Modulo II – Tópicos em Java - Extra

Prof. Ismael H F Santos

Ementa

- Modulo II - Tópicos em JAVA - Extra
  - Logging
  - Asserções
  - Classe java.lang.Object e a Interface Cloneable
  - Documentação com Javadoc

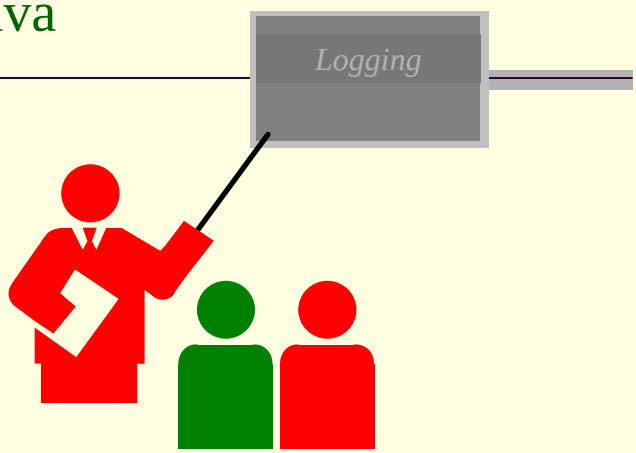
Bibliografia

- *Linguagem de Programação JAVA*
  - *Ismael H. F. Santos, Apostila UniverCidade, 2002*
- *The Java Tutorial: A practical guide for programmers*
  - *Tutorial on-line: http://java.sun.com/docs/books/tutorial*
- *Java in a Nutshell*
  - *David Flanagan, O´Reilly & Associates*
- *Just Java 2*
  - *Mark C. Chan, Steven W. Griffith e Anthony F. Iasi, Makron Books.*
- *Java 1.2*
  - *Laura Lemay & Rogers Cadenhead, Editora Campos*

Livros

- **Core Java 2**, Cay S. Horstmann, Gary Cornell
  - Volume 1 (Fundamentos)
  - Volume 2 (Características Avançadas)
- **Java: Como Programar**, Deitel & Deitel
- **Thinking in Patterns with JAVA**, Bruce Eckel
  - **Gratuito**. http://www.mindview.net/Books/TIJ/

POO-Java



Logging

- *Recurso introduzido no J2SDK 1.4*
- *Oferece um serviço que gera relatórios durante a execução de uma aplicação. Os relatórios são formados por eventos escolhidos pelo programador e podem ter como destino:*
  - *A tela (console), um arquivo, uma conexão de rede, etc.*
- *Os dados também podem ser formatados de diversas formas*
  - *Texto, XML e filtros*
- *As mensagens são classificadas de acordo com a severidade, em sete níveis diferentes. O usuário da aplicação pode configurá-la para exibir mais ou menos mensagens de acordo com o nível desejado*
- *Principais classes*
  - *java.util.logging.Logger e java.util.logging.Level*

Logger

- *Para criar um Logger, é preciso usar seu construtor estático:*

```
Logger logger = Logger.getLogger("com.meupacote");
```

- *Os principais métodos de Logger encapsulam os diferentes níveis de detalhamento (severidade) ou tipos de informação. Métodos log() são genéricos e aceitam qualquer nível*
  - `config(String msg)`
  - `entering(String class, String method)`
  - `exiting(String class, String method)`
  - `fine(String msg)`
  - `finer(String msg)`
  - `finest(String msg)`
  - `info(String msg)`
  - `log(Level level, String msg)`
  - `severe(String msg)`
  - `throwing(String class, String method, Throwable e)`
  - `warning(String msg)`

