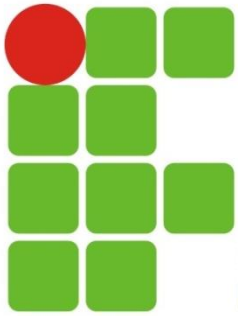


INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
RIO GRANDE DO NORTE

Análise e Projeto Orientados a Objetos

Diagrama de classes de projeto

Diretoria Acadêmica de Gestão e Tecnologia da Informação

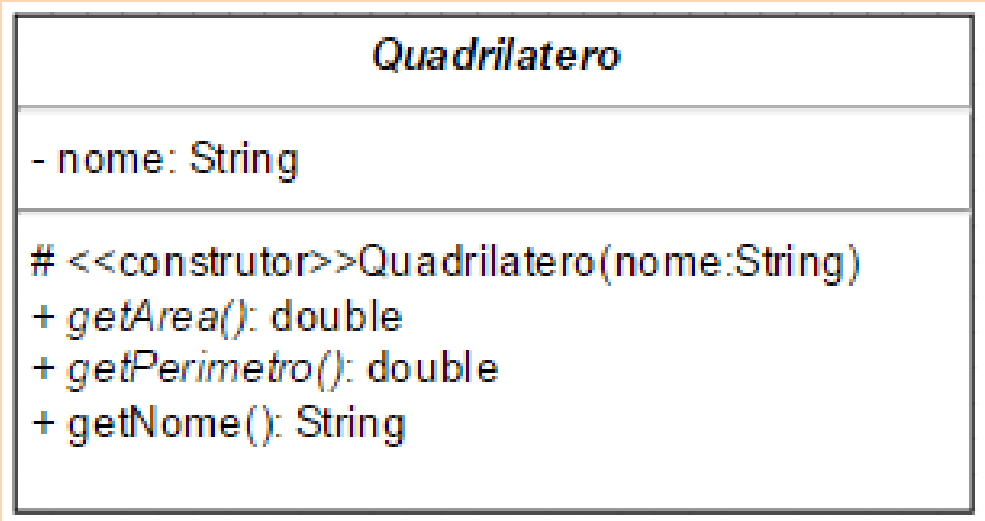
Curso de Tecnologia em Análise e Desenvolvimento de Sistemas

Introdução

- A seguir são apresentadas diversas (mas não todas) notações do diagrama de classes da UML. É considerada uma perspectiva de especificação ou implementação, ou seja, classes de software que foram ou serão implementadas em uma linguagem de programação.
- Nesta perspectiva, o diagrama é chamado de **diagrama de classes de projeto**.

Visibilidade de membros

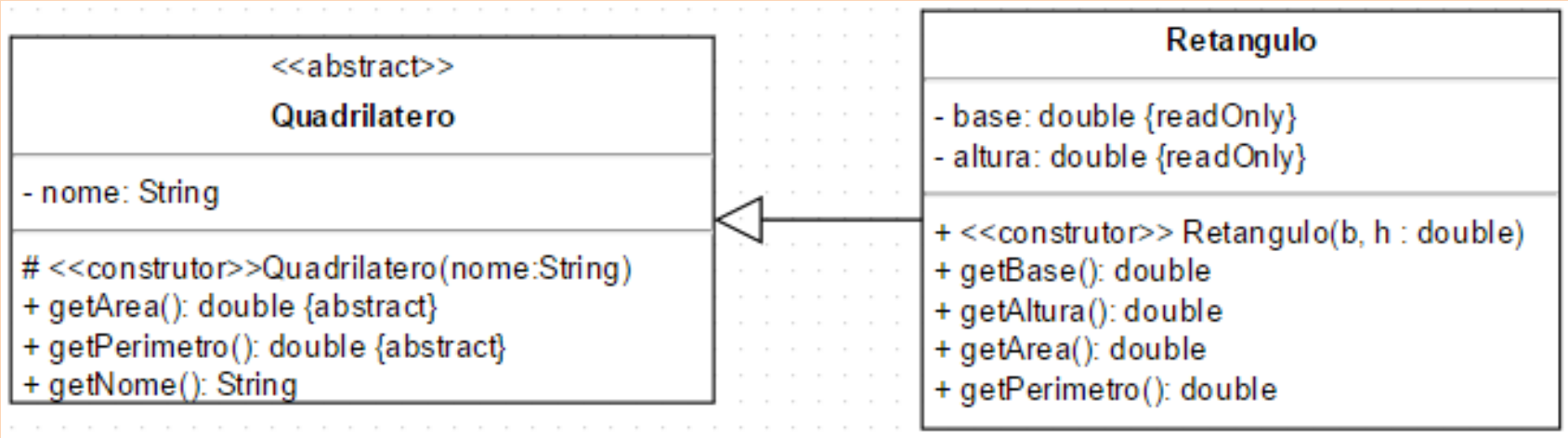
Símbolo	Visibilidade
+	Pública
-	Privada
#	Protegida
~	De pacote



