

□ Exceções

Uma exceção (**Exception**) é um sinal gerado pela máquina virtual de Java em tempo de execução indicando uma situação de erro da qual é possível recuperar. O objetivo da sua utilização é construir programas robustos com capacidade de correção e recuperação aquando da verificação de uma situação inesperada de erro em tempo de execução.

Um erro (**Error**) em Java corresponde a uma situação de erro da qual o programa não tem possibilidades de recuperar. Neste caso o interpretador envia uma mensagem de erro para o ecrã e termina a execução do programa.

Uma exceção é sinalizada ou “lançada” (“**thrown**”) a partir do ponto de código em que ocorreu, e diz-se capturada (“**caught**”) no ponto do código para o qual o controlo de execução do programa foi transferido.

1. Pretende-se um método que peça ao utilizador que introduza pelo teclado um valor inteiro, validando se esse valor está no intervalo cujos limites são dados como parâmetros. Para isso, resolva as seguintes questões:

a) Construa um método para uma classe de teste que dados como parâmetros dois inteiros, *i1* e *i2*, peça ao utilizador um valor no intervalo [*i1*, *i2*]. Se o valor lido do teclado estiver no intervalo, o método deve devolver o valor lido.

Se o valor não estiver no intervalo, o método deve lançar a exceção `ValorErrado` com a mensagem “Erro, o valor não está entre *i1* e *i2*”. Em *i1* e *i2* deverão estar os valores dos parâmetros. A exceção será tratada no programa que invocar o método.

b) Construa a classe de exceção, `ValorErrado`.

c) Na mesma classe construa um método `main` que invoque o método da alínea **a)** tratando a exceção.

d) Modifique o `main` tal que, se o valor introduzido estiver errado, deve repetir o pedido ao utilizador.

2. Pretende-se explorar a classe abaixo.

Ao tentar compilá-la, verificará que não o consegue fazer sem que cada método que possui a instrução `System.in.read()` inclua no seu cabeçalho a cláusula **throws IOException**. Terá ainda que importar o `package: java.io.IOException`.

- Isto acontece porque o método `read()` pode lançar uma exceção. A assinatura deste método é a seguinte: `public int read() throws IOException;`

Se o método que invoca o método `read()` não detetar e tratar a exceção localmente, então terá obrigatoriamente que lançar essa exceção para que ela seja tratada no método invocador ou noutra mais externo.

- Estude a classe e após as alterações necessárias execute o programa.

```

public class Exemplo1 {

    private static String leStr () /*throws IOException*/{
        int ch;
        String r = "";
        boolean fim = false;
        while (!fim){
            ch = System.in.read();
            if (ch < 0 || (char)ch=='\n')
                fim = true;
            else
                r = r + (char) ch;
        }
        return r;
    }

    public static void main(String[] args) /*throws IOException*/ {
        int index;
        String palavra = " ";
        String[] tabPal = new String[10];
        System.out.print("Introduza uma palavra: ");
        palavra=leStr();

        while (!palavra.equals("fim") ) {

            System.out.print("Introduza um índice (int): ");
            index = Integer.valueOf(leStr().trim()).intValue();
            tabPal[index]=palavra;
            System.out.print("Introduza uma palavra");
            System.out.println ("(para terminar escreva fim): ");
            palavra=leStr();
        }
    }
}

```

- e) Em vez de lançar a exceção, podemos tratá-la localmente usando as cláusulas try e catch:

- substitua o código do método leStr() por

```

...
while (!fim){
    try {
        ch = System.in.read();
        if (ch < 0 || (char)ch=='\n')
            fim = true;
        else
            r = r + (char) ch;
    }
    catch (java.io.IOException e){
        fim = true;
    }
}
return r;

```

...

- Verifique que já não precisa de lançar a exceção (**throws**) nem no método `leStr` nem no método `main`.

- f) Durante a execução do programa anterior podem facilmente ocorrer duas situações de erro:
- i) o utilizador pode introduzir um valor para o índice que esteja fora do intervalo de valores válidos ou
 - ii) introduzir um valor para o índice que não seja um inteiro.

- Simule estas duas situações e observe as mensagens de erro que ocorrem.

- Altere o programa de forma a detetar a primeira destas duas situações de erro (índice fora do intervalo), substituindo, no método `main`, a linha:

```
tabPal[index]=palavra;
```

por:

```
try {
    tabPal[index]=palavra;
}
catch (ArrayIndexOutOfBoundsException e){
    System.out.println(" 0 <= índice <= 9" );
}
```

- Altere o programa para que detete o erro de introdução de um valor não inteiro para o índice (exceção ***NumberFormatException***).

- Finalmente, altere o programa de forma a que quando o utilizador der um valor não válido para o índice, lhe seja pedido um novo valor até que introduza um valor correto.

3. Estude as classes abaixo e complete-as de acordo com as alíneas a), b) e c).

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

```

public class FrascoChocolates {
    int capacidade, conteudo;
    public FrascoChocolates(int cap, int cont) {
        capacidade = cap; conteudo = cont;
    }
}

```

a) Para a classe **FrascoChocolates**, construa um método, *retira*, que deverá receber uma quantidade (qtd) que, caso exista no frasco, deverá ser subtraída ao conteúdo existente. Caso a quantidade a retirar seja superior ao conteúdo, o método deverá gerar a exceção *FrascoVazio*.

A essa exceção deve associar a mensagem:

```
("OH! OH! guloso, queres demais "+qtd+" "+conteudo)
```

b) Para a classe **FrascoChocolates**, construa um método, *enche*, que deverá receber uma quantidade (qtd) que, se adicionada ao conteúdo for inferior ou igual à capacidade do frasco, deverá ser adicionada ao conteúdo deste. Caso contrário deverá ser gerada a exceção *FrascoCheio*. A essa exceção deve associar a mensagem:

```
("OH! OH! já chega, estou de dieta "+qtd+" "+conteudo);
```

```
}
```

- Construa agora a classe Teste abaixo,

```

public class Teste {
    public static void main (String args[]){
        int i, valor;
        int fornecedor[]={20, 80, 70, 60, 100, 50, 60, 20 };

```

```

        FrascoChocolates F = new FrascoChocolates(120, 50);

```

```

        for (i=0; i<8; i=i+2){
            try { valor=fornecedor[i];
                F.enche(valor);
                valor=fornecedor[i+1];
                F.retira(valor);
            }

```

c) **Capture as exceções** eventualmente geradas pelos métodos *enche* e *retira* e escreva a mensagem associada a cada exceção (método `getMessage()`).

```

        ...
        finally {
            System.out.println("fim da iteração"+ i );
        }
    } (*end for*)
} (*end main*)
} (*end class Teste*)

```

d) Antes de executar o programa indique qual será o seu output. Depois verifique a correção da sua solução.

4. Reveja as classes que implementou em exercícios das fichas anteriores.

- a. Faça o tratamento dos erros recuperáveis gerando e tratando exceções.
- b. Analise quais as situações em que fará sentido adiar o seu tratamento comunicando (lançando) a exceção ao método “chamador”.
- c. Será necessário criar novas classes de exceções para assinalar as situações identificadas? Se sim, faça-o!