

Curso Profissional: Programação de Informática

PSD – 11.º ano: UFCD 0816 - Programação de sistemas distribuídos - JAVA

Ficha de Trabalho 13

Ano letivo 22/23

Exceções

As exceções são objetos gerados pelo Java, durante a execução de um programa, para indicar que ocorreu um erro. Estes erros podem dever-se a problemas de código (erros de lógica) ou a falhas em tempo de execução, impossíveis de prever ou evitar.

A divisão por zero ou o acesso a um índice de array que está fora dos limites são exemplo de problemas de código. O acesso a um ficheiro inexistente ou a um servidor não disponível são exemplos de erros de execução.

Exemplo (programa sintaticamente correto que em execução para abruptamente):

```
class DivZero {
    public static void main(String[] args){
        int a=1, b=0;
        System.out.println(a/b); // divisão por zero!
        System.out.println ("Fim de programa");
    }
}
```

A divisão por zero provoca a interrupção do programa

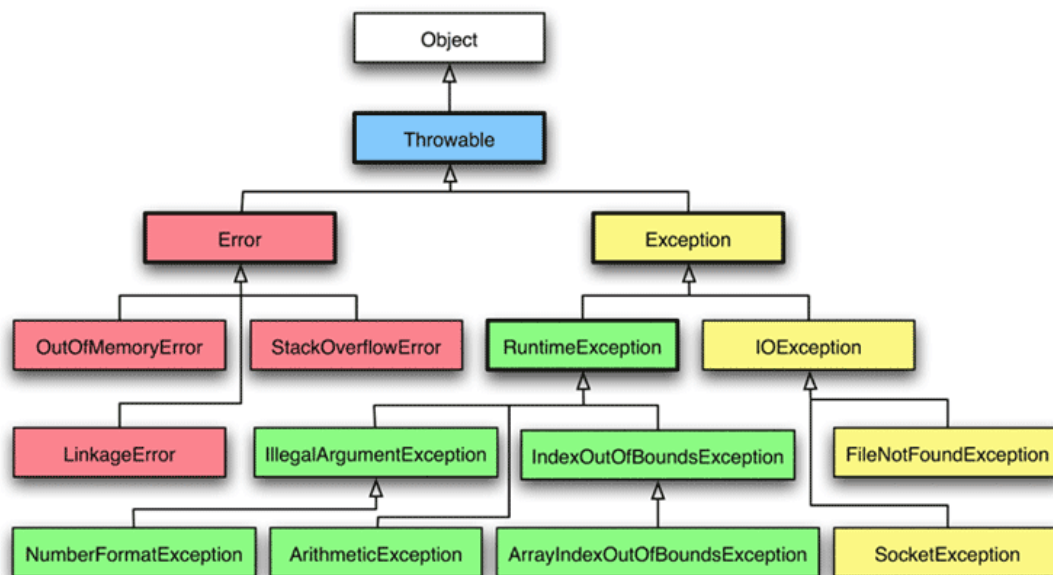
Console
<terminated> DivisaoZero (2) [Java Application] C:\Program Files\Java\jre1.8.0_101\bin\javaw.exe (29/12/2016, 15:06:12)
Exception in thread "main" java.lang.ArithmeticException: / by zero
at DivisaoZero.main(DivisaoZero.java:5)

Tipos de exceção

As exceções geradas pelo Java são instâncias de classes que são subclasses de "Exception".

Exemplos de exceções:

Erro	Exceção
Divisão por zero	ArithmeticException
Acesso a índice de array fora dos limites	ArrayIndexOutOfBoundsException
Acesso a ficheiro não existente	FileNotFoundException
Acesso a servidor indisponível	SocketException
Tipo de dados inválido	InputMismatchException



Sempre que se usam construtores ou métodos que possam dar origem a exceções, que serão lançadas pelo Java em tempo de execução do programa, deve-se utilizar o **bloco try/catch** para capturar os erros gerados e que assume a seguinte sintaxe:

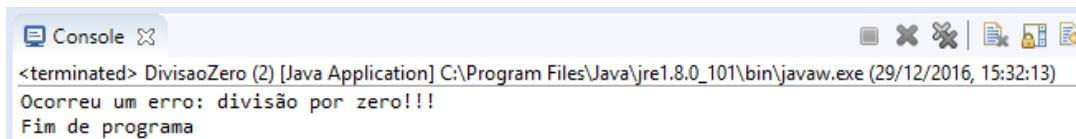
```
try {
    //código que poderá gerar o erro
}
catch (Exception* ex) {
    // código com o tratamento da exceção caso esta ocorra
}
```

*A exceção capturada dever ser uma instância de qualquer subclasse de “Exception”.

