



Escola Superior de Tecnologia
e Gestão de Viseu



Curso Técnico Superior Profissional Desenvolvimento para a Web e Dispositivos Móveis

Unidade Curricular:
Técnicas Avançadas de Programação

11 - Exceções

- Mesmo quando o código está bem implementado existem certas circunstâncias excepcionais que podem comprometer a execução do programa:
- Podem ser dados incorretos
- Pode ser um fim de leitura prematuro de um ficheiro quando se estava à espera de ler mais dados



11 - Exceções

- Tentar contemplar todas as situações possíveis no seio do programa e tomar todas as decisões que se impõem para evitar qualquer tipo de erro pode, mesmo que o programador não omita nenhuma situação, revelar-se uma tarefa fastidiosa e o código ficar ilegível tal é a quantidade de instruções de tratamento de circunstâncias excepcionais
- O JAVA dispõe de mecanismos designados de gestores de exceções que permitem:
 - distinguir/dissociar a “detecção de uma anomalia” do “tratamento de uma anomalia”
 - separar a gestão das anomalias do restante código - código mais legível
- Uma exceção é uma rotura da sequência desencadeada por uma instrução throw que tem uma expressão do tipo classe

11 - Exceções

- **[A] Exemplo de uma exceção:** Class Ponto, construtor com 2 argumentos, 1 método escreve(). Imaginar que se pretende apenas coordenadas positivas e quando inseridas negativas se desencadeie/ lance uma exceção com a ajuda da instrução **throw**

- (1) Como desencadear / lançar uma exceção com **throw**

- A instrução **throw** deve ter o objeto que posteriormente servirá para identificar a exceção em causa

11 - Exceções

Criação da classe ErrConst derivada da classe Exception
Para já artificial, i.e., vazia

```
class ErrConst extends Exception{  
}
```

Para lançar a exceção, ao nível do construtor (onde quero ver se as coordenadas são positivas) escrevo então a instrução **throw** seguinte:

```
throw new ErrConst();
```

O construtor da classe Ponto escreve-se da seguinte maneira:

```
class Ponto {  
    public Ponto(int a, int b) throws ErrConst {  
        if ( (a<0) || (b<0) ) {  
            throw ErrConst();  
        }  
        this.a = a;  
        this.b = b;  
    }  
}
```

Para indicar ao método que está/ficou susceptível de lançar uma exceção

11 - Exceções

- Reescrevendo a classe Ponto completa:

```
class Ponto {  
  
    public Ponto( int a, int b ) throws ErrConst {  
        if ( (a<0) || (b<0) ) {  
            throw new ErrConst();  
        }  
        this.a = a;  
        this.b = b;  
    }  
  
    public void escreve( ){  
        System.out.println("Nova coordenada: ( " + a + " ; " + b + " )");  
    }  
  
    private int a, b;  
}  
  
class ErrConst extends Exception{  
}
```

11 - Exceções

- (2) Utilização de um gestor de exceções
- Para gerir as eventuais exceções do tipo ErrConst que a sua utilização pode desencadear / lançar é necessário criar a seguinte estrutura de blocos:
- um primeiro bloco particular designado de “bloco **try**” que vai reunir as instruções as quais se arriscam de desencadear / lançar tal exceção

```
try {  
    // instruções  
}
```

11 - Exceções

- (2) Utilização de um gestor de exceções
- fazer seguir este bloco da definição dos diferentes gestores de exceções (neste exemplo apenas é necessário um) - todos estes blocos desde o **try** são contíguos. Cada definição do gestor é precedida de um cabeçalho introduzindo a palavra reservada **catch** (é como se catch fosse o nome de um método de gestão de erros)

```
catch ( ErrConst e) {  
    System.out.println( "Erro ao nível da entrada de dados!" );  
    System.exit( -1 );  
}
```

Aqui o gestor de erro mostra uma mensagem e termina o programa

11 - Exceções

- (3) O programa completo:

```
class Ponto {  
  
    public Ponto( int a, int b ) throws ErrConst {  
        if ( (a<0) || (b<0) ) {  
            throw new ErrConst();  
        }  
        this.a = a;  
        this.b = b;  
    }  
  
    public void escreve( ){  
        System.out.println("Nova coordenada: ( " + a + " ; " + b + " )");  
    }  
  
    private int a, b;  
}  
  
class ErrConst extends Exception{  
}
```

```
public class ExemploExcecao1 {  
  
    public static void main(String args[ ] ) {  
        try{  
            Ponto p1 = new Ponto(1, 4);  
            p1.escreve();  
            p1 = new Ponto(-3, 5);  
            p1.escreve();  
        }  
        catch(ErrConst e){  
            System.out.println( "Erro entrada dados!" );  
            System.exit( -1 );  
        }  
    }  
}
```

Nova coordenada: (1 ; 4)
Erro entrada dados!

