Aula VI - DWeb III

Introdução a Orientação a Objeto
Conceitos sobre a Orientação a Objeto
Classes
Encapsulamento
Público & Privado
Métodos Acessores
Representação gráfica da classe

Fontes:
https://distancia.qi.edu.br/course/view.php?id=359&section=1
https://www.devmedia.com.br/introducao-a-programacao-orientada-a-objetos-em-java/26452
Livro de Lógica de Programação - QI Faculdade & Escola Técnica - Técnico em Informática
http://ead.qi.edu.br/pluginfile.php/11857/mod_resource/content/0/Unidade%207%20-%20Encapsulamento%20e%20Visibilidade.pdf
https://www.devmedia.com.br/encapsulamento-em-java-primeiros-passos/31177
https://www.devmedia.com.br/encapsulamento-polimorfismo-heranca-em-java/12991
https://www.devmedia.com.br/metodos-atributos-e-classes-no-java/25404
http://distancia.qi.edu.br/mod/book/view.php?id=3375&chapterid=1868 - Disciplina de Algoritmos e Programação.
http://ead.qi.edu.br/pluginfile.php/11858/mod_resource/content/0/Unidade%208%20-%20M%C3%A9todo%20toString.pdf - Disciplina Lógica de Programação.
http://distancia.qi.edu.br/mod/book/view.php?id=3313&chapterid=1874 - Material professor Eduardo Reus - Disciplina de Algoritmos e Programação.
https://www.devmedia.com.br/como-criar-sobreposicoes-usando-o-metodo-tostring-em-java/29042

Programação Orientada a Objeto

Até o início da década de 70, o computador era utilizado somente por grandes empresas. Neste período, com a queda de preço dos computadores e a conseqüente proliferação de uso destes, cresceu a demanda por software. As técnicas de desenvolvimento de software utilizadas até então não eram suficientes para contornar problemas existentes no desenvolvimento de sistemas, principalmente quando desenvolvidos em grande escala, como então se exigia. Na verdade, pouco se possuía de técnicas que estivessem realmente sendo aplicadas.

1950 – 1960 Era do Caos	1970 – 1980 Era da Estruturação	1990 até agora Era dos Objetos
Saltos, gotos, variáveis não estruturadas, variáveis espalhadas ao longo do programa	If-then-else Blocos Registros Laços-While	Objetos Mensagens Métodos Herança

Paradigma Imperativo - Estruturado

Este paradigma é o primeiro paradigma a surgir e até hoje dominante. É um paradigma baseado na arquitetura Von Neumann. Segue o conceito de estruturação, onde tudo possui uma sequência passo a passo a ser seguida, modificando os dados a fim de chegar ao resultado esperado.

Exemplos: C, Pascal, Basic.

Foi neste contexto que surgiu a programação estruturada, seguida pelo conceito de desenvolvimento estruturado de sistemas. Esta metodologia tentava oferecer soluções para os problemas ligados ao desenvolvimento de sistemas, ao pregar a aplicação dos seguintes princípios:

- **Princípio da abstração:** para resolver um problema, o analista deveria analisá-lo separadamente dos demais aspectos, ou seja, abstrair os detalhes;
- **Princípio da formalidade:** o analista deveria seguir um caminho rigoroso e metódico para solucionar um problema;
- **Princípio de "dividir para conquistar"**: o analista deveria dividir o problema em partes menores, independentes e com possibilidade de serem mais simples de entender e solucionar;
- **Princípio da disposição hierárquica:** o analista deveria organizar os componentes da solução do problema na forma de uma árvore com estrutura hierárquica. O sistema seria entendido e construído nível a nível, onde cada novo nível acrescentaria mais detalhes.

