
ânima
EDUCAÇÃO
UNISUL
MODELAGEM DE SOFTWARE
Prof. Richard Henrique de Souza
Ministrante
Prof. Ricardo Ribeiro Assink
ricardo.assink@unisul.br
Prof. Rafael Lessa
rafael.lessa@unisul.br

Agenda

Modelagem UML e a Orientação a Objetos

- *Unified Modeling Language* – UML
- Paradigma: Orientação a Objetos
- Objetos e Classes de Objetos

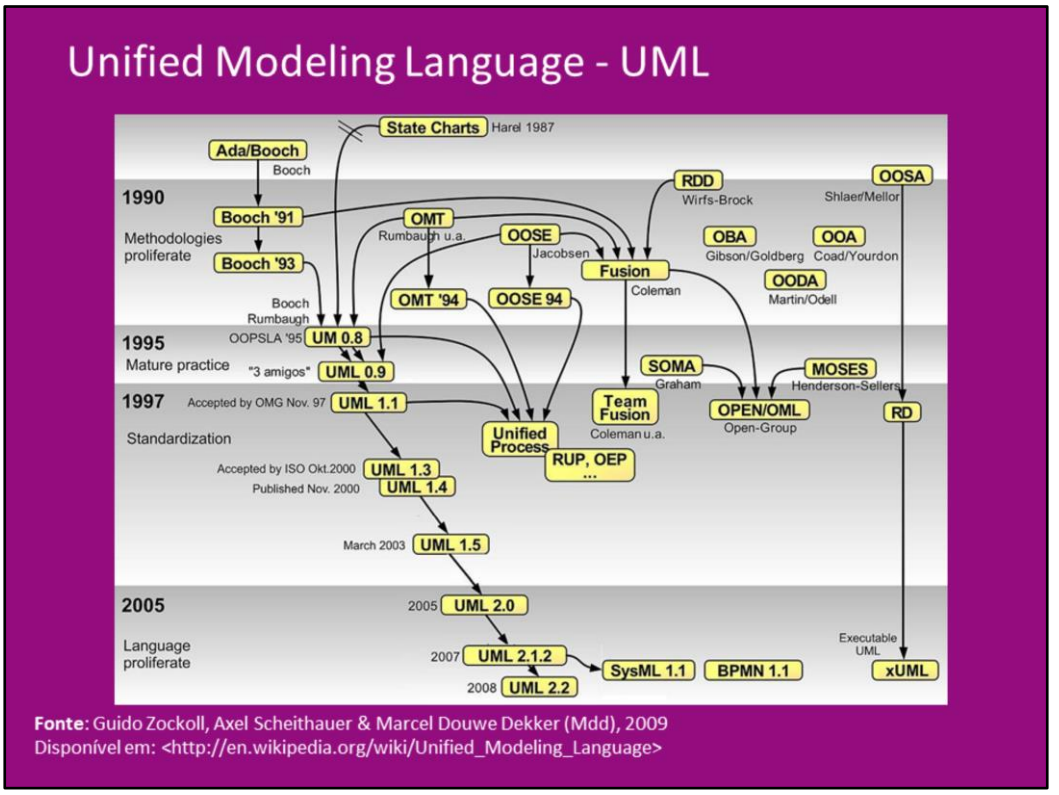
Unified Modeling Language - UML

Diversas notação, processos, e ferramentas para descrição do projeto orientado a objetos foram proposta nos anos 80 e 90.

A UML era uma tentativa de padronizar a modelagem para que qualquer sistema possa ser descrito corretamente de forma consistente e clara. Existiam várias metodologias de modelagem orientada a objetos que causavam uma guerra entre a comunidade de desenvolvedores orientado a objetos. A UML acabou com esta guerra trazendo as melhores ideias de cada uma destas metodologias, e mostrando como deveria ser a migração de cada uma para a UML.

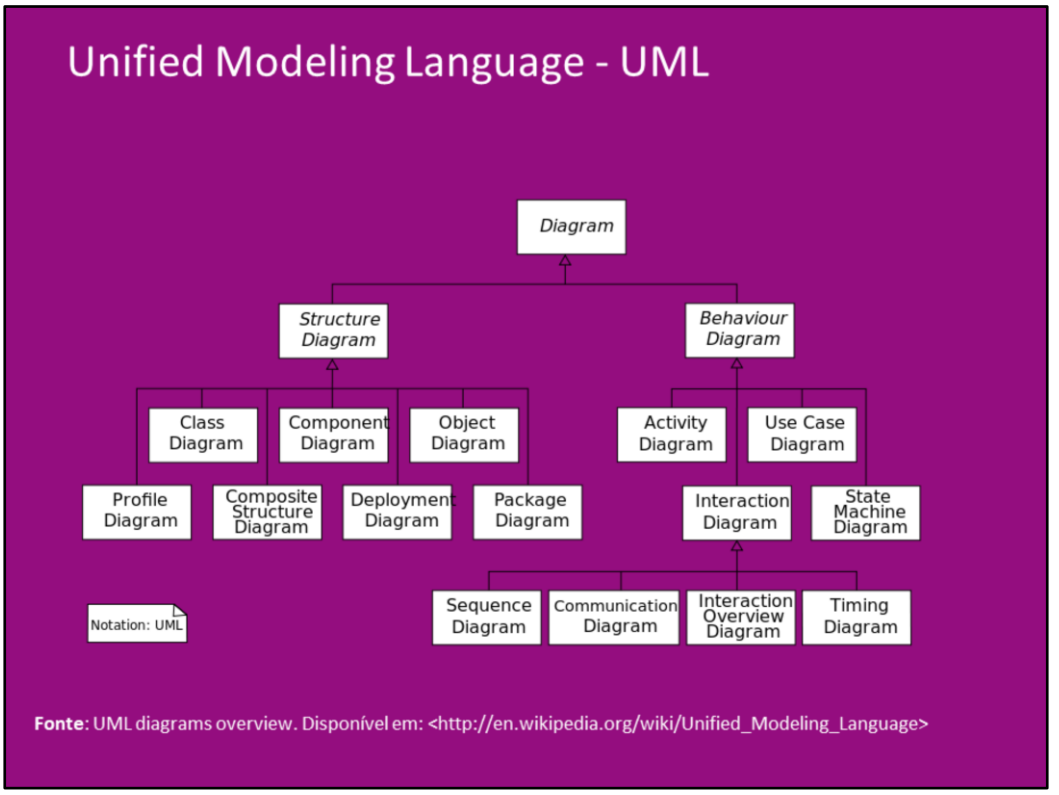
A UML foi desenvolvida por Grady Booch, James Rumbaugh, e Ivan Jacobson. Sendo que a OMG (*Object Management Group*), em 1997, unificou na UML diferentes notações existentes na época.

