Análise e Modelagem de Sistemas

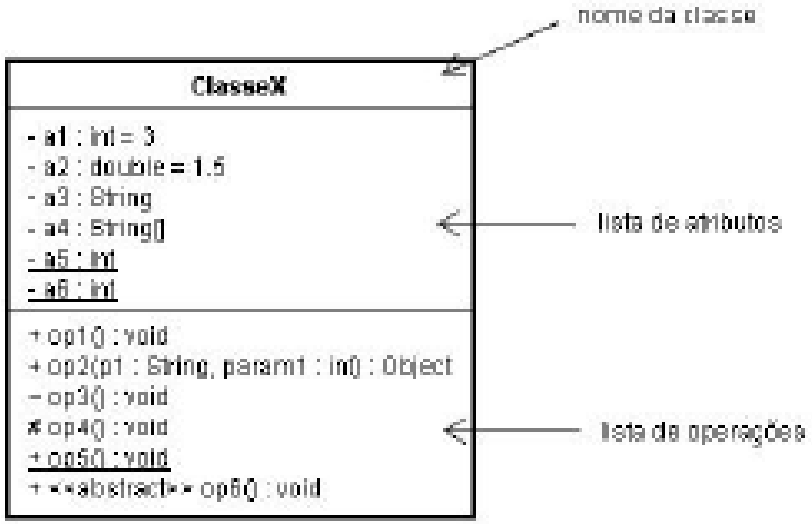
Prof. David S. Tosta

Agenda

- Revisão aula VI
- Encapsulamento e Herança
- Polimorfismo
- Abstração e classes abstratas