11 - Exceções

- (4) Primeiras propriedades da gestão de uma exceção
- O exemplo anterior:
 - apenas desencadeava e tratava / geria um só tipo de exceção
 - é possível gerir vários
 - o gestor da exceção não recebia nenhuma informação; recebia um objeto sem valor pelo que não o utilizava - é possível utilizar este objeto para comunicar uma informação ao gestor
 - para já não se vai explorar a class Exception
 - o gestor da exceção interrompia a execução do programa - é possível continuar a sua execução

11 - Exceções

- (4) Primeiras propriedades da gestão de uma exceção
- O gestor da exceção é independente dos métodos susceptíveis de desencadear a exceção, i.e., é possível atribuir uma gestão diferente de uma utilização a outra para a mesma classe exceção.
- No exemplo anterior:
 - (1) o utilizador via uma mensagem antes da interrupção do programa,
 - (2) mas o utilizador poderia não ler nada, ou
 - (3) o utilizador poderia ler um resultado por defeito

11 - Exceções

- (4) Primeiras propriedades da gestão de uma exceção
- Estrutura clássica dos blocos **try/catch**

```
void qualquerCoisa() {  
  
    try{ // aqui as exceções ErrConst são tidas em consideração  
        ...  
    }  
    catch ( ErrConst ) {  
        ...  
    }  
  
    ... // aqui as exceções ErrConst não são tidas em consideração  
  
    try{ // aqui as exceções ErrConst são tidas novamente em consideração  
        ...  
    }  
    catch ( ErrConst ) {  
        ...  
    }  
  
    ... // aqui as exceções ErrConst não são tidas novamente em consideração  
  
}
```

11 - Exceções

- **[B] Gestão de várias exceções**
- Considere-se o exemplo da classe Ponto para introduzir a análise de duas exceções
- O construtor vai lançar a mesma exceção
- O método desloca() vai agora garantir que não haverá coordenadas negativas, se tal for o caso é lançada a exceção (é criada a classe ErrDes):

```
class ErrDes extends Exception{  
}
```

```
public void desloca(int dx, int dy) throws ErrDes {  
    if ( (a+dx<0) || (b+dy<0) ) {  
        throw new ErrDes();  
    }  
    a += dx;  
    b += dy;  
}
```

11 - Exceções

- Exemplo

```
public class ExemploExcecao2 {  
  
    public static void main(String args[ ]) {  
        try{  
            Ponto p2 = new Ponto(1, 4);  
            p2.escreve();  
            p2.desloca(-3, 5);  
            p2 = new Ponto(-3, 5);  
            p2.escreve();  
        }  
        catch(ErrCont e){  
            System.out.println( "Erro entrada dados!" );  
            System.exit( -1 );  
        }  
        catch(ErrDes e){  
            System.out.println( "Erro em desloca!" );  
            System.exit( -1 );  
        }  
    }  
}
```

Nova coordenada: (1 ; 4)
Erro em desloca!

```
class Ponto {  
  
    public Ponto( int a, int b ) throws ErrConst {  
        if ( (a<0) || (b<0) ) {  
            throw new ErrConst();  
        }  
        this.a = a;  
        this.b = b;  
    }  
  
    public void escreve( ){  
        System.out.println("Nova coordenada: ( " + a + " ; " + b + " )");  
    }  
  
    public void desloca(int dx, int dy) throws ErrDes {  
        if ( (a+dx<0) || (b+dy<0) ) {  
            throw new ErrDes();  
        }  
        a += dx;  
        b += dy;  
    }  
  
    private int a, b;  
}  
  
class ErrConst extends Exception{  
}  
  
class ErrDes extends Exception{  
}
```

11 - Exceções

- **[C]** Transmissão de informação aos gestores de exceções
- É possível enviar informação ao gestor da exceção através:
 - do objeto que está na instrução throw
 - do construtor da classe exceção

