Tratamento de Exceções Entrada e Saída

Tecnicas de Programacao

José Antonio F. de Macêdo
jose.macedo@lia.ufc.br

MANIPULAÇÃO DE EXCEÇÕES

- Sua função é:
- transferir controle de onde ocorreu o erro para um manipulador de erro que possa lidar com a situação

- OBS:
  - Semelhante a C++ e Delphi
  - Mais flexível que On Error GoTo de VB

Exemplos de Exceções

- Erros de entrada do usuário
- digitação
- Erros de dispositivo
- impressora sem papel
- Limitações físicas
- disco cheio
- Erros de código
- acesso a pilha vazia

EXCEÇÕES

- Sinalizam condições de erro
- *Exceções Explícitas*
- devem ser tratadas
- sinalizam condições contornáveis
- *Exceções Implícitas*
- não precisam ser tratadas
- sinalizam condições geralmente incontornáveis

EXCEÇÕES

- Exceções são representadas por objetos
- Exceções são subclasses de **Throwable**
- **Exception**
- tipo base para *checked exceptions*
- **Error** & **RuntimeException**
- tipo base para *unchecked exceptions*
- **throw** lança uma exceção

EXCEÇÕES EXPLÍCITAS

- Precisam ser declaradas pelos métodos

```
class ListaE {
  private ListaE próximo;
  public void insere(ListaE e) throws Exception {
    if (e == null)
      throw new Exception("Elemento nulo");
    e.próximo = próximo;
    próximo = e;
  }
}
```

EXCEÇÕES IMPLÍCITAS

- Não precisam ser declaradas pelos métodos

```
class ListaE {
  private ListaE próximo;
  public void insere(ListaE e) {
    if (e == null)
      throw new Error("Elemento nulo");
    e.próximo = próximo;
    próximo = e;
  }
}
```

EXCEÇÕES EXPLÍCITAS

- Precisam ser tratadas pelos métodos

```
class ListaE {
  public void insere(ListaE e) throws Exception {
    ...
  }
  public void duploInsere(ListaE e1, ListaE e2) {
    insere(e2); // ERRO!
    insere(e1); // ERRO!
  }
}
```

EXCEÇÕES IMPLÍCITAS

- Não precisam ser tratadas pelos métodos

```
class ListaE {
  public void insere(ListaE e) {
    ... throw Error("Elemento nulo"); ...
  }
  public void duploInsere(ListaE e1, ListaE e2) {
    insere(e2); // OK!
    insere(e1); // OK!
  }
}
```

O QUE PRECISAMOS SABER ...

- Capturar as exceções lançadas pelos métodos que tentamos executar.
  - Comandos: **try, catch, finally**
- Anunciar uma exceção a qual pode ser lançada por um método.
  - Comando: **throws**
- Lançar e repassar exceções dentro de métodos.
  - Comando: **throw**

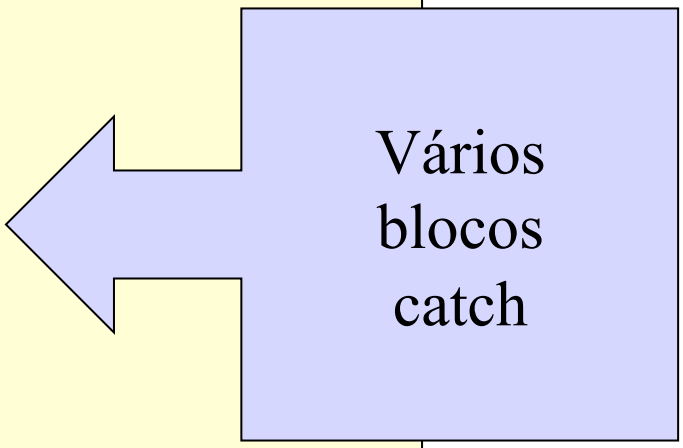
CAPTURAR EXCEÇÕES LANÇADAS

- Tentar executar um bloco de código
- No caso de erro, capturar exceções que foram lançadas
- finalmente realizar algum tipo de limpeza

```
try {
    // … execucar algo que possa causar uma exceção
}
catch ( TipoExcecao variavel) {
    // … tratar exceção
}
finally {
    // … ao final executar sempre este código
}
```

É possível tratar diversas exceções !

