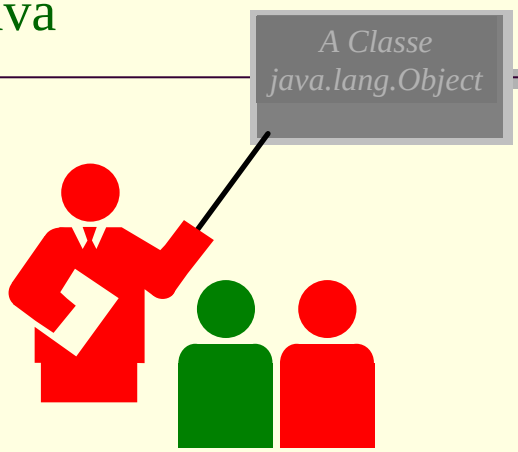
Modulo IIa
Extra: Object

Professor
Ismael H F Santos – ismael@tecgraf.puc-rio.br

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*

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;}
 }
}
```