Versão atual da UML 2.0 <http://www.uml.org>

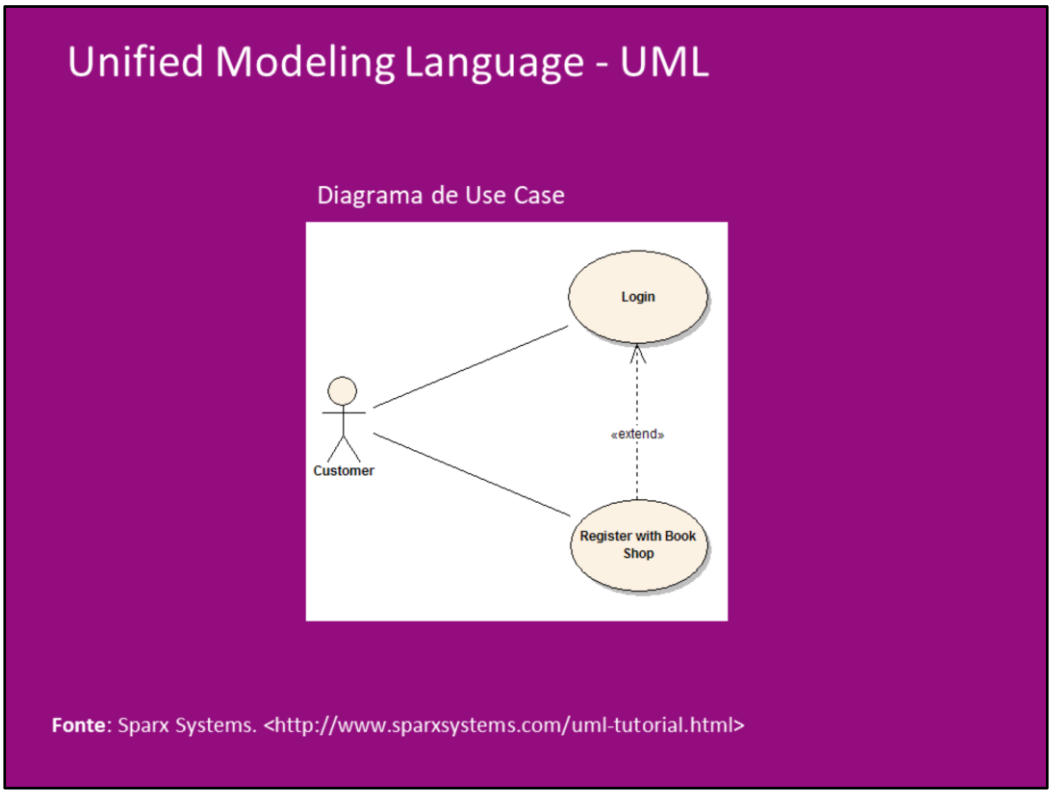


Fonte: Guido Zockoll, Axel Scheithauer & Marcel Douwe Dekker (Mdd), 2009
Disponível em: <http://en.wikipedia.org/wiki/Unified_Modeling_Language>

Descrição da imagem:
Em 1987 temos State Charts [Harel 1987]
Também na década de 80 temos Ada/Booch [Booch]
Entre 1990 até 1994 temos as methodohies proliferate: Booch ’91 , Booch ’93, OMT [Rumabaugh u.a], OMT ‘94, OOSE [Jacobsen], OOSE 94, RDD [wirfs-Brock], Fusion [Coleman], OBA [Gibson/Goldberg], OOSA [Shlaer/Mellor], OOA [Coad/Yourdon], OODA [Martin/ Odell]
Entre 1995 e 1996 temos a Mature practice: UM0.8 [Booch, Rumbaugh, OOPSLA ’95], UML 0.9 [“3 amigos”], SOMA [Graham], MODES [Henderson-Sellers.
Entre 1997 até 2003 temos a Standartization: UML 1.1 [Accepted by OMG nov. 97], Unified Process, Team Fusion [Coleman u.a.] OPEN/OML [Open-Group], RD, UML 1.3 [Accepted by ISO Okt. 2000], RUP, OPE,..., UML 1.4 [Published Nov. 2000], UML 1.5 [March 2003]
Após 2005, Language proliferate: UML 2.0 [2005], UML 2.1.2 [2007], SysML 1.1, BPMN 1.1, xUML [Executable UML], UML 2.2 [2008]



Descrição textual do DIAGRAMA
Notação UML
Primeiro temos o elemento “Diagram” no TOPO do diagrama, ligados a outros dois elementos: “Structure Diagram” e “Behaviour Diagram”, essa ligação é a de herença, onde “Diagram” é o elemento genérico.
O Elemento “Structue Diagram” está ligado aos elementos: “ Profle Diagram”, “Class Diagram”, “composite Structure Diagram”, “Component Diagram”, “Deployment Diagram”, “Object Diagram” e “Package Diagram”, também a ligação indica o relacionamento de herança, onde “Structue Diagram” é o elemento genérico.
O elemento “Behaviour Diagram” está ligado aos elementos “Activity Diagram”, ”Use case Diagram”, ”State Machine Diagram”, ”Interaction Diagram” , também a ligação indica o relacionamento de herança, onde “Behaviour Diagram” é o elemento genérico.
O elemento ”Interaction Diagram” está ligado aos elementos: “Sequence Diagram”, ”Communication Diagram”, ”Interaction Oberview Diagram”, ”Timing Diagram” , também a ligação indica o relacionamento de herança, onde “Interaction Diagram” é o elemento genérico.



Descrição da Figura: Diagrama de use case (Caso de uso)
Um boneco de nome “Customer”
Ligado por uma linha a dois balões (elipses) de nomes: “Login” e “Register with Book Shop”
Sendo que as elipses estão ligadas pelo relacionamento “extend”

Descrição da Figura:
Imagem com código de linguagens de programação.

Paradigma: Orientação a Objetos

A orientação a objetos é um modelo de análise, projeto e programação de sistemas de software baseado na composição e interação entre diversas unidades de software chamadas de objetos.

O paradigma da orientação a objeto tem como princípio a abstração de conceitos do mundo real.

Na qualidade de método de modelagem é tida como a melhor estratégia para se eliminar o "gap semântico", na medida em que a correlação da simbologia e conceitos abstratos do mundo real e das ferramentas (conceitos, terminologia, símbolos, grafismo e estratégias) fossem mais naturais e exatas.

Paradigma: Orientação a Objetos

Na programação orientada a objetos, implementa-se um conjunto de classes que definem os objetos presentes no sistema de software.

Cada classe determina o **comportamento** (definido nos métodos) e **estados possíveis** (atributos) de seus objetos, assim como o relacionamento com outros objetos.

Um projeto de software pode ser representado como um conjunto de objetos que interagem e gerenciam seus estados internos e suas operações.

Linguagem POO (*): C++, C♯, VB.NET, Java, Object Pascal, Python, SuperCollider, Ruby e Smalltalk.

* ActionScript, ColdFusion, Javascript, PHP (a partir da versão 4.0), Perl (a partir da versão 5) e Visual Basic (a partir da versão 4) são exemplos de linguagens de programação com suporte a orientação a objetos.

Paradigma: Orientação a Objetos

Temos as seguintes características:

- Objetos são **abstrações** de entidades do mundo real (ou de algum sistema) que se auto gerenciam.
- Objetos são independentes e **encapsulam** suas representações de estado e de informações.
- A funcionalidade de um sistema é expressa em termos de serviços que os objetos prestam.
- Objetos se comunicam através do envio de **mensagens**.
- Objetos podem ser distribuídos.

Conceitos da POO: Classes e objetos

Classe: Uma classe descreve a estrutura e o comportamento de um conjunto de objetos similares.

Uma classe é a descrição de um grupo de objetos com propriedades similares, comportamento comum, relacionamentos comuns com outros objetos e semântica comum.

Uma classe deve ser a representação de apenas uma abstração.

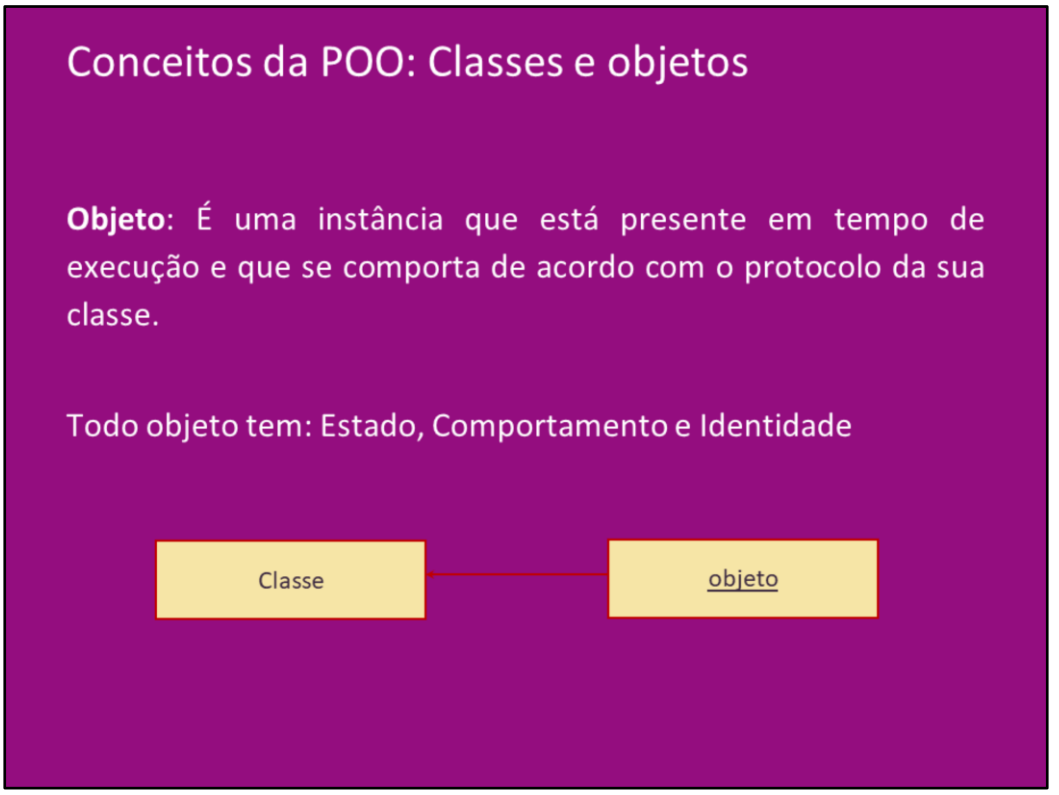
Conceitos da POO: Classes e objetos

Uma classe deveria capturar uma e somente uma abstração chave.

