Bacharelado em Engenharia de Software
Disciplina: Programação I
Professor Marcelo de Souza

Departamento de Engenharia de Software
Centro de Educação Superior do Alto Vale do Itajaí - CEAVI
Universidade do Estado de Santa Catarina - UDESC

Programação orientada a objetos
Conceitos básicos

"Qualquer um pode escrever um código que o computador entenda. Bons programadores escrevem códigos que os humanos entendam."

Martin Fowler

Orientação a objetos

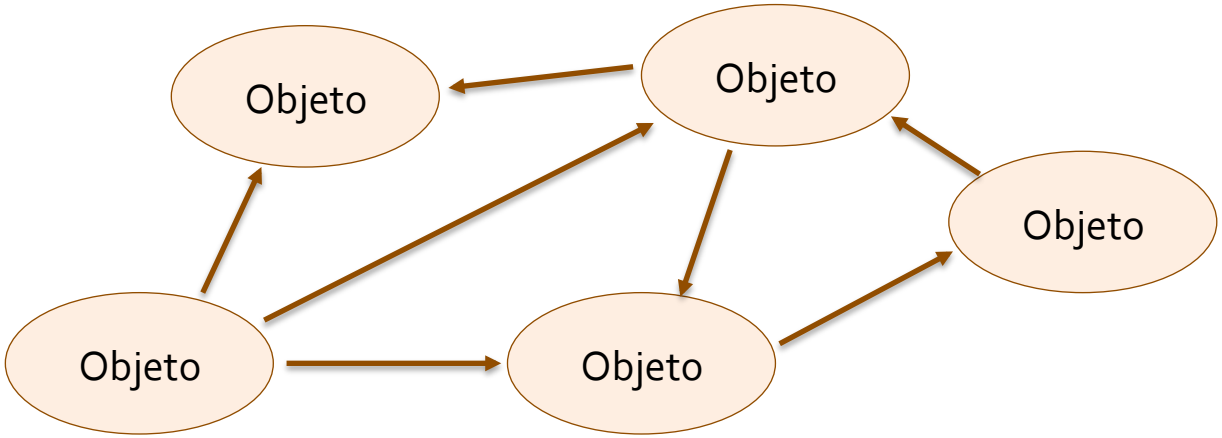
Histórico

- O paradigma orientado a objetos surgiu no final da década de 80.
- Alan Kay, um de seus idealizadores, formulou a chamada "analogia biológica", onde propôs que um sistema de software deveria funcionar como um ser vivo.
- Neste sistema, cada célula interage com outras células através do envio de mensagens para realizar um objetivo comum. Adicionalmente, cada célula funciona como uma unidade autônoma.

Orientação a objetos

Histórico

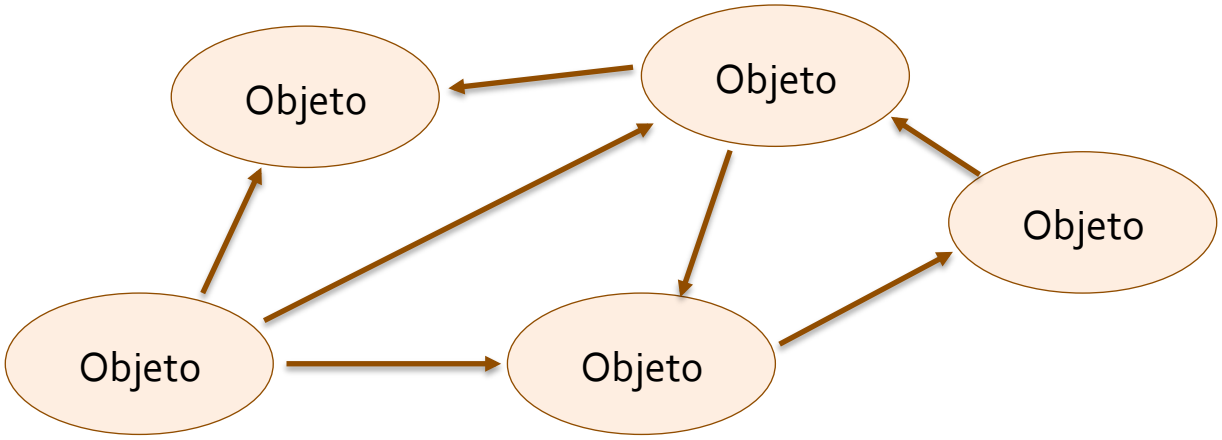
- O paradigma orientado a objetos surgiu no final da década de 80.
- Alan Kay, um de seus idealizadores, formulou a chamada "analogia biológica", onde propôs que um sistema de software deveria funcionar como um ser vivo.
- Neste sistema, cada célula interage com outras células através do envio de mensagens para realizar um objetivo comum. Adicionalmente, cada célula funciona como uma unidade autônoma.



Orientação a objetos

Fundamentos

- Qualquer coisa é um objeto.
- Objetos realizam tarefas através da requisição de serviços a outros objetos.
- Cada objeto pertence a uma determinada classe, que agrupa objetos similares.
- A classe define a estrutura e o comportamento dos seus objetos.
- Classes são organizadas em hierarquias.



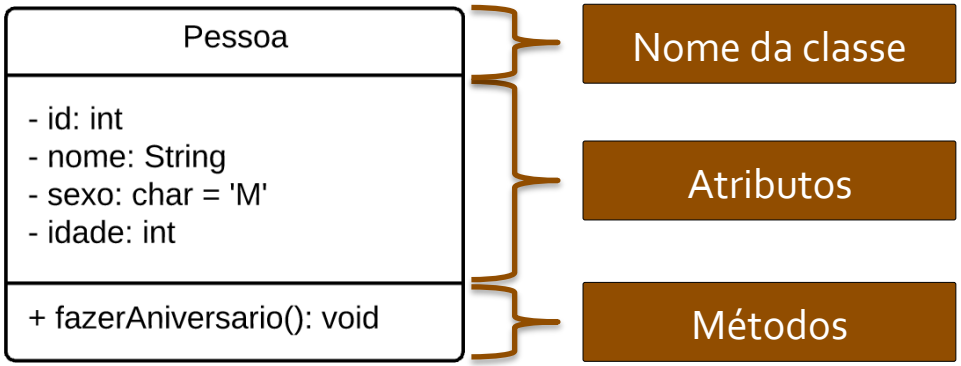
Classe

- Uma classe define a estrutura e o comportamento de um conjunto de objetos.
- A estrutura é definida pelos atributos da classe, enquanto o comportamento é definido pelos métodos que a classe implementa.

