Tratamento de Exceções

prof. Angelo C. Loula

Exceções

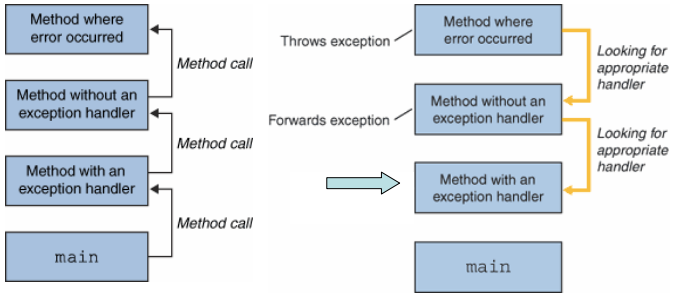
- O que é uma exceção?
  - Algo que ocorre durante a execução do programa que interrompe o fluxo normal das instruções do programa.
  - Quando ocorre um erro em um método, ele cria um objeto e envia para o sistema de execução. Esse objeto é a exceção, e contém informações sobre o erro. Esse processo se chama *lançar uma exceção* (*throwing na exception*).

Exceções

- Uma exceção é uma indicação da ocorrência de alguma ação fora do esperado
  - O usuário fornece um nome de arquivo inválido
  - Um arquivo contém dados corrompidos
  - Um arquivo está sendo usado e portanto está "bloqueado"
  - Um link de rede pode falhar
  - Um servidor pode não estar ativo
  - Um bug pode forçar o acesso a uma posição de array inexistente
  - Etc…

Exceções

- O que é uma exceção?
  - O sistema de execução procura então alguém, na pilha de chamadas, que irá *tratar essa exceção* (*exception handler*).



pilha de chamadas
(call stack)

Exceções

- O que é uma exceção?
  - Para um bloco de código tratar a exceção, ele deve *capturar a exceção.*
  - Se não houver tratamento, o programa é terminado
- Vantagens
  - Separação do tratamento de erro do restante do código
  - Propagação do erro na pilha de chamadas
  - Agrupamento e diferenciação de tipos de erro
    - ex: *FileNotFoundException* é um tipo de *IOException*

Como trabalhar com exceções

- Usa-se o bloco try/catch/finally

```
try { //tente executar...
   //meu codigo
} catch (E1 e1) { //pegue o erro e1
   ...
} catch (En en) { //pegue o erro e2
   ...
} finally { //ao final, execute sempre
   //codigo final
}
```


Como trabalhar com exceções

- O código dentro do ***try*** é o código a ser executado
  - Executa linha a linha até encontrar um problema
- O código dentro do ***catch*** é executado caso seja lançada exceção no *try*
- O código dentro do ***finally*** é sempre executado após o término do try e/ou catch
  - Opcional, ajuda a limpar a casa e liberar recursos!!!

Exemplo de código com exceções

```
public void execoes(){

    try {
           BufferedReader br = new BufferedReader(
                   new FileReader("arquivo.txt"));
           String line = br.readLine();
        } catch(FileNotFoundException e) {
           System.out.println("Arquivo não encontrado.");
        } catch(IOException e) {
           System.out.println("Não foi possível ler de arquivo.txt");
        }

}
```

- Execução:
  - Tenta ler o arquivo
  - Caso não exista, é tratado por um *catch*
  - Caso dê outro erro de leitura, é tratado por outro *catch*

Não tratamento de exceções

- Se uma exceção não for tratada:
  1. A execução passa para o final do método
  2. A exceção passa para o método que o chamou
  3. Se não for tratada, repete passos 1 e 2 até chegar ao "topo" e encerrar o programa

- Se existir um try/catch nesse caminho:
  - A execução passa para o início do catch e depois para o próxima linha de execução

Exceções

- Exceções são objetos comuns, portanto têm uma classe associada
  - Toda exceção herda de Exception
  - Classes com atributos e métodos
  - Dão informações sobre a exceção, causa, etc

```
try {
        BufferedReader br = new BufferedReader(
                new FileReader("arquivo.txt"));
        String line = br.readLine();
    } catch(FileNotFoundException e) {
        System.out.println(e.get);
    } catch(IOException e) {
        System.out.println("N
    }
```

```
getLocalizedMessage() String - Throwable
getMessage() String - Throwable
getCause() Throwable - Throwable
getClass() Class<?> - Object
getStackTrace() StackTraceElement[] - Throwable
```


Exceções

- O *catch* define qual exceção será tratada:
  - É tratado o catch do primeiro supertipo da exceção encontrada
    - Exception é o pai de todas as exceções. Logo, se for o primeiro *catch*, nenhum outro será executado
  - Conclusão: Sempre inicie com as exceções mais específicas, depois coloque as mais genéricas
    - Erro de compilação: Bloco não "alcançável"