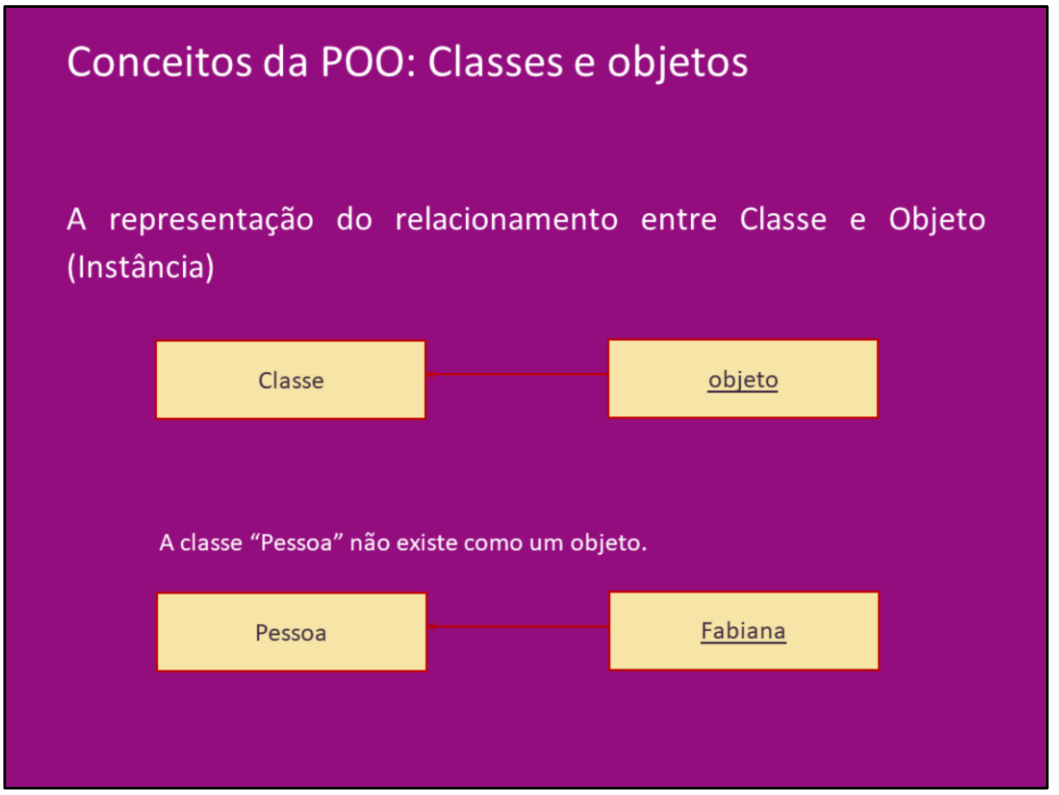
Classes podem ser identificadas examinando-se os substantivos dentro do sistema a ser modelado.

Uma classe deveria ser um substantivo singular que melhor caracteriza a abstração: Professor, Aluno, Turma, ...

Dificuldades na nomeação das classes podem indicar abstrações mal definidas.

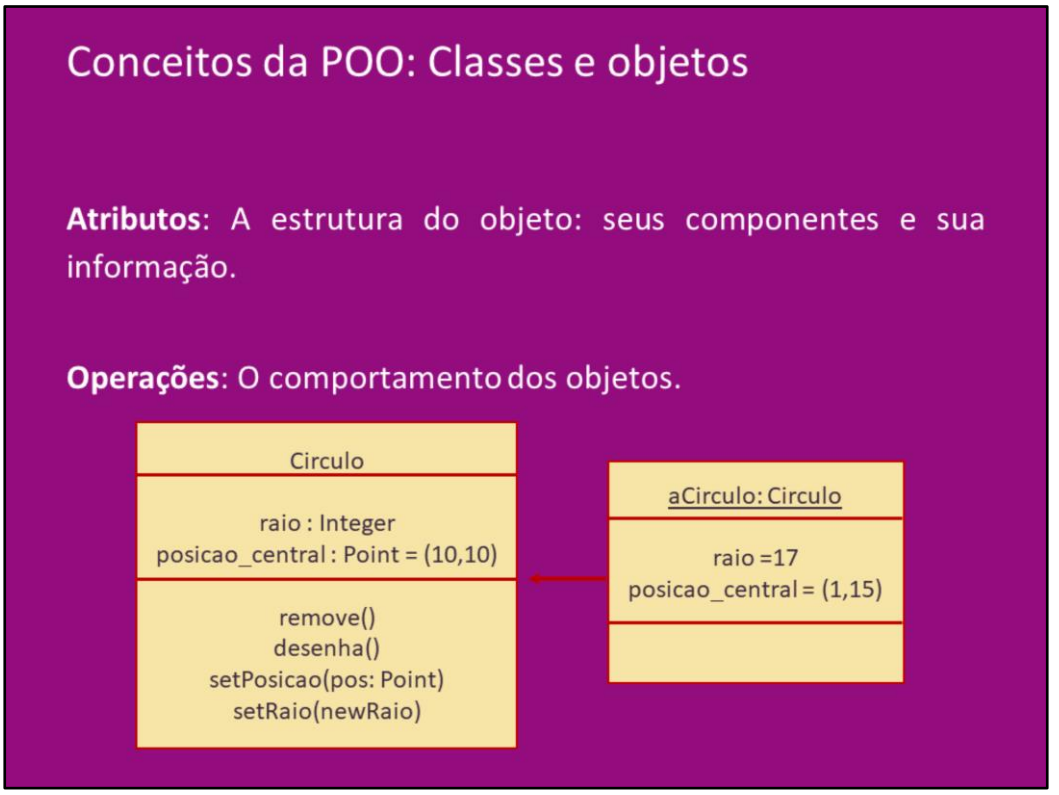


Descrição do diagrama
São dois retângulos ligados por uma linha.
O retângulo da direita representa uma Classe
E o da esquerda um Objeto,
Para diferenciar o que é classe e o que é objeto é utilizado o nome do objeto sublinhado.
Também é utilizado o a primeira letra do nome da classe em maiúsculo, já o objeto não há necessidade de capitalizar a primeira letra.