Exemplo Logging

- *Exemplo da documentação da Sun [J2SDK14]*

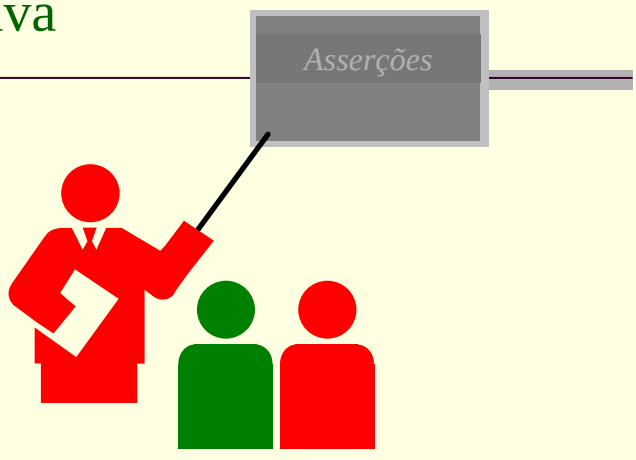
```
package com.wombat;
public class Nose{
   // Obtain a suitable logger.
   private static Logger logger =
      Logger.getLogger("com.wombat.nose");

   public static void main(String argv[]){
     // Log a FINE tracing message
     logger.fine("doing stuff");
     try{
        Wombat.sneeze();
     } catch (Error ex){ // Log the error
       logger.log(Level.WARNING,"trouble sneezing",ex);
     }
     logger.fine("done");
   }
}
```

Níveis de Severidade

- As seguintes constantes da classe Level devem ser usadas para indicar o nível das mensagens gravadas
  - Level.OFF (não imprime nada no log)
  - Level.SEVERE (maior valor - imprime mensagens graves)
  - Level.WARNING
  - Level.INFO
  - Level.CONFIG
  - Level.FINE
  - Level.FINER
  - Level.FINEST (menor valor - imprime detalhamento)
  - Level.ALL (imprime tudo no log)
- Quanto maior o valor escolhido pelo usuário, menos mensagens são gravadas.

POO-Java



Asserções – since JSDK 1.4.0

- Expressões booleanas que o programador define para afirmar uma condição que ele acredita ser verdade
  - Asserções são usadas para validar código (ter a certeza que um vetor tem determinado tamanho, ter a certeza que o programa não passou por determinado lugar, etc)
  - Melhoram a qualidade do código: tipo de teste caixa-branca
  - Devem ser usadas durante o desenvolvimento e desligadas na produção (afeta a performance)
  - Não devem ser usadas como parte da lógica do código
- Asserções são um recurso novo do JSDK1.4.0
  - Nova palavra-chave: assert
  - É preciso compilar usando a opção -source 1.4:
    - >javac -source 1.4 Classe.java
  - Para executar, é preciso habilitar afirmações (enable assertions):
    - >java -ea Classe

Asserções: sintaxe

- Asserções testam uma condição. Se a condição for falsa, um AssertionError é lançado
  - Sintaxe:
    - assert expressão;
    - assert expressãoUm : expressãoDois;
- Se primeira expressão for true, a segunda não é avaliada. Sendo falsa, um AssertionError é lançado e o valor da segunda expressão é passado no seu construtor.

Em vez de usar comentário...

```
if (i%3 == 0) {
   ...
} else if (i%3 == 1) {
   ...
} else { // (i%3 == 2)
   ...
}
```

... use uma asserção!.

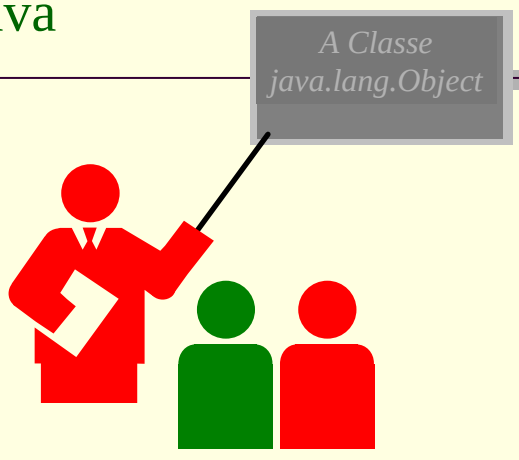
```
if (i%3 == 0) {
   ...
} else if (i%3 == 1) {
   ...
} else {
   assert i%3 == 2;
   ...
}
```

Asserções: exemplo

- Trecho de código que afirma que controle nunca passará pelo default:

```
switch(estacao) {
      case Estacao.PRIMAVERA:
         ...
      break;
      case Estacao.VERAO:
         ...
      break;
      case Estacao.OUTONO:
         ...
      break;
      case Estacao.INVERNO:
         ...
      break;
      default:
         assert false: "Controle nunca chegará aqui!";
}
```


