

ESQUEMA AULA PRÁTICA 9

□ 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 objectivo da sua utilização é construir programas robustos com capacidade de correcção 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. Implemente a classe Exemplo1 apresentada abaixo:

```
public class Exemplo1 {
    private static String leStr () {
        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[]){

        int index;
        String palavra = " ";
        String[] tabPal = new String[10];
        System.out.print("Introduza uma palavra: ");
        palavra=leStr();
        while (!palavra.equals("\r") ) {
            System.out.print("Introduza um índice: ");
            index = Integer.valueOf(leStr().trim()).intValue();
            tabPal[index]=palavra;
            System.out.print("Introduza uma palavra: ");
            palavra=leStr();
        }
    }
}
```

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 excepção. 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 excepção localmente, então terá obrigatoriamente que lançar essa excepção para que ela seja tratada no método invocador ou noutra mais externo.

2. Em vez de lançar a excepção, 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;
...

```

e verifique que já não precisa de lançar a excepção.

3. 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.
 - a. Altere o programa de forma a detectar 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" );
}

```

- b. Altere o programa de forma a detectar o erro de introdução de um valor não inteiro (exceção *NumberFormatException*).
 - c. 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 correcto.
4. Numa situação em que se quer assinalar uma situação de erro relacionada com a semântica do problema pode ser o próprio programador a lançar uma excepção.
 - a. Modifique a classe `FilaDeEspera`, implementada anteriormente, por forma a que a mesma tenha um limite para o número de elementos que pode comportar.
 - b. Faça com que o método `Object retirarElemento()` ao detectar uma situação de fila vazia lance uma excepção já existente (`EmptyStackException` do *package java.util*).
 - c. Por sua vez, o método `void adicionarElemento (Object e)` deverá lançar uma excepção, `FilaCheiaException`, definida pelo utilizador na classe `FilaCheiaException` por exemplo da seguinte maneira:

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

5. Implemente uma classe teste que lhe permita estudar o comportamento da classe `Fila` definida acima e o tratamento das excepções lançadas nessa mesma classe.
6. Estude as classes que se seguem 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) **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) **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);
```

```
}
```

```
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 correcção da sua solução.

Para estudar depois da aula:

7. Reveja as classes que implementou em exercícios 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 que identificou? Se sim, faça-o!