Diagramas de Classes

Projeto de Software

Fabio Morais

Diagramas de Classe

- Descrever tipos de objetos e seus relacionamentos
- Diferentes perspectivas com mais ou menos informação
  - Conceitual, Especificação e Implementação
- A perspectiva de implementação é a mais usada
  - Detalhes de atributos, visibilidade, navegabilidade, etc
  - Usada pelo time de desenvolvimento

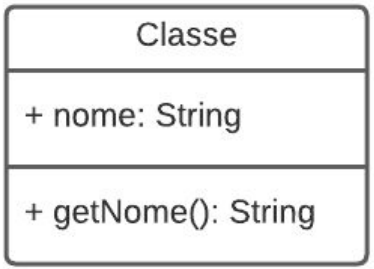
Diagramas de Classe

- Diagramas são formados por entidades e relacionamentos
  - Entidade: representa um objeto, conceito ou elemento
  - Relacionamento: é uma associação entre entidades

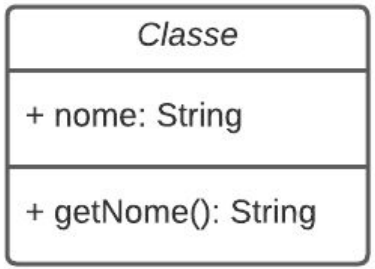
Entidades

- Representam graficamente Classes

Classes concretas



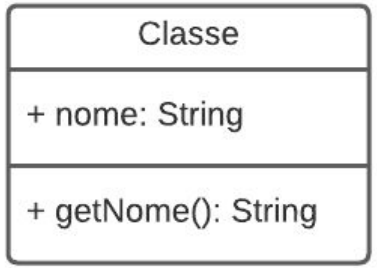
Classes abstratas



- A diferença é a fonte em *itálico* no nome classe

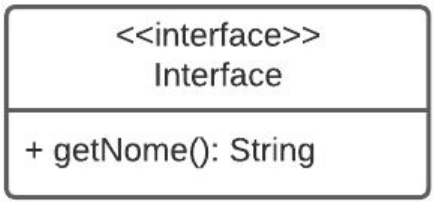
Entidades

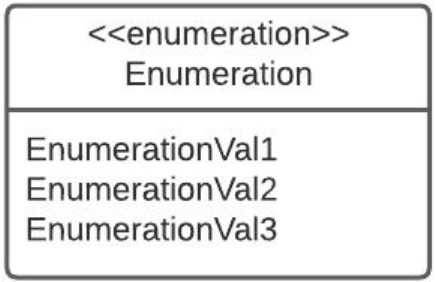
- Detalham informações sobre as Classes
  - Visibilidade de atributos e métodos
  - Parâmetros dos métodos e seus tipos
  - Tipos de retorno dos métodos



Entidades

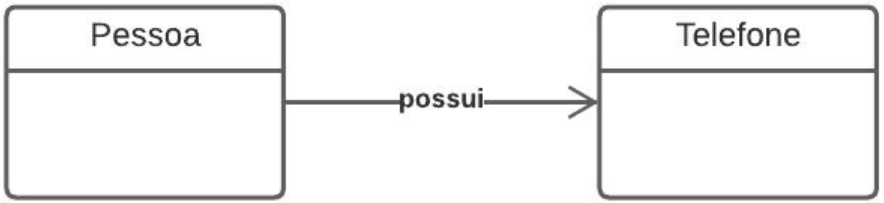
- Representam graficamente **Interfaces** e **Enumerators**



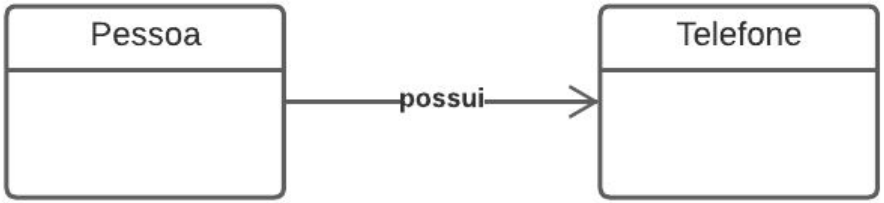


Relacionamentos

- **Papel** descreve o relacionamento em uma associação



Relacionamentos



- A **associação** permite descrever quem aponta para quem

```
public class Pessoa{
    private Telefone t;
    public Pessoa( ){}

    public void setTelefone(Telefone t){
      this.t = t;
    }
    public Telefone getTelefone( ) {
        return Telefone;
    }
}
```

```
public class Telefone {

    public Telefone( ){

      }

}
```

