Tratamento de Erros e Exceções

Controle de erros com Exceções

- *Exceções são*
  - *Erros de tempo de execução*
  - *Objetos criados a partir de classes especiais que são "lançados" quando ocorrem condições excepcionais*
- *Métodos podem capturar ou deixar passar exceções que ocorrerem em seu corpo*
  - *É obrigatório, para a maior parte das exceções, que o método declare quaisquer exceções que ele não capturar*
- *Mecanismo try-catch é usado para tentar capturar exceções enquanto elas passam por métodos*

- *1. Erros de lógica de programação*
  - *Ex: limites do vetor ultrapassados, divisão por zero*
  - *Devem ser corrigidos pelo programador*
- *2. Erros devido a condições do ambiente de execução*
  - *Ex: arquivo não encontrado, rede fora do ar, etc.*
  - *Fogem do controle do programador mas podem ser contornados em tempo de execução*
- *3. Erros graves onde não adianta tentar recuperação*
  - *Ex: falta de memória, erro interno do JVM*
  - *Fogem do controle do programador e não podem ser contornados*

- *Uma exceção é um tipo de objeto que sinaliza que uma condição excepcional ocorreu*
  - *A identificação (nome da classe) é sua parte mais importante*
- *Precisa ser criada com* ***new*** *e depois lançada com* ***throw***
  - ```
    IllegalArgumentException e = new
                IllegalArgumentException("Erro!");
    throw e;  // exceção foi lançada!
    ```
- *A referência é desnecessária. A sintaxe abaixo é mais usual:*
  - ```
    throw new IllegalArgumentException("Erro!");
    ```

- *Uma declaração throws (observe o "s") é obrigatória em métodos e construtores que deixam de capturar uma ou mais exceções que ocorrem em seu interior*

```
public void m() throws Excecao1, Excecao2 {...}
public Circulo() throws ExcecaoDeLimite {...}
```

- ***throws** declara que o método pode provocar exceções do tipo declarado (ou de qualquer subtipo)*
  - *A declaração abaixo declara que o método pode provocar qualquer exceção (nunca faça isto)*

```
public void m() throws Exception {...}
```

- *Métodos sobrepostos não podem provocar mais exceções que os métodos originais*

- *Uma exceção lançada interrompe o fluxo normal do programa*
  - *O fluxo do programa segue a exceção*
  - *Se o método onde ela ocorrer não a capturar, ela será propagada para o método que chamar esse método e assim por diante*
  - *Se ninguém capturar a exceção, ela irá causar o término da aplicação*
  - *Se em algum lugar ela for capturada, o controle pode ser recuperado*

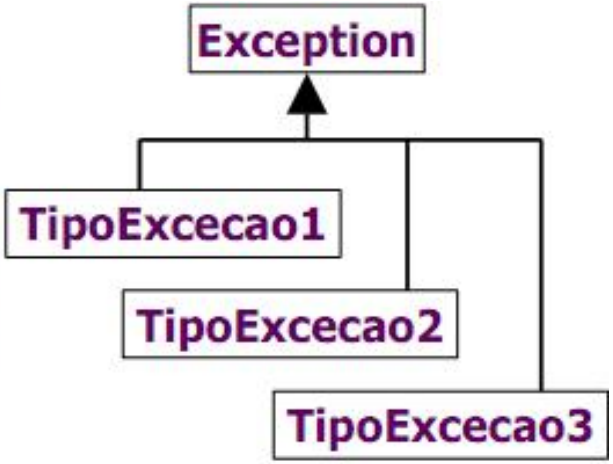
```
public class RelatorioFinanceiro {
    public void metodoMau() throws ExcecaoContabil {
        if (!dadosCorretos) {
            throw new ExcecaoContabil("Dados Incorretos");
        }
    }
    public void metodoBom() {
        try {
            ... instruções ...
            metodoMau();
            ... instruções ...
        } catch (ExcecaoContabil ex) {
            System.out.println("Erro: " + ex.getMessage());
        }
        ... instruções ...
    }
}
```

instruções que sempre serão executadas

instruções serão executadas se exceção não ocorrer

instruções serão executadas se exceção não ocorrer ou se ocorrer e for capturada

- *O bloco try "tenta" executar um bloco de código que pode causar exceção*
- *Deve ser seguido por*
  - *Um ou mais blocos catch(TipoDeExcecao ref)*
  - *E/ou um bloco finally*
- *Blocos catch recebem tipo de exceção como argumento*
  - *Se ocorrer uma exceção no try, ela irá descer pelos catch até encontrar um que declare capturar exceções de uma classe ou superclasse da exceção*
  - *Apenas um dos blocos catch é executado*




```
try {
    ... instruções ...
} catch (TipoExcecao1 ex) {
    ... faz alguma coisa ...
} catch (TipoExcecao2 ex) {
    ... faz alguma coisa ...
} catch (Exception ex) {
    ... faz alguma coisa ...
}

... continuação ...
```