Classe

- Uma classe define a estrutura e o comportamento de um conjunto de objetos.
- A estrutura é definida pelos atributos da classe, enquanto o comportamento é definido pelos métodos que a classe implementa.

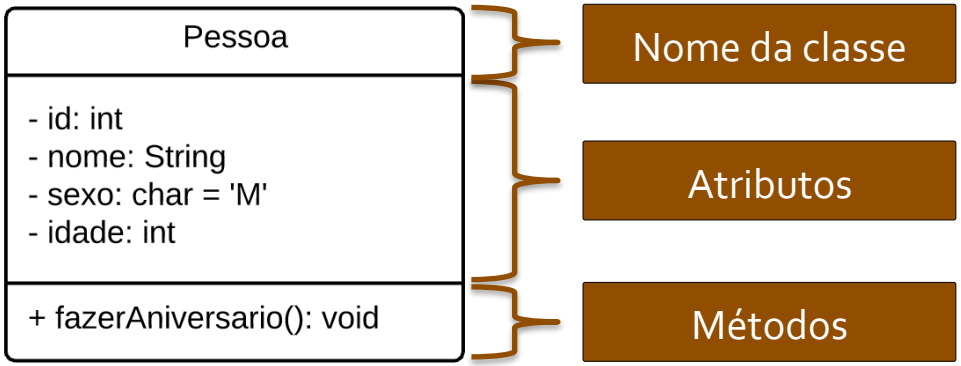
Representação de uma classe usando UML



Classe

Atributos

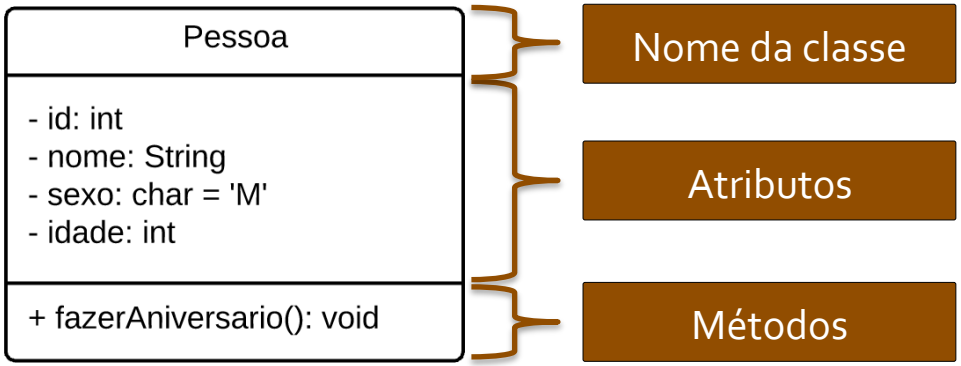
- Uma propriedade nomeada de uma classe, descrevendo uma faixa de valores que seus objetos poderão manter.
- Definem as características (estrutura) presentes nos objetos da classe.
- Os atributos dependem do domínio em questão.
- O valor de todos os atributos definem o **estado** do objeto.



Classe

Métodos

- Determinam o comportamento do objeto, ou seja, como ele age e reage.
  - Suas modificações de estado e interações com outros objetos.
- Os métodos definem o conjunto de operações que o objeto pode realizar. Estas operações podem ser solicitadas por outros objetos.

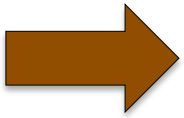


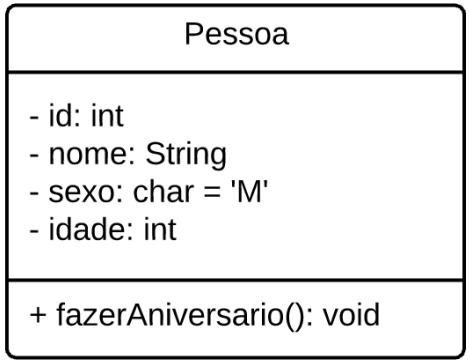
Classe

- Uma classe é uma **abstração** das características de algo do mundo real.
  - Abstrair significa focar nos elementos importantes de uma entidade ou processo, ignorando características e particularidades que não são de interesse a um determinado propósito.
  - Ao modelar uma entidade real através de uma classe, apenas as características relevantes da entidade são mantidas.
  - A definição do que é relevante e o que não é depende do contexto.

Classe

- Uma classe é uma **abstração** das características de algo do mundo real.
  - Abstrair significa focar nos elementos importantes de uma entidade ou processo, ignorando características e particularidades que não são de interesse a um determinado propósito.
  - Ao modelar uma entidade real através de uma classe, apenas as características relevantes da entidade são mantidas.
  - A definição do que é relevante e o que não é depende do contexto.