Na programação orientada a Objetos

Os objetos são a chave para a compreensão da tecnologia orientada a objeto. Olhe em volta agora e você vai encontrar muitos exemplos de objetos do mundo real: seu cachorro, sua mesa, sua televisão. Os objetos do mundo real partilham duas características: eles possuem estado e comportamento. Os cães têm estado (nome, cor, raça) e comportamento (latidos, cheirando, abanando o rabo). Bicicletas também

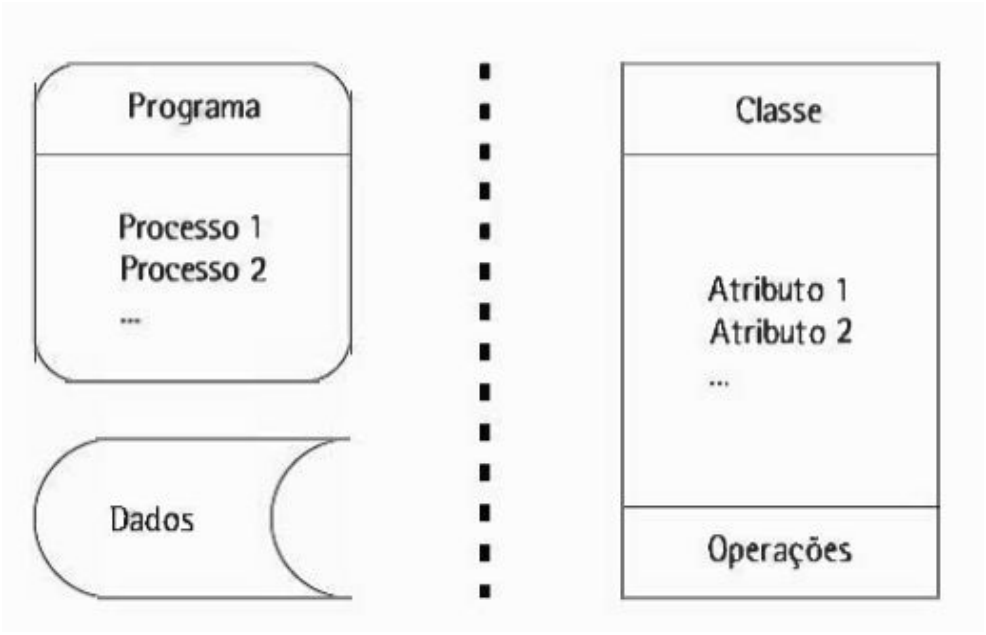
A programação orientada a objetos **representa uma mudança no enfoque da programação,** na forma como os sistemas eram vistos até então. Representa uma quebra de paradigma, revolucionando todos os conceitos de projeto e desenvolvimento de sistemas existentes anteriormente.

O enfoque tradicional para o desenvolvimento de sistemas e, por conseqüência, para a programação, **baseia-se no conceito de que um sistema é um conjunto de programas inter-relacionados que atuam sobre um determinado conjunto de dados que se deseja manipular de alguma forma para obter os resultados desejados.**

O enfoque da modelagem de sistemas por objetos procura enxergar o mundo como um conjunto de objetos que interagem entre si e apresentam características e comportamento próprios representados por seus atributos e suas operações. Os atributos estão relacionados aos dados, e as operações, aos processos que um objeto executa. Assim, supondo que se deseje desenvolver um sistema de controle de estoque para uma empresa, procura-se identificar os objetos relacionados

ao sistema, como os produtos, os pedidos de compra, os recibos, as pessoas etc., conforme está detalhado a seguir.

Pode-se dizer que é possível modelar, por meio da orientação a objetos, um setor, um departamento e até uma empresa inteira. Esse enfoque justifica-se, de forma resumida, pelo fato de que os objetos existem na natureza muito antes de haver qualquer tipo de negócio envolvido ou qualquer tipo de sistema para controlá-los. Equipamentos, pessoas, materiais, produtos, peças, ferramentas, combustíveis etc. existem por si sós e possuem características próprias determinadas pelos seus atributos (nome, tamanho, cor, peso) e um determinado comportamento no mundo real relacionado aos processos que eles sofrem.