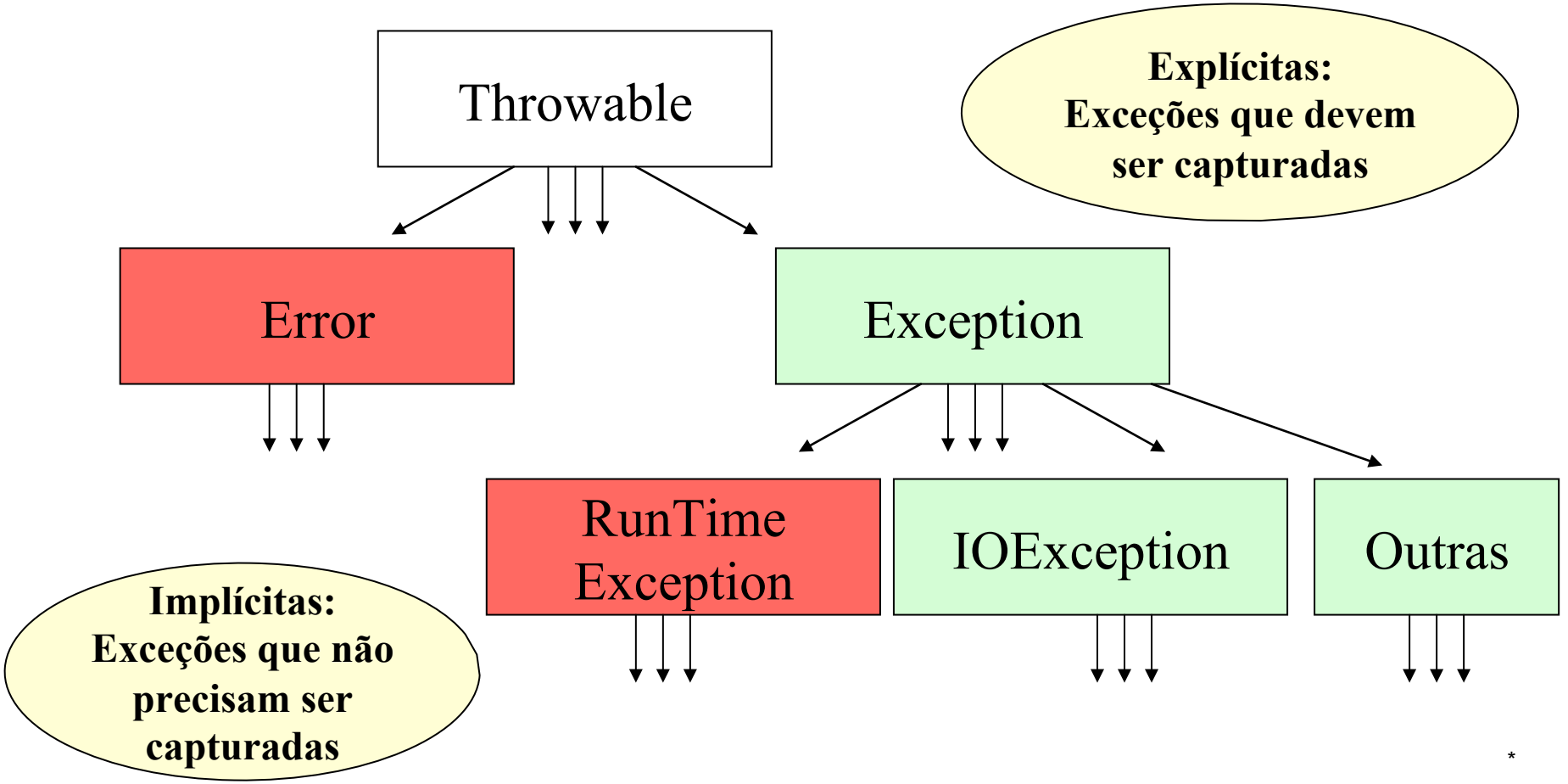
```
try {
   // … execucar algo que possa causar uma exceção
}
catch ( TipoExcecao1 variavel1) {
   // … tratar exceção1
}
catch ( TipoExcecao2 variavel2) {
   // … tratar exceção2
}
catch ( TipoExcecao3 variavel3) {
   // … tratar exceção3
}
finally {
  // … ao final executar sempre este código
}
```



