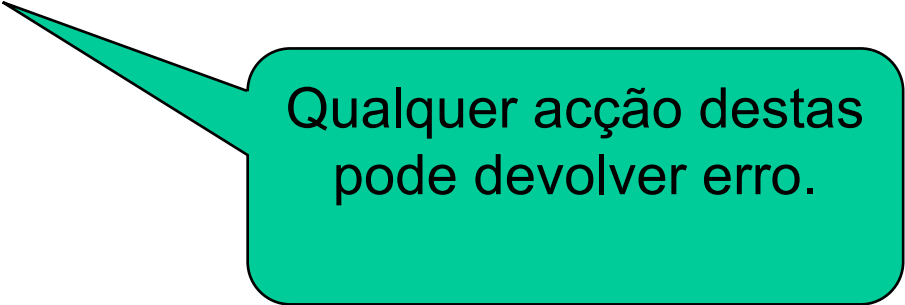
```
try {  
    // Code that might generate exceptions Type1,  
    // Type2 or Type3  
} catch(Type1 id1) {  
    // Handle exceptions of Type1  
} catch(Type2 id2) {  
    // Handle exceptions of Type2  
} catch(Type3 id3) {  
    // Handle exceptions of Type3  
} finally {  
    // Executada independentemente de haver ou não  
    // uma excepção  
}
```

Vantagens das Exceções

- Separação clara entre o código regular e o código de tratamento de erros
- Propagação dos erros em chamadas sucessivas
- Agrupamento de erros por tipos

Separação de código - exemplo (1)

```
readFile {  
    open the file;  
    determine its size;  
    allocate that much memory;  
    read the file into memory;  
    close the file;  
}
```



Qualquer acção destas
pode devolver erro.

Separação de código - exemplo (2)

```
errorCodeType readFile {  
    initialize errorCode = 0;  
    open the file;  
    if (theFileIsOpen) {  
        determine the length of the file;  
        if (gotTheFileLength) {  
            allocate that much memory;  
            if (gotEnoughMemory) {  
                read the file into memory;  
                if (readFailed) { errorCode = -1; }  
                } else { errorCode = -2; }  
            } else { errorCode = -3; }  
        close the file;  
        if (theFileDidntClose && errorCode == 0) {  
            errorCode = -4;  
        } else { errorCode = errorCode and -4; }  
    } else { errorCode = -5; }  
    return errorCode;  
}
```

Sem Exceções

Separação de código - exemplo (3)

```
readFile {  
    try {  
        open the file;  
        determine its size;  
        allocate that much memory;  
        read the file into memory;  
        close the file;  
    } catch (fileOpenFailed) {  
        doSomething;  
    } catch (sizeDeterminationFailed) {  
        doSomething;  
    } catch (memoryAllocationFailed) {  
        doSomething;  
    } catch (readFailed) {  
        doSomething;  
    } catch (fileCloseFailed) {  
        doSomething;  
    }  
}
```

Com Exceções

Propagação dos erros (1)

```
method1 {  
    call method2;  
}  
method2 {  
    call method3;  
}  
method3 {  
    call readFile;  
}
```

Solução sem
Exceções

```
method1 {  
    errorCodeType error;  
    error = call method2;  
    if (error)  
        doErrorProcessing;  
    else  
        proceed;  
}  
  
errorCodeType method2 {  
    errorCodeType error;  
    error = call method3;  
    if (error)  
        return error;  
    else  
        proceed;  
}  
  
errorCodeType method3 {  
    errorCodeType error;  
    error = call readFile;  
    if (error)  
        return error;  
    else  
        proceed;  
}
```

