

## Material de Apoio Aula 2

### Exceções

A ocorrência de falhas durante a execução de programas é uma situação bastante comum. As fontes são as mais diversas, podendo ser, por exemplo, entrada errônea de dados de entrada, divisão por zero ou mesmo acesso a uma posição indevida de memória. No entanto, um bom programa deve tolerar as falhas evitando que o programa aborte. A técnica empregada consiste em tratar as falhas de forma a recuperar um estado consistente da programação. Nesta terminologia, evita-se utilizar o termo erro, sendo mais apropriado indicar que uma exceção ao procedimento normal do programa foi observada, devendo ser tratada adequadamente. Para trabalhar este conceito, transcreva o programa apresentado na Figura 1 (download em <http://www.inf.unisinos.br/~gersonc/replab2/notas.java>), compile e execute-o. Observe as mensagens informadas na saída.

```
public class notas {
    public static void main(String[] args) {
        final int NUM_ALUNOS = 10;
        double [] vet = new double [NUM_ALUNOS] ;
        double acum = 0;

        System.out.println("Calcular a media de 10 notas.");
        System.out.println("Para simplificar, todo mundo tirou 10!!!");
        try {
            for( int cont = 0 ; cont <= NUM_ALUNOS ; cont++ )
                vet[cont] = 10;
        } catch( Exception e ) {
            System.out.print("Ups! Entrei uma nota que nao devia!");
            System.out.println(" Note: nao ha tratamento");
        }

        System.out.println("Observe a mesma falha, agora sem captura.");
        for( int cont = 0 ; cont <= NUM_ALUNOS ; cont++ )
            acum += vet[cont];
        System.out.println("Media = " + acum/NUM_ALUNOS);
    }
}
```

Figura 1. Exemplo de tratamento de exceções em Java.

O programa acima possui uma falha latente que é o caminhamento em uma posição inválida do array `vet`. Esta falha encontra-se em dois pontos. O primeiro deles é tratado, o segundo não. O trecho de código delimitado pelo bloco identificado pela palavra reservada `try` representa o trecho sobre o qual está sendo monitorada a ocorrência de exceções. O bloco identificado pela palavra reservada `catch` corresponde ao tratamento do erro propriamente dito – note que no exemplo apresentado acima o erro é representado por um objeto da classe `Exception`. No entanto, esta é uma classe genérica para exceções. A Figura 2 apresenta parte da hierarquia de classes de exceções.

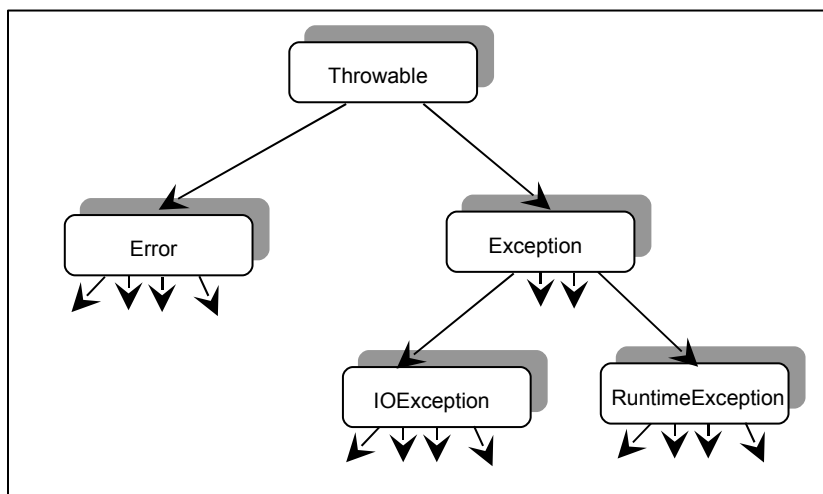


Figura 2. Estrutura hierárquica parcial das exceções em Java.

Estratégias mais complexas podem ser empregadas. Considere o trecho de código da Figura 3. Neste exemplo, duas situações de exceção são tratadas: falha no dispositivo de entrada e saída e falha devido ao usuário não ter digitado um número inteiro, conforme esperado no código.

```
try {
    BufferedReader teclado = new BufferedReader( new InputStreamReader(System.in) );
    System.out.println("Quantos anos tens?");
    String linha = teclado.readLine();
    int idade = Integer.parseInt(linha);
    idade++;
    System.out.println("Ano que vem tu teras: " + age + " anos!");
}
catch (IOException exception) {
    System.out.println("Erro de entrada e saida " + exception);
}
catch (NumberFormatException exception) {
    System.out.println("Nao digitado um numero como entrada");
}
```

Figura 3. Exemplo de tratamento de exceções em Java considerando diferentes falhas.