11 - Exceções

- do objeto que está na instrução throw

```
class Ponto {  
  
    public Ponto( int a, int b ) throws ErrConst {  
        if ( (a<0) || (b<0) ) {  
            throw new ErrConst(a, b);  
        }  
        this.a = a;  
        this.b = b;  
    }  
  
    public void escreve() {  
        System.out.println("Nova coordenada: ( " + a + " ; " + b + " )");  
    }  
  
    private int a, b;  
}  
  
class ErrConst extends Exception{  
  
    public ErrConst ( int abs, int ord ) {  
        this.abs = abs;  
        this.ord = ord;  
    }  
  
    public abs, ord;  
}
```

```
public class ExemploExcecao3 {  
  
    public static void main(String args[ ] ) {  
        try{  
            Ponto p3 = new Ponto(1, 4);  
            p3.escreve();  
            p3 = new Ponto(-3, 5);  
            p3.escreve();  
        }  
        catch(ErrCont e){  
            System.out.println( "Erro entrada dados!" );  
            System.out.println( "Coordenadas introduzidas: ( " +  
e.abs + " ; " + e.ord + " )" );  
            System.exit( -1 );  
        }  
    }  
}
```

```
Nova coordenada: ( 1 ; 4 )  
Erro entrada dados!  
Coordenadas introduzidas: ( -3 ; 5 )
```


11 - Exceções

- do construtor da classe exceção - usar o método getMessage() herdado da classe pai Exception

```
class Ponto {  
  
    public Ponto( int a, int b ) throws ErrConst {  
        if ( (a<0) || (b<0) ) {  
            throw new ErrConst("Erro entrada dados " + a + " ; " + b );  
        }  
        this.a = a;  
        this.b = b;  
    }  
  
    public void escreve( ){  
        System.out.println("Nova coordenada: ( " + a + " ; " + b + " )");  
    }  
  
    private int a, b;  
}  
  
class ErrConst extends Exception{  
  
    public ErrConst ( String mensagem ){  
        super (mensagem);  
    }  
  
}
```

```
public class ExemploExcecao4 {  
  
    public static void main(String args[ ] ) {  
        try{  
            Ponto p4 = new Ponto(1, 4);  
            p4.escreve();  
            p4 = new Ponto(-3, 5);  
            p4.escreve();  
        }  
        catch(ErrConst e){  
            System.out.println( "Erro entrada dados!" );  
            System.out.println( e.getMessage( ) );  
            System.exit( -1 );  
        }  
    }  
}
```

```
Nova coordenada: (1 ; 4)  
Erro entrada dados!  
Erro entrada dados -3 ; 5
```

11 - Exceções

- **[D]** O mecanismo da gestão das exceções

- (1) Fluxo de execução

Nos exemplos anteriores o gestor de exceção terminava a execução do programa (método `System.exit`)

Tal não é obrigatório

A execução pode continuar

```
public class ExemploExcecao5 {  
  
    public static void main(String args[] ) {  
        System.out.println( "Estou inicio do bloco try" );  
  
        try{  
            Ponto p5 = new Ponto(1, 4);  
            p5.escreve();  
            p5.desloca(-3, 5);  
            p5.escreve();  
        }  
        catch(ErrConst e){  
            System.out.println( "Erro entrada dados!" );  
        }  
        catch(ErrDes e){  
            System.out.println( "Erro deslocamento!" );  
        }  
  
        System.out.println( "Estou no fim do bloco try" );  
  
    }  
}
```

Estou inicio do bloco try
Nova coordenada: (1 ; 4)
Erro deslocamento!
Estou no fim do bloco try

11 - Exceções

- Fluxo de execução

- Um gestor de exceções pode ter uma instrução **return**
- A instrução **return** provoca a saída do método

```
int metodo( ){  
    ...  
    try{  
        ...  
    }  
    catch ( Exception X e ){  
        ...  
        return 0;  
    }  
    ...  
}
```

X é uma classe derivada da classe Exception

```
int metodo( ){  
    int n = 0  
    try{  
        float var;  
    }  
    catch ( Exception X e ){  
        // aqui não há acesso à variável var (está  
        // num bloco distinto deste)  
        // mas há acesso à variável n (está  
        // num bloco global)  
    }  
    ...  
}
```

11 - Exceções

- Escolha do gestor de exceção
- Quando uma exceção é lançada num bloco **try** procura-se pelos diferentes gestores associados aquele que corresponde ao objeto referido em throw
- A procura é feita pela ordem pelos quais os gestores aparecem
- É selecionado o primeiro que é do tipo exato do objeto ou de um tipo de base (polimorfismo)
- Esta possibilidade pode ser explorada para reagrupar várias exceções que se pretendem tratar de uma maneira mais fina