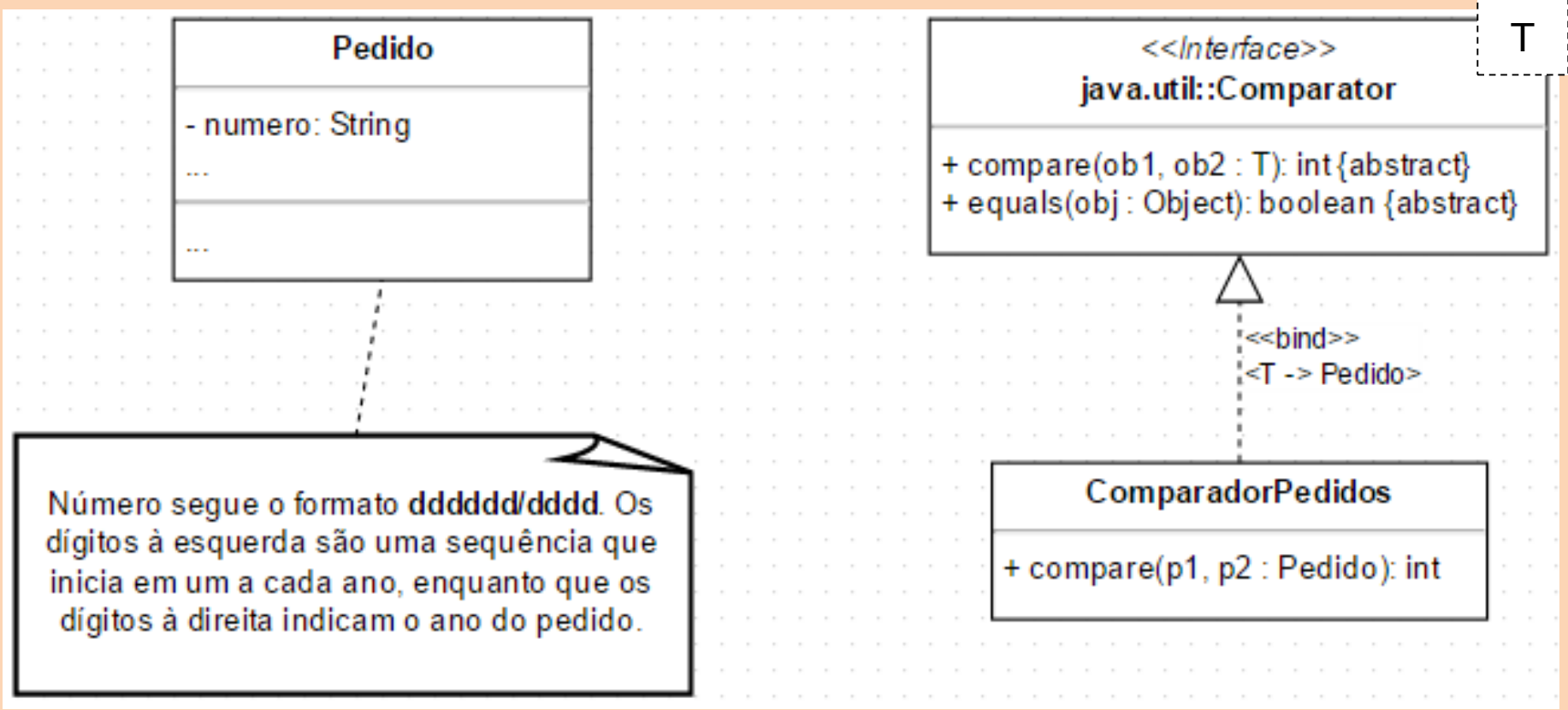

```java
public abstract class Quadrilatero{
  private String nome;
  protected Quadrilatero(String nome){
    this.nome = nome;
  }
  public abstract double getArea();
  public abstract double getPerimetro();
  public String getNome(){
    return nome;
  }
}
```