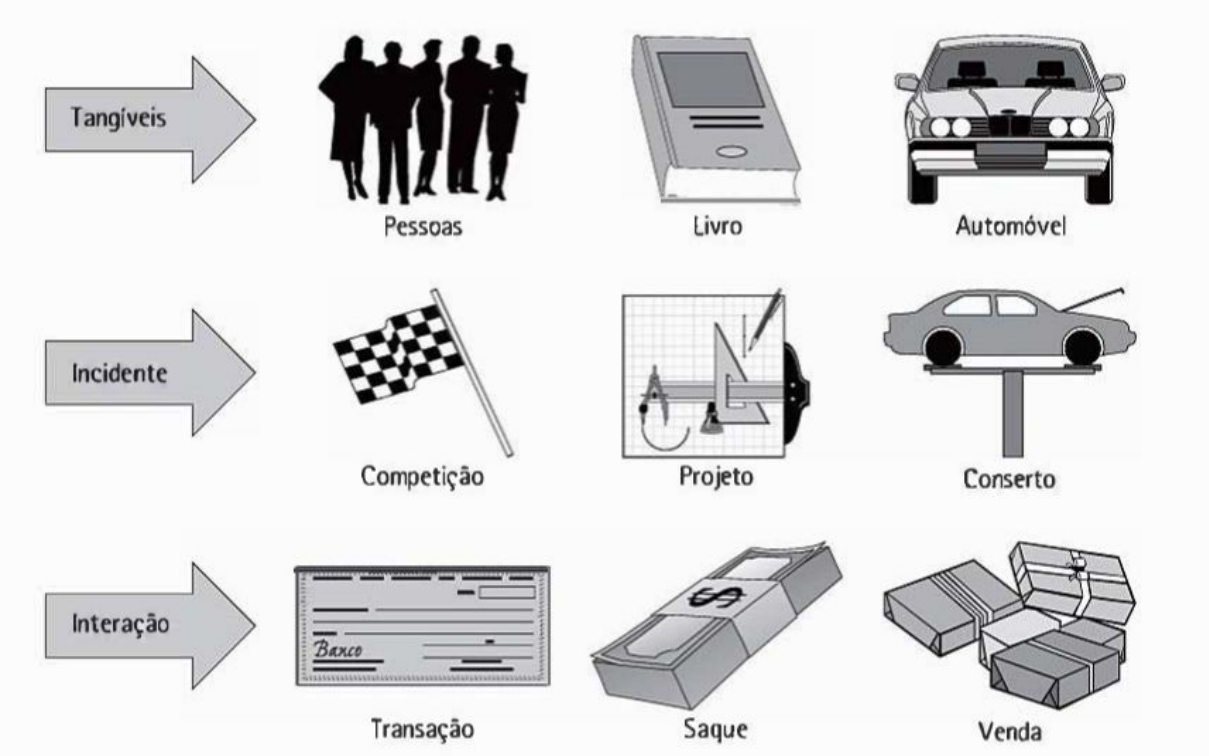


Programação Tradicional/Estruturada Programação Orientada a Objeto

O 4 pilares da Programação Orientada a Objeto (POO)

0 que é um Objeto

Um dos primeiros conceitos básicos da orientação a objetos é o do próprio objeto. Um objeto é uma extensão do conceito de objeto do mundo real, em que se podem ter coisas tangíveis, um incidente (evento ou ocorrência) ou uma interação (transação ou contrato).