No código da Figura 3 são capturadas duas exceções, através de objetos identificando as exceções definidas pelas classes `IOException` e `NumberFormatException`, ambas especializações da classe `Exception`.

### Exercícios

1. Implemente em uma classe o trecho de código apresentado na Figura 3. Compile e analise o comportamento de execução.
2. Modifique o programa da Figura 1. Inclua o tratamento de erros considerando a possibilidade de três diferentes exceções:
  - `IOException` : erro de entrada e saída;
  - `DivideByZero` : divisão por zero (quando `NUM_ALUNOS = 0`);
  - `NumberFormatException` : entrada de um valor não numérico para nota.
3. Redefina o código desenvolvido no Exercício 2 para contemplar todos os outros tipos de exceções.

### Disparando exceções

O recurso de exceções pode também ser utilizado pelo programador como parte da solução de sua implementação. De forma prática, exceções podem ser disparadas de forma explícita. Observe o código na Figura 4. O código do método `teste` informa explicitamente através da cláusula `throws` que a exceção identificada pela classe `Exception` pode ocorrer neste método.

```
class TesteThrows {
    public static void teste() throws Exception {
        System.out.println ("Executando o teste.");
        throw new Exception();
    }

    public static void main (String args []) {
        try {
            teste();
        }
        catch( Exception e ) {
            System.out.println ("Ups!!!");
        }
    }
}
```

Figura 4. Disparando uma exceção.

No código da Figura 4 a única atividade executada pelo método `teste` é o disparo da exceção, no entanto, em um caso real, a exceção deve ser invocada em resposta a um teste que verifica a ocorrência de um evento de exceção. Um exemplo de caso real pode ser dado no tratamento de parâmetros de métodos. Suponha um método que deva receber como parâmetro um valor numérico inteiro positivo para efetuar alguma operação matemática. Caso este valor seja negativo, o método não executa nenhuma operação, mas sinaliza a ocorrência da exceção identificada como `IllegalArgumentException`.

## Exercício

4. Implemente uma classe `Cofrinho`. Esta classe deve ter como atributo interno a informação sobre o saldo do cofrinho (considere um valor `double`). Implemente, além dos necessários métodos construtores, um método `Depositar` e um método `Sacar`. Garanta, utilizando mecanismo de exceção, que não será depositado um valor negativo na conta. Utilize `IllegalArgumentException`. Implemente um método `main` que utilize objetos da classe `Cofrinho` e realize o tratamento de exceções correspondente.

### Criando suas próprias exceções

O programador não está preso em utilizar as exceções disponíveis em Java, ele pode criar suas próprias estendendo a classe `Exception`. Um exemplo da definição de uma exceção é dado na Figura 5. Neste código, a classe `MinhaExcecao` é a nova exceção definida. Como informação adicional, é possível informar um código de erro. Observe que a classe define também o método `toString`, que permite apresentar um texto informativo.

```
class MinhaExcecao extends Exception {
    private int cod;

    public MinhaExcecao() {
        cod = 0;
    }
    public MinhaExcecao( int a ) {
        cod = a;
    }
    public String toString() {
        return "Ocorreu a MinhaExcecao com codigo ["+ cod + "];"
    }
}
```

Figura 5. Definição de uma nova exceção.

O uso das classes de exceções definidas pelo usuário é o mesmo uso que se faz com as exceções próprias do Java. Para ilustrar, a Figura 6 apresenta um exemplo de código utilizando a exceção definida na Figura 5.

```
class TesteExcecao {
    public static void main( String args[] ) {
        try {
            int a=11;
            if( a > 10 ) {
                MinhaExcecao minhaExc = new MinhaExcecao( a );
                throw minhaExc;
            }
        }
        catch (MinhaExcecao e) {
            System.out.println ("Excecao capturada: " + e);
        }
    }
}
```

Figura 6. Uso da exceção definida pelo programador.

## Exercício

5. Implemente uma exceção para tratar a situação em que há a tentativa de fazer um saque e deixar o valor negativo. Utilize esta exceção na implementação do método `Sacar` da classe `Cofrinho` (Exercício 4). Implemente um método `main` adequado.