11 - Exceções

- Escolha do gestor de exceção
- Exemplo: suponha-se que as exceções ErrConst e ErrDes são derivadas da mesma classe ErrPonto

```
class ErrPonto extends Exception { ... }  
  
class ErrConst extends ErrPonto { ... }  
  
class ErrDes extends ErrPonto{ ... }
```

11 - Exceções

- Escolha do gestor de exceção
- Considerando um método xpto() qualquer que desencadeie as exceções do tipo ErrPonto e ErrDes

```
void xpto( ){  
    ...  
    throw ErrConst;  
    ...  
    throw ErrDes;  
    ...  
}
```

11 - Exceções

- Escolha do gestor de exceção

- Um programa que utiliza o método `xpto()` pode gerir as exceções que ele é susceptível de desencadear das seguintes maneiras:

```
try {  
    ... // assumir que é usado xpto()  
}  
catch ( ErrPonto e) {  
    ... // tratam-se tanto as exceções do tipo ErrPonto e como as do tipo ErrDes  
}
```

11 - Exceções

- Escolha do gestor de exceção
- Um programa que utiliza o método `xpto()` pode gerir as exceções que ele é susceptível de desencadear das seguintes maneiras:

```
try {  
    ... // assumir que é usado xpto()  
}  
catch ( ErrConst e) {  
    ... // trata-se a exceção do tipo ErrConst apenas  
}  
catch ( ErrDes e) {  
    ... // trata-se a exceção do tipo ErrDes apenas  
}
```


11 - Exceções

- Escolha do gestor de exceção

- Um programa que utiliza o método `xpto()` pode gerir as exceções que ele é susceptível de desencadear das seguintes maneiras:

```
try {  
    ... // assumir que é usado xpto()  
}  
catch ( ErrConst e) {  
    ... // trata-se a exceção do tipo ErrConst apenas  
}  
catch ( ErrPonto e) {  
    ... // tratam-se as exceções do tipo ErrPonto ou derivadas (outras que não ErrConst)  
}
```

11 - Exceções

- Escolha do gestor de exceção

- Um programa que utiliza o método `xpto()` pode gerir as exceções que ele é susceptível de desencadear das seguintes maneiras:

```
try {  
    ...  
}  
catch ( ErrPonto e) {  
    ...  
}  
catch ( ErrConst e) {  
    ...  
}
```

Erro de compilação uma vez que
`ErrConst` nunca se atingiria

11 - Exceções

- Encaminhamento das exceções

- Assim que um método desencadeie uma exceção, procura-se primeiro que tudo um eventual gestor no bloco **try** que contenha a instrução **throw** correspondente
- O gestor de exceções raramente está no método que lançou a exceção - um dos objetivos do tratamento de exceções é precisamente separar o lançamento das exceções da gestão das exceções
- Nos exemplos estudados anteriormente a procura no bloco try é feita no método chamante. Por exemplo, a exceção ErrConst lançada pelo construtor de Ponto era tratada não no bloco try deste construtor mas no bloco try do método main que tinha chamado o construtor

11 - Exceções

- A cláusula throws

- Java impõe a regra seguinte:

Todo o método susceptível de lançar uma exceção que não a trate localmente deve mencionar o seu tipo numa cláusula **throws** no seu cabeçalho

- Assim sabe-se exatamente a que exceções o método é susceptível

11 - Exceções

- O bloco finally
- O desencadeamento de uma exceção provoca uma ligação incondicional a um dos gestores qualquer que seja o nível a que se encontre. A execução prossegue com as instruções deste gestor
- O JAVA permite a introdução de um bloco particular a seguir a um bloco **try** que é sempre executado:
- seja no final natural do bloco try, se nenhuma exceção foi desencadeada (pode tratar-se de uma instrução de ligação incondicional tal como **break** ou **continue**)
- seja depois do gestor de exceções (na condição óbvia deste não ter provocado a paragem da execução do programa)

11 - Exceções

- O bloco finally

- O bloco é introduzido com a palavra reservada **finally** e deve obrigatoriamente ser colocado a seguir ao último gestor de exceção
- Não tem qualquer interesse se as exceções são tratadas localmente

```
try {  
    ...  
}  
catch ( Exception E e) {  
    ...  
}  
finally {  
    // instruções apto  
}
```