Fonte da Imagem do Livro Lógica de Programação - Sandra Puga

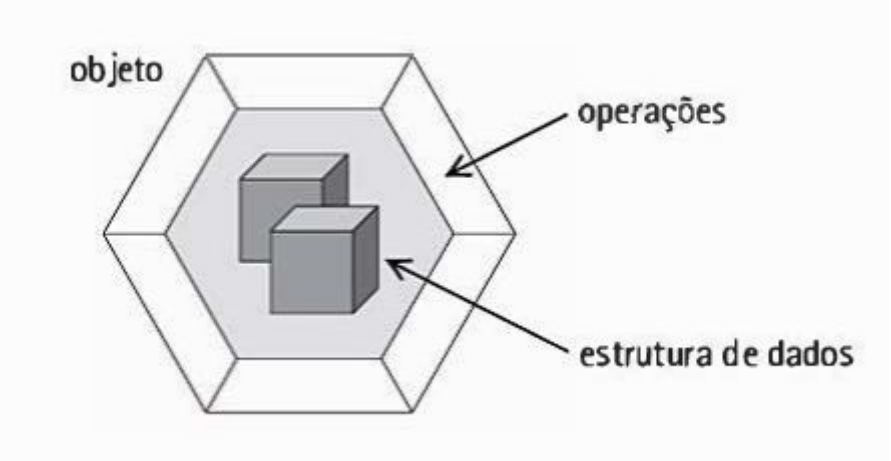
Por exemplo, em um sistema acadêmico em que **João é um aluno objeto e Carlos é um professor objeto que ministra aulas objeto da disciplina objeto algoritmos,** para que João possa assistir às aulas da disciplina do prof. Carlos, ele precisa fazer uma matricula objeto no curso objeto de computação.

Têm-se as ocorrências de objetos mencionados.
• **tangíveis: aluno e professor;**
• **incidente: curso, disciplina, aula;**
• **interação: matrícula.**

A identificação dos objetos em um sistema depende do nível de abstração de quem faz a modelagem, podendo ocorrer a identificação de diferentes tipos de objetos e diferentes tipos de classificação desses objetos. Não existe um modelo definitivamente correto; isso vai depender de quem faz a modelagem e de processos sucessivos de refinamento, até que se possa encontrar um modelo adequado a sua aplicação.

Como visualizar um Objeto?

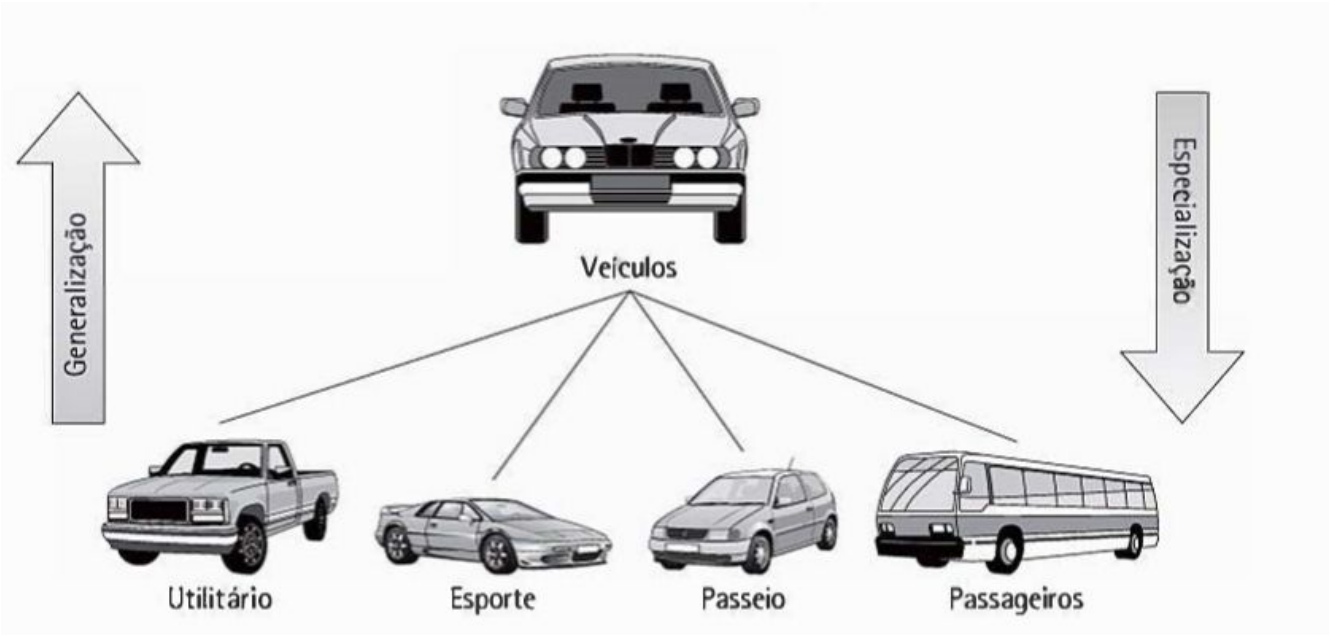
Pode-se imaginar um objeto como algo que guarda dentro de si os dados ou informações sobre sua estrutura (seus atributos) e possui um comportamento definido pelas suas operações.
objeto
operações
estrutura de dados