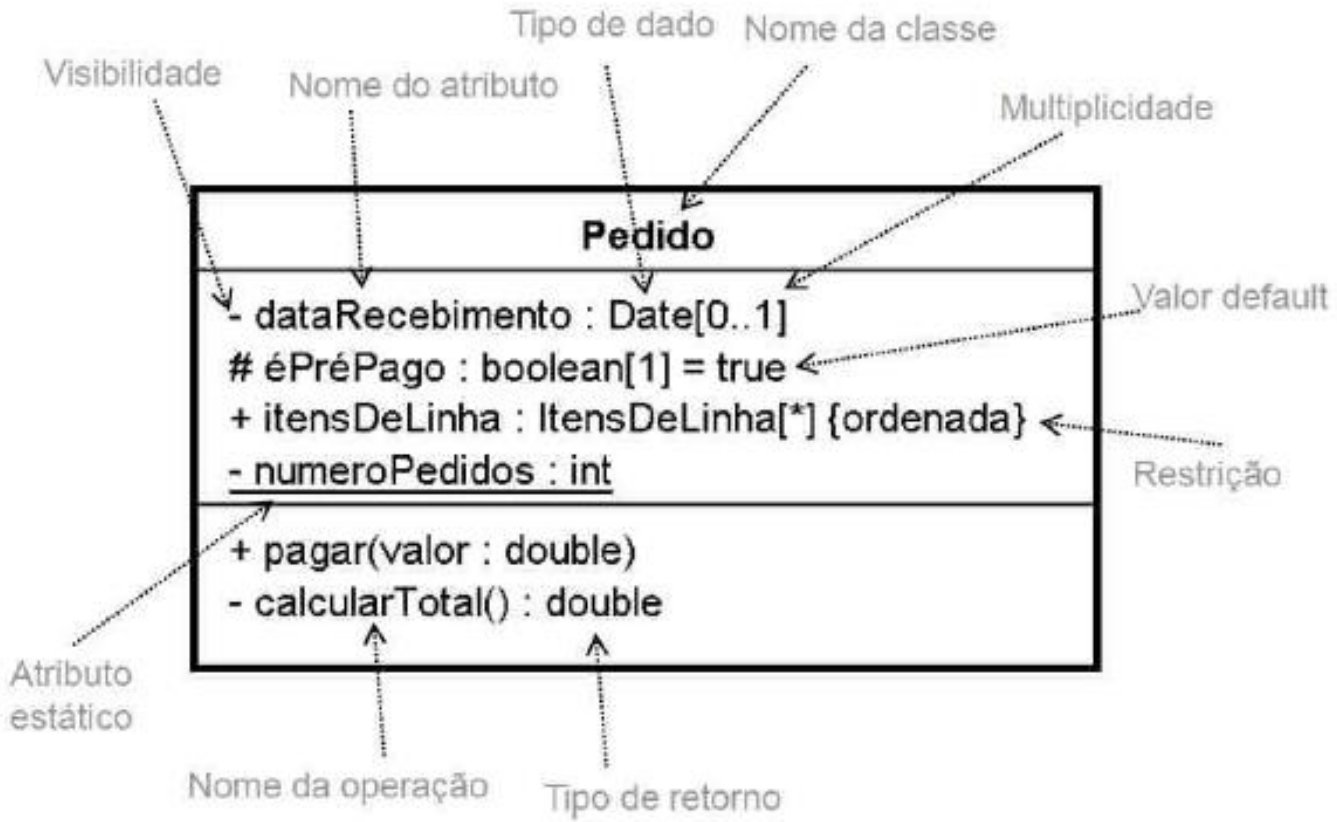
Revisão Aula VI

- O que é uma classe ?
  - Uma classe descreve um conjunto de objetos que têm os mesmos atributos, as mesmas operações e a mesma **semântica**;
  - Graficamente uma classe é representada por um retângulo dividido em três seções: nome, lista de atributos e lista das operações.



Revisão Aula VI

- Estrutura de uma Classe



Revisão Aula VI

- Exemplo de uma classe em Java

```
public class Pessoa {
  String nome;
  double dinheiroNaCarteira;

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

  public void gastar(double valor) {
    dinheiroNaCarteira -= valor;
  }

  public void receber(double valor) {
    dinheiroNaCarteira += valor;
  }
}
```

Revisão Aula VI

- O que é um objeto ?
  - Um objeto, em programação orientada a objetos, é uma instância (ou seja, um exemplar) de uma classe
  -
  - Chama-se instância de uma classe, um objeto cujo comportamento e estado são definidos pela classe.
  - As instâncias de uma classe compartilham o mesmo conjunto de atributos, embora sejam diferentes quanto ao conteúdo desses atributos.

Revisão Aula VI

- Exemplo de um objeto (classe pessoa)

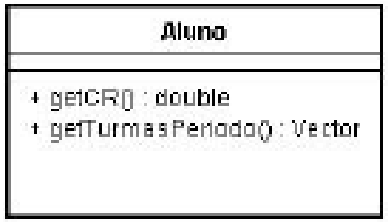
```
public static void main(String args[]) {

    // Cria um Objeto de Pessoa.
    Pessoa pVitor;
    pVitor = new Pessoa("Vitor Fernando Pamplona"));
}
```

Revisão Aula VI

- Atributo
  - É uma propriedade nomeada de uma classe que define um domínio de valores que um objeto – instância da classe – pode armazenar;
  - A definição de um atributo é válida para todos os objetos de uma classe;
  - Os valores de todos os atributos de um objeto determinam o estado do mesmo.

Revisão Aula VI

- Operações - Métodos
  - É a declaração de um serviço que pode ser solicitado às instâncias de uma classe;
  - A declaração de uma operação é válida para todos os objetos de uma classe;
  - Uma implementação de uma operação é chamada de **método**.



Revisão Aula VI

- Visibilidade
  - **público** – pode ser referenciado por qualquer elemento de modelagem externo ao classificador em questão;
  - **protegido** – pode ser referenciado pelos elementos de modelagem internos ao classificador em questão e pelos descendentes deste (generalização);
  - **privado** – pode ser referenciado **apenas** pelos elementos de modelagem internos ao classificador em questão;
  - **pacote** – pode ser referenciado por qualquer elemento de modelagem declarado no mesmo pacote do classificador em questão. Para os elementos externos, o elemento é tratado como **privado**.

  - Tipos de visibilidade:
    - pública "+"
    - protegida "**#**"
    - privada "-"
    - pacote "~"

Revisão Aula VI

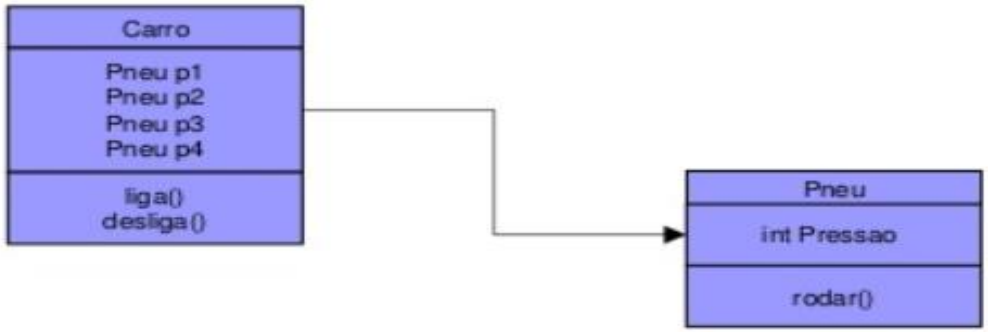
- Associação
  - Uma associação é um relacionamento estrutural cujo significado – semântica – é representativo para o problema sendo modelado;
  - Uma associação representa um conjunto de ligações entre os objetos das classes que participam da associação;
  - Ela define as regras de conexão entre estes objetos;
  - Uma associação binária é representada por uma linha sólida que conecta as duas classes envolvidas.