Herança

```java
public class Retangulo extends Quadrilatero{
  private final double base, altura;
  public Retangulo(double b, double h){
    super("Retângulo");
    this.base = b;
    this.altura = h;
  }
  public double getBase(){ return base; }
  public double getAltura(){ return altura; }
  public double getArea(){ return base * altura; }
  public double getPerimetro(){ return base *2 + altura * 2; }
}
```



Implementação de interface (realização)



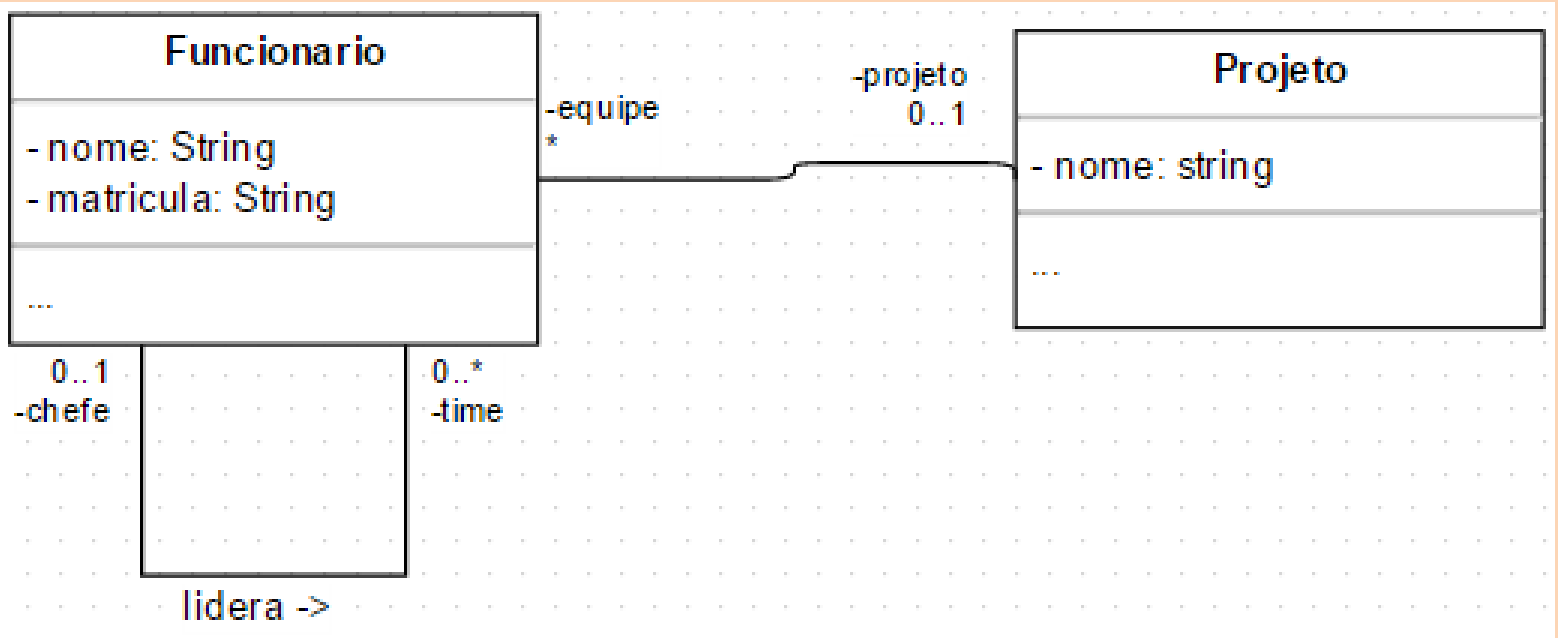
Implementação de interface (realização)

```
package java.util;
public interface Comparator<T>{
   ...
}
```

```
public class ComparadorPedido implements java.util.Comparator<Pedido>{
  public int compare(Pedido p1, Pedido p2){
    int anoP1 = Integer.parseInt(p1.getNumero().substring(7));
    int anoP2 = Integer.parseInt(p2.getNumero().substring(7));
    if(anoP1 != anoP2)
      return anoP1-anoP2;
    int seqP1 = Integer.parseInt(p1.getNumero().substring(0, 6));
    int seqP2 = Integer.parseInt(p2.getNumero().substring(0, 6));
    return seqP1-seqP2;
  }
}
```

