Introdução a UML

Diagrama de Classes

Vinicius Takeo Friedrich Kuwaki
Universidade do Estado de Santa Catarina

UDESC
UNIVERSIDADE
DO ESTADO DE
SANTA CATARINA

Diagrama de Classes

- Modela a visão estrutural do projeto de um sistema de forma estática;
- Representa as classes, interfaces e a forma como eles se relacionam;
- Importantes para a documentação e visualização;

Seções

UDESC

Pacotes

- Modularizam um sistema;
- Separam as partes de um sistema;
- Facilitam a organização;
- Um dos principais princípios da orientação a objeto, pois aumenta a reutilização de código;
- Por convensão, nomes de pacotes começam com letra minúscula;
- Pacotes abrigam classes, interfaces, relacionamentos, e vários outros componentes;

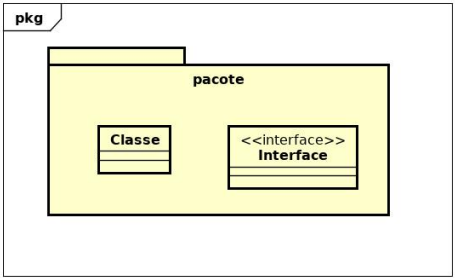

Figura 1: Representação de um Pacote

Pacotes

```
package nomePacote;
```

Listing 1: Declaração de um pacote

- Normalmente IDE's como Eclipse e Netbeans declaram os pacotes automaticamente;

Seções

Classes

- Unidade mais importante da orientação a objetos;
- Representada por um retângulo;
- Contém:
  - Nome:
    - Por convenção, as classes iniciam com letra maiúscula;
  - Atributos: representam os atributos das classes:

UML	Java
- nome : String	private String nome

  - Métodos: representam as funcionalidades que a classe possui:

UML	Java
+ soma(a:int,b:int):int	int soma(int a , int b) height

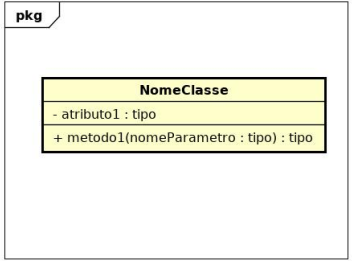

Figura 2: Representação de uma Classe

Classes Abstratas

- Classes que não podem ser instânciadas;
- Tema de aulas futuras;
- No diagrama de classe UML são classes que possuem o nome em itálico;

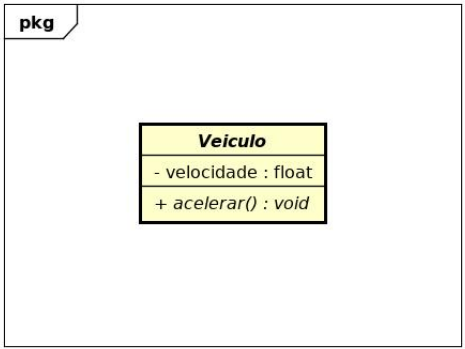

Figura 3: Classe Abstrata Veículo, contento um método abstrato a ser implementado pela classe que a extender. Note que o nome da classe e o método estão em itálico.

Seções

Encapsulamento

- Conceito importante em orientação a objeto;
- Java é uma das linguagens que implementa encapsulamento;
- UML possui 4 formas:

UML	Java	Visibilidade
-	private	membros da própria classe
+	public	qualquer um
#	protected	membros da própria classe e classes filhas (herança)
~	package	membros do mesmo pacote

Tabela 2: Encapsulamento UML

Seções

UDESC

Generalização

- Representa um relacionamento de herança;
- Em Java é associada a palavra extends;
- Na UML é representado por uma seta pintada de branco;

