Formabase
Aprenda passo a passo

Java
Exceções

Metodologia

- Interprete o documento calmamente e com atenção.
- Acompanhe a execução do exercício no seu computador.
- Não hesite em consultar o formador para o esclarecimento de qualquer questão.
- Não prossiga para o ponto seguinte sem ter compreendido totalmente o ponto anterior.
- Caso seja necessário, execute várias vezes o exercício até ter compreendido totalmente o processo.

Conteúdo programático

1. Exceções

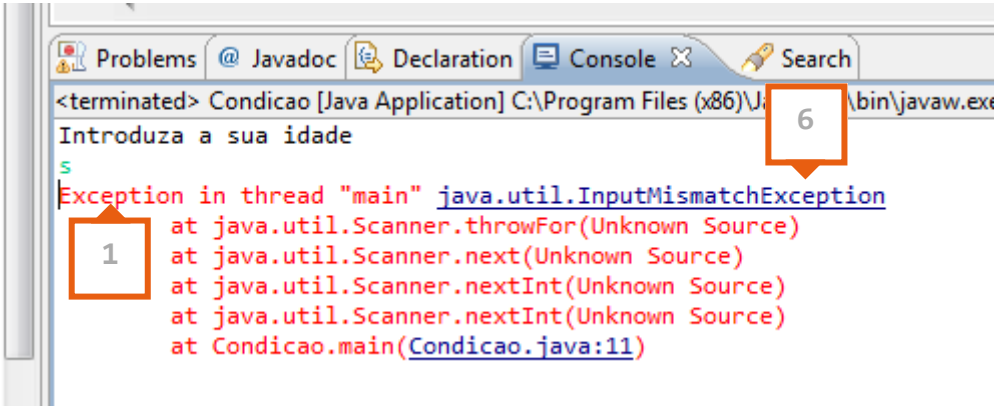
As exceções numa linguagem de programação são mecanismos de controlo de erros. Existem para verificar e tratar situações de erro que podem ocorrer em vários pontos do nosso programa. Isto é mais frequente em situações de leitura de dados, assim como se vê na linha abaixo.

```
int numero = teclado.nextInt();
```

Neste caso estamos à espera de ler um número inteiro da consola. O que acontece se o utilizador não inserir um número mas sim um texto? Este é um dos casos onde é gerado um erro ou, por outras palavras, é lançada uma exceção.

- **Execute** a classe `Condicao.java`

- Para a idade solicitada pelo programa, **introduza** um **texto**:



> Repare como o programa terminou abruptamente com um erro.

O erro é na verdade uma **exceção** 1 e existe no Java uma classe `Exception` que a representa. Quando esta é lançada, é criado internamente um objeto desta classe e são-lhe atribuídas informações relativas ao tipo de erro ocorrido. Consequentemente, a execução do programa é interrompida. Existe, no entanto, uma construção no Java que nos permite executar um determinado bloco de código e capturar as exceções que aí possam ser lançadas:

```
try {

      2

}
```

```
catch (Exception e){
      3
}
```

A **área** 2 representa a área com o código que queremos executar e que pode lançar exceções. É na **área** 3 que criamos o código que deve ser executado quando ocorre uma exceção. Se ocorrer uma exceção em qualquer uma das **linhas da área** 2 , a execução desse código pára imediatamente e o programa salta para as **instruções da área** 3 .

Vamos ver este conceito aplicado a uma situação real:

- **Reescreva** a classe `Condicao.java` de forma a ficar como o código seguinte:

```
import java.util.Scanner;

public class Condicao {

   public static void main(String[] args) {

      int idade;

      try{
          System.out.println("Introduza a sua idade");
          Scanner teclado = new Scanner(System.in);

          idade = teclado.nextInt();
      4
          if (idade<19){
              System.out.println("Você é um jovem");
          }
          else {
              System.out.println("Você é um adulto");
          }
```

```
        }
        catch(Exception e){

            System.out.println("Idade invalida");
        }
    }
}
```

- **Teste** esta alteração introduzindo **valores de texto** para a idade

> Repare que agora já não aparece a mensagem de texto com o erro anterior, mas sim a mensagem "Idade invalida". Quando na linha de **leitura do valor** 4 ocorre um erro a execução normal do programa pára passando a executar a **primeira linha do bloco de tratamento de exceções** 5 .

Cada tipo de erro está associado a uma exceção específica que não a classe `Exception`. Isto porque existem várias classes derivadas de `Exception` que definem erros mais particulares, como por exemplo **InputMismatchException** 6 , que identifica um erro relacionado com o tipo de dados introduzidos pelo utilizador. Desta forma, podemos dar um tratamento diferente a cada tipo de exceção que possa ocorrer no mesmo bloco de código. Para isso é necessário criar vários blocos de `catch`, do mais específico ao mais genérico.