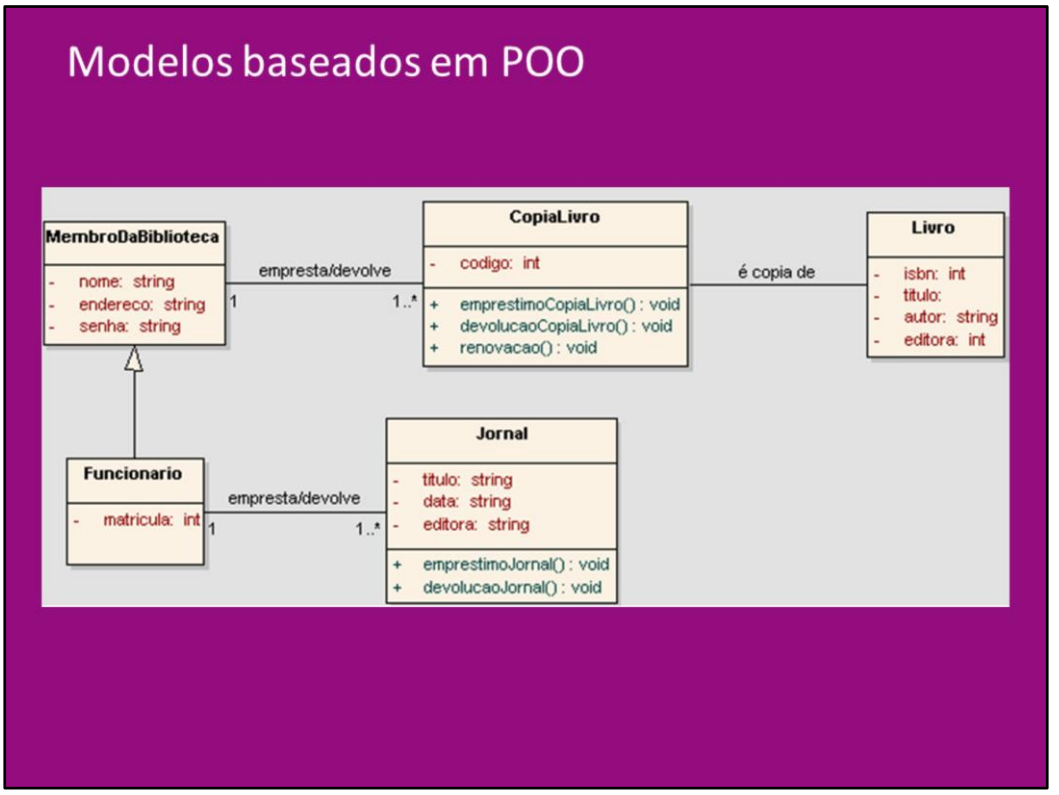


Descrição do diagrama 1:
É um template de diagrama, onde dois retângulos ligados por uma linha.
O retângulo da direita representa uma Classe
E o da esquerda um Objeto,
Para diferenciar o que é classe e o que é objeto é utilizado o nome do objeto sublinhado.
Também é utilizado o a primeira letra do nome da classe em maiúsculo, já o objeto não há necessidade de capitalizar a primeira letra.

Descrição do diagrama 2:
dois retângulos ligados por uma linha.
O retângulo da direita representa uma Classe Pessoa
E o da esquerda um Objeto Fabiana, sabe-se que é um objeto porque o nome Fabiana está sublinhado.



Descrição do diagrama
dois retângulos ligados por uma linha.
O retângulo da direita representa uma Classe Circulo
E o da esquerda um Objeto aCirculo: Circulo, sabe-se que é um objeto porque o nome Fabiana está sublinhado e para indicar que aCirculo é um objeto da Classe Circulo utiliza-se os : Circulo
Observação: Caso não se queira dar um nome para o objeto pode-se apenas colocar os : e o nome da classe, exemplo :Circulo.
Além do nome da classe e o Objeto, neste diagrama a Classe circulo está indicando seus atributos e suas operações, para isso o retângulo é dividido em três partes: a do Topo é nome, no caso Circulo; a parte do meio possui os atributos, neste caso são raio: Integer e posicao_central: Point = (10,10); e a parte de baixo onde ficam as operações, neste caso são remove(), desenha(), setPosicao(pos: Poit), e setRaio(newRaio).
Neste diagrama o objeto aCirculo: Circulo está indicando seu estado, para isso o retângulo é divido em duas partes: a parte de cima com o nome do objeto, neste caso aCirculo: Circulo e a parte de baixo contendo o estado do objeto, neste caso é raio = 17 e posicao_central = (1,15).



Descrição do Diagrama
Existe cinco retângulos indicando cinco classes.
Temos a classe Jornal com os atributos – titulo: String, -data: String, - editora: String e com as operações: + emprestimoJornal(): void e +devolucaoJornal(): void.
Observação: o sinal de menos significa que o atributo é privado e o sinal de + significa que a operação é publica. E o termo void significa que a operação não possui retorno.

A segunda classe é a de Funcionário que possui o atributo –matricula: int. A Classe Funcionário está ligada com a classe Jornal por uma linha, indicando uma associação com nome empresta/devolve com cardinalidade 1 do lado do Funcionário e cardinalidade 1..* do lado de Jornal.

A terceira classe é ade MembroDaBiblioteca que possui os atributos – nome: String, -endereço: String e –senha: String. A Classe MembroDaBiblioteca está ligada a classe Funcionário pelo relacionamento de herança, onde a classe MembroDaBiblioteca é a classe genérica e a Classe Funcionario é a classe especializada.

A quarta classe é a CopiaLivro que possui como atributo –código: int e as operações +emprestimoCopiaLivro(): void, +devolucaoCopiaLivro(): void e + renovação(): void.
A classe CopiaLivro está ligada com a classe MembroDaBilioteca por uma associação chamada de empresta/devolve onde há uma cardinalidade de 1 no lado da classe MembroDaBibliotaca e a cardinalidade de 1..* do lado da classe CopiaLivro.

A quinta Classe é a a classe Livro que contêm os atrubutos –isbn: int, - titulo: String, - autor: String e – editora: int. A Classe Livro esta ligada com a classe CopiaLivro por uma associação chamada de “é copia de” e a qual não foi definida as cardinalidades.

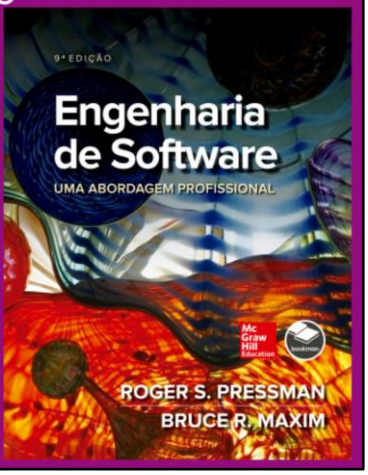
Busca Ativa

- Leitura das páginas 114 até a página 125 e

Das páginas 141 até 144 do livro

Pressman, Roger, S. e Bruce R. Maxim. *Engenharia de software*.

Disponível em: Minha Biblioteca, (9th edição). Grupo A, 2021.



Descrição da imagem:
Capa do livro Pressman, Roger, S. e Bruce R. Maxim. *Engenharia de software*.

Disponível em: Minha Biblioteca, (9th edição). Grupo A, 2021.

Conceitos da POO: Classes e objetos

Visibilidade da classe

Especifica como os atributos e/ou operações poderão ser acessados por outros objetos

Atributos e operações podem ter visibilidades diferentes

- Privada (–) : atributos e operações só podem ser acessados na implementação da própria classe
- Pública (+) : atributos e operações são visíveis dentro da própria classe e para todas as outras classes
- Protegida (#) : será discutida após discutirmos o conceito de herança

Visibilidade

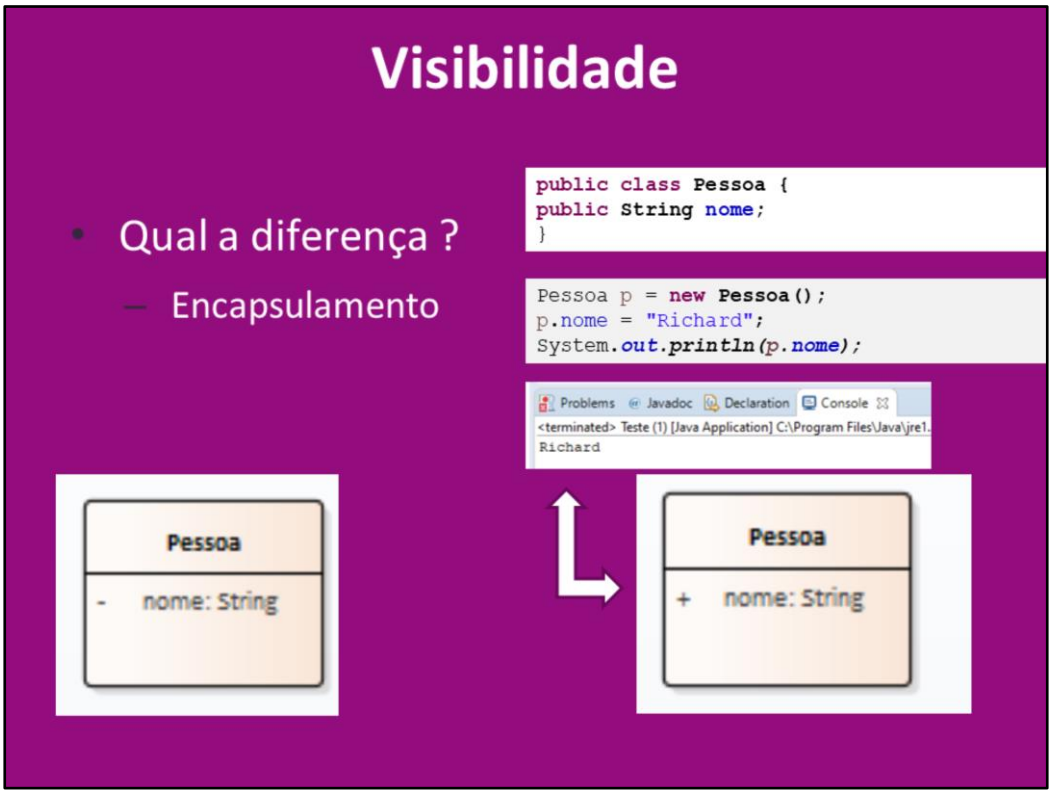
- Qual a diferença ?

