Orientação a Objetos

Pacotes, Construtores e This

Profa.: Márcia Sampaio Lima

EST - UEA

Packages

- **Pacotes (Packages)**
  - São usados para organizar as classes semelhantes.
  - A grosso modo, são apenas pastas ou diretórios do sistema operacional onde ficam armazenados os arquivos fonte de Java.
  - Java possui um pacote padrão que é utilizado quando programamos, mesmo embora não seja recomendado usá-lo (default package).
- **Criar Pacotes**
  - A criação de pacotes é opcional. **Package** é usado em Java para agrupar classes.

Packages

- Definição feita pela palavra reservada **package**.
- **Package** deve ser a primeira linha de comando a ser compilada no código fonte da classe.
- Se tivéssemos criado um pacote chamado **modelo** e fossemos criar uma classe nesta pasta (pacote), o começo de nosso código seria: **package modelo**;.

Packages - Exemplo

```
package animais;

public class Gato {

   String nome;

   public void miar(){
      System.out.println("Miou..");
   }
}
```

Packages - Exemplo

```
package animais;

public class Cao {

  String nome;

  public void latir(){
     System.out.println(" Latiu..");
  }
}
```

Convenções de Nomenclatura do Programa Java - Concatenação

- A concatenação é a prática de criar strings com palavras juntas em maiúsculas e sem espaço.

- Concatenação com minúsculas
  - Por exemplo: exemploConcatenacao.
- Concatenação com maiúscula
  - Por exemplo: ExemploConcatenacao.

Convenções de Nomenclatura do Programa Java

- O nome do pacote é definido antes de uma instrução de importação na concatenação com minúsculas.
- As instruções de importação são definidas abaixo do nome do pacote.
- O nome da classe é um nome que usa concatenação com maiúscula.

Convenções de Nomenclatura do Programa Java - Concatenação

- Os nomes de variáveis são curtos, mas significativos em concatenação com minúsculas.
- Os nomes de constantes são declarados em letras maiúsculas com o modificador **final**.
- Métodos são verbos nomeados em concatenação com minúsculas.

Packages - Exemplo

```
package animais;

public class Cao {

   String nome;

   public void latir(){
      System.out.println(" Latiu..");
   }
}
```

Nome Classe

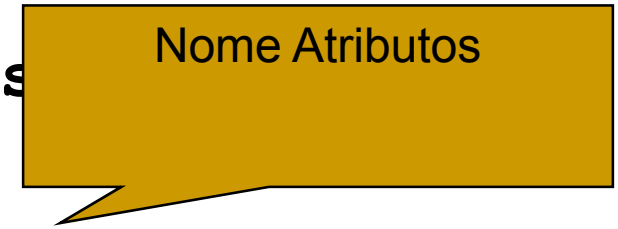
Packages - Exemplo

```
package animais;

public class

   String nome;

   public void latir(){
      System.out.println(" Latiu..");
   }
}
```



Packages - Exemplo

```
package animais;

public class Cao {

   String nome;

   public void latir(){
      System.out.println(" Latiu..");
   }
}
```

Packages - Exemplo

```
package execucao;
Import animais.Gato*;
public class Execucao {

  public static void main (String [] args){
     Gato gato = new Gato();
     gato. Miar();
  }
}
```

Packages - Importar

- **Importar Pacotes**
  - Java possui vários pacotes com outros pacotes internos e várias classes já prontas para serem utilizadas.
  - Exemplo:
    - O pacote AWT (*Abstract Windowing Toolkit*) possui as classes necessárias para se criar um ambiente gráfico API (Janelas).
  - Para utilizar as milhares de classes contidas nos inúmeros pacotes de Java devemos ou nos referenciar diretamente a classe ou importá-la.

Import

- Você pode importar uma única classe ou um pacote inteiro.
- Você pode incluir várias instruções de importação.
- As instruções de importação seguem a instrução do pacote e precedem a declaração da classe.
  - import java.util.Scanner;

Packages - Importar

- **Importar Pacotes**
  - Usamos o comando **import**.
  - Para separar um pacote de seu sub-pacote usamos um ponto.
    - Ex.: import javax.swing.JOptionPane;
    - Isso irá importar apenas JOptionPane do pacote SWING.