  - Uma seta de direcionamento não tem nenhum significado maior; apenas determina o sentido de leitura da associação.

Revisão Aula VI

- Associação – Entendendo melhor

  - Associação ocorre quando uma classe possui atributos do tipo de outra classe.



Nota : Neste caso estamos dizendo que carro possui pneu (4 pneus)

Revisão Aula VI

- Associação – Entendendo melhor

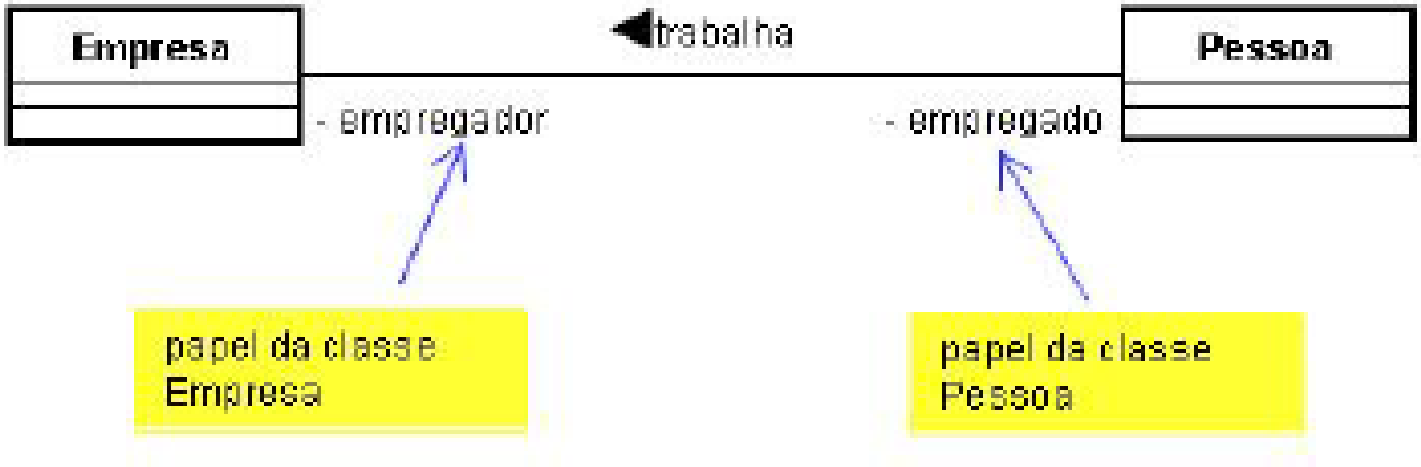
- A associação pode ser representada em Java da seguinte forma:

```
public class Pneu {
        int Pressao;

        void roda() {
                System.out.println("Pneu em movimento");
        }
}
```

```
public class Carro {
        Pneu p1;
        Pneu p2;
        Pneu p3;
        Pneu p4;

        void liga() {
                System.out.println("Carro ligado");
        }

        void desliga() {
                System.out.println("Carro desligado");
        }
}
```

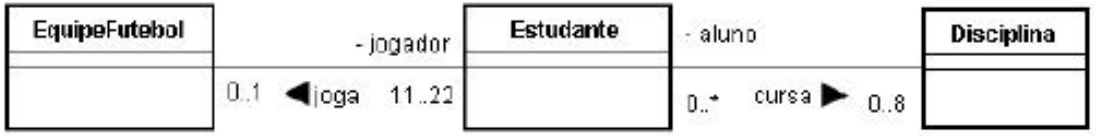
Revisão Aula VI

- Papeis
  - Quando uma classe participa de uma associação, ela desempenha um **papel** nesse relacionamento;



Revisão Aula VI

- Multiplicidades - Exemplo
  - Um **estudante** pode pertencer ou não a uma **equipe de futebol**;
  - Um **equipe de futebol** tem entre 11 e 22 **estudantes** jogando;
  - Um **estudante** pode cursar no máximo oito **disciplinas**, ou estar com a matrícula trancada;
  - Uma **disciplina** pode ser cursada por um número indeterminado de **estudantes**.