Descrição da imagem
São exibidas duas classes
A primeira é a Classe Pessoa com o atributo – nome: String
A segunda é a Classe Pessoa com o atributo + nome: String
A diferença entre as figuras é a apenas o sinal “-” no atributo da primeira classe e “+” no atributo da segunda classe

Visibilidade

- Qual a diferença ?
  - Encapsulamento

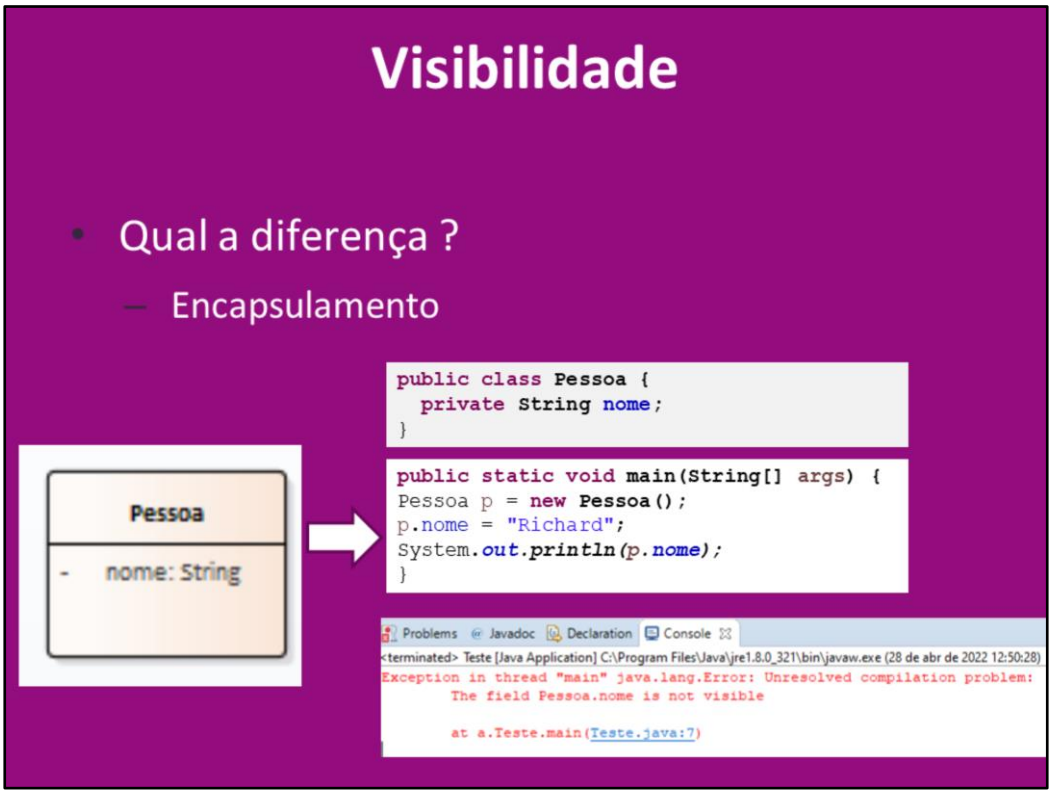
Descrição da imagem
São exibidas duas classes
A primeira é a Classe Pessoa com o atributo – nome: String
A segunda é a Classe Pessoa com o atributo + nome: String
A diferença entre as figuras é a apenas o sinal “-” no atributo da primeira classe e “+” no atributo da segunda classe



Descrição da imagem
São exibidas duas classes e o resultado da execução classe que tem o atributo publico (+)
A primeira é a Classe Pessoa com o atributo – nome: String
A segunda é a Classe Pessoa com o atributo + nome: String
A diferença entre as figuras é a apenas o sinal "-" no atributo da primeira classe e "+" no atributo da segunda classe

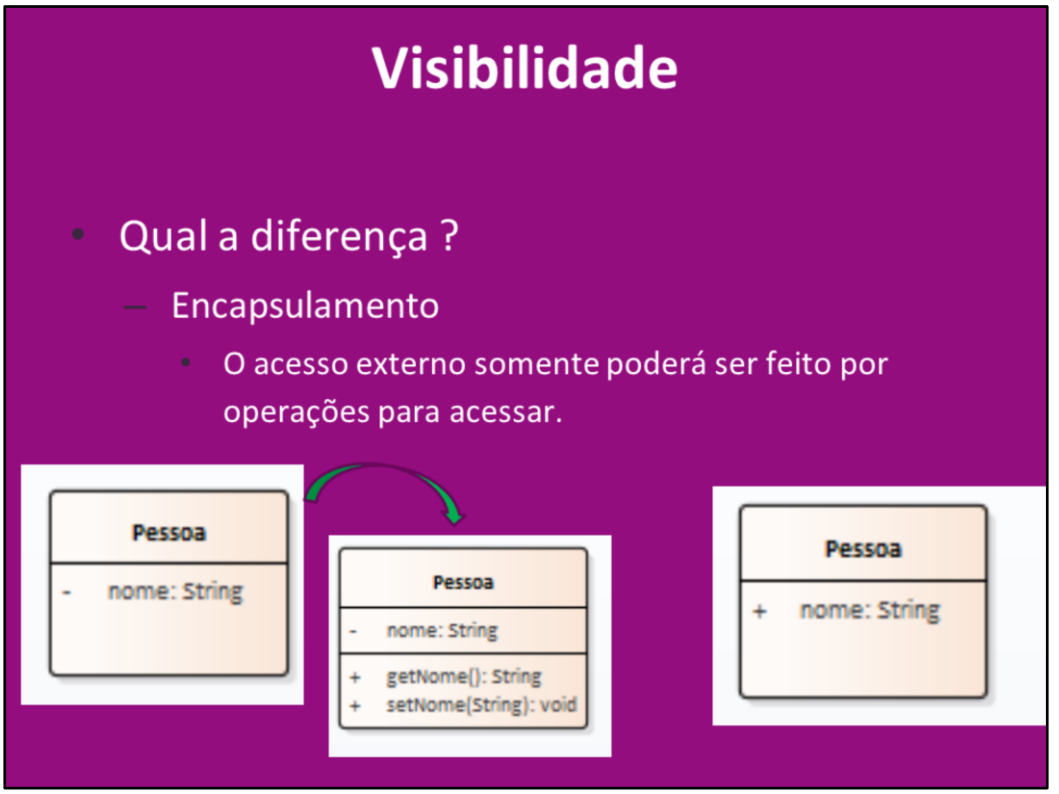
A imagem da execução do programa que implementa o atributo publico mostra no console a palavra "Richard"

```
...
Pessoa p = new Pessoa();
p.nome = "Richard";
System.out.println(p.nome);
...

public class Pessoa {
public String nome;
}
```



Descrição da imagem
Mostra a Classe Pessoa com o atributo – nome: String
E resultado da implementação dessa classe, utilizando o mesmo código do slide anterior
O resultado é um erro: " Exception in thread "main" java.lang.Error: ....."