Generalização

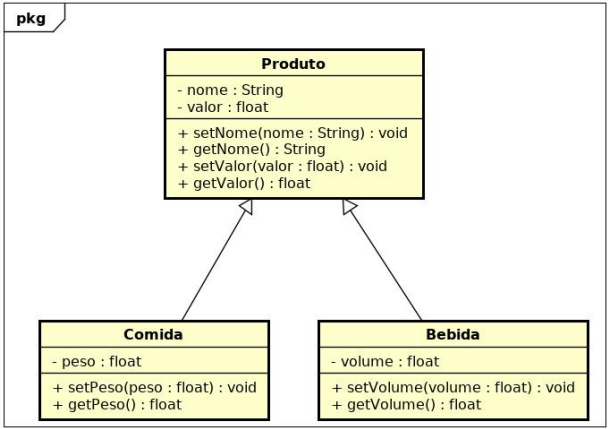

Figura 4: Classe Pai: Produto, Classes Filhas: Comida e Bebida

Generalização

- As setas partes das classes filhas em direção a classe Pai;
- É possível haver vários níveis de herança (mas isso será abordado na aula de Herança);
- Veja o código-fonte em Java das classes representadas no diagrama:

```java
public class Produto{
    private String nome;
    private float valor;

    public String getNome(){
        return this.nome;
    }

    public float getValor(){
        return this.valor;
    }

    public void setValor(float valor){
        this.valor = valor;
    }

    public void setNome(String nome){
```

```
        this.nome = nome;
    }
}
```

Listing 2: Classe Produto

```
class Comida extends Produto{

    private float peso;

    public void setPeso(float peso){
        this.peso = peso;
    }

    public float getPeso(){
        return this.peso;
    }

}
```

Listing 3: Classe Comida extends Produto

Generalização

```
class Bebida extends Produto{

    private float volume;

    public void setVolume(float volume){
        this.volume = volume;
    }

    public float getVolume(){
        return this.volume;
    }

}
```

Listing 4: Classe Bebida extends Produto

Associação

- Especifica um relacionamento entre entidades;
- Podendo determinar quantidade (cardinalidade);
- E até atribuir semântica/significado (função);
- Sendo esses relacionamentos com navegação ou não;

Associação - Função

- Especifica o significado do relacionamento;
- Não obrigatório;
- Porém, deixa o diagrama mais rico;
- Pois atribui o significado da relação entre essas classes;

Associação - Função

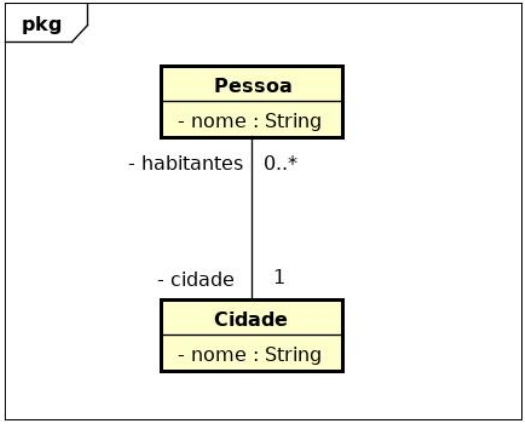

Figura 5: Relacionamento entre a Classe Pessoa e a Classe Cidade

Associação - Função

- No diagrama foram omitidos os métodos getters e setters, pois são inerentes a programação orientada a objetos em Java;
- As palavras abaixo das classes descrevem a semântica que o relacionamento exerce;
  - Uma Pessoa terá uma Cidade;
  - Uma Cidade terá habitantes;

```
public class Pessoa{
    private String nome;
    private Cidade cidade;
}
```

Listing 5: Classe Pessoa

```
public class Cidade{
    private String nome;
    private Pessoa[] habitantes;
}
```

Listing 6: Classe Cidade

Associação - Função

- Observe que a classe Pessoa é um atributo da classe Cidade assim como o inverso;
- Os relacionamentos de associação são caracterizados pelos atributos das classes;
- Novamente, como o objetivo desse exemplo é exemplificar a semântica, os métodos getters e setters não estão presentes no código e nem no diagrama, mas são inerentes a linguagem Java.