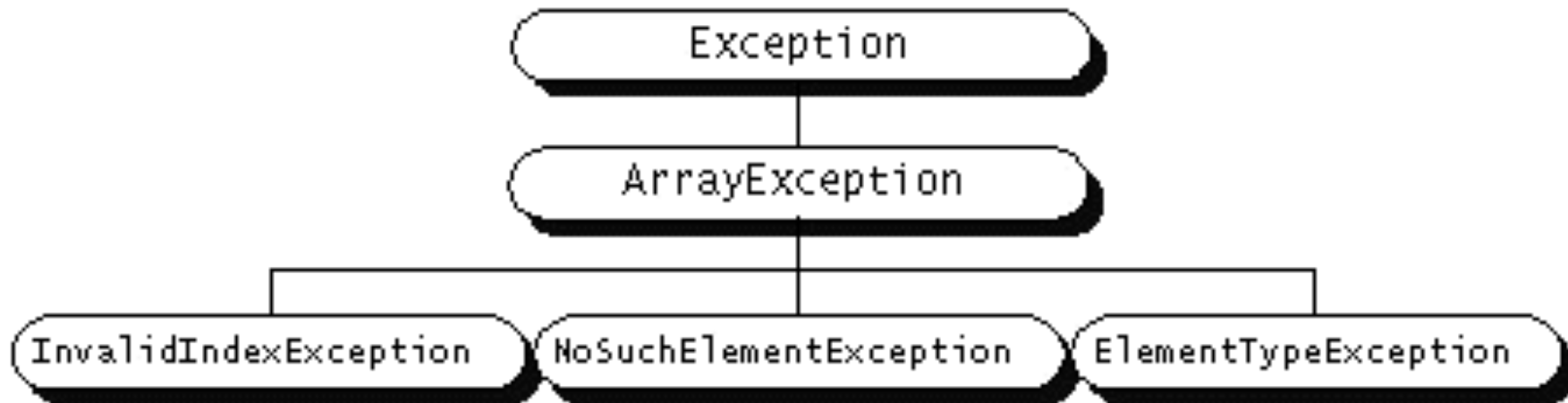
Propagação dos erros (2)

```
method1 {  
    call method2;  
}  
method2 {  
    call method3;  
}  
method3 {  
    call readFile;  
}
```

Solução com
Exceções

```
method1 {  
    try {  
        call method2;  
    } catch (exception) {  
        doErrorProcessing;  
    }  
}  
  
method2 throws exception {  
    call method3;  
}  
  
method3 throws exception {  
    call readFile;  
}
```

Agrupamento de erros por tipos



```
catch (InvalidIndexException e) {  
    . . .  
}
```

Diferenciação

```
catch (ArrayException e) {  
    . . .  
}
```

Agrupamento

Exceções - Hierarquia de Classes

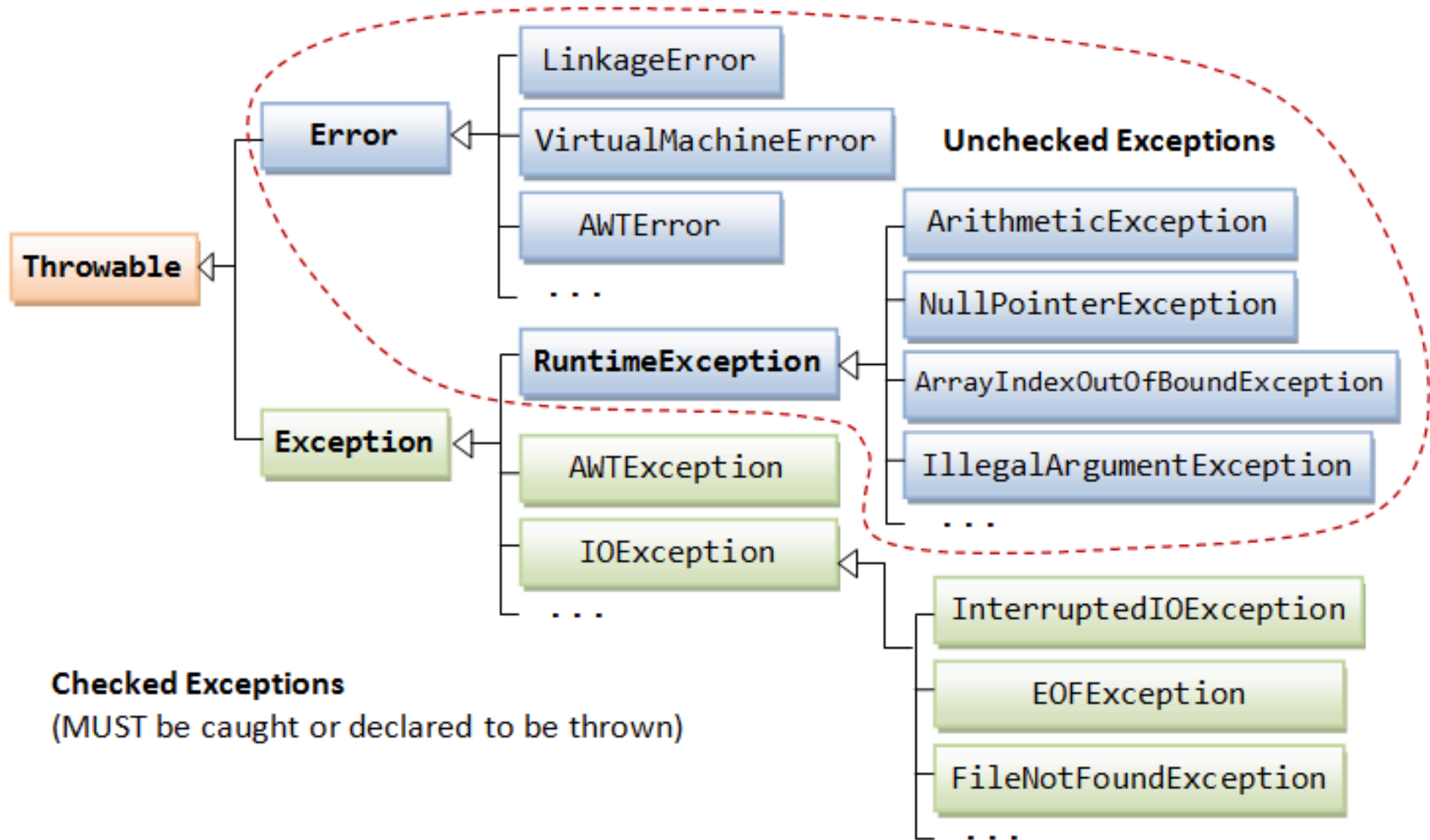
```
try {  
    ...  
}  
catch (NullPointerException e){  
    ...  
}  
catch (IndexOutOfBoundsException e){  
    ...  
}  
catch (ArithmeticException e){  
    ...  
}  
catch (Exception e){  
    ...  
}
```