Visualização do objeto
Os dados ficam protegidos pela interface, que se comunica com os demais objetos do sistema. Nessa interface, representada pela camada mais externa de nosso modelo, estão as operações. Todo tipo de alteração nos dados do objeto (atributos) somente poderá ser feito por meio das operações, que recebem as solicitações externas, fazem as alterações nos dados (se permitidas) e retornam outras informações para o meio externo.

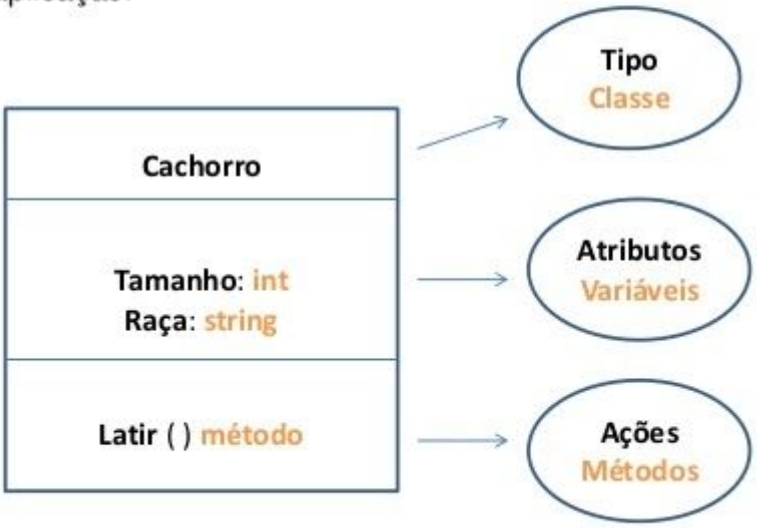
O Conceito de Classe

O conceito de classes é muito importante para o entendimento da orientação a objetos. **Uma classe é uma coleção de objetos que podem ser descritos por um conjunto básico de atributos e possuem operações semelhantes.** Falamos em um conjunto básico de atributos e operações semelhantes, pois veremos adiante que nem todos os objetos da mesma classe precisam ter exatamente o mesmo conjunto de atributos e operações. **Quando um objeto é identificado com atributos e operações semelhantes em nosso sistema, diz-se que pode ser agrupado em uma classe. Esse processo é chamado de generalização.**
Por outro lado, pode ocorrer que um objeto, ao ser identificado, constitua-se, na verdade, de uma classe de objetos, visto que dele podem se derivar outros objetos. Já esse processo é chamado de especialização.