Multiplicidade

- Representa de objetos que irão se relacionar;
- Utilizando as seguintes notações a quantidade:
  - **0..1**: nenhuma ou uma;
  - **0..***: nenhuma ou muitas;
  - **1**: apenas uma;
  - **1..***: uma ou muitas;
  - ***: muitas (indeterminado);
  - **n**: muitas (determinado);
- Observe o exemplo a baixo:

Multiplicidade

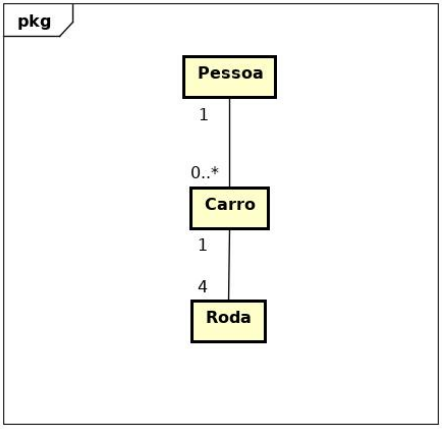

Figura 6: Classe Pessoa se relacionando com as Classes Carro e Roda e suas respectivas cardinalidades. Note que uma Pessoa pode ter nenhum ou mais carros, um Carro porém pertence a somente uma Pessoa. Assim como a Roda pertence a só um Carro e um Carro possui 4 rodas.

Multiplicidade

- A cardinalidade/multiplicidade em Java é representada na quantidade de instâncias do objeto que a classe pode ter;

```
public class Pessoa{

    private List<Carro> carros;

}
```

Listing 7: Classe Pessoa

- Note que a classe Pessoa possui uma lista de Carros como atributo, permitindo 0 ou mais Carros;

Multiplicidade

```
public class Carro{

    private Pessoa dono;
    private Roda[] rodas = new Roda[4];

}
```

Listing 8: Classe Carro

- Já a Classe Carro pode ter um e somente um dono, e 4 rodas;

```
public class Roda{

    private Carro carro;

}
```

Listing 9: Classe Roda

Multiplicidade

- Entretanto a classe Roda possui apenas um Carro (não tem como uma roda pertencer a dois carros);

Navegação

- Observe que até o momento os relacionamentos de associação apresentados não possuiam setas;
- Entretanto é comum o uso das setas em relacionamentos;
- Pois elas determinam que apenas uma classe possui outra;
- A classe do qual a seta parte, possui um atributo da classe que a seta aponta;
- Observe o exemplo a seguir:

Navegação

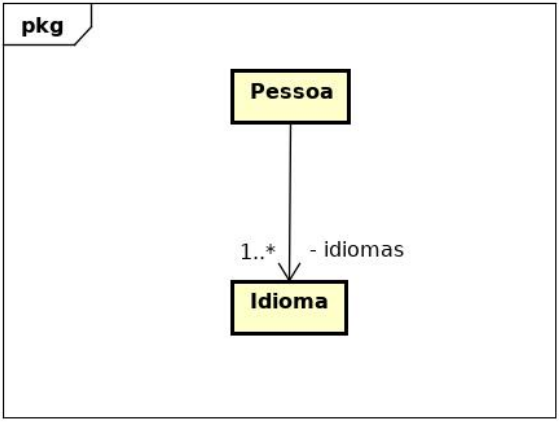

Figura 7: Observe que a seta parte da classe Pessoa em direção a classe Idioma, logo quem possui instancias da classe Idioma é a classe Pessoa e não o contrário.

Navegação

```
public class Pessoa{

    private List<Idioma> idiomas;

}
```

Listing 10: Classe Pessoa

```
public class Idioma{

    private char[] alfabeto;

}
```

Listing 11: Classe Idioma

- Observe que uma pessoa possui vários idiomas, mas um idioma não possui uma pessoa;
- É como se o idioma não soubesse da existência da pessoa;

Agregação

- O relacionamento de agregação é uma associação;
- Porém, com um significado mais forte;
- É um relacionamento do tipo parte/todo;
- Onde o todo é constituído das partes.
- É representado por um losângo branco;

Agregação

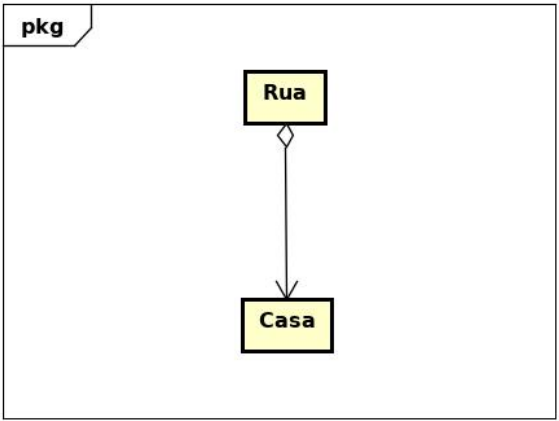

Figura 8: Relacionamento de Agregação entre a classe Rua e classe Casa.