**A ordem dos "catch"
é importante**

```
try {  
    ...  
}  
catch (Exception e){  
    ...  
}  
catch (NullPointerException e){  
    ...  
}  
catch (IndexOutOfBoundsException e){  
    ...  
}  
catch (ArithmeticException e){  
    ...  
}
```

Má Solução

Exceções - Hierarquia de Classes



Classe Exception

- A classe Exception é derivada da classe Throwable
- Podemos usar a classe base java.lang.Exception para capturar qualquer exceção
- Podemos regenerar nova exceção de forma a ser tratada num nível superior

```
catch(Exception e) {  
    System.out.println("caught an exception");  
}
```

```
catch(Exception e) {  
    System.out.println("Exception was thrown");  
    throw e;  
}
```

Especificação da Excepção

- Quando desenhamos métodos que possam gerar exceções devemos assinalá-las explicitamente

```
void f() throws TooBigException,  
              TooSmallException,  
              DivByZeroException {  
  
    //...  
  
}
```

Criar Novas Excepções

- Podemos usar o mecanismo de herança para personalizar algumas excepções

```
class MyException extends Exception {  
    // interface base  
    public MyException() {}  
    public MyException(String msg) {  
        super(msg) ;  
    }  
    // podemos acrescentar construtores e dados  
}
```


Exemplo

```
public class Rethrowing {  
    public static void f() throws Exception {  
        System.out.println("exception in f()");  
        throw new Exception("thrown from f()");  
    }  
    public static void g() throws Exception {  
        try { f();  
        } catch(Exception e) {  
            System.out.println(" exception in g()");  
            throw e;  
        }  
    }  
    public static void main(String[] args) {  
        try { g();  
        } catch(Exception e) {  
            System.out.println("Caught in main");  
        }  
    }  
}
```

Tipos de Exceções

- checked
 - Se invocarmos um método que gere uma *checked exception*, temos de indicar ao compilador como vamos resolvê-la:
 - 1) Resolver *try .. catch* ou
 - 2) Propagar *throw*
- unchecked
 - São erros de programação ou do sistema (podemos usar Asserções nestes casos)
 - são subclasses de `java.lang.RuntimeException` ou `java.lang.Error`

Boas Práticas

- Usar exceções apenas para condições excepcionais
 - Uma API bem desenhada não deve forçar o cliente a usar exceções para controlo de fluxo
 - Uma exceção não deve ser usada para um simples teste

X

```
try {  
    s.pop();  
} catch (EmptyStackException es) {...}
```

V

```
if (!s.empty()) s.pop();    // melhor!
```

Boas Práticas

- Usar preferencialmente exceções standards
 - `IllegalArgumentException`
 - - valor de parâmetros inapropriado
 - `IllegalStateException`
 - - Estado de objecto incorrecto
 - `NullPointerException`
 - `IndexOutOfBoundsException`
- Tratar sempre as exceções (ou delegá-las)

X `try {
 // .. código que pode causar exceções
} catch (Exception e) {}`