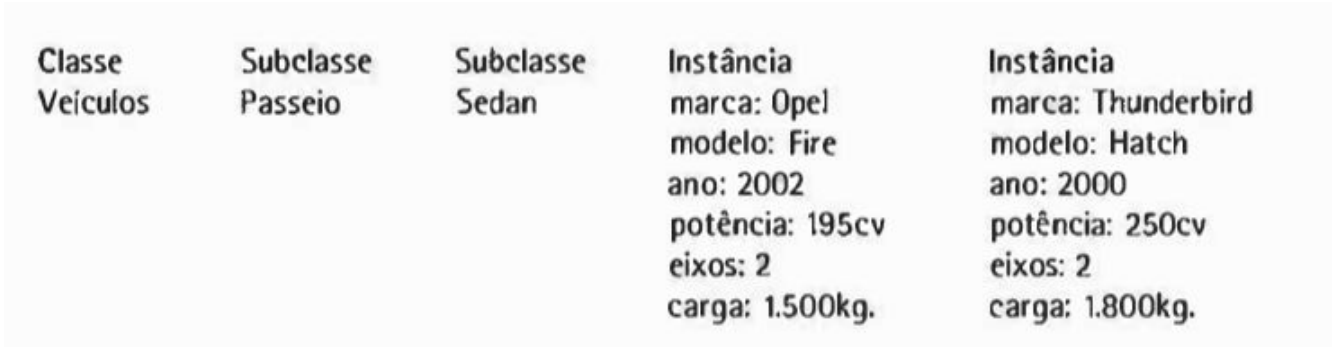
O que é uma Classe?

Representa-se a classe como um projeto do objeto, ou seja, objeto é a instancia de uma classe, antes de ser criado um objeto deve-se definir a classe na qual ele pertence. A partir da classe podemos construir objetos na memória do computador que executa a aplicação.



Instâncias de Objetos

Quando se fala em classes de objetos, está sendo considerado que se podem incluir objetos em cada uma delas. Como exemplo, considere que será modelado um sistema para uma revendedora de veículos que comercializa os veículos conforme a imagem do esquema citado.
Cada novo veículo adquirido pela revendedora seria cadastrado no sistema obedecendo a sua classificação. Supondo que o veículo seja um automóvel de passeio do tipo sedan, cria-se um novo objeto dessa classe, que será chamada de uma instância de objeto, conforme o seguinte esquema:



Encapsulamento

De acordo com o site Devmedia, o encapsulamento é a coisa mais importante que devemos saber sobre a diferença que existe entre ele e a ocultação de informações, devido à grande confusão que sempre existiu entre estas duas definições. A ocultação de informações é considerada parte do encapsulamento, mas se fizermos uma pesquisa na internet, podemos encontrar a seguinte definição para encapsulamento: **Um mecanismo da linguagem de programação para restringir o acesso a alguns componentes dos objetos, escondendo os dados de uma classe e tornando-os disponíveis somente através de métodos.**

No desenvolvimento de software **orientado a objeto**, temos este recurso que auxilia a padronização e controle da criação dos códigos das classes; esse recurso tem o nome de encapsulamento. Agora vamos aprender sobre este conceito e seus derivados, **como a visibilidade de atributos e métodos e os métodos modificadores e assessores.**