Exemplo - Captura de exceções

```
int divisao (int a, int b) {
    int resultado = 0;
    try {
      resultado = a / b;
    }
    catch (ArithmeticException e) {
      System.out.println ("\tOcorreu um erro de divisão");
    }
    finally {
      System.out.println ("\tSempre vai apresentar esta mensagem");
    }
    return resultado;
  }
```

Hierarquia de Herança de Exceção

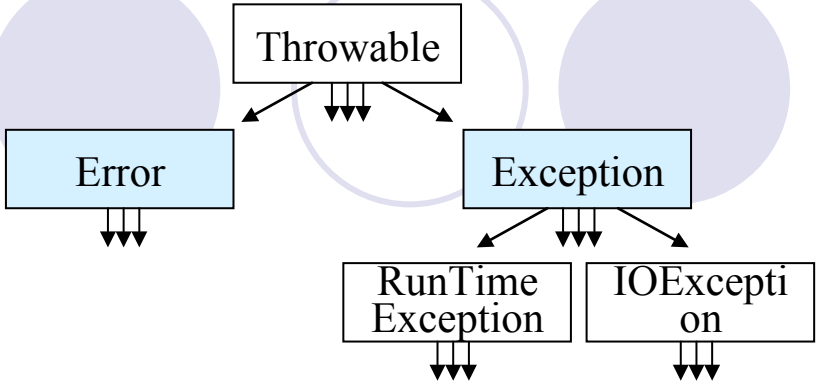


IMPORTANTE

- Exceção Implícita:
- derivada de Error ou de RunTimeException
- Exceção Explícita:
- todas as outras

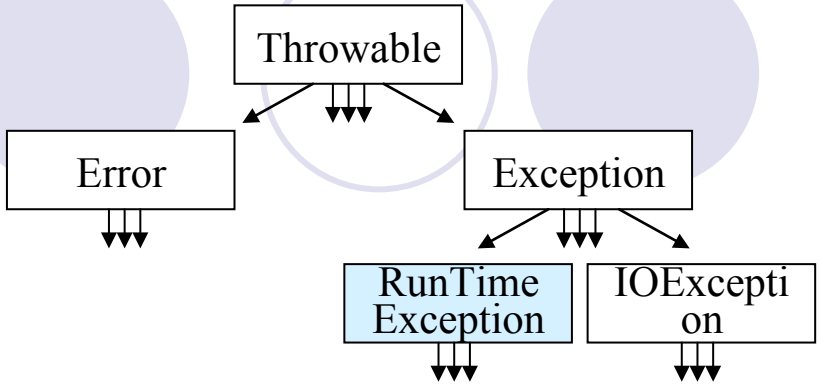
- Importante:
- Implícitas = fora de seu controle ou derivam de condições que deveriam ter sido tratadas por vc
- Explícitas = são as que vc deve tratar!
- Um método deve declarar todas as exceções explícitas que passa

COMENTÁRIOS



- Throwable
- Error
  - descreve erros internos e falta de recursos em tempo de execução (raro)
  - programador não trata nem passa objetos deste tipo
- Exception
  - Programador deve tratar esta hierarquia
  - RuntimeException:
    - conversor errado, acesso a array fora dos limites
    - regra: A CULPA é NORMALMENTE do programador !
  - IOException
    - tentativa de ler além do fim de arquivo, URL mal formatada
  - *Suas classes de exceção*