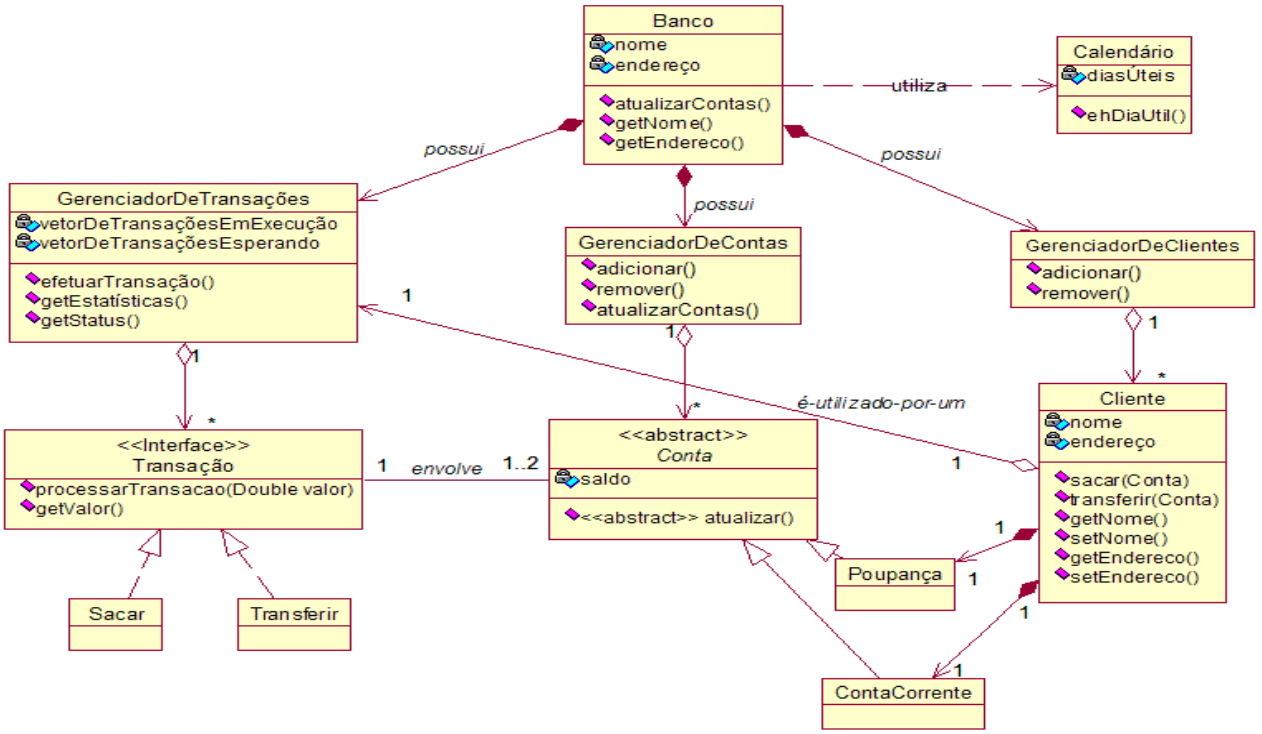


Revisão Aula VI

- Perspectivas
  - Um diagrama de classes pode oferecer três perspectivas, cada uma para um tipo de observador diferente. São elas:
    - Conceitual - Representa os conceitos do domínio em estudo. Perspectiva destinada ao cliente.
  - Especificação
    - Tem foco nas principais interfaces da arquitetura, nos principais métodos, e não como eles irão ser implementados.
    - Perspectiva destinada as pessoas que não precisam saber detalhes de desenvolvimento, tais como gerentes de projeto.

Revisão Aula VI

- Perspectivas
  - Implementação - a mais utilizada de todas (exemplo)
    - Aborda vários detalhes de implementação, tais como navegabilidade, tipo dos atributos, etc. Perspectiva destinada ao time de desenvolvimento.



Encapsulamento

- Encapsulamento
  - O conceito do encapsulamento consiste em “esconder” os atributos da classe de quem for utilizá-la. Isso se deve por dois motivos principais.
  - O primeiro é para que alguém que for usar a classe não a use de forma errada como, por exemplo, em uma classe que tem um método de divisão entre dois atributos da classe - se o programador java não conhecer a implementação interna da classe, ele pode colocar o valor zero no atributo do dividendo.

Encapsulamento

- Encapsulamento
  - Se a classe estiver corretamente encapsulada podemos impedir que o programador faça isso. Esse tipo de implementação é feito via os métodos get e set.

