Programação Orientada a Objetos em

Flávio Miguel Varejão

Departamento de Informática

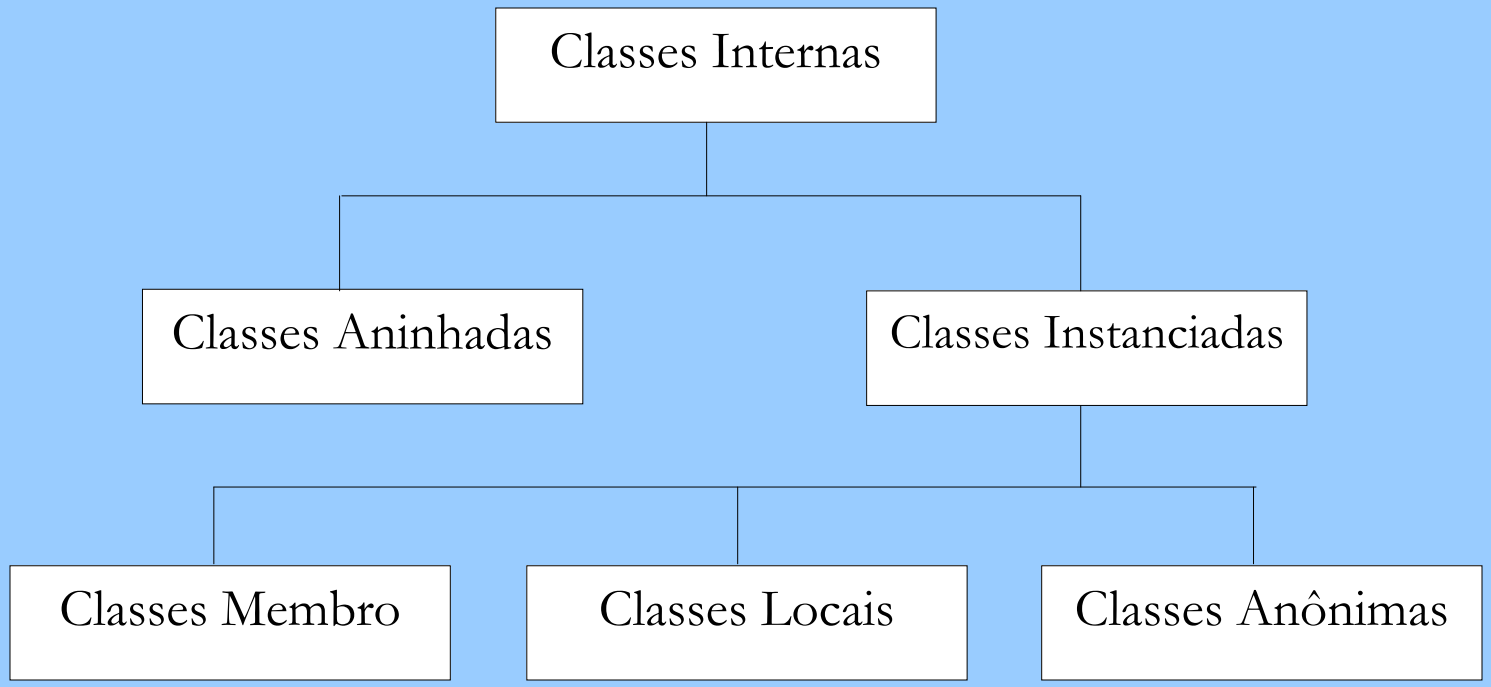
UFES

Classes Internas

- Classes declaradas dentro de outras classes
- Motivação
  - Legibilidade
    - proximidade de classes logicamente relacionadas
  - Ocultamento
    - classes auxiliares de implementação não se tornam acessíveis para outras classes do pacote
  - Redigibilidade
    - acesso livre aos atributos das classes associadas

Classes Internas

- Quatro tipos:



Classes Internas

- Arquivo *.java* onde classe interna é declarada gera vários *.class*
  - *Externa.java*
    - *Externa.class*
    - *Externa$Interna.class*
- Classes internas não podem ter o mesmo nome das classes que as contém