Aplicação:

Exemplo 1 (try catch):

```
public class DivisaoZero {
    public static void main(String[] args){
        int a=1, b=0;
        try{
            System.out.println((a/b)); // divisão por zero!
        }
        catch (Exception c){
            System.out.println ("Ocorreu um erro: divisão por zero!!!");
        }
        System.out.println ("Fim de programa");
    }
}
```



```
Console
<terminated> DivisaoZero (2) [Java Application] C:\Program Files\Java\jre1.8.0_101\bin\javaw.exe (29/12/2016, 15:32:13)
Ocorreu um erro: divisão por zero!!!
Fim de programa
```

Bloco finally

Sempre que existe um bloco de código que deve ser executado independentemente de ter ocorrido ou não uma exceção, deve-se utilizar a palavra reservada **finally**:

```
try {
    //código que poderá gerar o erro
}
catch (Exception* ex) {
    // código com o tratamento da exceção caso esta ocorra
}
finally {
    // código final, executado quer haja, ou não, uma exceção
}
```

Exemplo 2:

```
public class ConverteString {
    public static void main(String[] args){
        try{
            String s= "123abc";
            int i=Integer.parseInt(s);
            System.out.println(i);
        }
        catch (NumberFormatException nfe){
            System.out.println ("O formato do n.º que está a converter é inválido!!!");
            // nfe.printStackTrace(); // a)
            // System.out.println(nfe.toString()); // b)
            // System.out.println(nfe.getMessage()); // c)
        }
        finally{
            System.out.println ("Fim de programa");
        }
    }
}
```

NOTAS:

- a) Se usarmos o método `printStackTrace()` é mostrado a pilha armazenada na memória da origem do erro, podendo-se analisar todo o “rasto” do erro, de onde partiu até onde chegou, com grande utilidade para efetuar a depuração do código.

```
O formato do n.º que está a converter é inválido!!!
java.lang.NumberFormatException: For input string: "123abc"
    at java.lang.NumberFormatException.forInputString(Unknown Source)
    at java.lang.Integer.parseInt(Unknown Source)
    at java.lang.Integer.parseInt(Unknown Source)
    at ConverteString.main(ConverteString.java:7)
Fim de programa
```

- b) se usarmos o método `toString()` é apenas apresentado o tipo de exceção e a mensagem de erro.

```
O formato do n.º que está a converter é inválido!!!
java.lang.NumberFormatException: For input string: "123abc"
Fim de programa
```

- c) Se usarmos o método `getMessage()` obtemos apenas a mensagem de erro.

```
O formato do n.º que está a converter é inválido!!!
For input string: "123abc"
Fim de programa
```

Palavras reservadas `throws` e `throw`

Por vezes pode acontecer existir a possibilidade de serem geradas exceções num método, mas não ser definido nenhum mecanismo para as tratar. Nesse caso, um método pode definir que a exceção, caso ocorra, seja enviada para o método que o chamar. Para isso, é utilizada a palavra-chave *throws*:

Nome_do_metodo() throws IOException { ...

A palavra-chave *throws* está presente na assinatura (cabeçalhos) dos construtores e métodos nativos do Java que podem provocar exceções, por exemplo sempre que se utiliza o método `System.in.read()`, como já observámos na ficha 6, ex6):

```
import java.io.IOException;
...
public class M9F3ex3 {
    public static void main(String[] args) throws IOException {
    ...
    char converterEm = (char)System.in.read();
    ...
}
```

O próprio programador pode provocar uma nova exceção, utilizando para tal a palavra reservada **throw**:

```
if (condição)
    throw new AminhaExcecao();
```

Assim que a condição se verificar, é gerada uma exceção da classe **AminhaExcecao**. Qualquer classe de exceções terá que ser necessariamente uma subclasse da classe **throwable**.

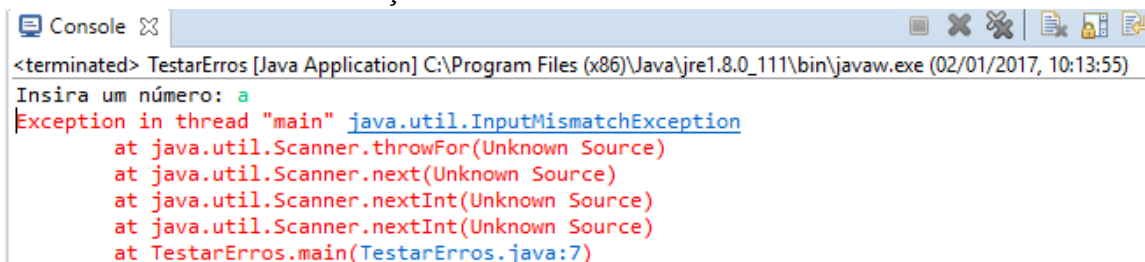
EXERCÍCIOS

1. No Eclipse, cria um projeto, de nome **F13TestarErros**. Cria uma classe com o nome **TestarErros** que tenha o método **main**.
2. Dentro do método *main*, inclui o seguinte código:

```
public static void main(String[] args) {
    Scanner entrada = new Scanner(System.in);
    System.out.print("Insira um número: ");
    int n = entrada.nextInt();
    System.out.println("O número introduzido foi "+n);
}
```