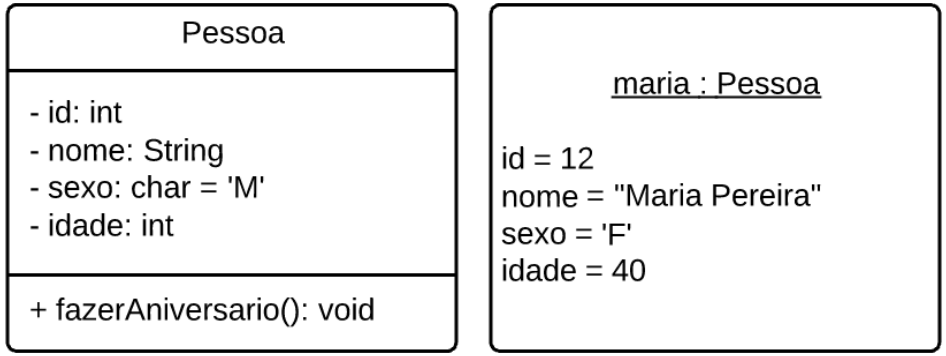




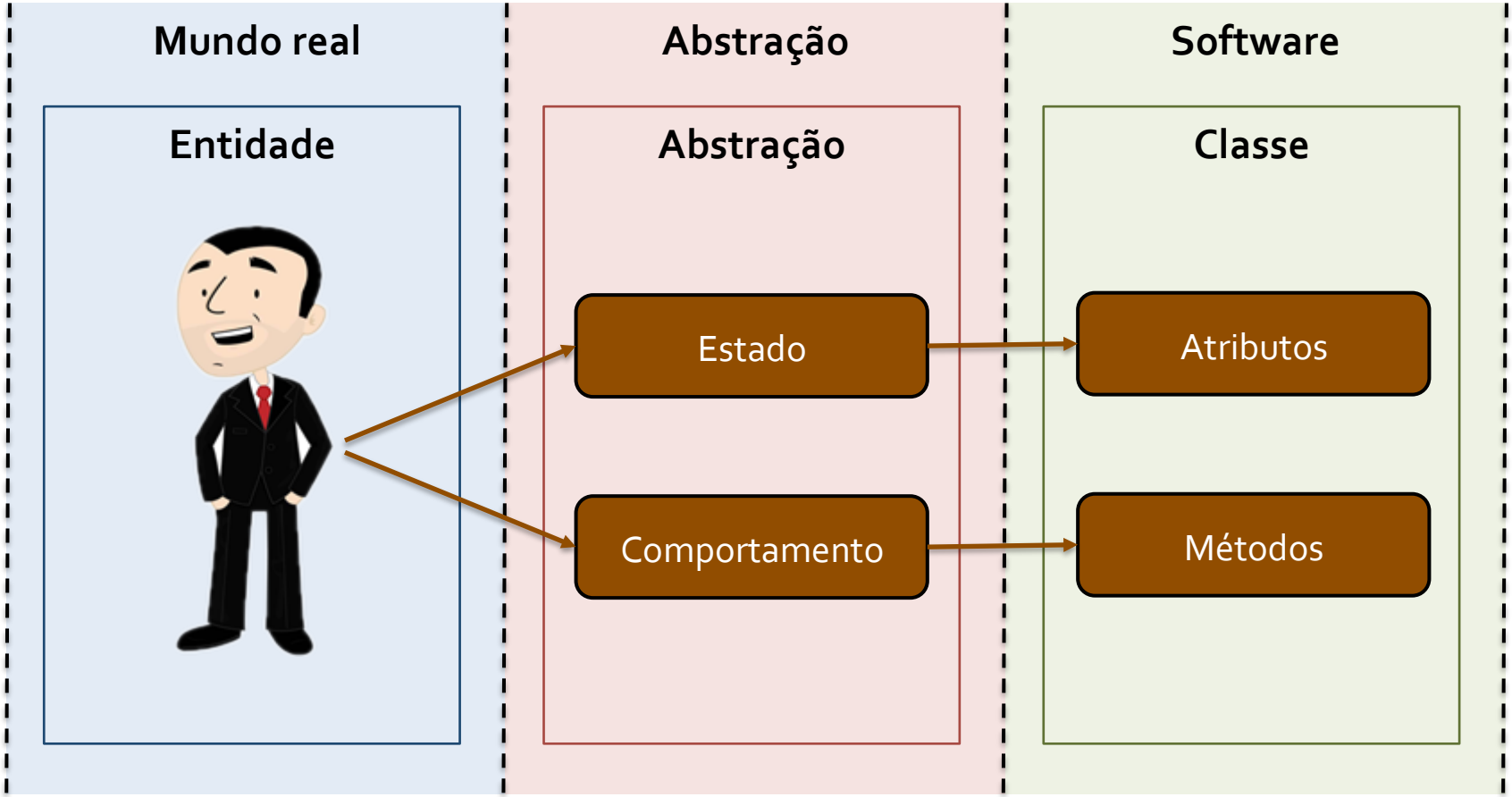
Exemplo: quais as características relevantes de uma pessoa para um sistema de empréstimos? E para um sistema de academia?

Objeto

- Objetos são instâncias de classes.
  - Enquanto uma classe é uma abstração, um objeto é uma manifestação concreta dessa abstração.
- É uma entidade que possui **estado** (os valores dos seus atributos), **comportamento** (a implementação dos seus métodos) e **identidade**.



Criação de classes



Criação de classes

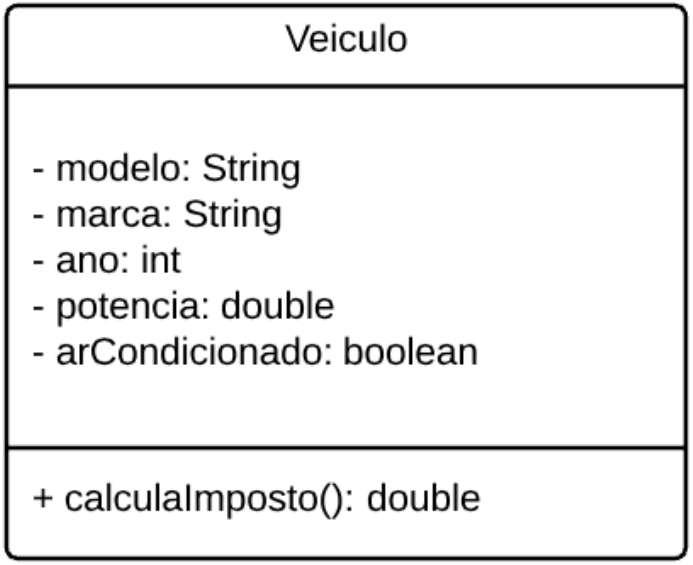
Exemplo

- Representação de um **veículo** (entidade) para uma **concessionária** (contexto).
- **Abstração:** quais as características relevantes da entidade?
  - **Atributos:** modelo, marca, ano, cor, potência, ar condicionado.
  - **Métodos:** método que calcula o imposto com base no ano do veículo.