RUNTIMEEXCEPTION



- RuntimeException
- NullPointer, Security, NegativeArrayIndex, Arithmetic, IndexOutOfBounds, ...
- *não é necessário tentar capturar estas exceções*
- *não é necessário especificá-las na cláusula throws*
- Outras subclasses de Exception
- ou
  - lida com exception onde ela pode ocorrer (try { ... } catch { ... })
- ou
  - passa adiante, mas deve fazer declaração em throws

ANUNCIANDO UMA EXCEÇÃO

Método não apenas informa valores a serem retornados, informa também o que pode sair errado

Indica que este método pode lançar uma exceção

```
public static void sleep (long t) throws InterruptedException
```

Tipo da exceção a ser lançada

Um método que executar o método ***sleep*** deve:

- 1. capturar a exceção lançada pelo ***sleep*** usando ***try-catch***

ou

- 2. anunciar a exceção lançada pelo ***sleep*** e repassá-la usando o comando ***throw***.

CAPTURANDO UMA EXCEÇÃO

- capturar a exceção lançada pelo ***sleep*** usando ***try-catch***

```
…
void meuMetodo ( ) {
    try {
            x.sleep(10);
    }
    catch (InterruptedException e) {
            // Tratamento
    }
}
…
```

EXEMPLO DE TRY-CATCH-FINALLY

```
void readFile(String name) throws IOException {
  FileReader file = null;
  try {
    file = new FileReader(name);
    ... // lê o arquivo
  } catch (Exception e) {
    System.out.println(e);
  } finally {
    if (file != null) file.close();
  }
}
```

REPASSANDO UMA EXCEÇÃO

- anunciar a exceção lançada pelo ***sleep*** e repasso-a.

```
…
void meuMetodo ( ) throws InterruptedException {
    x.sleep(10);
}
…
```

Quem executar o método ***meuMetodo( )*** deverá capturar e tratar a exceção ***InterruptedException***

REPASSANDO UMA EXCEÇÃO (OUTRA FORMA)

- anunciar a exceção lançada pelo ***sleep*** e repasso usando o comando ***throw***.

```
…
void meuMetodo ( ) throws InterruptedException {
    try {
            x.sleep(10);
    }
    catch (InterruptedException e) {
            // Faço alguma coisa ...
            throw e;
    }
}
…
```

Repasso a exceção capturada

Quem executar o método ***meuMetodo( )*** deverá capturar e tratar a exceção ***InterruptedException***

Quais exceções um método pode lançar ?

- Um método só pode lançar as exceções explícitas listadas na cláusula ***throws*** ou subclasses dessas exceções
- Um método pode lançar qualquer exceção implícita, mesmo que ela não tenha sido declarada na cláusula ***throws***.

```
…
void meuMetodo ( ) throws Exception {
    if ( i == 0 )
       throw new IOException();
    if (i == -1)
       throw new ArithmeticException ();
}
```

CRIANDO NOVAS EXCEÇÕES

- Para criar novas exceções basta criar classes que extendam as classes de exceções ou suas subclasses.
- Dependendo da superclasse voce poderá ter uma exceção implícita ou explícita.

```
class MinhaExcecao extends Exception {

}
```

USANDO AS NOVAS EXCEÇÕES

```
class testeExcecao {
   void meuMetodo ( ) throws MinhaExcecao {
   }

   void outroMetodo ( ) {
         try {
               meuMetodo();
         } catch (MinhaExcecao e) {
               // trato
         }
   }
}
```

```
class MinhaExcecao
      extends Exception {

}
```

Construtor pode lançar exceção

```
class Person{
  int age;
  Person(int age) throws AgeErrorException{
   if (age<0)
       throw new AgeErrorException("invalid age");
   this.age = age;
  }

  public void setAge(int age) throws AgeErroException{
  if (age<0)
       throw new AgeErrorException("invalid age");
   this.age = age;
  }
}
```

Dica de Depuração

- Objetos de Exceção
- podem imprimir “stack trace” para permitir verificar exceção
- fornecem msg (string) com alguns detalhes
- podem ter outras propriedades específicas de subclasses

```
try
{
    …
}
catch (Throwable t)
{
    t.printStackTrace ();
    throw t;
}
```