3. Executa o código anterior e introduz um número.
4. Executa novamente o código mas introduz uma letra.

Obtém-se um erro de execução:



```
<terminated> TestarErros [Java Application] C:\Program Files (x86)\Java\jre1.8.0_111\bin\javaw.exe (02/01/2017, 10:13:55)
Insira um número: a
Exception in thread "main" java.util.InputMismatchException
    at java.util.Scanner.throwFor(Unknown Source)
    at java.util.Scanner.next(Unknown Source)
    at java.util.Scanner.nextInt(Unknown Source)
    at java.util.Scanner.nextInt(Unknown Source)
    at TestarErros.main(TestarErros.java:7)
```

5. Provocando um erro é possível descobrir qual a classe da Exceção gerada, assim podemos evitar que o programa termine colocando o seguinte código:

```
public static void main(String[] args) {
    Scanner entrada = new Scanner(System.in);
    System.out.print("Insira um número: ");
    try {
        n = entrada.nextInt();
        System.out.println("O número introduzido foi "+n);
    }
    catch( java.util.InputMismatchException e ) {
        System.out.println("Não introduziu um número inteiro");
    }
}
```

6. Introduz o seguinte código:

```
int numeros[] = { 20, 30, 40 };
numeros[6] = 4;
```

7. Executa o código anterior e verifica que ocorreu um erro devido a tentativa de acesso ao array fora dos seus limites (array números, apenas tem 3 posições de 0 a 2).

```
<terminated> TestarErros [Java Application] C:\Program Files (x86)\Java\jre1.8.0_111\bin\javaw.exe
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 6
    at TestarErros.main(TestarErros.java:6)
```

8. Coloca o código necessário para escrever “Acesso fora de limites.” Mais a mensagem de erro.

```
int numeros[] = { 20, 30, 40 };
try {
    numeros[6] = 4;
}
catch( java.lang.ArrayIndexOutOfBoundsException e ) {
    System.out.println("Acesso fora de limites (" + e.getMessage() + ")");
}
```

9. Alternativamente, podemos utilizar uma exceção genérica:

```
int numeros[] = { 20, 30, 40 };
try {
    numeros[6] = 4;
}
catch(Exception e ) {
    System.out.print("Ocorreu um erro:" + e); //funciona de modo semelhante ao toString
}
```

✗ CRIAÇÃO DE EXCEÇÕES PELO UTILIZADOR

10. Dentro da classe **TestarErros** acrescenta um novo método estático:

```
public static double ler_nota_secundario() throws Exception {
    double n;
    Scanner entrada = new Scanner(System.in);
    System.out.print("Insira um número: ");
    n = entrada.nextInt();

    if( n < 0 || n > 20 )
        throw new Exception("Valor deve estar entre 0 e 20.\n");
    return n;
}
```

11. No main coloca o seguinte código:

```
double n;
try {
    n = ler_nota_secundario();
    System.out.println("O número introduzido foi "+n);
}
catch( java.util.InputMismatchException e ) {
    System.out.print("Não introduziu um número");
}
catch( Exception e) {
    System.out.print("Erro: "+e.getMessage());
}
```

12. Insere o número 2 e depois o número 25. Avalia o resultado.

13. Insere o número 2 e depois uma letra. Avalia o resultado.

14. Acrescenta um ciclo infinito de modo a tentar novamente ler o número, se ocorrer um erro. Quando se consegue ler um número termina o ciclo com break.

```
double n;
for(;;) // ou usar while(true)
{
    try {
        n = ler_nota_secundario();
        System.out.println("O número introduzido foi "+n);
        break;
    }
    catch( java.util.InputMismatchException e ) {
        System.out.println("Não introduziu um número");
    }
    catch( Exception e) {
        System.out.println("Erro: "+e.getMessage());
    }
}
```

15. Faz uma cópia do programa **TestarErros**. Renomeia-o para **TestarErros2**. Altera o programa para ler quatro notas, dizer a sua maior nota, menor nota e média. Elimina o código desnecessário.
16. Acrescenta a **TestarErros2** código para calcular o número de vezes que se tentou introduzir um número. Dica: utiliza *finally*.
17. Altera o programa **TestarErros2** para se poder optar por notas em percentagem (0 a 100) ou valores (0 a 20).
18. *Envia os ficheiros.java do projeto, para a classroom.*