Encapsulamento

- Encapsulamento

```
package com.devmedia.model;

public class Divisao {

	private int num1;
	private int num2;

	public void divisao() {
		System.out.println("A divisao e: " + (num1 / num2));
	}

	public int getNum1() {
		return num1;
	}

	public void setNum1(int num1) {
		this.num1 = num1;
	}

	public int getNum2() {
		return num2;
	}

	public void setNum2(int num2) {
		if (num2 == 0) {
			num2 = 1;
		} else {
			this.num2 = num2;
		}
	}
```

Encapsulamento

- Encapsulamento
  - O outro motivo é de manter todo o código de uma determinada classe encapsulada dentro dela mesmo.
  - Por exemplo, se existe uma classe Conta, talvez seja melhor não permitir que um programador acesse o atributo saldo diretamente, nem mesmo com os métodos get e set, mas somente por operações, como saque, depósito e saldo.

Encapsulamento

- Encapsulamento
  - Veja o exemplo de implementação da classe Conta, onde só é possível acessar a atributo saldo pelas operações disponibilizadas na classe e os outros atributos podem ser acessados via métodos get e set.

Encapsulamento

- Encapsulamento

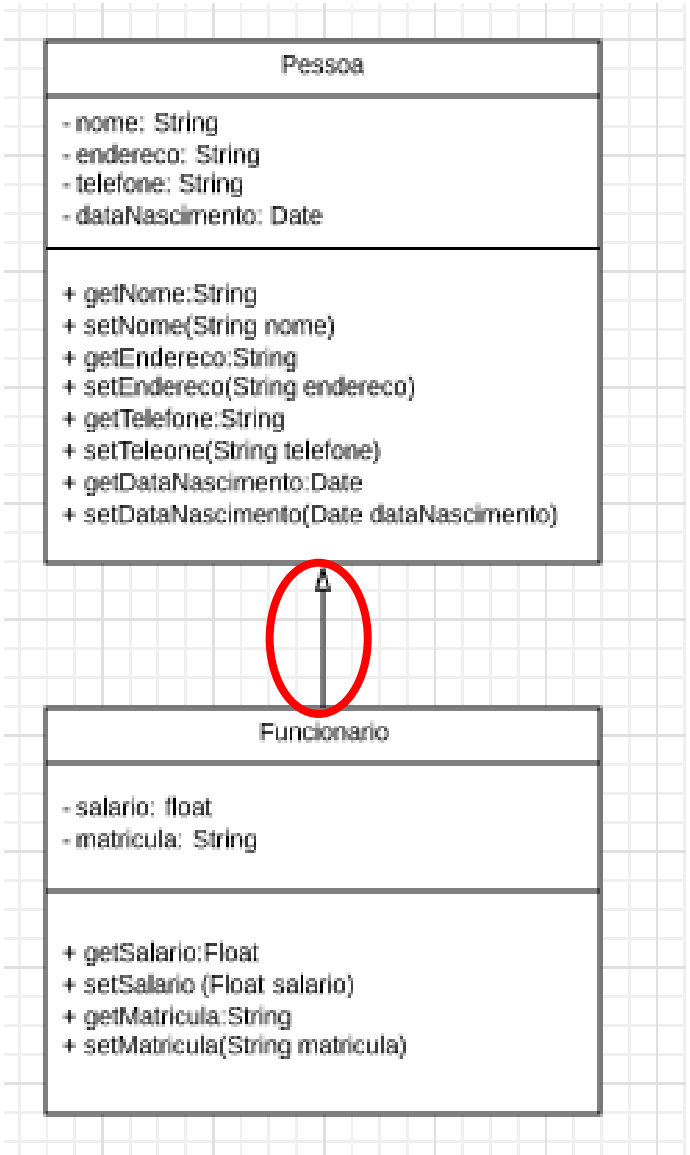
```
package com.devmedia.model;
public class Conta {
    private String agencia;
    private String numero;
    private float saldo;

    public void saque(float valor) {
        if (saldo < valor) {
            System.out.println("O saldo é insuficiente!");
        } else {
            saldo -= valor;
        }
    }
    public String getAgencia() {
        return agencia;
    }
    public void setAgencia(String agencia) {
        this.agencia = agencia;
    }
    public String getNumero() {
        return numero;
    }
    public void setNumero(String numero) {
        this.numero = numero;
    }
    public void deposito(float valor) {
        saldo += valor;
    }
    public void saldo() {
        System.out.println("O saldo é: " + saldo);
    }
}
```

Herança

- Herança
  - Relacionamento entre um elemento mais geral e um mais específico. Onde o elemento mais específico herda as propriedades e métodos do elemento mais geral.
  - A herança é uma relação de generalização.
  - Como a relação de dependência, ela existe só entre as classes.
  - Um objeto particular não é um caso geral de um outro objeto, só conceitos (classes no modelo a objetos) são generalização de outros conceitos.