Classes Aninhadas

- Usadas para agrupar classes relacionadas
- Classes iguais as outras, apenas declaradas internamente a uma classe
- Declaração deve incluir palavra *static*
- Referenciada através de

  Externa.Interna
- Não pode haver classes aninhadas declaradas dentro de uma classe interna
- Classes aninhadas podem ter classes internas
- Especificadores de acesso são aplicados

Classes Aninhadas

```
// Externa.java
public class Externa {
  int i;
  static int j;
  public static class Interna {
    int k;
    public void naoFazNada() {
      //! i = 10; ** i nao eh estatico **
      j = 10;
    }
  }
  public void tambemNaoFazNada() {}
}
```

Classes Aninhadas

```
// TestaAninhada.java
public class TestaAninhada {
  public static void main (String [] args) {
     Externa e = new Externa();
     e.tambemNaoFazNada();
     Externa.Interna ei = new Externa.Interna();
     ei.naoFazNada();
  }
}
```

Classes Aninhadas

```
// Externa.java
public class Externa {
  private int i;
  private static int j;
  public static class Interna {
    int k;
    public void naoFazNada(Externa e) {
      e.i = 3;
      //! i = 10; ** i nao estatico **
      j = 10;
    }
  }
  public void tambemNaoFazNada() {
     System.out.println("i = " + i + "\nj= " + j);
  }
}
```

Classes Aninhadas

```
// TestaAninhada.java
public class TestaAninhada {
  public static void main (String [] args) {
     Externa e = new Externa();
     e.tambemNaoFazNada();
     Externa.Interna ei =
          new Externa.Interna();
     ei.naoFazNada(e);
     e.tambemNaoFazNada();
  }
}
```

Classes Instanciadas

- Toda instância destas classes internas é associada internamente a uma instância da classe externa
- Os métodos da classe interna podem se referir aos atributos da classe externa (mesmo os *private*) e vice-versa
- Não podem possuir atributos e métodos de classe

Classes Membro

- Declaradas onde são declarados os atributos da classe externa
- Uso de *this* dentro da classe interna se refere a objeto da classe interna
- Para acessar o objeto da classe externa dentro da interna usa-se a sintaxe *Externa.this*
- Nova sintaxe para permitir a criação de objetos das classes membro: *this.new Interna();*

Classes Membro

```
public class A {
   public String nome = "a";
   public class B {
      public String nome = "b";
      public class C {
         public String nome = "c";
         public void imprime() {
            System.out.println (nome);
            System.out.println (this.nome);
            System.out.println (C.this.nome);
            System.out.println (B.this.nome);
            System.out.println (A.this.nome);
         }
      }
   }
}
```

Classes Membro

```
class Instanciadas {
  public static void main (String [] args) {
    A a = new A();
    A.B b = a.new B();
    A.B.C c = b.new C();
    c.imprime();
  }
}
```

Classes Membro

```
public class Embrulho1 {
  class Conteudo {
    private int i = 11;
    public int valor() { return i; }
  }
  class Destino {
    private String etiqueta;
    Destino(String aonde) { etiqueta = aonde; }
    String lerEtiqueta() { return etiqueta; }
  }
```

Classes Membro

```
  public void envio(String dest) {
    Conteudo c = new Conteudo();
    Destino d = new Destino(dest);
    System.out.println(d.lerEtiqueta());
  }
  public static void main(String[] args) {
    Embrulho1 emb = new Embrulho1();
    emb.envio("Tanzania");
  }
}
```

Retorno de Classes Membro

```
public class Embrulho1 {
  class Conteudo {
    private int i = 11;
    public int valor() { return i; }
  }
  class Destino {
    private String etiqueta;
    Destino(String aonde) {
      etiqueta = aonde;
    }
    String lerEtiqueta() { return etiqueta; }
  }
  public Destino destinatario(String s) {
    return new Destino(s);
  }
```

Retorno de Classes Membro

```
  public Conteudo cont() {
    return new Conteudo();
  }
  public void envio(String dest) {
    Conteudo c = cont();
    Destino d = destinatario(dest);
    System.out.println(d.lerEtiqueta());
  }
}
```