...
```
public static void main(String[] args) {
Pessoa p = new Pessoa();
p.nome = "Richard";
System.out.println(p.nome);
}
...
public class Pessoa {
  private String nome;
}
```



Descrição da imagem
São exibidas três classes
A primeira é a Classe Pessoa com o atributo – nome: String
A segunda é a Classe Pessoa com o atributo + nome: String
A diferença entre as figuras é a apenas o sinal "-" no atributo da primeira classe e "+" no atributo da segunda classe

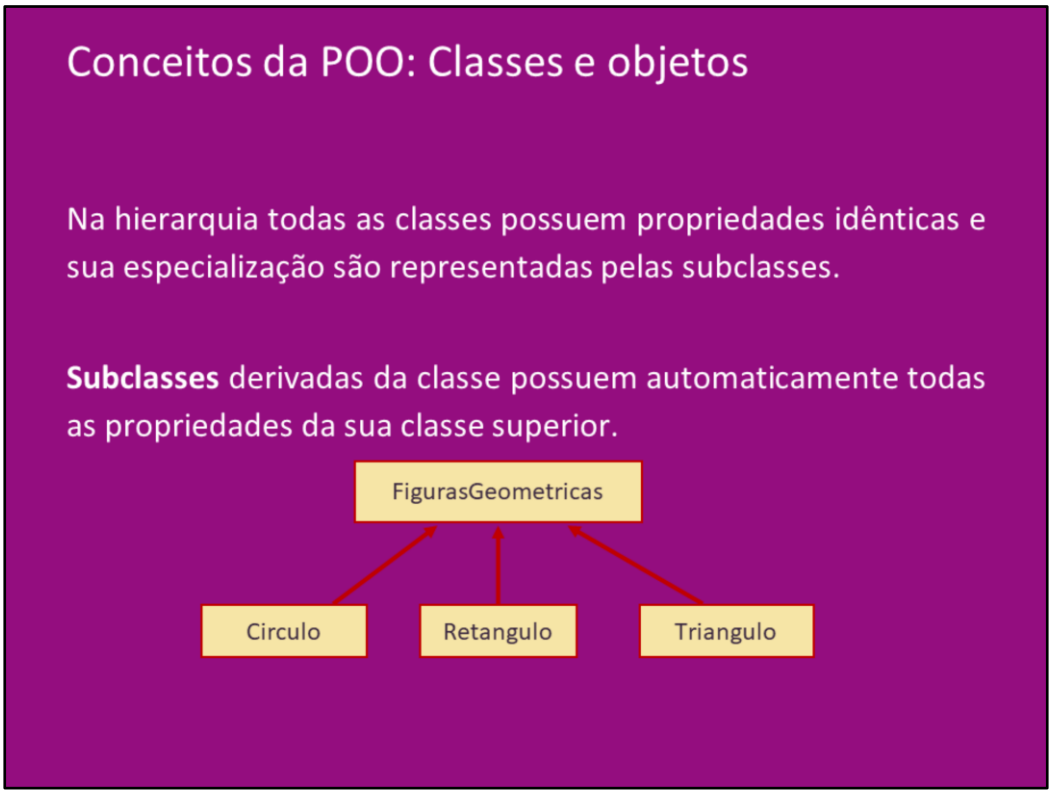
E a terceira é a evolução da Classe com o atributo privado ("-"), onde é adicionado as operações + getNome(): String e + setNome(String): void.

Busca Ativa

- Leitura da página 97 até a página 141 do livro

Horstmann, Cay. *Conceitos de Computação com Java*. Disponível em: Minha Biblioteca, Grupo A, 2009.

Descrição da imagem
Capa do livro: Horstmann, Cay. *Conceitos de Computação com Java*. Disponível em: Minha Biblioteca, Grupo A, 2009.



Descrição da imagem:
A Classe FigurasGeometricas está relacionada com outras três classes pelo relacionamento de herança, o qual é uma linha com um triangulo na ponta que liga a classe genérica, neste caso a classe genérica é a classe FigurasGeométricas. As outras três classes são Circulo, Retangulo e Triangulo, essas são ditas classes especializadas da classe FigurasGeometricas devido ao relacionamento de herança.

Conceitos da POO: Classes e objetos

Superclasse é a classe que fornece os atributos e operações.

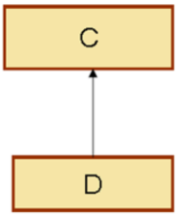
As relações:

- FigurasGeometricas é uma **generalização** de Retangulo.

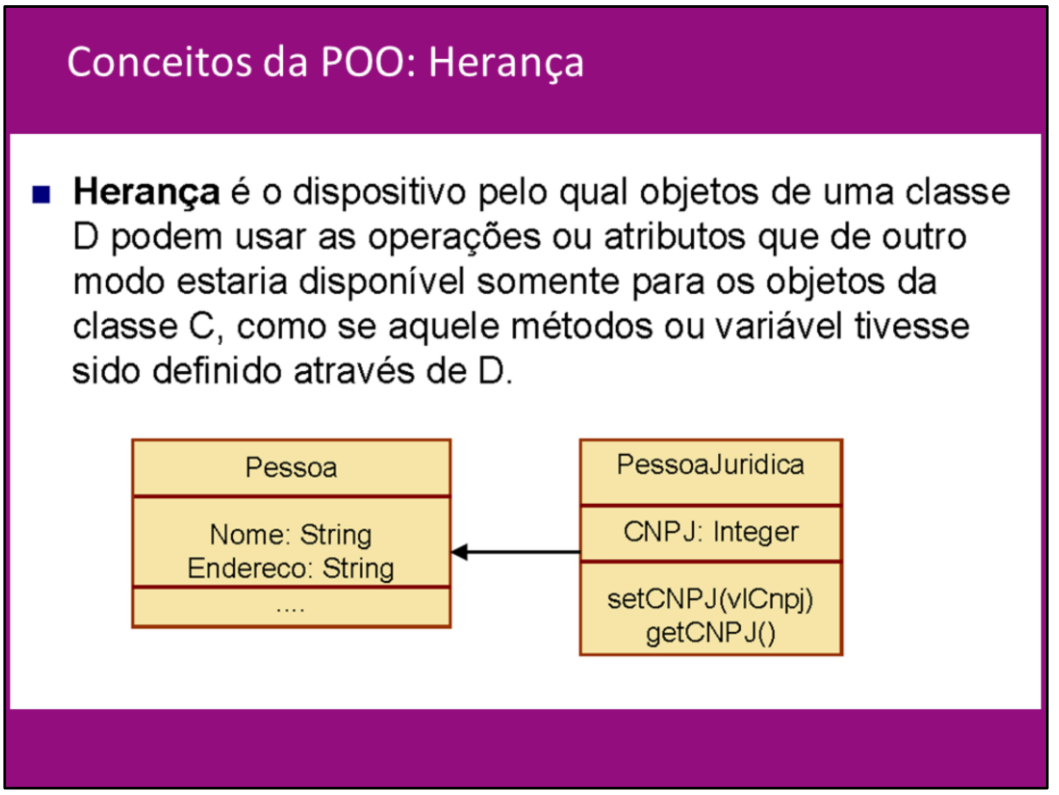
- Retangulo é uma **especialização** de FigurasGeometricas

Conceitos da POO: Herança

- O que você faria se tivesse escrito uma classe C e depois descobrisse uma classe D quase igual a C, exceto com alguns métodos extras?
  - Uma solução seria duplicar todos os métodos de C e colocá-los em D. (Isso não seria retrabalho? E a manutenção?)
  - A *Melhor Solução* seria fazer com que a classe D de certa forma "pedisse para usar os métodos" da classe C. Chamamos isso de **Herança.**



Descrição da imagem:
Inicia com a Pergunta: O que você faria se tivesse escrito uma classe C e depois descobrisse uma classe D quase igual a C, exceto com alguns métodos extras?
Opção 1: Uma solução seria duplicar todos os métodos de C e coloca-los em D. (Isso não seria retrabalho? E a manutenção?)
Opção 2: A Melhor Solução seria fazer com que a classe D de certa forma "pedisse para usar os métodos" da classe C. Chamamos isso de Herança.
Imagem de dois retângulos C e D ligados com uma linha com um triangulo na ponta do lado do retângulo C.