=

```
try {  
    ...  
}  
catch ( Exception E e) {  
    ...  
}  
  
// instruções apto
```

11 - Exceções

- O bloco finally

- O bloco é introduzido com a palavra reservada **finally** e deve obrigatoriamente ser colocado a seguir ao último gestor de exceção
- Não tem qualquer interesse se as exceções são tratadas localmente

```
try {  
    ...  
}  
catch ( Exception E e) {  
    ...  
}  
finally {  
    // instruções apto  
}
```

=

```
try {  
    ...  
}  
catch ( Exception E e) {  
    ...  
}  
  
// instruções apto
```

11 - Exceções

- O bloco finally

- Situação bem diferente no exemplo seguinte. Se uma exceção Ex se produz em xpto(), as instruções A são executadas, devido ao bloco **finally**, antes de ir para o gestor de exceções apropriado
- Sem a presença da palavra reservada finally as instruções A só seriam executadas se não houvesse exceções lançadas no bloco **try**

```
void xpto ( ) throws Ex {  
    ...  
    try {  
        ...  
    }  
    finally {  
        // instruções A  
    }  
    ...  
}
```


11 - Exceções

- O bloco finally

- O bloco **finally** ganha importância no que diz respeito à aquisição de recursos
- Uma ação que necessita de uma ação contrária para continuar a execução do programa: a criação de um objeto; a abertura de um ficheiro; a interdição de um ficheiro partilhado
- A todos os recursos adquiridos num programa deve ser possibilitado a sua libertação mesmo em caso de exceção
- O bloco **finally** permite tratar este problema uma vez que é suficiente colocar as instruções de libertação de recursos de todos os recursos alocados no bloco **try**

11 - Exceções

- **[E] As exceções standard**
- O JAVA dispõe de diversas classes predefinidas derivadas da classe Exception que são usadas por métodos standard
- Por exemplo a classe IOException e as suas derivadas são utilizadas por métodos de entrada e saída
- Existem classes de exceções que são utilizadas pela máquina virtual para detetar anomalias, tais como, um índice de uma tabela fora do limite, um índice de uma tabela negativo, etc.
- As exceções standard classificam-se em duas categorias:
 - (1) As exceções explícitas;
 - (2) As exceções implícitas;

11 - Exceções

- **[E] As exceções standard**
- (1) As exceções explícitas: (também designadas sob controlo ou verificadas) correspondem aos exemplos anteriores. Elas devem ser tratadas por um método ou mencionadas na cláusula throw
- (2) As exceções implícitas: (fora de controlo) não tem de ser mencionadas na classe throw e não é obrigatório tratá-las (mas podem ser)

11 - Exceções

- **[E] As exceções standard**
- Esta classificação separa as exceções susceptíveis de se produzirem em qualquer parte do código daquelas onde se distingue claramente uma fonte potencial de exceção
- O surgimento de um índice superior ou uma divisão por zero, e.g., pode aparecer / produzir-se em qualquer lado do programa
- Um erro de leitura só surgirá se não forem utilizados os métodos apropriados / bem conhecidos

11 - Exceções

- **[E] As exceções standard**
- Exemplo: Exceção standard `NegativeArraySizeException` e `ArrayIndexOutOfBoundsException` e utilização novamente do método `getMessage()`

```
public class ExemploStandard{
    public static void main(String[] args){
        Scanner sc = new Scanner(System.in)
        try{
            int t[ ] :
            System.out.Print("Dimensão = ");
            int n = sc.nextInt( );
            t = new int[n];
            System.out.print("Indice: ");
            int i = sc.nextInt( );
            t[i] = 12;
            System.out.print("Fim Normal");
        }
    }
}
```

```
        catch (NegativeArraySizeException e){
            System.out.Println("Erro ind neg: " +
            e.getMessage());
        }
        catch (ArrayIndexOutOfBoundsException
        e){
            System.out.Println("Erro ind gra: " +
            e.getMessage());
        }
    }
}
```

Corrida 1:
Dimensão = -3
Erro ind neg:

Corrida 2:
Dimensão = 5
Indice: 6
Erro ind gra: 6

11 - Exceções

- **[E] As exceções standard**
- Pacote standard - java.lang: Exception explicit / implicit
- Pacote java.io: IOException
- Pacote java.awt: AWTException; RuntimeException
- Pacote java.util: Exception explicit: TooManyListenersException; Exception implicit: RuntimeException