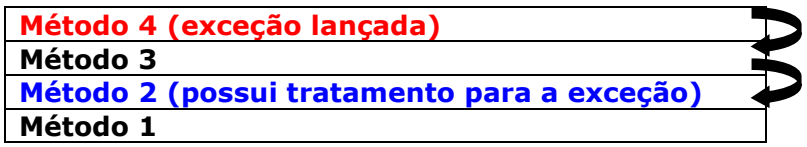
Universidade do Estado de Santa Catarina
Curso de Bacharelado em Ciência da Computação
Disciplina: Programação Orientada a Objetos (POO)

Exceções

1) Visão Geral

- Exceções são utilizadas para sinalizar uma irregularidade na execução de um método;
- Tipos de irregularidade que podem gerar exceção:
  - O arquivo que você tentou abrir não existe
  - A conexão de rede foi interrompida
  - O valor de algum atributo da classe foi setado com um valor inválido
  - Etc...
- Quando uma exceção é lançada, a execução do método em questão é interrompida. Uma vez interrompida, a execução retorna para a pilha de execução de chamadas, e vai "desempilhando" até encontrar algum método apto a tratar esta exceção (ou até o encerramento do programa):



- Quando trabalhamos com exceções, em geral, precisamos:
  - De uma classe que represente a exceção
  - Lançar a exceção
  - Capturar a exceção (tratá-la)

2) Exceções em Java

2.1. Representando exceções

- As exceções em Java são representadas por classes. A API Java fornece uma hierarquia de classes que representam exceções:

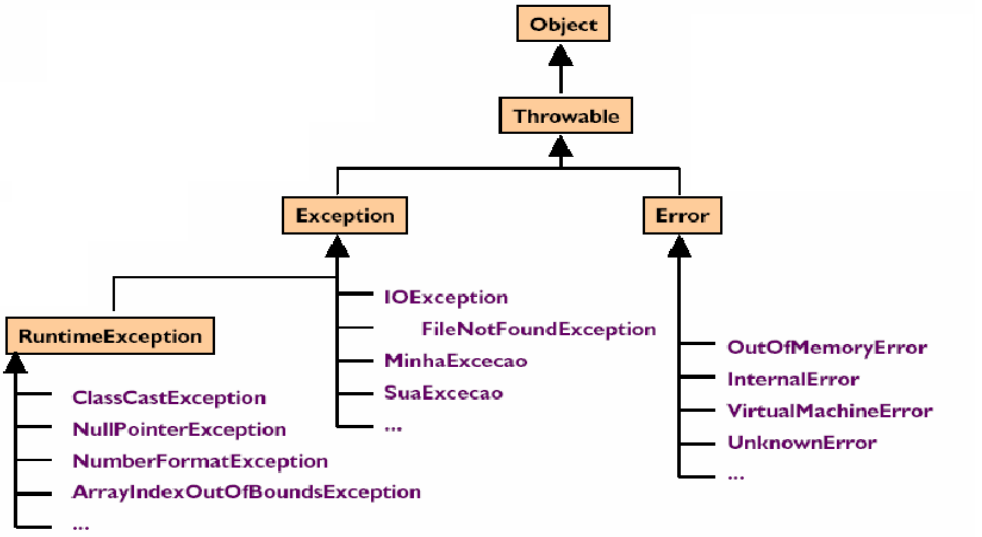


Figura 1. Principais exceções Java

- O programador pode definir uma classe para representar uma exceção. No entanto é necessário que tal classe estenda de alguma das classes de exceção do Java (Figura 1).

- Exemplo1: Criação de exceção pelo programador

```
public class InvalidNumberException extends Exception {

    public InvalidNumberException(int num, int min, int max) {
      super("Number "+num+" is out of range (Number must be in ["+
                  min+", "+max+"] interval)");
    }
}
```

2.2 Lançando Exceções

- Para lançar a exceção é preciso usar a instrução ***throw*** sucedida pela chamada ao construtor da exceção.
- Além disto, em geral, é necessário que os métodos que lançam exceção definam qual exceção estão lançando na assinatura do método. Para isto utilizamos a palavra chave ***throws*** sucedida do nome da exceção. No exemplo 2: throws InvalidNumberException. Maiores detalhes na seção 2.4
- Exemplo2: Lançando exceção criada pelo programador

```
public ProcessadorDatas {
      Calendar dataProcessada;

      public void setDataProcessada(int dia, int mes, int ano) throws
InvalidNumberException{
          if (mes <1 || mes>12)
                throw new InvalidNumberException(mes, 1, 12) ;
          dataProcessada = new GregorianCalendar(ano, mes, dia) ;
      }
}
```

- Exemplo3: Lançando exceção da API

```
public ProcessadorDatas {
      Calendar dataProcessada;

      public void setDataProcessada(int dia, int mes, int ano) throws
Exception {
          if (mes <1 || mes>12) {
                throw new Exception(("Month "+mes+" is out of range"
                   + "(Month must be in [1,12] interval)") ;
          }
          dataProcessada = new GregorianCalendar(ano, mes, dia) ;
      }
}
```

2.3. Capturando Exceções

- Exceções são capturadas por meio do bloco try-catch
- O bloco de código **try** é executado até que uma exceção seja lançada ou até que seja finalizado o processamento do bloco
- O bloco **catch** deve estar sempre relacionado a um bloco **try**. Os blocos **catch** capturam uma exceção específica que é determinada na sua assinatura, ou uma exceção filha da que está determinada.
- Exemplo4: Capturando exceção:

```
public class Teste {

    public void testeData(int d, int m, int a) {
          processadorDatas pd = new ProcessadorDatas();
          try {
              pd. setDataProcessada(a, m, d);
          } catch(InvalidNumberException ine) {
              System.out.println(ine);
          }
    }
}
```

- Adicionalmente pode-se utilizar a instrução **finally**. Tal instrução define um bloco de código que **sempre** será executado, mesmo que uma exceção seja lançada. Um bloco **finally** pode suceder um bloco catch ou diretamente um bloco **try**
- Exemplo5:

```
...
try {
    pd. setDataProcessada(a, m, d);
} catch(InvalidNumberException ine) {
    System.out.println(ine);
} finally {
    System.out.println("No final tudo dá certo!");
}
....
```

- É importante destacar que em um mesmo bloco **try** podem ocorrer diversas exceções. Portanto para um bloco **try** podemos ter diversos blocos **catch** para o tratamento de todas as exceções que são lançadas dentro dele**.**

2.4. Tipos de Exceção Java

- Java possui dois tipos de exceções:
    - ***Checked Exceptions*** são exceções que devem ser usadas para representar falhas contornáveis. Devem ser declaradas pelos métodos que as lançam e precisam ser tratadas (a menos que explicitamente passadas adiante). A classe Exception da API é uma checked exception. Portanto para criarmos uma checked exception precisamos torná-la filha de Exception.
    - ***Unchecked Exceptions*** são exceções que devem ser usadas para representar falhas incontornáveis. Não precisam ser declaradas e nem tratadas. As classes Error e RuntimeException são unchecked exceptions da API Java.

- Para declararmos as exceções na assinatura de método utilizamos a palavra-chave **throws** sucedida pela lista de exceções que podem ser lançadas por tal método
- O método `setDataProcessada` do exemplo 2 deveria obrigatoriamente declarar a exceção InvalidNumberException:

```
...
public void setDataProcessada(int dia, int mes, int
     ano) throws InvalidaNumberException {
....
```

- O mesmo deve ser feito pelo exemplo 3!!