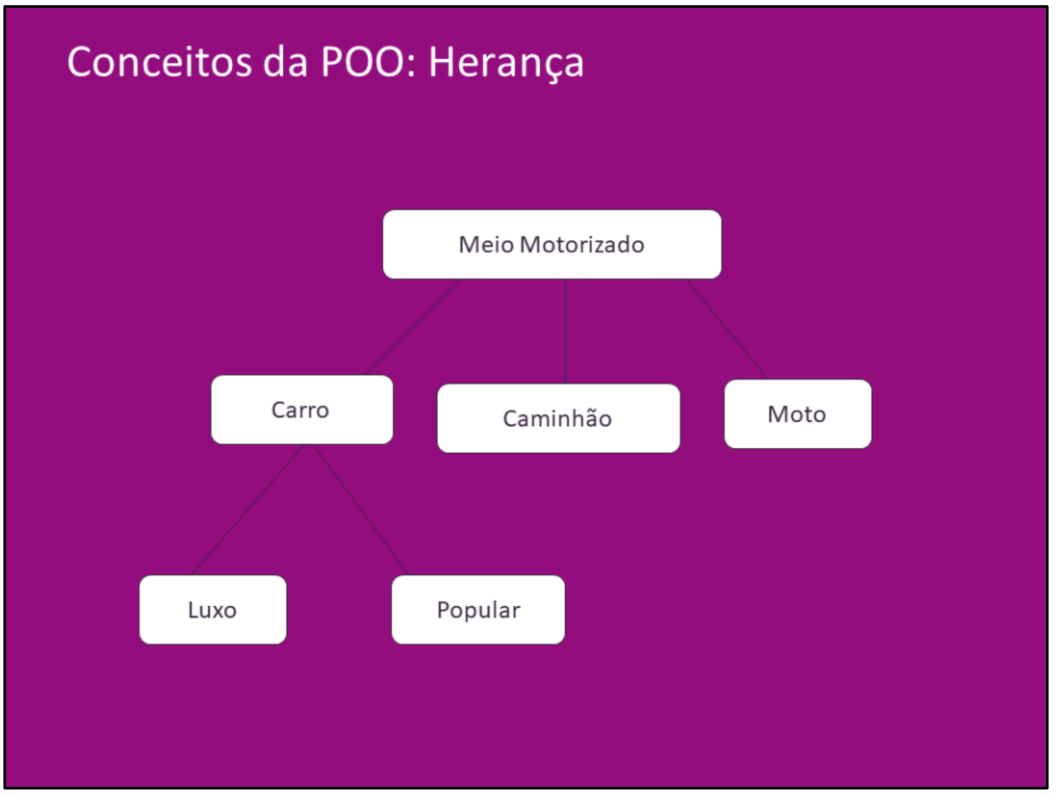
Descrição da Figura:
Herança é o dispositivo pelo qual objetos de uma classe D podem usar as operações ou atributos que de outro modo estaria disponível somente para os objetos da classe C, como se aquele métodos ou variável tivesse sido definido através de D.
No diagrama exemplo tem duas classes a primeira é a classe Pessoa com os atributos nome: String e endereço: String e a segunda classe é a PessoaJuridica com o atributo CNPJ: Integer e as operações setCNPJ(vlCnpj) e getCNPJ(). Há o relacionamento de herança entre as classes Pessoa e PessoaJuridica, sendo que a classe Pessoa é a classe genérica e a classe PessoaJuridica é a classe especializada. O relacionamento é definido como uma linha com uma seta (triangulo) na ponta. A seta (ou triangulo) está no lado da classe Pessoa, indicando assim que a classe Pessoa é a classe genérica.

Conceitos da POO: Herança

É um relacionamento de **especialização** e **generalização**(“é um” ou “tipo de”).

Herança define uma hierarquia de abstrações na qual uma subclasse herda de uma ou mais superclasses:

- **Herança simples**: a subclasse herda de uma única superclasse.
- **Herança múltipla**: a subclasse herda de duas ou mais Superclasses.

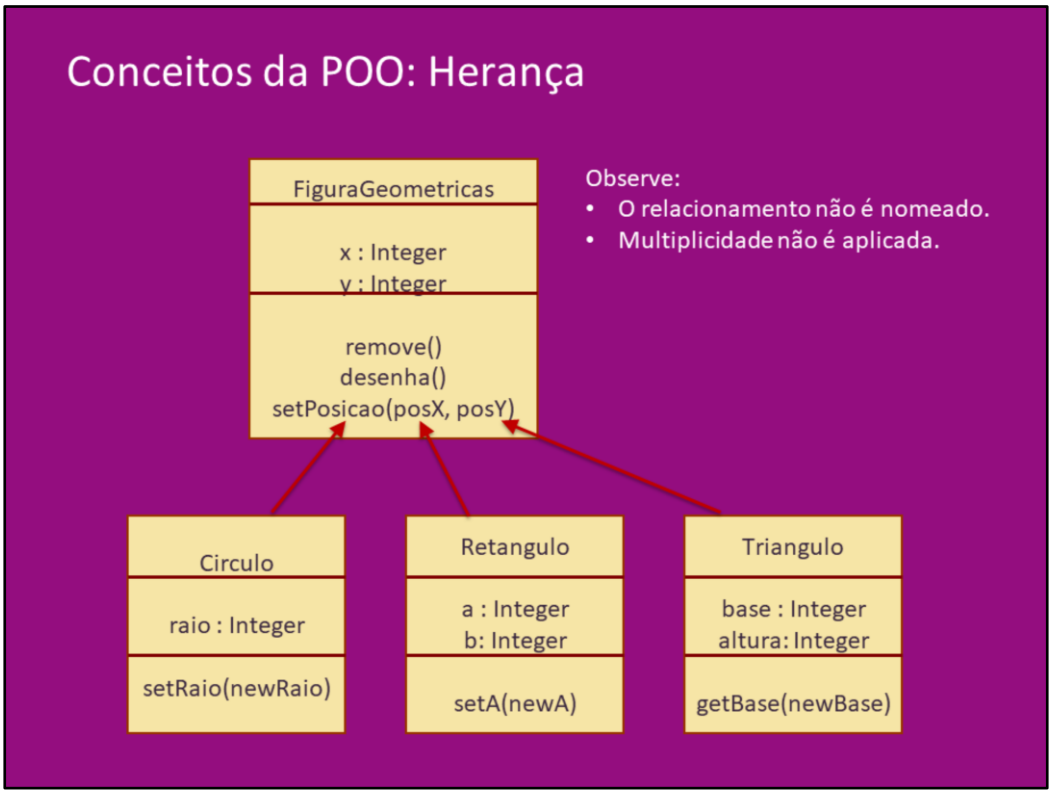


Descrição do diagrama
Tem 6 retângulos para indicar as classes, sendo que a classe no topo é a classe Meio Motorizado que está ligada as classes Carro, caminhão e Moto, sendo que a classe Carro está ligado as classes Luxo e Popular.

Conceitos da POO: Herança

Na hierarquia todas as classes possuem propriedades idênticas e sua especialização são representadas pelas subclasses. A hierarquia de classes segue a visão individual sobre o domínio a ser modelado.

Subclasses derivadas da classe possuem automaticamente todas as propriedades da sua classe superior.



Descrição do Diagrama
Possui 4 classes,
Primeiro a classe genéria FiguraGeometricas com os atributos x: Integer e y: Integer e com as operações remove(), desenha() e setPosicao(posX, posY).
A classe FiguraGeometricas está ligada outras 3 classes pelo relacionamento de herança.
As outras três classes são Circulo com o atributo raio: Integer e a operação setRaio(newRaio), e classe Retangulo com os atributos a: Integer e b: Integer e as operação seta(newA) e a Classe Triangulo com os atributos base: Integer e altura: Integer e a operação getBase(newBase)