Agregação

- Observe que uma Rua é constituida de Casas;

```
public class Rua{

    private List<Casa> casas;

}
```

Listing 12: Classe Rua

```
public class Casa{

    private int numero;

}
```

Listing 13: Classe Casa

- Note também que é exatamente igual a um relacionamento de associação;
- Porém representa parte/todo.

Composição

- Assim como a Agregação é uma Associação, a Composição é uma Agregação;
- Logo, ela também é um relacionamento de parte/todo;
- Entretanto, um objeto não existe sem o outro.
- É representado pelo losângo preto;
- E quando a parte é destruída, o todo também é;
- Linguagens que lidam com alocação de memória tratam esse relacionamento melhor;
- Em Java não é necessário se preocupar com isso, porém linguagens como C++, quando a parte ou o todo é destrúido, a contra-parte também é;

Composição

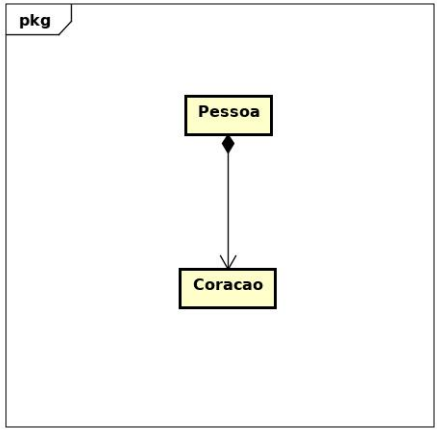

Figura 9: Relacionamento de Composição entre uma Pessoa e um Coração

Composição

- Note que uma pessoa não existe sem o seu coração;

```
public class Pessoa{

    private Coracao coracao;

    public Pessoa(Coracao coracao){
        this.coracao = coracao;
    }

}
```

Listing 14: Classe Pessoa

- Então, quando uma pessoa é "construída"ela "vem"com o seu coração;

Composição

```
public class Coracao{

    public void bater(){
        System.out.println("Contraindo");
        System.out.println("Retraindo");
    }

}
```

Listing 15: Classe Coracao

Dependência

- Quando uma entidade usa informações e serviços de outra entidade;
- Mas não necessáriamente o inverso;
- Comum na utilização de pacotes;

Dependência

- Observe que a classe Main possui um relacionamento de dependência com o pacote dados.
- Esse relacionamento existe pois a classe Main usa pelo menos alguma classe do pacote dados, no caso, instancia objetos do tipo Pessoa para realizar o cadastro no pacote de negocio;

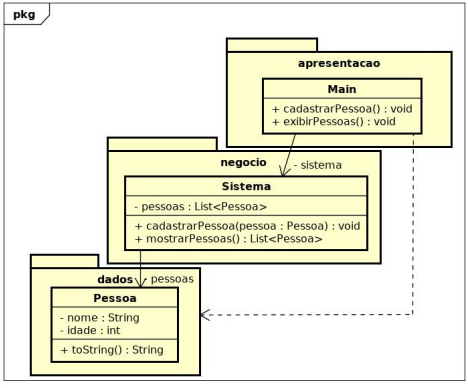

Figura 10: Representação de um relacionamento de dependência entre pacotes

Dependência - Código Fonte

```java
package dados;

public class Pessoa {

    private String nome;
    private int idade;

    public Pessoa(String nome, int idade) {
        this.nome = nome;
        this.idade = idade;
    }

    public String toString() {
        return "Nome: " + this.nome + "\nIdade: " + this.idade;
    }

}
```

- Observe que a classe Pessoa pertence ao package dados;
- E que a classe Sistema (que pertence ao package negócio), possui instancias de classes do pacote de dados (relacionamento de associação).