- *O bloco try não pode aparecer sozinho*
  - *deve ser seguido por pelo menos um catch ou por um finally*
- *O bloco finally contém instruções que devem se executadas independentemente da ocorrência ou não de exceções*

```
try {
  // instruções: executa até linha onde ocorrer exceção
} catch (TipoExcecao1 ex) {
   // executa somente se ocorrer TipoExcecao1

} catch (TipoExcecao2 ex) {
   // executa somente se ocorrer TipoExcecao2
} finally {
   // executa sempre ...
}

// executa se exceção for capturada ou se não ocorrer
```

- *A não ser que você esteja construindo uma API de baixo-nível ou uma ferramenta de desenvolvimento, você so usará exceções do tipo (2) (veja página 3)*
- *Para criar uma classe que represente sua exceção, basta estender java.lang.Exception:*

```
class NovaExcecao extends Exception {}
```

- *Não precisa de mais nada. O mais importante é herdar de Exception e fornecer uma identificação diferente*
  - *Bloco catch usa nome da classe para identificar exceções.*

- *Você também pode acrescentar métodos, campos de dados e construtores como em qualquer classe.*
- *É comum é criar a classe com dois construtores*

```
class NovaExcecao extends Exception {
    public NovaExcecao () {}
    public NovaExcecao (String mensagem) {
        super(mensagem);
    }
}
```

- *Esta implementação permite passar mensagem que será lida através de toString() e getMessage()*

- *Construtores de Exception*
  - *Exception ()*
  - *Exception (String message)*
  - *Exception (String message, Throwable cause) [Java 1.4]*
- *Métodos de Exception*
  - *String getMessage()*
    - *Retorna mensagem passada pelo construtor*
  - *Throwable getCause()*
    - *Retorna exceção que causou esta exceção [Java 1.4]*
  - *String toString()*
    - *Retorna nome da exceção e mensagem*
  - *void printStackTrace()*
    - *Imprime detalhes (stack trace) sobre exceção*

