Java

Professor: Marcelo Faustino

Nome: _____ Data: ____/____/____

Exceções

Em um mundo perfeito, os usuários nunca digitariam dados de forma errada, os arquivos que eles escolhessem sempre existiriam e os programas nunca teria erros. Infelizmente sabemos que esse mundo não existe. O que existe é um mundo real, onde temos que lidar com erros a todo instante.

Encontrar erros é desagradável. Se um usuário perder todo o seu trabalho durante uma sessão de um programa que você fez, seja devido a uma falha de programação, seja devido a circunstâncias externas, ele provavelmente não vai querer usar o seu programa mais. Por esse motivo, você no mínimo deveria:

- Notificar o usuário de um erro
- Salvar o trabalho
- Permitir que o usuário saia do programa de forma adequada

A linguagem Java dispõe de um mecanismo que permite o tratamento dos erros. Esse mecanismo é chamado tratamento de exceções.

Como lidar com erros:

Suponha que ocorra um erro, enquanto seu programa em Java está sendo executado. O erro poderia ser causado por um arquivo contendo informações erradas, por uma conexão de rede ruim ou pela tentativa de acessar um índice inválido de um array. Se uma operação não puder ser concluída por causa de um erro, o programa deveria:

- Retornar a um estado seguro e permitir que o usuário execute outras tarefas.
- Permitir que o usuário salve todo o seu trabalho e finalize o programa de forma correta.

Para lidar com situações excepcionais em um programa, é preciso levar em conta os erros e os problemas que podem ocorrer. Que tipos de problemas você precisaria considerar:

Erros de entrada de dados do usuário: Além dos erros de digitação, que são inevitáveis, alguns usuários gostam de complicar as coisas, não seguindo as instruções. O usuário pode, por exemplo digitar uma URL em um formato inválido.

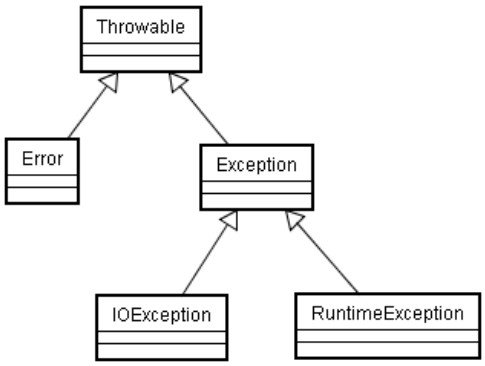
Erros de dispositivo: Nem sempre os dispositivos operam da forma que deveriam. A impressora pode estar desligada, ou a página que estamos tentando acessar pode ter sido removida.

Limitações físicas: Os discos podem ficar cheios, ou a memória disponível pode acabar.

Erros de programação: Você pode escrever um método que tenta acessar um índice inválido de um array.

Hierarquia das exceções:

Em Java, um objeto de exceção é sempre instância de uma classe derivada de Throwable:



Observe que todas as exceções descendem de Throwable, mas a hierarquia se divide imediatamente em duas: Error e Exception.

A hierarquia Error, descreve erros internos e a exaustão de recursos do sistema de tempo de execução Java. Pouco se pode fazer se um erro interno ocorrer.

Ao se programar em Java, deve focar a hierarquia Exception. Ela também de divide em duas: As que são RuntimeException e as que não são. Uma RuntimeException ocorre devido a um erro de programação, tais como:

- Conversão de tipos errada (Cast).
- Acesso a um array além dos limites.
- Acesso a um ponteiro nulo.

As exceções que não derivam de RuntimeException incluem:

- Tentar ler além do final de um arquivo.
- Tentar abrir um URL incorreto.
- Tentar encontrar um objeto Class através de uma string que não denota uma classe existente.