POO-Java



Classe java.lang.Object

- Modelo de Classes de Java
  - Conforme já dissemos anteriormente em Java a classe Object é a raiz da hierarquia de classes à qual todas as classes existentes pertencem. Quando não declaramos que uma classe estende outra, ela implicitamente estende Object;
  - Uma das vantagens de termos uma superclasse comum, é termos uma funcionalidade comum a todos os objetos. Principais métodos de Object

```
public boolean equals(Object obj)
public String toString()
public int hashCode()
protected Object clone() throws CloneNotSupportedException
public void wait() throws InterruptedException
public void notify()
```


Redefinindo métodos de Object

- Há vários métodos em Object que podem ser sobrepostos pelas subclasses
  - A subclasse que você está estendendo talvez já tenha sobreposto esses métodos mas, alguns deles, talvez precisem ser redefinidos para que sua classe possa ser usada de forma correta
- Métodos que devem ser sobrepostos
  - boolean equals(Object o): Defina o critério de igualdade para seu objeto
  - int hashCode(): Para que seu objeto possa ser localizado em Hashtables
  - String toString(): Sobreponha com informações específicas do seu objeto
  - Object clone(): se você desejar permitir cópias do seu objeto

Redefinindo métodos de Object

- Determine quais os critérios (que propriedades do objeto) que podem ser usados para dizer que um objeto é igual a outro
  - *raio, em um objeto Círculo*
  - *número de série, em um objeto genérico*
  - *nome, sobrenome, departamento, para um objeto Empregado*
  - *A chave primária, para um objeto de negócio*
- Implemente o equals(), testando essas condições e retornando true apenas se forem verdadeiras (false, caso contrário)
  - *Garanta que a assinatura seja igual à definida em Object*

Redefinindo métodos de Object

- *instanceof é um operador usado para comparar uma referência com uma classe*
  - *A expressão será true se areferência for do tipo de uma classe ou subclasse testada e false, caso contrário*
- *Exemplo: sobreposição de equals()*

```
class Point {
  private int x, y;
  public boolean equals(Object obj) {
    if ( obj instanceof Point ) {
      Point ponto = (Point)obj;
      if ( ponto.x == this.x && ponto.y == this.y) {
     return true;
    }
    }
  return false;
 }
}
```

Agora posso usar: if(p1.equals(p2))... !

Sobrecarregando toString()

- toString() deve devolver um String que possa representar o objeto quando este for chamado em uma concatenação ou representado como texto
  - *Decida o que o toString() deve retornar*
  - *Faça chamadas super.toString() se achar conveniente*
  - *Prefira retornar informações que possam identificar o objeto (e não apenas a classe)*
  - *toString() é chamado automaticamente em concatenções usando a referência do objeto*

Sobrecarregando hashCode()

- *hashCode() deve devolver um número inteiro que represente o objeto*
  - *Use uma combinação de variáveis, uma chave primária ou os critérios usados no equals()*
  - *Número não precisa ser único para cada objeto mas dois objetos iguais devem ter o mesmo número.*
- *método hashCode() é chamado automaticamente quando referências do objeto forem usadas em coleções do tipo hash (Hashtable, HashMap)*
- *equals() é usado como critério de desempate, portanto, se implementar hashCode(), implemente equals() também.*

Interface Cloneable

- *Interface Cloneable*
  - *Usada para permitir que um objeto seja clonado/copiado. Não possui declaração de métodos (Marker Interface).*
  - *Indica para o método Object.clone() que o mesmo pode fazer uma cópia campo a campo quando tiver que clonar (criar via copia) uma nova instância da classe.*
  - *Como fazer:*

    class MyClass implements Cloneable

    class MyClass extends SuperClass implements Cloneable

Interface Cloneable

- *Interface Cloneable (cont.)*
  - Exemplo:

```
Point p1, p2, p3;
p1 = new Point(0, 0);
p2 = p1;        // p2 e p1 se referenciam ao mesmo obj
p3 = (Point)p1.clone(); // p3 novo obj criado com os valores p1
```

Sobrecarregando Clone()

- *clone() é chamado para fazer cópias de um objeto*

Circulo c = new Circulo(4, 5, 6);
Circulo copia =(Circulo)c.clone();

- *Se o objeto apenas contiver tipos primitivos como seus campos de dados, é preciso*
  - *1. Declarar que a classe implementa Cloneable*
  - *2. Sobrepor clone() e invocar o método Object.clone() que faz uma copia bit-wise dos atributos do objeto*