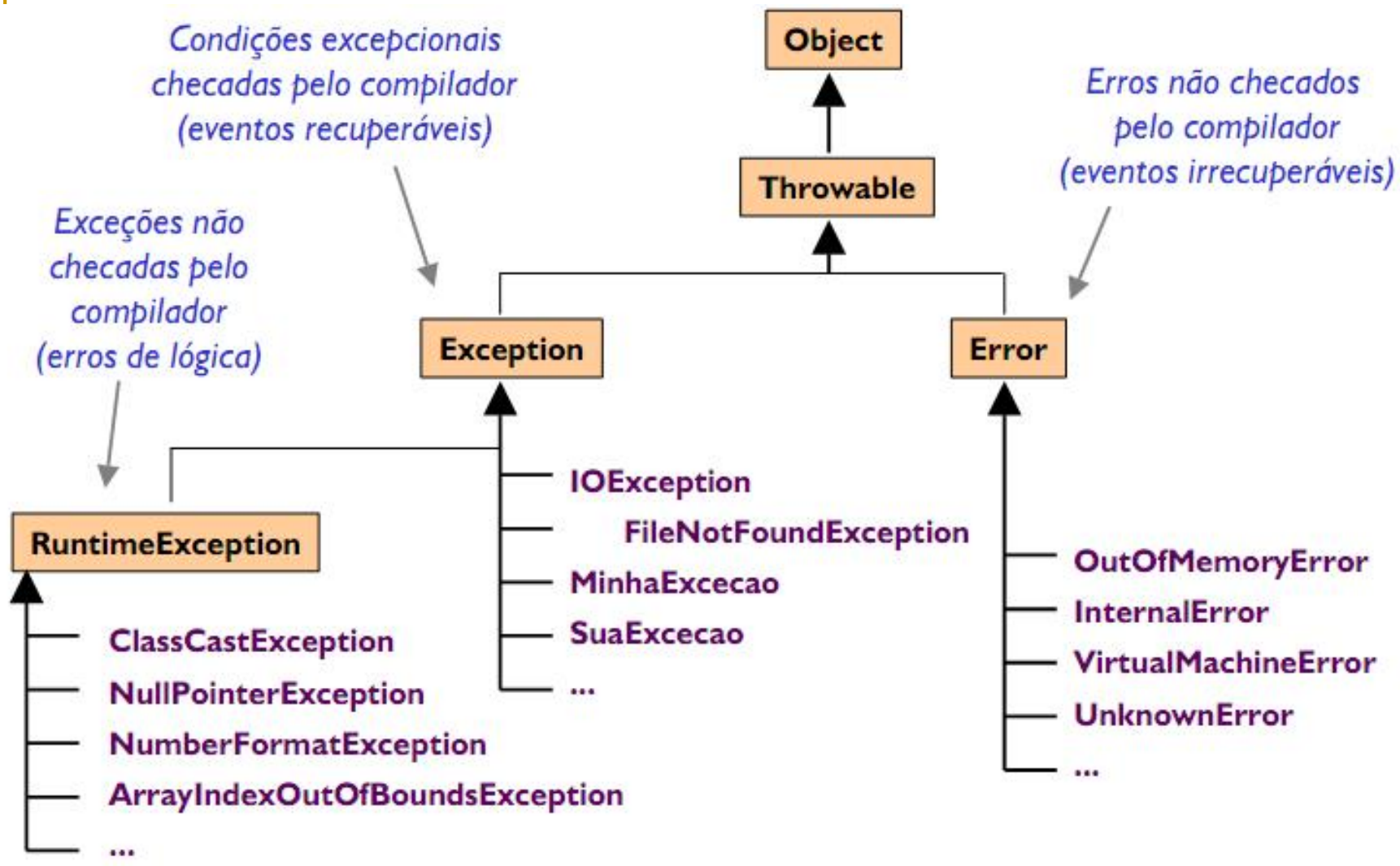
- Se, entre os blocos catch, houver exceções da mesma hierarquia de classes, as classes mais específicas (que estão mais abaixo na hierarquia) devem aparecer primeiro
  - Se uma classe genérica (ex: Exception) aparecer antes de uma mais específica, uma exceção do tipo da específica jamais será capturado
  - O compilador detecta a situação acima e não compila o código
- Para pegar qualquer exceção (geralmente isto não é recomendado), faça um catch que pegue Exception

```
catch (Exception e) { ... }
```

- *Às vezes, após a captura de uma exceção, é desejável relançá-la para que outros métodos lidem com ela*
  - *Isto pode ser feito da seguinte forma*

```
public void metodo() throws ExcecaoSimples {
   try {
      // instruções
   } catch (ExcecaoSimples ex) {
      // faz alguma coisa para lidar com a exceção
      throw ex;  // relança exceção
   }
}
```

- *RuntimeException* e *Error*
  - *Exceções não verificadas em tempo de compilação*
  - *Subclasses de Error não devem ser capturadas (são situações graves em que a recuperação é impossível ou indesejável)*
  - *Subclasses de RuntimeException representam erros de lógica de programação que devem ser corrigidos (podem, mas não devem ser capturadas: erros devem ser corrigidos)*
- *Exception*
  - *Exceções verificadas em tempo de compilação (exceção à regra são as subclasses de RuntimeException)*
  - *Compilador exige que sejam ou capturadas ou declaradas pelo método que potencialmente as provoca*


Condições excepcionais
checadas pelo compilador
(eventos recuperáveis)
Object
Erros não checados
pelo compilador
(eventos irrecuperáveis)
Throwable
Exceções não
checadas pelo
compilador
(erros de lógica)
Exception
Error
IOException
FileNotFoundException
MinhaExcecao
SuaExcecao
...
RuntimeException
OutOfMemoryError
InternalError
VirtualMachineError
UnknownError
...
ClassCastException
NullPointerException
NumberFormatException
ArrayIndexOutOfBoundsException
...

- *Não tratar exceções e simplesmente declará-las em todos os métodos evita trabalho, mas torna o código menos robusto*
- *Mas o pior que um programador pode fazer é capturar exceções e fazer nada, permitindo que erros graves passem despercebidos e causem problemas dificílimos de localizar no futuro.*
- ***NUNCA** escreva o seguinte código:*

```
try {
// .. código que pode causar exceções
} catch (Exception e) {}
```

- *Ele pega até NullPointerException, e não diz nada. O mundo se acaba, o programa trava ou funciona de modo estranho e ninguém saberá a causa a não ser que mande imprimir o valor de e, entre as chaves do catch.*
- *Pior que isto só se no lugar de Exception houver Throwable.*