Capturando exceções:

Para capturar uma exceção, usamos o bloco protegido try...catch:

```
try {
    <código que pode gerar exceção>;
} catch (<TipoExceção> e) {
    <manipulador para a exceção>;
}
```

Exemplo:

```
3  public class TesteExcecoes {
4
5      public static double divide(int dividendo, int divisor) {
6          double resultado = 0;
7
8          try {
9              resultado = dividendo / divisor;
10         } catch (Exception e) {
11             System.out.println("Erro: divisao por 0.");
12         }
13         return resultado;
14     }
15
16     public static void main(String[] args) {
17
18         double valor = divide(10,0);
19         System.out.println(valor);
20     }
21 }
```

Anunciando exceções que um método lança:

A idéia é simples: Um método informa ao compilador que tipos de exceção ele pode retornar. Para isso usamos a palavra reservada **throws**.

```
public static double metodo(<Tipo> <argumento>)
        throws <Tipo Exceção que pode retornar> {
```

Exemplo:

```
3  public class TesteExcecoes1 {
4
5      public static double divide(int dividendo, int divisor)
6              throws Exception {
7
8          double resultado = 0;
9          resultado = dividendo / divisor;
10         return resultado;
11     }
12
13     public static void main(String[] args) {
14
15         double valor;
16
17         try {
18             valor = divide (10,0);
19             System.out.println(valor);
20         } catch (Exception e) {
21             System.out.println("Erro de divisao por 0");
22         }
23     }
24 }
```

Lançando exceções:

Imagine que em determinada situação, tenha acontecido algo excepcional com seu código e você decida que, nessa situação, você precisa parar o fluxo do programa. Você pode lançar uma exceção. Para lançar uma exceção em Java, usamos a palavra reservada **throw**.

```
<tipo exceção> <nome> = new <tipo exceção>;
throw <nome>;
```

Exemplo:

```
3  public class TesteExcecoes2 {
4
5      public static double divide(int dividendo, int divisor)
6              throws Exception {
7
8          double resultado = 0;
9          if (divisor < 0) {
10             Exception exception =
11                     new Exception("Proibido divisao por numero negativo");
12             throw exception;
13         }
14         resultado = dividendo / divisor;
15         return resultado;
16     }
17
18     public static void main(String[] args) {
19
20         double valor;
21
22         try {
23             valor = divide (10,0);
24             System.out.println(valor);
25         } catch (Exception e) {
26             System.out.println("Erro de divisao por 0");
27         }
28     }
29 }
```

Obs.: Não é necessário anunciar erros internos da linguagem Java (exceções derivadas de Error), pois qualquer código poderia lançar esse tipo de exceção. Da mesma forma, não se deve lançar exceções derivadas de RuntimeException. As exceções que extendem de Error, ou de RuntimeException são chamadas exceções não verificadas. As demais são chamadas de exceções verificadas. A regra para as exceções é simples:

Um método precisa declarar todas as exceções verificadas que ele lança.

Se o método lançar uma exceção verificada que não foi declarada, o compilador Java irá reclamar e emitir uma mensagem de erro.

A cláusula finally:

Suponhamos que você queria que determinado pedaço do seu código seja executado independente de ocorrer ou não uma exceção. Em Java, usamos a cláusula **finally** para isso:

```
try {
    <código que pode gerar exceção>;
} catch (<TipoExceção> e) {
    <manipulador para a exceção>;
} finally {
    <código que será executado de qualquer forma>;
}
```

Obtendo informações sobre a exceção:

As classes de exceção possuem dois métodos que nos ajudam a ober informações sobre elas, são eles:

- getMessage(): retorna a mensagem de erro
- printStackTrace() : imprime a pilha de métodos que estavam na fila para a execução:

Exemplo:

```
3  public class UsandoExcecoes {
4
5      public static void main(String[] args) {
6          try {
7              metodo1();
8          } catch (Exception e) {
9              System.err.println("Mensagem de erro: " + e.getMessage() + "\n");
10             e.printStackTrace();
11         }
12     }
13
14     public static void metodo1() throws Exception {
15         metodo2();
16     }
17     public static void metodo2() throws Exception {
18         metodo3();
19     }
20     public static void metodo3() throws Exception {
21         Exception exception = new Exception("Excecao lancada no metodo 3");
22         throw exception;
23     }
24 }
```