Criação de classes

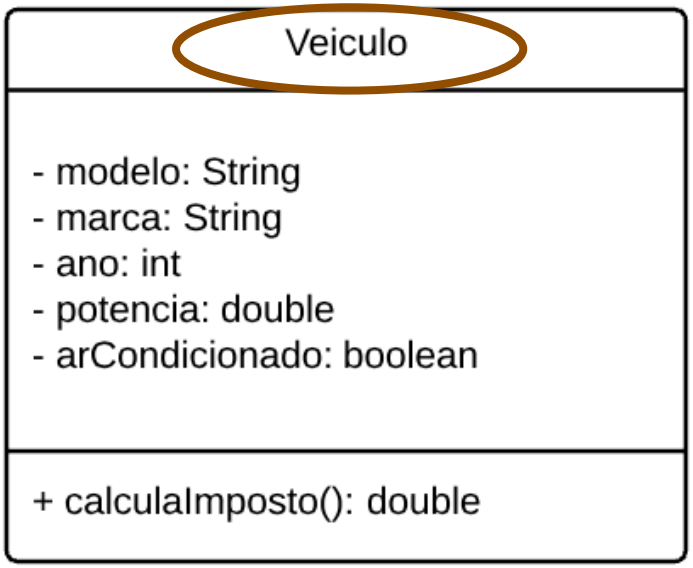
Exemplo

- Representação de um **veículo** (entidade) para uma **concessionária** (contexto).
- **Abstração:** quais as características relevantes da entidade?
  - **Atributos:** modelo, marca, ano, cor, potência, ar condicionado.
  - **Métodos:** método que calcula o imposto com base no ano do veículo.



Criação de classes

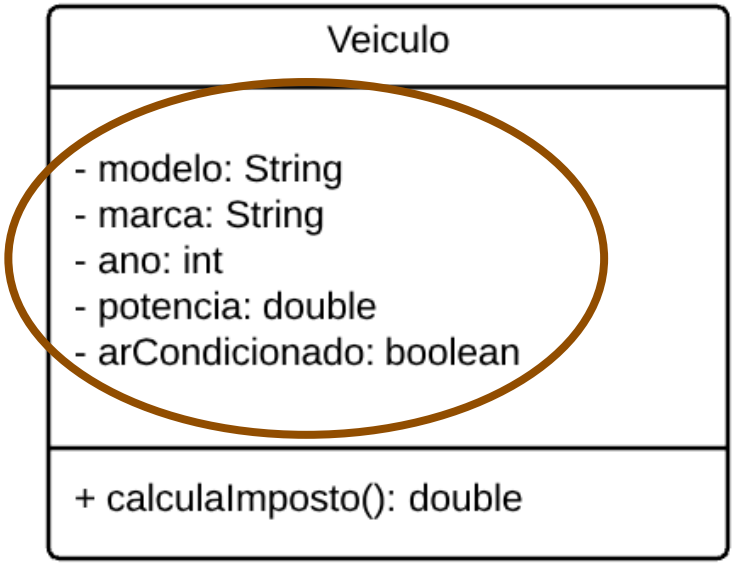
Nome da classe



Deve ser um substantivo no singular, que defina a satisfatoriamente a abstração.

Criação de classes

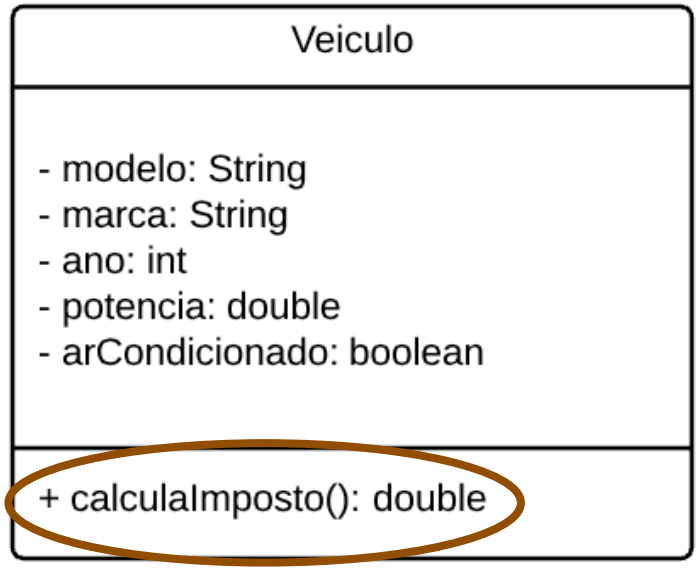
Atributos



Sintaxe é composta pelo nome do atributo, seguido do seu valor. Opcionalmente pode ser definido um valor inicial.

Criação de classes

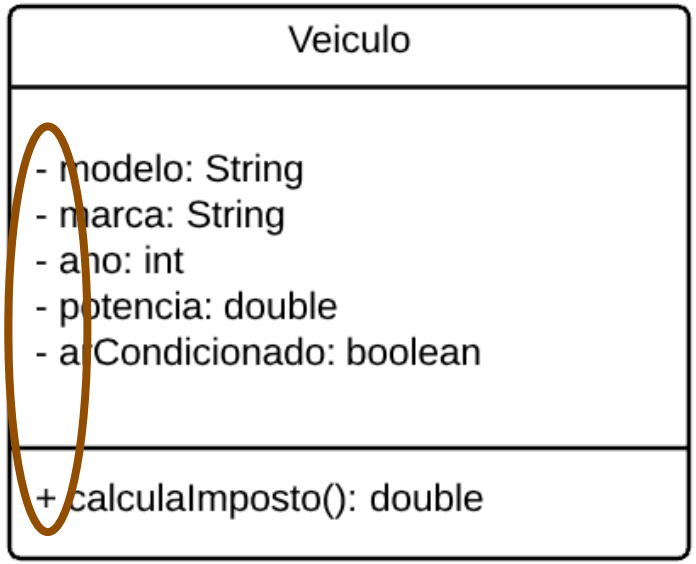
Métodos



Sintaxe é composta pelo nome do método, seguido da lista de parâmetros (se houver) e o tipo de retorno.