Associações

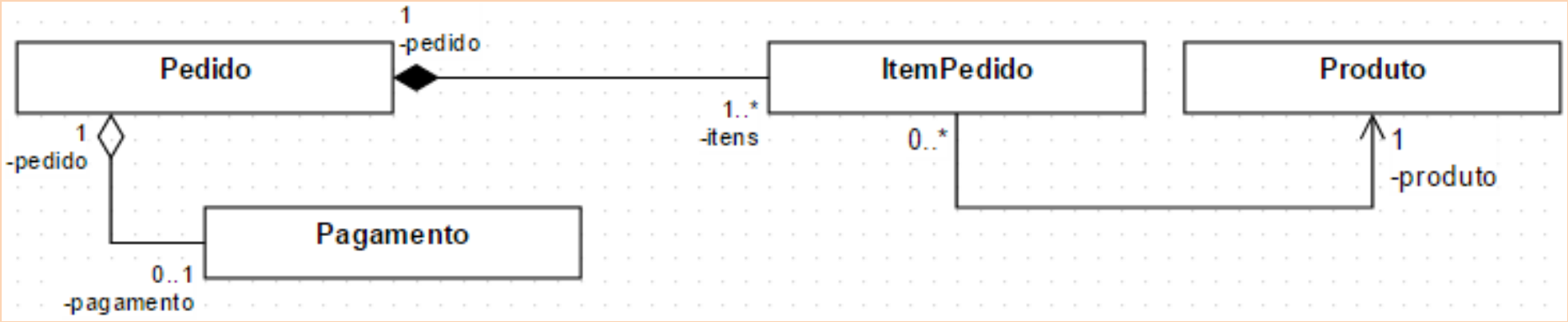
- Uma associação mostra um vínculo entre diferentes instâncias.




```
public class Funcionario{
  private String nome, matricula;
  private Funcionario chefe;
  private Projeto projeto;
  private Collection<Funcionario> time;
  ...
}
public class Projeto{
  private String nome;
  private Collection<Funcionario> equipe;
}
```

Agregação e composição

- São associações do tipo todo-parte.
  - Os objetos parte complementam os objetos todo.
- Agregação: o todo pode existir sem as partes. As partes podem se agregar ao mesmo tempo com mais de um objeto todo.
- Composição: o todo não existe sem as partes. Os objetos parte pertencem a uma única composição. Caso o todo seja destruído, as partes também o são (ou são alocadas para outro objeto todo).
- Dica de Martin Fowler: não se preocupe com agregação. Use apenas composição.




```
public class Pedido{
  private Pagamento pagamento;
  private Collection<ItemPedido> itens;
  public Pedido(Collection<ItemPedido> itens){
    this.itens = itens;
  }
  ...
}
public class ItemPedido{
  private Pedido pedido;
  private Produto produto;
  ...
}
public class Pagamento{
  private Pedido pedido;
  ...
}
```