Retorno de Classes Membro

```
public class Embrulho2 {
  public static void main(String[] args) {
    Embrulho1 p = new Embrulho1();
    p.envio("Tanzania");
    Embrulho1 q = new Embrulho1();
    Embrulho1.Conteudo c = q.cont();
    Embrulho1.Destino d =
            q.destinatario("Bornéo");
  }
}
```

Classes Membro e Upcast

```
// Destino.java
public interface Destino {
  String lerEtiqueta();
}
// Conteudo.java
public interface Conteudo {
  int valor();
}
// Embrulho3.java
public class Embrulho3 {
  private class PConteudo implements Conteudo {
    private int i = 11;
    public int valor() { return i; }
  }
```

Classes Membro e Upcast

```
  protected class PDestino implements Destino {
    private String etiqueta;
    private PDestino(String aonde) {
      etiqueta = aonde;
    }
    public String lerEtiqueta() { return etiqueta; }
  }
  public Destino dest(String s) {
    return new PDestino(s);
  }
  public Conteudo cont() {
    return new PConteudo();
  }
}
```

Classes Membro e Upcast

```
class TesteInternas {
  public static void main(String[] args) {
    Embrulho3 p = new Embrulho3();
    Conteudo c = p.cont();
    Destino d = p.dest("Tanzania");
    // Não é possível acessar uma classe privada:
    //! Embrulho3.PConteudo pc = p.new PConteudo();
  }
}
```

Classes Locais

- Declaradas dentro de um bloco de código em Java
- Só são visíveis e usáveis dentro do bloco onde foram declaradas
- Podem usar apenas as variáveis locais e os parâmetros que são *final*
- Vantagem é declaração no ponto de uso - melhora a legibilidade

Classes Locais

```
// Embrulho4.java
public class Embrulho4 {
  public Destino dest(String s) {
    class PDestino implements Destino {
      private String etiqueta;
      private PDestino(String aonde) {
        etiqueta = aonde;
      }
      public String lerEtiqueta() { return etiqueta; }
    }
    return new PDestino(s);
  }
}
```

Classes Locais

```
// Lista.java
public class Lista {

  private class No {
    Object info;
    No prox;
    No (Object o) {
      info = o;
      prox = null;
    }
  }

  No prim, marc;
  int tam;
```

Classes Locais

```
public Lista() {
   prim = marc = null;
   tam = 0;
 }

 public void inicio() {
   marc = prim;
 }

 public Object proximo () {
   if (marc != null) marc = marc.prox;
   return marc;
 }
```

Classes Locais

```
public void inclui (Object o) {
   No p = prim, q = null, n = new No(o);
   while (p != null) {
     q = p;
     p = p.prox;
   }
   if (q == null) {
     prim = n;
   } else {
     q.prox = n;
   }
   tam++;
 }
 // outros metodos
}
```

Classes Anônimas

- Classes locais sem nome
- Usadas quando classe só é necessária uma única instância da classe - exemplo de GUIs
- *.java* gera *Externa$1.class*
- Não possui construtores
  - *new* seguido de nome de classe da qual herda
  - *new* seguido de nome de interface - herda de *Object*
- Parâmetros na criação são passados para o construtor da superclasse

Classes Anônimas

```
// Embrulho5.java
public class Embrulho5 {
  public Conteudo cont() {
    return new Conteudo() {
      private int i = 11;
      public int valor() { return i; }
    }; // Ponto e Vírgula obrigatório neste caso
  }
  public static void main(String[] args) {
    Embrulho5 p = new Embrulho5();
    Conteudo c = p.cont();
  }
}
```

Classes Anônimas

```
buttonPane.setLayout(new FlowLayout());
button = new JRadioButton("Single");
button.addActionListener(new AbstractAction() {
    public boolean isEnabled() { return true; }
    public void actionPerformed(ActionEvent e) {
     tree.getSelectionModel().setSelectionMode
         (TreeSelectionModel.SINGLE_TREE_SELECTION);
    }
});
```

Classes Locais X Classes Anônimas

- Locais
  - mais de uma instância
  - construtores são necessários
- Anônimas
  - pouco código
  - somente uma instância
  - uso imediato após a declaração
  - nome da classe não aumenta legibilidade