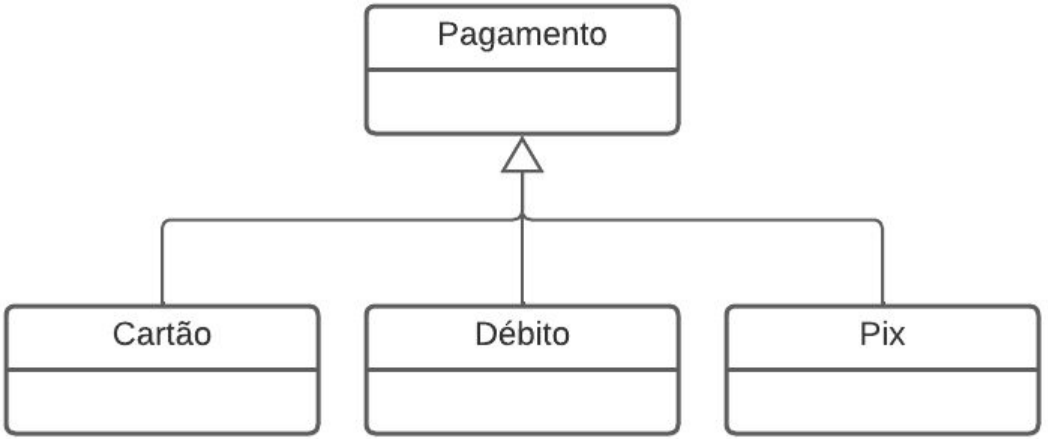
Relacionamentos

- **Multiplicidade** descreve o tamanho da associação

Tipo	Significado
0..1	Zero ou uma instância do objeto. A notação n..m indica n para m instâncias
0..* ou *	Não existe limite para o número de instâncias
1	Exatamente uma instância
1..*	Ao menos uma instância

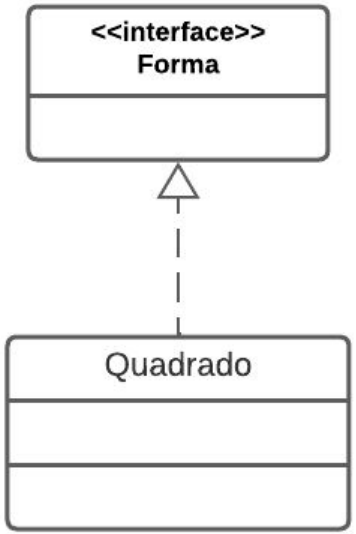
Relacionamentos

- **Herança ou Generalização** indica que uma classe é um subtipo ou tipo especial de outra (herança de tipo e implementação)



Relacionamentos

- **Implementação** indica que uma classe implementa uma interface (herança de tipo)

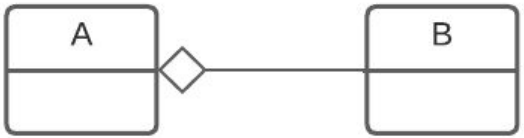


Relacionamentos

- **Agregação**: um objeto é parte de um outro
    - A parte pode existir sem o todo

```
public class A{
    private B b;
    public A( ){}

    public void setB(B b){
      this.b = b;
    }
    public B getB( ) {
        return b;
    }
}
```

```
public class B {

    public B( ){

      }

}
```



Relacionamentos

- **Composição**: um objeto tem outro como valor
    - A parte não pode existir sem o todo. Se o todo some a parte some junto

```
public class A{
    private B b;
    public A( ){

      this.b = new B();

    }
}
```

```
public class B {

    public B( ){

      }

}
```

Referências

- http://www.dsc.ufcg.edu.br/~jacques/cursos/map/html/uml/diagramas/classes/classes1.htm
- https://engsoftmoderna.info/cap4.html
- https://www.ibm.com/docs/pt-br/rsas/7.5.0?topic=structure-class-diagrams