Criação de classes

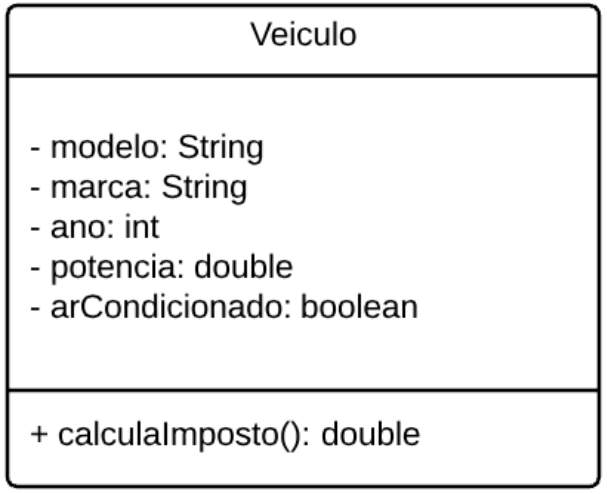
Modificadores de acesso



Definem quais objetos podem acessar os valores armazenados nos atributos ou chamar a execução dos métodos.

Criação de classes

Implementação de classes

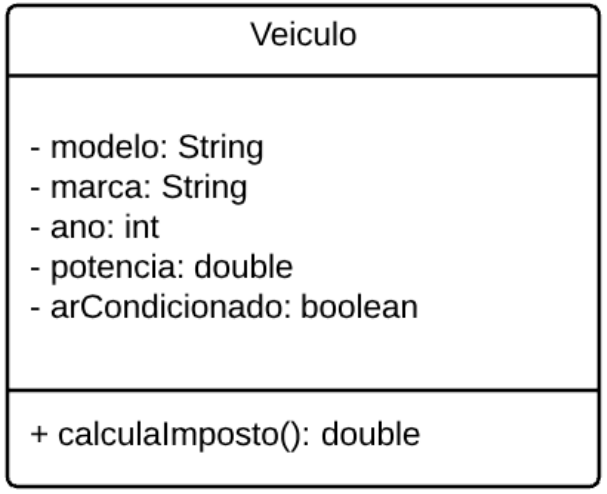



```
public class Veiculo {

    private String modelo;
    private String marca;
    private int ano;
    private double potencia;
    private boolean arCondicionado;

    public double calculaImposto() {
        if(ano < 2010)
            return 500d;
        return 700d;
    }
}
```

Criação de classes

Implementação de classes

```
public class Veiculo {

    private String modelo;
    private String marca;
    private int ano;
    private double potencia;
    private boolean arCondicionado;

    public double calculaImposto() {
        if(ano < 2010)
            return 500d;
        return 700d;
    }
}
```

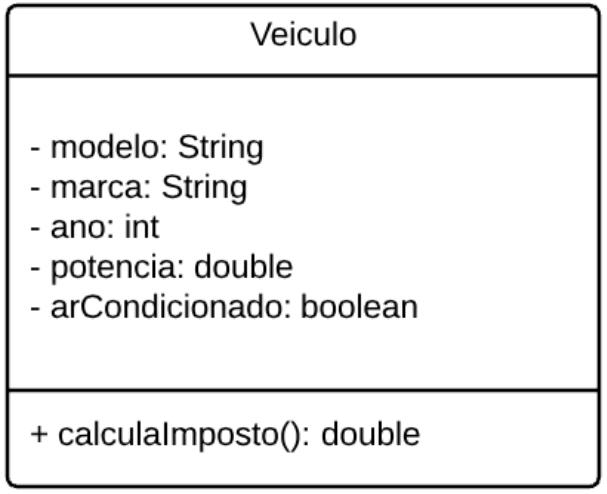


Classe é definida através dos termos **public class**, seguido do nome da classe.

Criação de classes

Implementação de classes

```
public class Veiculo {

    private String modelo;
    private String marca;
    private int ano;
    private double potencia;
    private boolean arCondicionado;

    public double calculaImposto() {
        if(ano < 2010)
            return 500d;
        return 700d;
    }
}
```

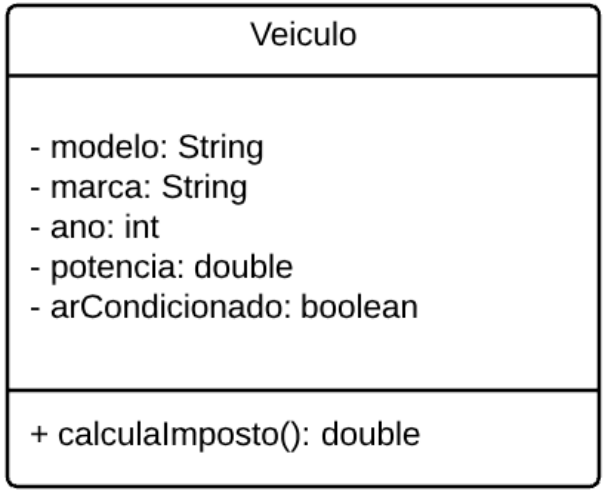


Os atributos são definidos pelo seu modificador, seu tipo e seu identificador.