Herança

- Herança



Herança

- Herança
  - A herança é um tipo de relacionamento que define que uma classe "é um" de outra classe como, por exemplo, a classe Funcionário que é uma Pessoa.
  - Assim um Funcionário tem um relacionamento de herança com a classe Pessoa.
  - Em algumas linguagens, como C, é possível fazer herança múltipla, isto é, uma classe pode herdar de diversas outras classes.

Herança

- Herança
  - Porém, em Java isso não é permitido pois cada classe pode herdar de apenas outra classe.
  - O próximo exemplo mostra como implementar a herança entre classes: a classe Pessoa definida será utilizada como base para as outras classes.
  - Será definida a classe Funcionário, com os atributos salario e matricula. Como é possível observar, para implementar herança em Java usaremos a palavra reservada extends.

Herança

- Classe Pessoa

```
package com.devmedia.model;
import java.util.Date;
public class Pessoa {
	private String nome;
	private String endereco;
	private String telefone;
	private Date dataNascimento;
	public String getNome() {
		return nome;
	}
	public void setNome(String nome) {
		this.nome = nome;
	}
	public String getEndereco() {
		return endereco;
	}
	public void setEndereco(String endereco) {
		this.endereco = endereco;
	}
	public String getTelefone() {
		return telefone;
	}
	public void setTelefone(String telefone) {
		this.telefone = telefone;
	}
	public Date getDataNascimento() {
		return dataNascimento;
	}
	public void setDataNascimento(Date dataNascimento) {
		this.dataNascimento = dataNascimento;
	}
}
```

Herança

- Classe Funcionário

```
package com.devmedia.model;

public class Funcionario extends Pessoa {

	private float salario;
	private String matricula;

	public float getSalario() {
		return salario;
	}

	public void setSalario(float salario) {
		this.salario = salario;
	}

	public String getMatricula() {
		return matricula;
	}

	public void setMatricula(String matricula) {
		this.matricula = matricula;
	}

}
```

Herança

- Exemplo de trecho de código utilizando herança

```
public class Teste{
    public static void main(String [] args) {
        Funcionario operador = new Funcionario();
        operador.setNome("Marcos");
        operador.setEndereco("Av. Rio Branco, 234 - Rio de Janeiro");
        operador.setTelefone("(21) 3456-3455");
        operador.setMatricula("A1234");
        operador.setSalario(5400);
    }
}
```

Herança

- Tipos de acesso aos atributos
  - Sobre a herança em Java é importante saber que existem quatro tipos de acesso aos atributos:
    - public: que permite que métodos e atributos sejam acessados diretamente de qualquer classe;
    - private: que permite que métodos e atributos sejam acessados apenas dentro da classe;
    - protected: que permite que métodos e atributos sejam acessados apenas dentro da própria classe e em classes filhas;
    - tipo de acesso padrão, que permite que métodos e atributos sejam acessadas por qualquer classes que esteja no mesmo pacote.

Polimorfismo

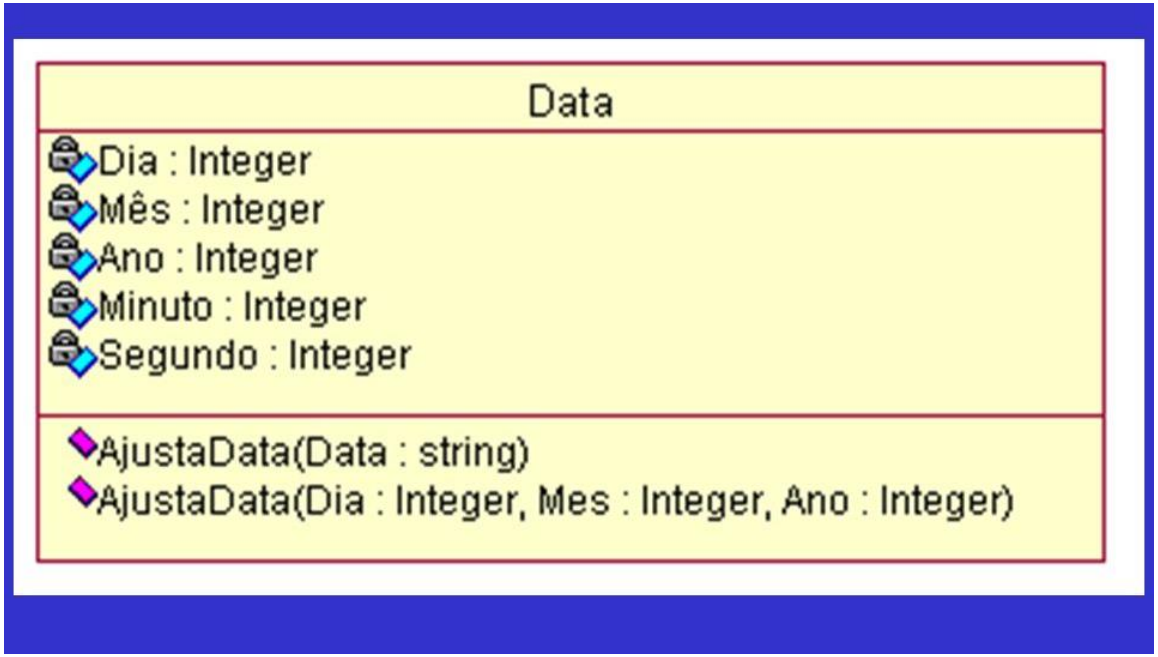
- O que é polimorfismo ?
  - É a característica única de linguagens orientadas a objetos que permite que diferentes objetos respondam a mesma mensagem cada um a sua maneira.
  - Em termos de programação, polimorfismo representa a capacidade de uma única referência invocar métodos diferentes, dependendo do seu conteúdo.
  - De maneira geral o polimorfismo permite a criação de programas mais claros, pois elimina a necessidade de darmos nomes diferentes para métodos que conceitualmente fazem a mesma coisa, e também programas mais flexíveis, pois facilita em muito a extensão dos mesmos.