Exceções

- FileNotFoundException é um tipo (herda) de IOException

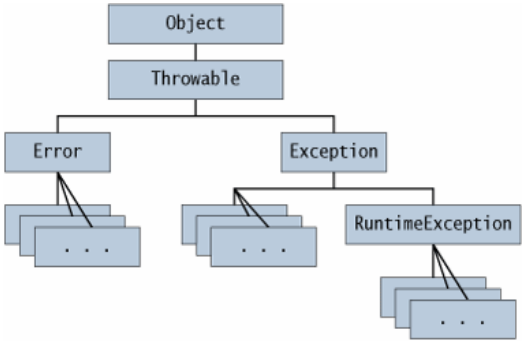
```
public void execoes(){

    try {
            BufferedReader br = new BufferedReader(
                    new FileReader("arquivo.txt"));
            String line = br.readLine();
        } catch(FileNotFoundException e) {
            System.out.println("Arquivo não encontrado.");
        } catch(IOException e) {
            System.out.println("Não foi possível ler de arquivo.txt");
        }

}
```

Tipos de Exceções

- Runtime Exceptions: em tempo de execução
- Error: causas externas à aplicação
- Checked Exceptions: exceções verificadas



java.lang
Class Exception

java.lang.Object
  └ java.lang.Throwable
    └ java.lang.Exception

All Implemented Interfaces:
Serializable

Direct Known Subclasses:
AclNotFoundException, ActivationException, AlreadyBoundException, ApplicationException, AWTException, BackingStoreException, BadLocationException, CertificateException, ClassNotFoundException, CloneNotSupportedException, DataFormatException, DestroyFailedException, ExpandVetoException, FontFormatException, GeneralSecurityException, GSSException, IllegalAccessException, InstantiationException, InterruptedException, IntrospectionException, InvalidMidiDataException, InvalidPreferencesFormatException, InvocationTargetException, IOException, LastOwnerException, LineUnavailableException, MidiUnavailableException, MimeTypeParseException, NamingException, NoninvertibleTransformException, NoSuchFieldException, NoSuchMethodException, NotBoundException, NotOwnerException, ParseException, ParserConfigurationException, PrinterException, PrintException, PrivilegedActionException, PropertyVetoException, RefreshFailedException, RemarshalException, RuntimeException, SAXException, ServerNotActiveException, SQLException, TooManyListenersException, TransformerException, UnsupportedAudioFileException, UnsupportedCallbackException, UnsupportedFlavorException, UnsupportedLookAndFeelException, URISyntaxException, UserException, XAException

java.lang
Class Error

java.lang.Object
  └ java.lang.Throwable
    └ java.lang.Error

All Implemented Interfaces:
Serializable

Direct Known Subclasses:
AssertionError, AWTError, CoderMalfunctionError, FactoryConfigurationError, LinkageError, ThreadDeath, TransformerFactoryConfigurationError, VirtualMachineError

Tipos de Exceções

- Runtime Exceptions: Em tempo de execução
  - Ocorridas em tempo de execução
  - Verifica erros gerais que podem ou não acontecer
  - Não deveriam acontecer: normalmente associadas a bugs e erros de programação
  - Você não é obrigado a tratar (try/catch), pois não deveriam acontecer
  - Lance somente se sabe que quem chamou não pode se recuperar do erro
- Mais comuns:
  - NullPointerExeption (usando um objeto não instanciado)
  - ArrayOutOfBoundsException (acessando uma posicao de um array que não existe)

Trabalhando com RuntimeExceptions

```
try {

   Aluno aluno = turma.getAlunoSabeNada();
   System.out.println("O aluno que não sabe nada é " +
   aluno.getNome());

} catch (NullPointerException e) {

   System.out.println("Erro:" +  e.getMessage());
   e.printStackTrace();

}
```

Tipos de Exceções

- Checked Exceptions: Exceções verificadas
  - Verificadas em tempo de compilação
  - Verificadas pela assinatura e chamada dos métodos
  - Checa se você fez o tratamento das exceções que podem ocorrer na execução da aplicação
  - Ex.: Ao tentar conectar com um banco de dados, deve-se obrigatoriamente tratar se o servidor está disponível

Tratamento de Exceções

- Exceções verificadas permitem ao programador remover o código de tratamento de erros da “linha principal” da execução do programa.
  - Melhora a clareza do programa
  - Separa o tratamento de erros das regras de negócio
  - Permite gerenciar e padronizar como retornar as mensagens ao usuário

Tratamento de Exceções

- Podemos capturar (catch) e/ou lançar a exceção para outro método tratar:

```
throw someThrowableObject;
```

- Exemplo

```
public Object pop() {
  Object obj;
  if (size == 0) {
     throw new EmptyStackException(); }
  obj = objectAt(size - 1);
  setObjectAt(size - 1, null);
  size--;
  return obj;
}
```