Criação de classes

Implementação de classes

```
public class Veiculo {

    private String modelo;
    private String marca;
    private int ano;
    private double potencia;
    private boolean arCondicionado;

    public double calculaImposto() {
        if(ano < 2010)
            return 500d;
        return 700d;
    }
}
```

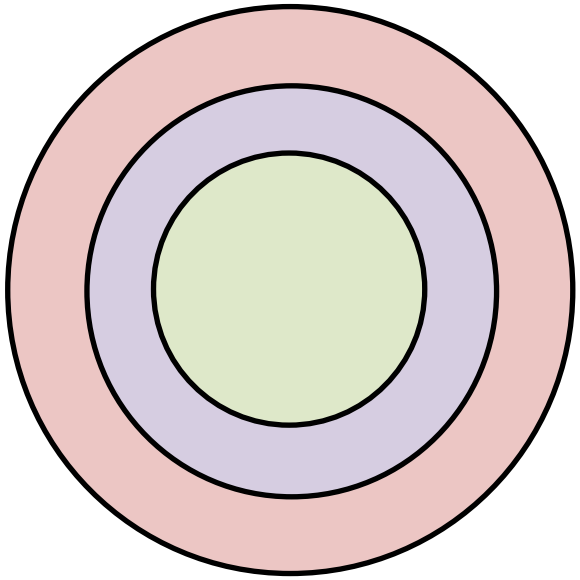


Os métodos são definidos pelo seu modificador, identificador, a lista de parâmetros e sua implementação.

Encapsulamento

Modificadores de acesso

- **Privado (-):** O elemento é acessado apenas pela classe que o define.
- **Protegido (#):** O elemento é acessado pela classe que o define e também por suas subclasses.
- **Público (+):** Qualquer objeto pode acessar o elemento.

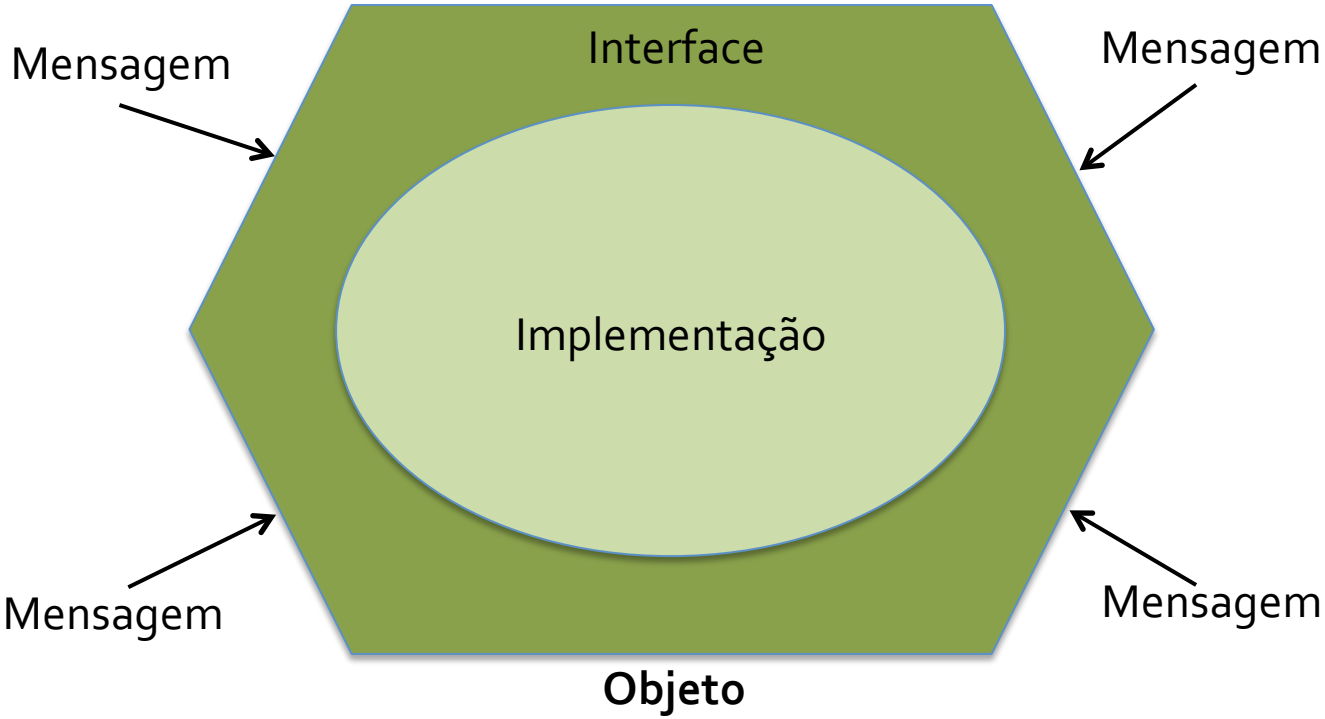


Encapsulamento

- Consiste em agrupar e empacotar os detalhes internos da abstração e torná-los inacessíveis para entidades externas.
- **Exemplos**

Encapsulamento

- Consiste em agrupar e empacotar os detalhes internos da abstração e torná-los inacessíveis para entidades externas.



Encapsulamento

```java
public class ContaBancaria {

    public double saldo;

    public boolean saque(double valor) {

        if(saldo >= valor){
            saldo -= valor;
            return true;
        }

        return false;
    }
}
```

```java
public class ContaBancaria {

    private double saldo;

    public boolean saque(double valor) {

        if(saldo >= valor){
            saldo -= valor;
            return true;
        }

        return false;
    }
}
```

Qual o problema com a primeira classe?

Métodos acessores

- Por conta do encapsulamento, os atributos das classes devem ser privados.
- Os métodos acessores (*get* e *set*) são responsáveis por devolver os valores dos atributos quando solicitado, bem como alterar seus valores quando solicitado.
- **Getters** (ex: *getNome()*)
  - Responsáveis por devolver o valor de um atributo do objeto.
  - Geralmente não possuem argumentos e possuem um tipo de retorno.
  - São públicos por padrão.
- **Setters** (ex: *setNome(String nome)*)
  - Responsáveis por realizar a modificação dos valores dos atributos.
  - Geralmente possuem como argumento o valor a ser atribuído e não possuem retorno.
  - São públicos por padrão.

Métodos acessores

- Por conta do encapsulamento, os atributos das classes devem ser privados.
- Os métodos acessores (*get* e *set*) são responsáveis por devolver os valores dos atributos quando solicitado, bem como alterar seus valores quando solicitado.

```
public class Veiculo {

    private String modelo;

    public void setModelo(String modelo) {
        this.modelo = modelo;
    }

    public String getModelo() {
        return this.modelo;
    }
}
```

Construtores e destrutores

O que acontece quando um objeto é criado – *pessoa = new Pessoa()*?