Packages - Importar

- **Importar Pacotes**
  - O asterísco (*) serve para importar todos os sub-pacotes e classes do pacote que está definido.
    - Ex.: import java.awt.*;
    - Isso importará todos os sub-pacotes pertencentes ao pacote AWT.
    - **Como o * importa todos os sub-pacotes, o consumo de memória será alto e, provavelmente, não usaremos todas as classes de todos os pacotes importados. Por isso, o recomendado é sempre importar apenas o pacote que será utilizado.**

Exemplo Import

```
import java.util.Date;                  // Import the individual class.

import java.util.*;                     //Import the entire package.

import java.util.Date;                  //Import using multiple statements.
import java.util.Calendar;
```

Import

É sempre uma boa pratica o uso da instrução import de forma explícita (java.util.List) ao invés do uso da forma implícita (java.util.*).

Essa é uma boa pratica porque favorece ao programador que ele determine rapidamente quais classes foram importadas.

Packages - Importar

```
package modelo;

public class Calculadora {

   public void geraRandomico(){

       System.out.println(Math.random());

   }

}
```

Packages - Importar

```
package calcular;

import modelo.Calculadora;


public class Execucao {

   public static void main (String [] args){
        Calculadora calc = new Calculadora();
        calc. geraRandomico();
   }

}
```

Conflitos no Import

- Problema na importação de classes homonimas porém de pacotes diferentes:

Listagem 1. Importação com erro de compilação

```
package com.wordpress.mballem.artigos.imports;

import java.awt.List;
import java.util.List;
import java.util.ArrayList;

public class TesteImports {
        public static void main(String[] args) {
                List awtList = new List();
                List utilList = new ArrayList();
        }
}
```

Conflitos no Import

- No método main() temos duas variáveis locais e ambas do tipo List.
  - awtList se referente a classe java.awt.List
  - utilList se refere a classe java.util.List.

- Como o compilador saberia qual classe está sendo usada em determinado momento de sua declaração?

Conflitos no Import

- Solução: incluir, em pelo menos uma das declarações do método main() a importação da classe desejada e eliminar a instrução import referente a esta importação.

Listagem 2. Corrigindo a importação – 1ª Forma.

```
import java.awt.List;
import java.util.ArrayList;

public class TesteImports {
        public static void main(String[] args) {
                List awtList = new List();
                java.util.List utilList = new ArrayList();
        }
}
```

- Solução:

Listagem 3. Corrigindo a importação – 2ª Forma.

```
import java.util.ArrayList;
import java.util.List;

public class TesteImports {
        public static void main(String[] args) {
                java.awt.List awtList = new java.awt.List();
                List utilList = new ArrayList();
        }
}
```

Construtores

- **Método Construtor**
  - O método construtor é desenvolvido da mesma forma que uma função.
  - Possui o mesmo nome da classe.
  - Isso se deve ao fato de que um objeto deve ser construído cada vez que chamamos a classe.
  - E a responsabilidade de fazer isso é do construtor.
  - Sempre que criamos uma classe, Java automaticamente vincula um método construtor padrão interno com o mesmo nome da classe, mas sem inicializar nenhum atributo.

Construtores

- Construtor:
  - É um procedimento (semelhante a um método) especial cuja principal função é criar um objeto.
  - Não retorna valor algum, nem void!
  - Pode receber parâmetros.
  - Possui o nome igual ao nome da classe.

Construtores

- Construtor:
  - Você pode declarar mais de um construtor em uma declaração de classe.
  - Não é necessário declarar um construtor, o Java fornecerá um construtor (em branco) padrão para você.
  - Se você declarar um ou mais construtores, o Java não fornecerá um construtor padrão.

Construtores

```
public class Gato {

   String nome;
   double peso;
   int idade;

   public Gato(){
        nome = "";
        peso = 0;
        idade = 0;
   }

   public Gato(String Pnome, double Ppeso, int Pidade){
        nome = Pnome;
        peso = Ppeso;
        idade = Pidade;
   }
        ........
}
```

Construtores com Parâmetros

- Se você criar um construtor com parâmetros (o parêntese NÃO ficará vazia), você também poderá inicializar as variáveis entre o { e }.
- Esse construtor inicializará as variáveis de classe com os valores que são enviados a partir da principal classe.

Construtores

```
public class Gato {

   String nome;
   double peso;
   int idade;

   public Gato(){
        nome = "";
        peso = 0;
        idade = 0;
   }

   public Gato(String Pnome, double Ppeso, int Pidade){
        nome = Pnome;
        peso = Ppeso;
        idade = Pidade;
   }
        ........
}
```

- Exemplo

```
public Student(int x, String n, String s, double g){
        studentid = x;
        name = n;
        ssn = s;
        gpa = g;
}
```

Construtores

```
package animais;


public class principal {

   public static void main(String[] args) {

      Gato cat1 = new Gato();
      cat1.nome = "fred";
      cat1.idade = 60;
      cat1.peso = 2;

      Gato cat2 = new Gato("tom",40,1);

   }
}
```

This – auto-referência

- *This* é usado para fazer auto-referência ao próprio contexto em que se encontra.