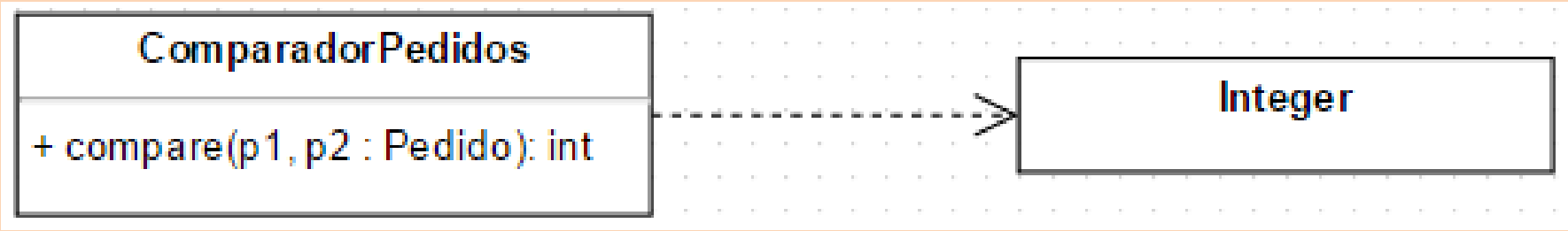
Como o relacionamento entre **ItemPedido** e **Produto** é unidirecional, na classe **Produto** não há um atributo do tipo **ItemPedido**. Assim, um **Produto** não é capaz de saber quais instâncias de **ItemPedido** referem-se a ele.

Dependência

- Indica que de alguma forma uma classe usa ou conhece (depende de) uma outra classe. É um relacionamento mais fraco do que a associação, herança ou realização.
- No código, geralmente indica que a classe dependente usa a outra classe em um método como tipo de parâmetro, de retorno ou de variável local.

Dependência

```java
public class ComparadorPedido implements java.util.Comparator<Pedido>{
  public int compare(Pedido p1, Pedido p2){
    int anoP1 = Integer.parseInt(p1.getNumero().substring(7));
    int anoP2 = Integer.parseInt(p2.getNumero().substring(7));
    if(anoP1 != anoP2)
      return anoP1-anoP2;
    int seqP1 = Integer.parseInt(p1.getNumero().substring(0, 6));
    int seqP2 = Integer.parseInt(p2.getNumero().substring(0, 6));
    return seqP1-seqP2;
  }
}
```



Referências

- FOWLER, Martin. **UML essencial**, 3ª ed. Porto Alegre: Bookman, 2005.
- GUEDES, Gileanes T. A. **UML 2: guia de consulta rápida**, 2ª ed. São Paulo: Novatec, 2005.
- LARMAN, Craig. **Utilizando UML e padrões**, 3ª ed. Porto Alegre: Bookman, 2007.
- uml-diagrams.org. **Class Diagrams**. Disponível em <http://www.uml-diagrams.org/class-diagrams.html>. Acesso em 23 de janeiro de 2015.
- eTutorials.org. **Parameterized Class**. Disponível em <http://etutorials.org/Programming/UML/Chapter+6.+Class+Diagrams+Advanced+Concepts/Parameterized+Class/>. Acesso em 23 de janeiro de 2015.
- VIEIRA, Sidney. **UML: aspectos de projetos em diagramas de classes**. Disponível em <http://sidneyvieira.kinghost.net/abas/disciplinas/download/ESI/ES_DC_projeto.pdf>. Acesso em 23 de janeiro de 2015.
- Ferramenta de diagramação utilizada: https://www.draw.io