1. Inicialização default dos campos (null, 0, false).
2. Chamada recursiva aos construtores de cada superclasse (até Object).
    1. Inicialização default dos campos das superclasses.
    2. Execução do conteúdo dos métodos construtores de cada superclasse (desde Object).
3. Execução do conteúdo do método construtores da classe.

Construtores e destrutores

O que acontece quando um objeto é criado – *pessoa = new Pessoa()*?

1. Inicialização default dos campos (null, 0, false).
2. Chamada recursiva aos construtores de cada superclasse (até Object).
    1. Inicialização default dos campos das superclasses.
    2. Execução do conteúdo dos métodos construtores de cada superclasse (desde Object).
3. **Execução do conteúdo do método construtores da classe.**

É possível definir o que será realizado quando um objeto da classe é criado.

Construtores e destrutores

Método construtor

- Quando não especificado um método construtor, o Java atribui um construtor padrão (sem argumentos e sem implementação).
- Quando o programador define um método construtor, o construtor default deixa de existir.
- O método construtor não possui um tipo de retorno e seu nome é idêntico ao nome da classe.

Construtores e destrutores

Método construtor

- Quando não especificado um método construtor, o Java atribui um construtor padrão (sem argumentos e sem implementação).
- Quando o programador define um método construtor, o construtor default deixa de existir.
- O método construtor não possui um tipo de retorno e seu nome é idêntico ao nome da classe.

```
public class Pessoa {

    private String nome;
    private int idade;

    public Pessoa() {
        //Neste exemplo, são definidor valores
        //iniciais para cada objeto criado
        this.nome = "Não especificado";
        this.idade = 1;
    }
}
```

Construtores e destrutores

Método construtor

- Quando não especificado um método construtor, o Java atribui um construtor padrão (sem argumentos e sem implementação).
- Quando o programador define um método construtor, o construtor default deixa de existir.
- O método construtor não possui um tipo de retorno e seu nome é idêntico ao nome da classe.

```
public class Pessoa {

    private String nome;
    private int idade;

    public Pessoa() {
        //Neste exemplo, são definidor valores
        //iniciais para cada objeto criado
        this.nome = "Não especificado";
        this.idade = 1;
    }
}
```

O termo **this** é usado para acessar a instância do objeto atual. É usado para diferenciar atributos de variáveis locais.

Construtores e destrutores

Método construtor

- É possível (e comum) definir argumentos no método construtor, que são passados no momento em que o objeto é criado.
- Com isso, valores já podem ser armazenados nos atributos da instância.

```
public class Pessoa {

    private String nome;
    private int idade;

    public Pessoa() {
        this.nome = "Não especificado";
        this.idade = 1;
    }

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

Construtores e destrutores

Método construtor

- É possível (e comum) definir argumentos no método construtor, que são passados no momento em que o objeto é criado.
- Com isso, valores já podem ser armazenados nos atributos da instância.

```
public class Pessoa {

    private String nome;
    private int idade;

    public Pessoa() {
        this.nome = "Não especificado";
        this.idade = 1;
    }

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

Esta prática é chamada de **sobrecarga de métodos**.

Podem ser definidos diferentes construtores, desde que seus argumentos sejam diferentes.

Construtores e destrutores

Método construtor

- É possível (e comum) definir argumentos no método construtor, que são passados no momento em que o objeto é criado.
- Com isso, valores já podem ser armazenados nos atributos da instância.

```
public class Pessoa {

    private String nome;
    private int idade;

    public Pessoa() {
        this.nome = "Não especificado";
        this.idade = 1;
    }

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

O **this** é utilizado para diferenciar entre o atributo da classe e o parâmetro recebido.

Construtores e destrutores

Método destrutor

- No Java, os objetos são destruídos (ou seja, eliminados da memória) automaticamente, quando não existe mais nenhum ponteiro para ele.
  - Este mecanismo é chamado Garbage Collector.
- É possível definir um método a ser executado na destruição do objeto.

Construtores e destrutores

Método destrutor

- No Java, os objetos são destruídos (ou seja, eliminados da memória) automaticamente, quando não existe mais nenhum ponteiro para ele.
  - Este mecanismo é chamado Garbage Collector.
- É possível definir um método a ser executado na destruição do objeto.

```
public class Pessoa {

    private String nome;
    private int idade;

    public void finalize() {
        System.out.println("Objeto destruído: " + this.getNome());
    }
}
```

Sobrecarga de operações

- Assim como métodos construtores, métodos comuns também podem ser sobrecarregados.
  - Ou seja, diferentes métodos com o mesmo nome e diferentes assinaturas (quantidade e tipos dos argumentos).

```
public class Acumulador {
    private long somaInteiros;

    public Acumulador() {
        this.somaInteiros = 0;
    }

    public Acumulador(long valorInicial) {
        this.somaInteiros = valorInicial;
    }

    public void soma(int valor) {
        this.somaInteiros += valor;
    }

    public void soma(String valor) {
        this.somaInteiros += Integer.parseInt(valor);
    }
}
```

Sobrecarga de operações

- Assim como métodos construtores, métodos comuns também podem ser sobrecarregados.
  - Ou seja, diferentes métodos com o mesmo nome e diferentes assinaturas (quantidade e tipos dos argumentos).