- *This* sempre será a própria classe ou o objeto já instanciado.

This – auto-referência

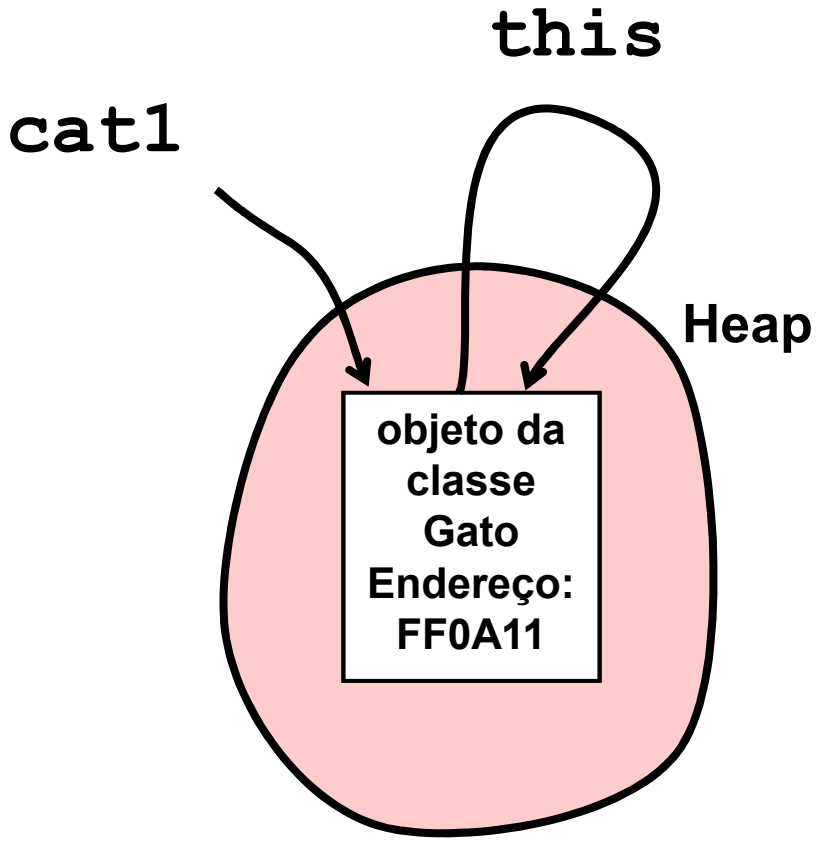
- Vejamos uma instanciação:

```
Gato cat1 =
        new Gato("gatinho", 2,1);
```

- Quando um objeto gato é instanciado, a variável **cat1** é uma referência para o objeto gato real armazenado na memória.
- O **this** é uma referência que um objeto possui que aponta para o seu próprio endereço.

This – auto-referência

- A idéia é esta...



This – auto-referência

- Por exemplo, se criarmos um método que receba um argumento com o mesmo nome do atributo da classe devemos diferenciar ambos com o *this*. Como *this* se refere ao contexto empregado, então o usamos para identificar os atributos da classe.

This – auto-referência

```
public class Gato {

    String nome;
    double peso;
    int idade;

    public Gato(){
        this.nome = "";
        this.peso = 0;
        this.idade = 0;
    }

    public Gato(String nome, double peso, int idade){
        this.nome = nome;
        this.peso = peso;
        this.idade = idade;
    }
}
```

This – auto-referência

```
public Gato(String nome, double peso, int idade){
      this.nome = nome;
      this.peso = peso;
      this.idade = idade;
   }
```

- O **this** aqui, foi usado para diferenciar os nomes dos atributos, dos nomes dos parâmetros do construtor. Se os nomes fossem diferentes, não seria necessário o uso do **this**.

Exercício

Calculadora
Double soma(double,double); Double subtracao(double,double); Double multiplicacao(double,double); Double divisao(double,double); Double exponenciacao(double,double); Double raizquadrada(double); Double geraRandomico(double,double) { ((LS-LI)*Math.randon())+LI} Double log(double);

Exercício

- Criar um vetor de 6 Gatos.
- Inicializar os atributos de gatos através do método construtor.
- Mostrar todos os dados de todos os gatos usando estrutura de repetição FOR.