1. No Eclipse, cria um projeto de nome Java2F1Contas
2. Cria a classe Conta adicionando o atributo saldo, do tipo double e com modificador de acesso privado :

```
public class Conta {  
    private double saldo;  
}
```

3. Cria os métodos Getter e Setter para o atributo saldo e os métodos deposita(double) e levanta(double) que atualize o saldo de uma conta.

```
public abstract class Conta {  
    private double saldo;
```

```
    public void deposita(double valor) {  
        this.saldo += valor;  
    }  
    public void levanta(double valor) {  
        this.saldo -= valor;  
    }  
    ...  
}
```

4. Adiciona mais um método na classe Conta, que atualiza essa conta de acordo com uma taxa percentual fornecida.

```
class Conta {  
    private double saldo;
```

```
    ...  
    public void atualiza(double taxa) {
```

```
        this.saldo += this.saldo * taxa;
```

```
    }  
}
```

5. Cria duas subclasses da classe Conta: ContaCorrente e ContaPoupanca. Ambas terão o método atualiza reescrito:
 - A ContaCorrente deve atualizar-se com o dobro da taxa e a ContaPoupanca deve atualizar-se com o triplo da taxa.

```
public class ContaCorrente extends Conta{
    public void atualiza(double taxa) {
        setSaldo(getSaldo()+ ( getSaldo() * taxa * 2));
    }
}
```

Porquê usar este código e não:
`this.saldo += this.saldo * taxa*2;`
Resposta: _____

```
public class ContaPoupanca extends Conta {
    public void atualiza(double taxa) {
        setSaldo(getSaldo()+ ( getSaldo() * taxa * 3));
    }
}
```

6. Na classe ContaCorrente, reescreve o método deposita para descontar a taxa bancária de dez centimos:

```
public class ContaCorrente extends Conta {
    public void atualiza(double taxa) {
        setSaldo(getSaldo()+ ( getSaldo() * taxa * 2));
    }
    public void deposita(double valor) {
        setSaldo( getSaldo() + valor - 0.10);
    }
}
```

7. Cria uma classe com método main, de nome TestaContas e instancia essas classes. Atualiza-as e observa o resultado. Algo como:

```
public class TestaContas {
    public static void main(String[] args) {
        Conta cc = new ContaCorrente();
        Conta cp = new ContaPoupanca();
        cc.deposita(1000);
        cp.deposita(1000);
        cc.atualiza(0.01);
        cp.atualiza(0.01);
        System.out.println(cc.getSaldo());
        System.out.println(cp.getSaldo());
    }
}
```

7. Acrescenta o código abaixo e executa de novo o programa.

```
cc.levanta(1100); // queremos que seja gerada uma exceção
cp.deposita(-100); // queremos que seja gerada uma exceção
System.out.println(cc.getSaldo());
System.out.println(cp.getSaldo());
```


8. Para prevenir depósitos ou levantamentos inválidos (respetivamente menores que zero e superiores ao saldo existente), altera o código da forma seguinte:

```
public void levanta(double valor) throws Exception {  
    if (this.saldo < valor)  
        throw new IllegalArgumentException();  
    else  
        this.saldo -= valor;  
}  
-----  
public void deposita (double valor) throws Exception {  
    if (valor < 0)  
        throw new IllegalArgumentException();  
    else  
        this.saldo += valor;  
}
```

9. Usar try/catch/finally, para a captura dos erros gerados e usar printf para formatar a saída de dados.

```
public class TestaContas {  
    public static void main(String[] args) throws Exception {  
  
        Conta cc = new ContaCorrente();  
        Conta cp = new ContaPoupanca();  
        cc.deposita(1000);  
        cp.deposita(1000);  
        cc.atualiza(0.01);  
        cp.atualiza(0.01);  
  
        try {  
            cp.deposita(-100);  
        }  
        catch (IllegalArgumentException e) {  
            System.out.println("Depósito inferior a zero!");  
        }  
  
        try {  
            cc.levanta(1100);  
        }  
        catch (IllegalArgumentException e) {  
            System.out.println("Saldo Insuficiente!!!");  
        }  
        finally {  
            System.out.printf("Saldo da conta poupança: %.2f €\n", cp.getSaldo());  
            System.out.printf("Saldo da conta corrente: %.2f €", cc.getSaldo());  
        }  
    }  
}
```

Caso as instruções do bloco finally estivessem dentro do bloco try seriam executadas?

Envia todos os ficheiros com extensão java criados em Java2F1Contas para a classroom.

Bibliografia:

<https://www.w3schools.com/java>

<https://www.tutorialspoint.com/java/>

Jesus, C. (2013). Curso Prático de Java. Lisboa: FCA

Coelho, P (2016). Programação em JAVA – Curso Completo. Lisboa: FCA