Polimorfismo

- Polimorfismo Estático - Exemplo
  - Polimorfismo pode ser de duas formas, estático ou dinâmico:
  - Polimorfismo Estático: Ocorre quando na definição de uma classe criamos métodos com o mesmo nome, porém com argumentos diferentes. Dizemos neste caso que o método está sobrecarregado (overloading).
  - A decisão de qual método chamar é tomada em tempo de compilação, baseada nos argumentos que foram passados.

Polimorfismo

- Polimorfismo Estático - UML



Polimorfismo

- Polimorfismo Estático - Exemplo

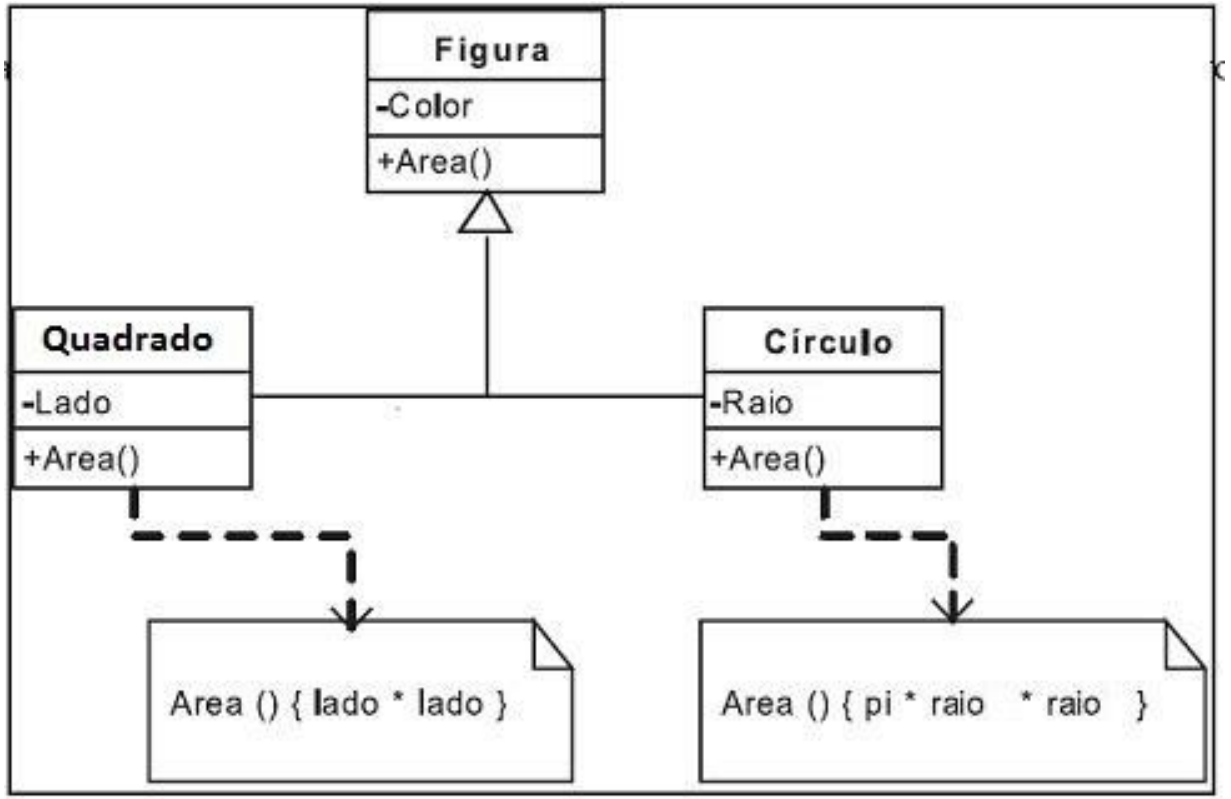
```
package com.devmedia.model;
public class Pessoa {
    private String nome;
    private String endereco;
    private String telefone;

    ......
    ......
    ......

    public String recuperarIdade(Pessoa p){
    ....
    }

    public Integer recuperarIdade(Pessoa pessoa){
    ....
    }

    public Integer recuperarIdade(String pessoa){
    ....
    }
```