Para exemplificar este conceito vamos criar uma nova classe.

```
import java.util.Scanner;

public class Calculadora {
    public static void main(String[] args) {

        Scanner teclado = new Scanner(System.in);

        System.out.println("Insira o primeiro numero");
        int num1 = teclado.nextInt();
```

```
        System.out.println("Insira a operação");
        char oper = teclado.next().charAt(0);

        System.out.println("Insira o segundo numero");
        int num2 = teclado.nextInt();

        double res=0;

        switch(oper){
            case '+': res = num1 + num2; break;
            case '-': res = num1 - num2; break;
            case '*': res = num1 * num2; break;
            case '/': res = num1 / num2; break;
        }

        System.out.println("O resultado é -> " + res);
    }
}
```

- **Compile** e **teste** a classe

- **Teste** com **valores de texto** para os números e **teste** também uma **divisão por zero**

  Repare a exceção é diferente para cada um destes casos.

- **Altere** o código que se encontra realçado:

```
import java.util.InputMismatchException;
import java.util.Scanner;
```

```
public class Calculadora {
    public static void main(String[] args) {

        Scanner teclado = new Scanner(System.in);

      try {
         System.out.println("Insira o primeiro numero");
            int num1 = teclado.nextInt();

            System.out.println("Insira a operação");
            char oper = teclado.next().charAt(0);

            System.out.println("Insira o segundo numero");
            int num2 = teclado.nextInt();

            double res=0;

            switch(oper){
                case '+': res = num1 + num2; break;
                case '-': res = num1 - num2; break;
                case '*': res = num1 * num2; break;
                case '/': res = num1 / num2; break;
            }

            System.out.println("O resultado é -> " + res);
      }
      catch (InputMismatchException e){
          System.out.println("Erro nos tipos dos dados introduzidos");
      }
      catch (ArithmeticException e){
          System.out.println("Não pode dividir por zero");
      }
      catch (Exception e){
          System.out.println("Ocorreu um erro inesperado no programa");
      }
```

```
    }
}
```

- **Compile** a classe e **teste** com os mesmos valores que geraram **erros** anteriormente

Quando é lançada uma exceção, o compilador tenta encontrar, de cima para baixo, o bloco que trata a exceção do tipo que foi lançado. Caso isto não aconteça, e como todas as exceções derivam de `Exception`, inevitavelmente será capturada no último bloco.

Podemos também usar o próprio objeto da exceção para guardar ou mostrar informações da Exceção.

- Volte a **alterar** o código destacado:

```
catch (ArithmeticException e){
    System.out.println(e.getMessage());
    //System.out.println("Não pode dividir por zero");
}
```

- **Teste** esta alteração utilizando uma **divisão por zero**

O que fizemos nesta última alteração foi mostrar a mensagem interna do objeto exceção. Podemos também aceder aos seus métodos e atributos para retirar mais conclusões. É ainda possível especificar o bloco de código `finally` que é sempre executado, independentemente se é lançada ou não uma exceção:

```
try{

}
catch (ExpcetionTipo1 e){
```

```
}
catch (ExceptionTipoN e){

}
catch (Exception e){

}
finally{

}
```

Isto torna-se útil em alguns cenários específicos. Imaginemos que, no nosso programa, pretendemos abrir um ficheiro para escrita de dados. Se ocorrer uma exceção no programa, antes do fecho deste ficheiro, a informação que até então foi gravada, pode perder-se. Com o bloco `finally` podemos evitar esta situação, definindo as instruções de fecho do ficheiro.

Existem vários tipos de classes que derivam da classe `Exception` e ainda outros subtipos que derivam dessas. Sempre que tivermos dúvida de quais as corretas, devemos consultar a API do Java. A seguir temos uma tabela com as exceções mais comuns e os seus significados:

Exceção	Significado	Deriva de
ArithmeticException	Ocorrência de um erro de operação matemática, como por exemplo divisão por zero	RuntimeException
ClassNotFoundException	A aplicação não consegue encontrar a classe principal a executar	Exception
IllegalArgumentException	O método recebeu parâmetros incorretos	RuntimeException
IndexOutOfBoundsException	Indica que um índice de um objeto excedeu a quantidade de elementos desse objeto (ex: array, string, etc.)	RuntimeException
InputMismatchException	Erro na interpretação do tipo de dados	RuntimeException
IOException	Significa que ocorreu uma exceção de input/output (leitura ou escrita de dados)	Exception
NullPointerException	Ocorre quando se tenta efetuar operações num objeto que possui o valor null	RuntimeException
ParseException	Ocorreu um erro na interpretação de um texto	Exception