Tratamento de Exceções

- Exceções em cascata: podemos responder a uma exceção, lançando outra exceção
- A primeira **causa** a segunda exceção
- Métodos de Throwable para causas

```
Throwable getCause()
Throwable initCause(Throwable)
Throwable(String, Throwable)
Throwable(Throwable)
```

Tratamento de Exceções

- Indicando causas

```
try {
   //instructions
} catch (IOException e) {
    throw new SampleException(
          "Other IOException", e);
}
```

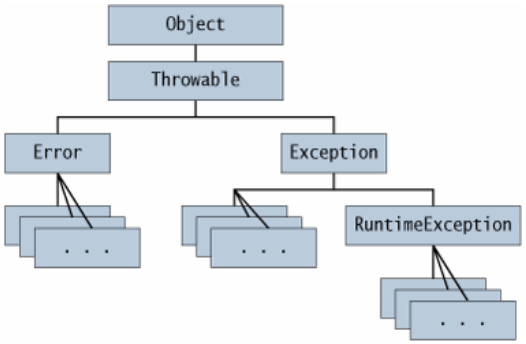

Tratamento de Exceções

- Indicando causas

```
try {
   //instructions
} catch (IOException e) {
    e2=new OtherSampleException();
    e2.initCause(e);
    throw e2;
}
```


Tratamento de Exceções

- Podemos criar nossas próprias exceções como subclasse de Exception



Exemplo

```
public class Conta {
  private String numero;
  private double saldo;

  void debitar(double valor) {
     saldo = saldo - valor;
  }
```

Ação fora do esperado: cliente tenta sacar mais dinheiro do que possui.

Como evitar débitos acima do limite permitido?

Possível Alternativa

```
class Conta {
   private String numero;
   private double saldo;

   void debitar(double valor) {
        if (valor <= saldo)
                saldo = saldo - valor;
 }
}
```

- Como avisar ao cliente que não tem saldo?

Outra Alternativa

Aviso ao objeto que tentou debitar, que a operação não é possível – Lanço uma exceção

```
class Conta {
  private String numero;
  private double saldo;

  void debitar(double valor) {
     if (valor <= saldo)
               saldo = saldo - valor;
          else System.out.print("Saldo
  Insuficiente");
  }
}
```

- E se eu quiser reutilizar esse código em outra aplicação?
- E essa aplicação tiver interface visual?
- E se for uma aplicação *mobile*?
- E se for uma aplicaçào web?

Trabalhando com Exceções verificadas (1)

- Primeiramente, criamos a classe referente à Exceção que queremos tratar
  - Saldo para débito insuficiente

```
public class SaldoInsuficienteException extends
  Exception {
   SaldoInsuficienteException() {
      super("Saldo Insuficiente!");
   }
}
```


Trabalhando com Exceções verificadas (2)

- Depois, as classes com seus métodos
  - Podemos usar a mesma exceção para diferentes métodos (e.g. transferir())

```
public class Conta {
  /* ... */
  void debitar(double v) throws SaldoInsuficienteException {
    if (v <= saldo)
       saldo = saldo - v;
    else {
       throw new SaldoInsuficienteException();
   }
}
```

Trabalhando com Exceções verificadas (3)

- Por fim, tratamos as exceções na hora de executar os métodos

```
/*...*/
try {
  conta.debitar(90.00);
} catch (SaldoInsuficienteException e) {
   System.out.println(e.getMessage());
}
```


Considerações

- O método que chamou o método que lança a exceção pode:
  1. Tratar a exceção
     - No exemplo, imprimindo a informação
  2. *Jogar a exceção para cima*
     - Adicionando um throws na assinatura do método
- Com a opção 2, a responsabilidade para o tratamento passa a ser do método que a chamou
  - O desenvolvedor escolhe onde tratar a exceção: na interface gráfica, por exemplo

Tratamento de Exceções

- Boas práticas

```
public void consumeAndForgetAllExceptions(){
   try {
       ...some code that throws exceptions
   } catch (Exception ex){
       ex.printStacktrace();
   }
}
```

  - Qual o problema desse tratamento?
  - Ocorre a captura do erro, imprime o erro na tela e segue adiante como se nada tivesse acontecido!
  - Sempre dê um tratamento adequado ou lance para cima!

Adaptado de http://onjava.com/onjava/2003/11/19/exceptions.html

Tratamento de Exceções

- Três regras
  - Seja específico
    - Lance exceções específica do tipo de erro
    - Use a hierarquia e herança
  - Lance o mais breve possível
    - Assim que o problema puder ser detectado lance a exceção
    - ex: se já puder determinar que o nome do arquivo é nulo, lance a exceção e não chame a abertura do arquivo
  - Capture o mais tarde possível
    - Capture somente quando puder tratá-lo adequadamente
    - O programa deve se recuperar do erro no lugar apropriado
    - ex: capturar erro de abertura de arquivo, e ali mesmo imprimir a mensagem e pilha de chamada

Adaptado de http://today.java.net/pub/a/today/2003/12/04/exceptions.html