Programação Orientada a Objetos

Erros e Exceções

Dalton Serey

Motivação

- Um programa **interage** com seu ambiente
  - usuários
  - sistema operacional
  - outros programas
- Problema: o ambiente não é confiável
  - usuários fornecem dados errados/inválidos
  - arquivos esperados podem não existir
  - programas podem não cumprir contratos
  - e o próprio programador comete erros

Conceito de Exceção

- Tais situações são chamadas de **exceções**
- O problema é que, por serem exceções, é fácil escrever programas que não as tratam
- Há casos em que
  - ...são erros realmente não esperados
  - ...e outros em que, embora improváveis, tais situações podem ser previstas

Tipos de Exceções

- Falhas catastróficas
  - falta de energia
  - crash de disco
- Situações absolutamente previsíveis
  - divisão por zero
  - uso de referências nulas
- Exceções intermediárias
  - são previsíveis
  - mas de tratamento mais complexo

Como tratar erros?

- O que se espera de um programa, quando tais situações são alcançadas?
  - **exemplo**: o usuário digitou uma série de dados, mas em certo ponto digitou um valor inválido; o que deve ocorrer?
- Espera-se que:
  - o programa se recupere, retorne a um estado seguro e que possa continuar sendo usado
  - ou que salve dados do usuário e que finalize a operação de forma adequada

Detecção de erros

- Para permitir que um erro seja tratado, o programa deve ser capaz de **detectá-lo**
- Isso pode ser tão simples como usar um if

```
if ( divisor == 0 )
   // erro detectado
x = x / divisor;
```

Detecção de erros

- Problema: nem sempre um erro pode ser **detectado** e **tratado** no mesmo lugar
- Precisamos de algum esquema de comunicação entre cliente e provedor
- Em alguns casos, usa-se o valor de retorno como "código" para indicar certas situações

```
linha = entrada.readLine();
if ( linha == null )
   // acabou a entrada de dados
System.out.println(linha);
```


Detecção de Erros

- Problema: há casos em que todo valor de retorno tem significado válido
  - exemplo: métodos que retornam inteiros
  - em alguns casos, -1 pode ser usado
- Solução: **lançar exceções**
  - é uma **saída especial** dos métodos
  - detectado o erro, o **método é interrompido**
  - tipo de **retorno especial**
  - **execução é desviada** para o tratamento

O Objeto Exceção

- Problema: conciliar a detecção e o tratamento de exceções
- Idéia:
  - criar um objeto com a informação referente à exceção no momento da detecção
  - "lançar" o objeto para o código designado para fazer o tratamento da exceção
- A linguagem provê o suporte para deslocar a execução do thread adequadamente

Exceções são declaradas

- Tal como o valor de retorno, as exceções que podem ser lançadas por um método devem ser declaradas
- Isso permite que o programador cliente se prepare para a possível exceção
  - o código provedor detecta e lança exceções,
  - mas é o código cliente quem as trata
- Para declarar que um método lança uma exceção, usamos a cláusula `throws`

Exceções são declaradas

- Na classe `BufferedReader`, há um método para ler linhas, `readLine()`
- Sua assinatura indica que uma exceção do tipo `IOException` pode ser lançada
- Isso é necessário porque o método lê de um recurso externo que pode falhar...

```
public String readLine()
throws IOException
```


Blocos Try/Catch

- Como faz parte da assinatura, o compilador obriga o cliente a manipular a exceção
- Por exemplo, o código abaixo não compila!

```
String linha;
while ( (linha=in.readLine()) != null )
  System.out.println(linha);
```

Blocos Try/Catch

- Eis o código corrigido com o necessário bloco try/catch que manipula a possível exceção

```
String linha;
try {
   while ( (linha=in.readLine()) != null )
      System.out.println(linha);
} catch (IOException e) {
   e.printStackTrace();
}
```


Tratamento de Exceções

- Há três coisas que podemos fazer no catch:
  - tratar a exceção
  - silenciar o erro (não o faça!)
  - relançar (ou propagar) a exceção

```
try {
   código que pode gerar exceção
} catch (Exception e) { } // silencia!
```


Tratamento de Exceções

- Há três coisas que podemos fazer no catch:
  - tratar a exceção
  - silenciar o erro (não o faça!)
  - relançar (ou propagar) a exceção

```
try {
   código que pode gerar exceção
} catch (Exception e) {
   faz tratamento local...
   throw e; //...e relança!
}
```


Tratamento de Exceções

- Há três coisas que podemos fazer no catch:
  - tratar a exceção
  - silenciar o erro (não o faça!)
  - relançar (ou propagar) a exceção

```
try {
   código que pode gerar exceção
} catch (Exception e) {
   //lança outra exceção...
   throw new OutraException();
}
```


Definindo Novas Exceções

- Em geral, podemos usar as exceções disponíveis na API de Java
- Há casos, contudo, em que vale a pena definir exceções específicas para nossa aplicação
- Para fazer isso, devemos estender a classe `Exception` de Java
  - ou a que for mais conveniente
  - veja que quase não há código!

```
class MinhaException extends Exception {}
```


Definindo Novas Exceções

- Esse esquema só cria o construtor default...
- Para adicionar o construtor com String
  - `super(msg)` chama o construtor correspondente da superclasse

```
class MinhaException extends Exception {
  public MinhaException() {}
  public MinhaException(String msg) {
    super(msg)
  }
}
```

RuntimeException

- Algumas exceções não precisam ser declaradas para que sejam lançadas
- Por exemplo, ao acessarmos uma referência null, um `NullPointerException` será lançado, mesmo sem o uso de `throw`
- Problema: tais exceções são lançadas e relançadas por todo o código até o main do programa!
- Conclusão: implicam erros de programação!

Programas Multi-threads

- Threads também são objetos
- Assim, podem ser criadas e manipuladas diretamente por nossos programas!
- Isso permite que, quando conveniente, façamos programas que operem com mais de uma thread ao mesmo tempo...
- A questão é: pra quê?

Exercício

- Escreva um programa que acessa um objeto null. Capture a exceção. Faça o mesmo com uma divisão por zero (uma vez com zero inteiro e outra com zero double).
- Escreva um programa que resolva equações de segundo grau, lendo **a**, **b** e **c** da entrada padrão. A classe que representa equações de segundo grau deve ter métodos delta, numRaizes, raiz1 e raiz2 (raiz1 e raiz2 lançam RaizIndefinidaException se for o caso). A saída do programa consiste em delta, raiz1 e raiz2 (se estiverem definidos) com duas casas decimais.