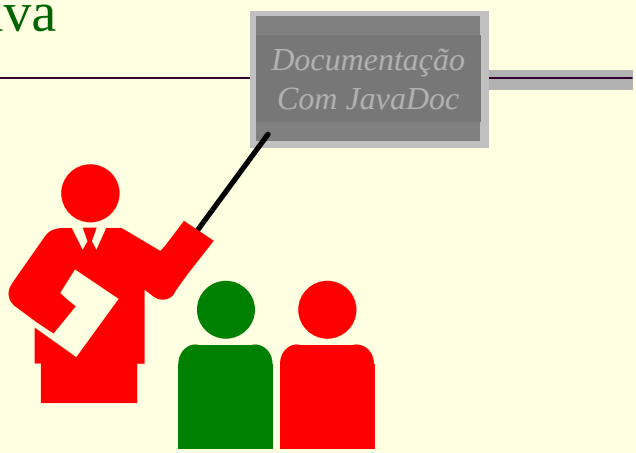
```
public Object clone() {
  try {
    return super.clone();
  } catch (CloneNotSupportedException e) {
    return null;
  }
}
```


Sobrecarregando Clone()

- Se o objeto contiver campos de dados que são referências a objetos, é preciso fazer cópias desses objetos também

```
public class Circulo {
 private Point origem;
 private double raio;
 public Object clone() {
  try {
    Circulo c = (Circulo)super.clone();
    c.origem = (Point)origem.clone(); // Point clonável!
    return c;
  } catch (CloneNotSupportedException e) {return null;}
 }
}
```

POO-Java



Documentação

- Ferramenta **javadoc**
- Documentação a partir de comentários, colocados no código fonte
- Formato HTML: permite visualização via *browser*
- Manual do usuário × Guia de referência
- A saída (conteúdo e formato) gerada pelo Javadoc pode ser customizada através do uso de doclets.

Uso do **javadoc**

- Comentário especiais /** **... */**
  - se referem ao próximo identificador definido
  - permitem o uso de *tags* HTML
- Parágrafos especiais
  - documentam assinaturas de métodos
  - fazem referências cruzadas
  - sinalizam código depreciado
  - identificam autoria

Exemplo

```
package org.enterprise.myapp;
/**
 * Classe exemplo. Documentada com javadoc.
 * <p>
 * Exemplo de uso tags HTML (see the {@link Exemplo2} class)
 * @since 1.0
 * @see org.enterprise.myapp
 */
public class Exemplo {
  /**
   * Divide um número por dois. Esse método retorna a
   * divisão inteira por 2 do número fornecido.
   * @param i número a ser dividido.
   * @return resultado da divisão inteira por 2.
   */
  public int div2(int i) { return i/2; }
}
```

Parágrafo **@param**

- Documenta um parâmetro de um método
- Recebe o nome do parâmetro e sua descrição
- Exemplo:

```
@param id Identificador a ser buscado
```


Parágrafo **@return**

- Documenta o valor de retorno de um método
- Recebe a descrição do valor
- Exemplo:

```
@return Nome do elemento encontrado
```

Parágrafo **@exception**

- Documenta uma exceção gerada por um método
- Recebe o tipo da exceção e sua descrição
- Exemplo:

```
@exception IdNotFound Identificador não encontrado
```


Parágrafo **@see**

- Cria de uma referência cruzada
- Recebe o nome de um identificador
- Exemplo:

```
@see estruturas.Coleção#insere
```

Parágrafo **@deprecated**

- Marca um identificador como depreciado
- Qualquer código que utilize o identificador receberá um aviso em tempo de compilação
- Recebe uma descrição
- Exemplo:

```
@deprecated Esse método foi descontinuado. Use o método
   xxx.
```


Parágrafo **@author**

- Identifica o autor do código
- Recebe um nome
- Exemplo:

```
@author João José
```

Parágrafo **@version**

- Identifica a versão do código
- Recebe o identificador da versão
- Exemplo:

```
@version 1.0β
```


Parágrafo **@since**

- Especifica a versão onde o identificador foi introduzido
- Recebe o identificador da versão
- Exemplo:

```
@since 1.0β
```