Dependência - Código Fonte

```
package negocio;

import java.util.LinkedList;

import dados.Pessoa;

public class Sistema {

    private LinkedList<Pessoa> pessoas = new LinkedList<Pessoa>();

    public void cadastrarPessoa(Pessoa p) {
        this.pessoas.add(p);
    }

    public LinkedList<Pessoa> mostrarPessoas() {
        return this.pessoas;
    }

}
```

- Já a classe Main, utiliza a classe Pessoa para enviar dados ao package sistema;
- Além de utilizar o método toString() da classe Pessoa;

Dependência - Código Fonte

- É esse uso que caracteriza a dependência entre a classe Pessoa e a classe Main.

```
package apresentacao;

import java.util.Scanner;
import java.util.LinkedList;

import dados.Pessoa;
import negocio.Sistema;

public class Main {

    private static Sistema sistema = new Sistema();
    private static Scanner s = new Scanner(System.in);

    public static void main(String[] args) {

        int opcao = -1;

        while (opcao != 0) {

            System.out.println("Escolha uma opcao:");
            System.out.println("0 - Sair");
            System.out.println("1 - Cadastrar Pessoa");
```

Dependência - Código Fonte

```
            System.out.println("2 - Exibir Pessoas");

            opcao = s.nextInt();

            switch (opcao) {
            case 0:
                break;
            case 1:
                sistema.cadastrarPessoa(novaPessoa());
                break;
            case 2:
                exibirPessoas();
                break;
            default:
                System.out.println("Valor incorreto!");
                break;
            }
        }
    }

    public static Pessoa novaPessoa() {

        System.out.println("Digite o nome da pessoa");
        String nome = s.nextLine();
```

Dependência - Código Fonte

```
        nome = s.nextLine();

        System.out.println("Digite a idade da pessoa");
        int idade = s.nextInt();

        return new Pessoa(nome, idade);

    }

    public static void exibirPessoas() {
        for (Pessoa pessoa : sistema.mostrarPessoas()) {
            System.out.println(pessoa.toString());
        }
    }

}
```

Realização

- Relacionamento entre uma interface e uma classe;
- Uma classe implementa uma interface;
- Representado por uma seta trajeda pintada de branco;
- Por convensão, iterfaces começam com nome minúsculo;

Realização

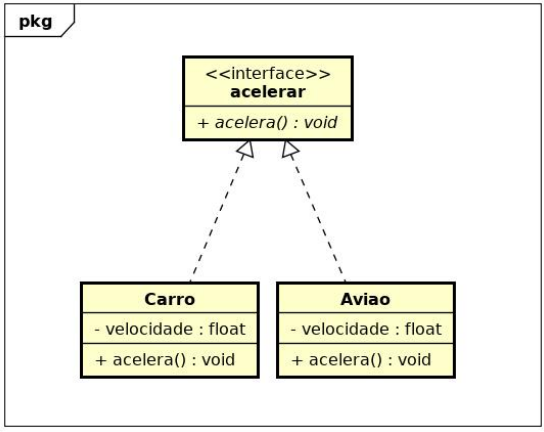

Figura 11: As classes Carro e Avião implementam a interface acelerar

Realização

```
public interface acelerar{

    public void acelera();

}
```

Listing 16: Interface acelera

- Em Java, para uma classe implementar uma interface, ela precisa utilizar a palavra reservada implements após o nome da classe;
- Uma classe que implementa uma interface, precisa obrigatóriamente implementar todos os métodos da interdace;

Realização

```
public class Carro implements acelerar{

    private float velocidade;

    public void acelera(){
        velocidade += 10;
    }

}
```

Listing 17: Classe Carro

Realização

```
public class Aviao implements acelerar{

    private float velocidade;

    public void acelera(){
        velocidade += 50;
    }
}
```

Listing 18: Classe Aviao

Duvidas:
Vinicius Takeo Friedrich Kuwaki
vinicius.kuwaki@edu.udesc.br
github.com/takeofriedrich

UDESC
UNIVERSIDADE
DO ESTADO DE
SANTA CATARINA