```
public class Acumulador {
    private long somaInteiros;

    public Acumulador() {
        this.somaInteiros = 0;
    }

    public Acumulador(long valorInicial) {
        this.somaInteiros = valorInicial;
    }

    public void soma(int valor) {
        this.somaInteiros += valor;
    }

    public void soma(String valor) {
        this.somaInteiros += Integer.parseInt(valor);
    }
}
```

O construtor vazio inicializa a soma com zero, enquanto o segundo construtor permite que um valor inicial seja atribuído.

Sobrecarga de operações

- Assim como métodos construtores, métodos comuns também podem ser sobrecarregados.
  - Ou seja, diferentes métodos com o mesmo nome e diferentes assinaturas (quantidade e tipos dos argumentos).

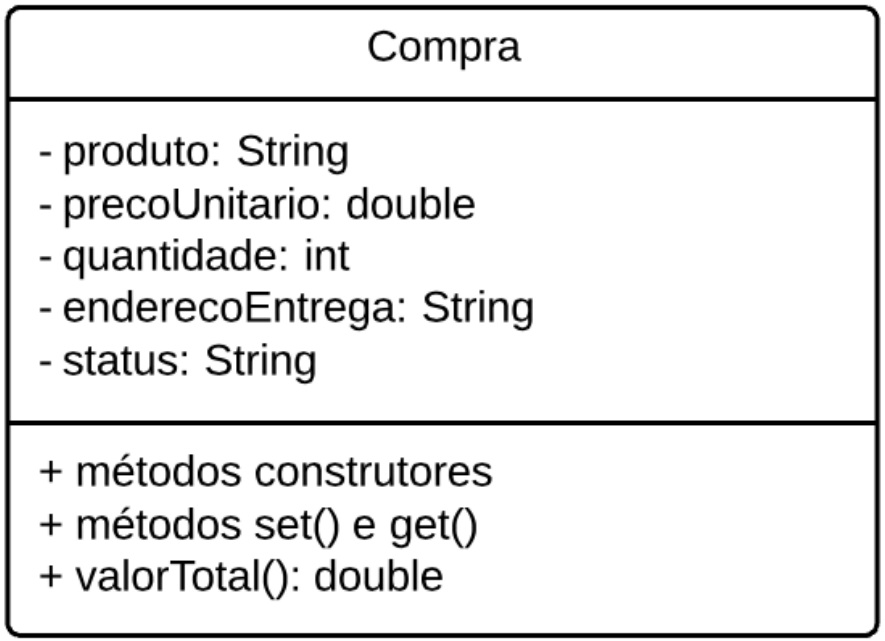
```
public class Acumulador {
    private long somaInteiros;

    public Acumulador() {
        this.somaInteiros = 0;
    }

    public Acumulador(long valorInicial) {
        this.somaInteiros = valorInicial;
    }

    public void soma(int valor) {
        this.somaInteiros += valor;
    }

    public void soma(String valor) {
        this.somaInteiros += Integer.parseInt(valor);
    }
}
```

No momento da chamada, o Java verifica qual método deve ser utilizado (conforme os parâmetros passados).

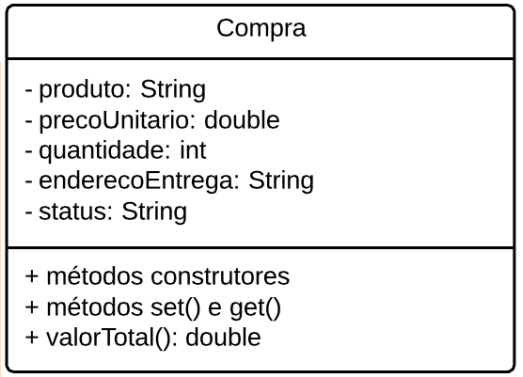
Exemplo

- **Contexto:** um sistema para gerenciar compras feitas pela Internet. O sistema deverá armazenar dados das compras em uma lista.

- **Abstração da entidade compra**

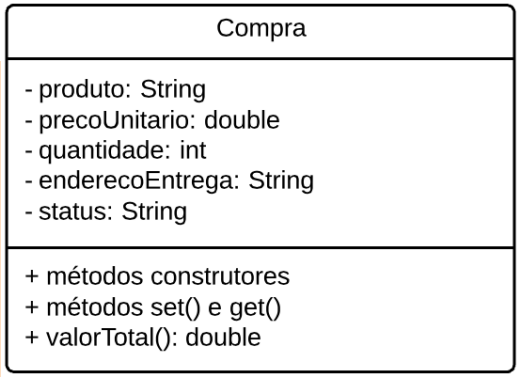


Exemplo




```java
public class Compra {

    private String produto;
    private double precoUnitario;
    private int quantidade;
    private String enderecoEntrega;
    private String status;

    public Compra() {}

    public Compra(String produto, double precoUnitario, int quantidade,
            String enderecoEntrega, String status) {

        this.produto = produto;
        this.precoUnitario = precoUnitario;
        this.quantidade = quantidade;
        this.enderecoEntrega = enderecoEntrega;
        this.status = status;
    }

    public double valorTotal() {
        return this.precoUnitario * this.quantidade;
    }

    //Métodos set() e get()
}
```

Exemplo




```java
public class ExemploCompra {

    private static List<Compra> compras = new ArrayList<Compra>();

    public static void main(String[] args) {
        Compra compra = new Compra();
        compra.setProduto("Camiseta XPTO - TAM M");
        compra.setPrecoUnitario(140.0);
        compra.setQuantidade(2);
        compra.setEnderecoEntrega("Rua Getúlio Vargas, 200 - Ibirama, SC - 89140-000");
        compra.setStatus("Pedido realizado");

        compras.add(compra);

        Compra compra2 = new Compra("Tênis ABC 42", 300.0, 1,
                                    "Caixa Postal 27 - 89140-970", "Pagamento confirmado");

        compras.add(compra2);

        for(int i = 0; i < compras.size(); i++) {
            Compra c = compras.get(i);
            System.out.println("Compra: " + c.getProduto() + " a " + c.getPrecoUnitario()
                               + "/unidade, totalizando " + c.valorTotal() + ".");
        }
    }
}
```

```
Compra: Camiseta XPTO - TAM M a 140.0/unidade, totalizando 280.0.
Compra: Tênis ABC 42 a 300.0/unidade, totalizando 300.0.
```

Referências

DEITEL, H. M. **Java: como programar**. H. M Deitel e P. J. Deitel - 8a ed. Porto Alegre: Prentice-Hall, 2010.

Leitura complementar

TutorialsPoint Java (http://www.tutorialspoint.com/java).