Polimorfismo

- O que é polimorfismo ?
  - Polimorfismo Dinâmico: Esta associado com o conceito de herança e ocorre quando uma subclasse redefine um método existente na superclasse.
  - Dizemos neste caso que o método foi sobreescrito (overriding) na subclasse.
  - A decisão de qual método executar é tomada somente em tempo de execução.

Polimorfismo

- Polimorfismo Dinâmico - UML



Polimorfismo

- Polimorfismo Dinâmico - Exemplo

```
public class Forma {
	private Integer base;
	private Integer altura;
	private Integer raio;


	public Double calcularArea() {
		return base * altura;
	}

	public Integer getBase() {
		return base;
	}
	public void setBase(Integer base) {
		this.base = base;
	}
	public Integer getAltura() {
		return altura;
	}
	public void setAltura(Integer altura) {
		this.altura = altura;
	}
	public Integer getRaio() {
		return raio;
	}
	public void setRaio(Integer raio) {
		this.raio = raio;
	}
}
```

Polimorfismo

- Polimorfismo Dinâmico - Exemplo

```
public class Circulo extends Forma {
        public Double calcularArea() {
                return 3,14 * getRaio()* getRaio();
        }
}
```

Abstração e Classe Abstrata

- O que é Abstração?
  - É a habilidade de concentrar nos aspectos essenciais de um contexto qualquer, ignorando características menos importantes ou acidentais.
  - Em modelagem orientada a objetos, uma classe é uma abstração de entidades existentes no domínio do sistema de software.
  - Por exemplo, imaginamos a abstração referente a classe Animais. Há várias entidades na classe Animais como Anfíbios, Répteis e Mamíferos que são também sub-classes da classe Animais.

Abstração e Classe abstrata

- O que é Classe abstrata?
  - As classes abstratas são as que não permitem realizar qualquer tipo de instância.
  - São classes feitas especialmente para serem modelos para suas classes derivadas.
  - As classes derivadas, via de regra, deverão sobrescrever os métodos para realizar a implementação dos mesmos.
  - As classes derivadas das classes abstratas são conhecidas como classes concretas.

Abstração e Classe abstrata

- O que é Classe abstrata?
  - Para ter um objeto de uma classe abstrata é necessário criar uma classe mais especializada herdando dela e então instanciar essa nova classe.
  - Os métodos da classe abstrata devem então serem sobrescritos nas classes filhas.

Abstração e Classe abstrata

- O que é Classe abstrata?
  - Outro fato importante de classes abstratas é que elas podem conter ou não métodos abstratos, que tem a mesma definição da assinatura de método encontrada em interfaces. Ou seja, uma classe abstrata pode implementar ou não um método.
  - Os métodos abstratos definidos em uma classe abstrata devem obrigatoriamente ser implementados em uma classe concreta.
  - Mas se uma classe abstrata herdar outra classe abstrata, a classe que herda não precisa implementar os métodos abstratos.

Abstração e Classe abstrata

- Classe abstrata - UML

- A classe Trabalhador está em itálico e por isso é definida como uma classe abstrata
- A operação calcularCusto também está em itálico e é considerada uma operação abstrata
- Não se pode criar objetos a partir da classe Trabalhador, já que ela é uma classe abstrata
- É possível se criar objetos a partir das classes Pedreiro e Serralheiro
- As classes Pedreiro e Serralheiro são obrigadas a implementar a operação abstrata calcularCusto.
- A operação verificarSituacao não é uma operação abstrata e por isso, as classes Pedreiro e Serralheiro não a precisa implementar

Abstração e Classe abstrata

- Classe abstrata - Exemplo

```
abstract class Animal{
    abstract String getHabitat();
    public String getRaca(){
        return "Raça indefinida";
    }
}

class Cachorro extends Animal{
    public String getHabitat(){
        return "";
    }
}

class Gato extends Animal{
    public String getHabitat(){
        return "indefinido";
    }
    public String getRaca(){
        return "Munchkin";
    }
}
```