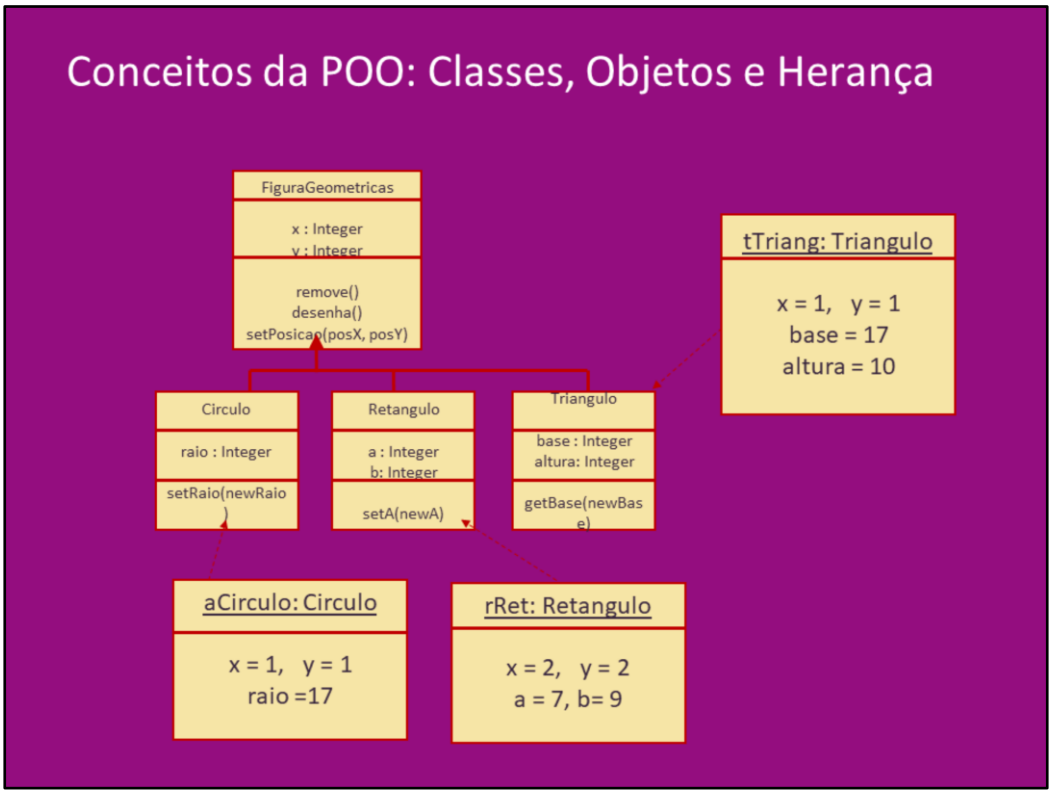
Conceitos da POO: Herança

Herança não relaciona objetos individuais

- O relacionamento não é nomeado.
- Multiplicidade não é aplicada.

Uma subclasse pode:

- Adicionar atributos, operações e relacionamentos.
- Redefinir operações herdadas.



Descrição do Diagrama
Possui 4 classes e o estado de 3 objetos
Primeiro a classe genéria FiguraGeometricas com os atributos x: Integer e y: Integer e com as operações remove(), desenha() e setPosicao(posX, posY).
A classe FiguraGeometricas está ligada outras 3 classes pelo relacionamento de herança.
As outras três classes são Circulo com o atributo raio: Integer e a operação setRaio(newRaio), e classe Retangulo com os atributos a: Integer e b: Integer e as operação seta(newA) e a Classe Triangulo com os atributos base: Integer e altura: Integer e a operação getBase(newBase)

É representado o estado do objeto aCirculo: Circulo com x=1, y=1 e raio =17.
É representado o estado do objeto rRet: Retangulo com x=2, y=2, a=7, b=9
É representado o estado do objeto tTriang: Triangulo com x=1, y=1, base =17 e altura=10

Note que os objetos herdaram os atributos x e y.

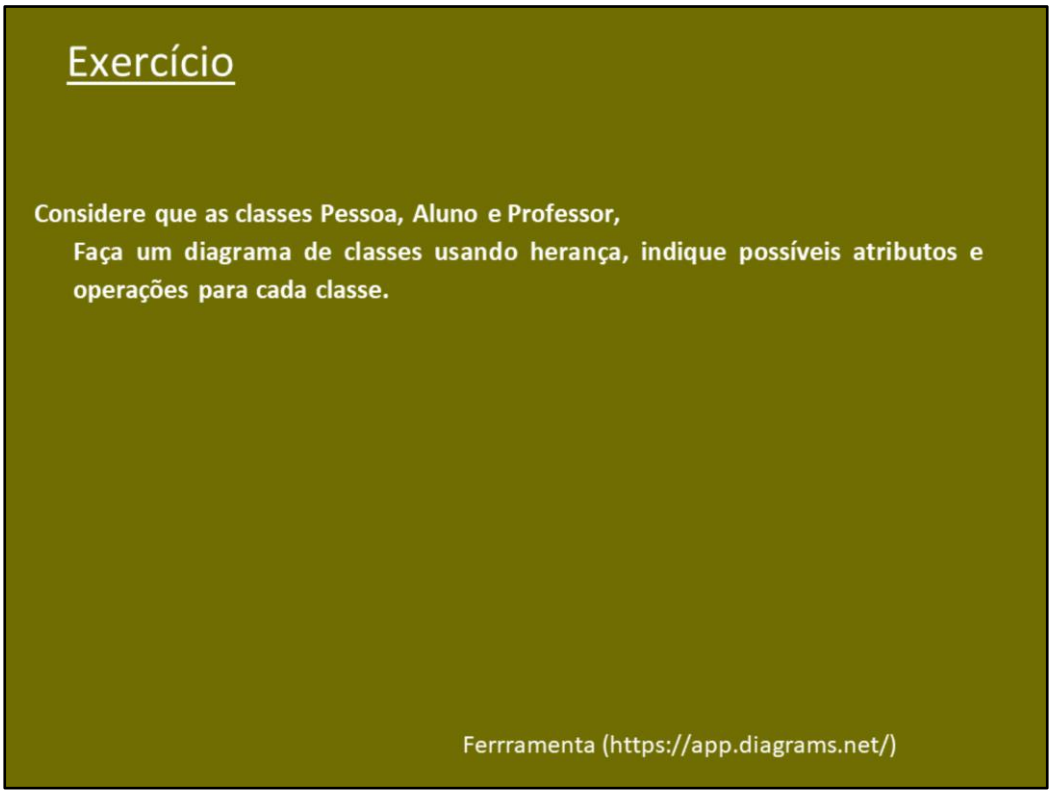
Diagrama de classes

Objetos são as instâncias de classe.

Atributos de classe são informações compartilhadas por todos os objetos da classe.

Atributos de objeto são informações específicas do objeto, usado para definir seu estado.

Operações são serviços que podem ser requeridos de um objeto Sendo descrito por sua assinatura (nome da operação)



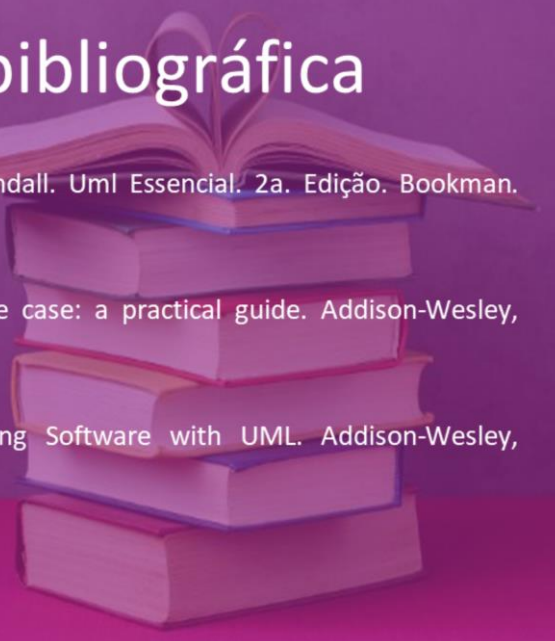
Exemplo Usando a notação gotUML

```
ClassDiagram [frame=true framecolor=steelblue label="Class Diagram"] {

 abstract class Pessoa {
  private id : int
  private nome: String
  private idade: int
  public void mostraDados()
 }

 class Aluno {
  private curso: String
  private fase: int
  public verificarFormando (): boolean
  public verificarFormando (fase: int): boolean
}

 class Professor {
  private dalario : double
  private titulo : String
  public calcularImposto (porcentagem: double): double

 }
```

Aluno -g-> Pessoa
Professor -g-> Pessoa

}

Referência bibliográfica



FOWLER, Martin e SCOTT, Kendall. Uml Essencial. 2a. Edição. Bookman. Porto Alegre, 2000.

SCHNEIDER, Geri. Applying use case: a practical guide. Addison-Wesley, 1998.

OESTEREICH, Bernd. Developing Software with UML. Addison-Wesley, 1999.


ânima
EDUCAÇÃO
UNISUL
CRÉDITOS
COORDENAÇÃO
Vera Rejane Niedersberg Schuhmacher
PROFESSORES
Rafael Lessa
Daniella Vieira