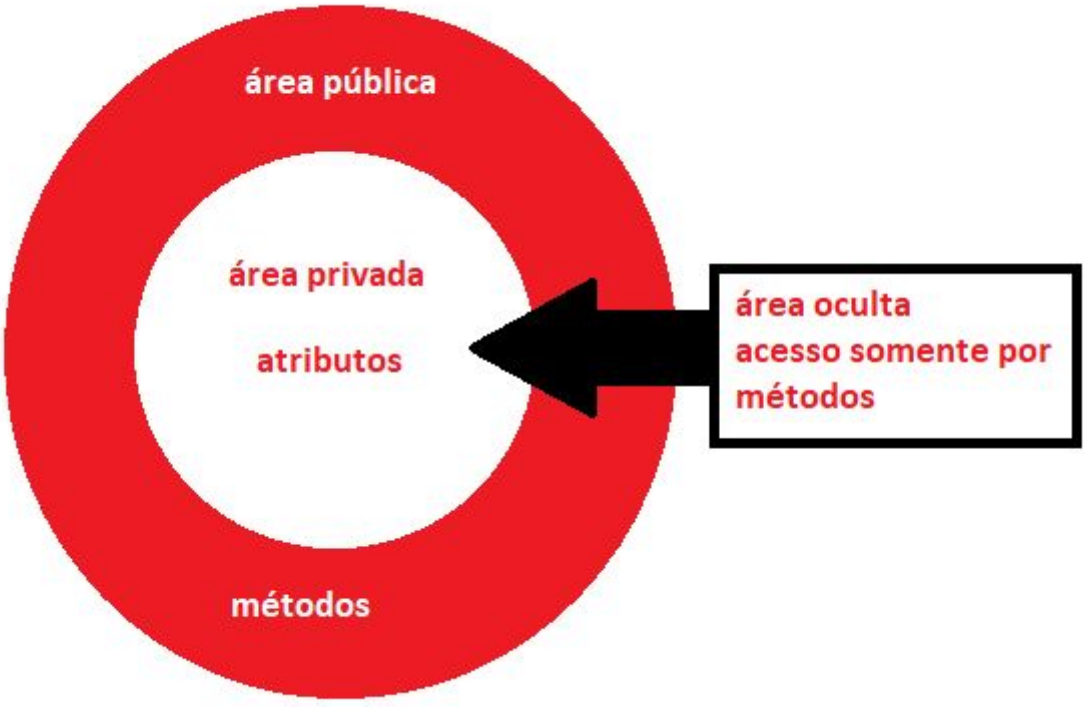
Imagine que temos uma **classe ContaBancaria**. Podemos, portanto, **possuir no sistema vários objetos desta classe, ou seja, várias contas**.

Se deixarmos os dados (atributos) públicos, simplesmente todas as classes do programa poderão visualizar e modificar qualquer dado de qualquer conta, sem nenhum tipo de controle.

Ao ocultarmos os dados, estamos protegendo contra alteração indevida. É uma prática comum deixar todos os atributos da classe ocultos, para protegê-los.

Para entendermos melhor o encapsulamento, vamos analisar o seguinte: a sua carteira é pública? Ou seja, qualquer pessoa tem acesso a ela, qualquer um abre e tem acesso ao seu conteúdo? Provavelmente sua resposta será não, afinal nossa carteira é privada, é algo só seu e somente você deve ter acesso a ela, assim você tem o controle do que tem dentro dela. Se alguém quiser algo da sua carteira, terá de pedir a você, certo?

Através do encapsulamento e recursos de visibilidade, o objeto esconde seus dados de outros objetos e permite que os dados sejam acessados por intermédio de seus próprios métodos. Isso é chamado de ocultação de informações (information hiding).



“Cada objeto encapsula uma estrutura de dados e métodos. Uma estrutura de dados encontra-se no centro de um objeto. Os dados do objeto não podem ser acessados, exceto através destes métodos.” (MACORATTI)

Visibilidade

Para **proteger os dados** de uma classe encapsulada, **precisamos alterar a sua visibilidade.** A visibilidade nada mais é do que a maneira que acessamos e enxergamos os dados da nossa classe.

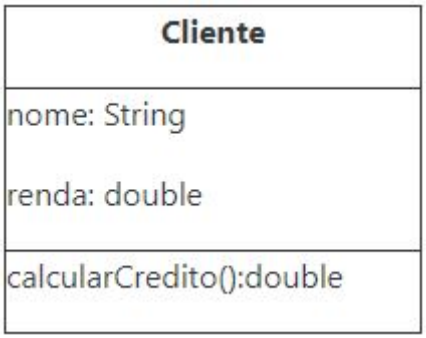
Quando a visibilidade de um atributo/método é pública (“public”), com a qual até o momento estávamos trabalhando, este pode ser visto e acessado de qualquer parte do programa.

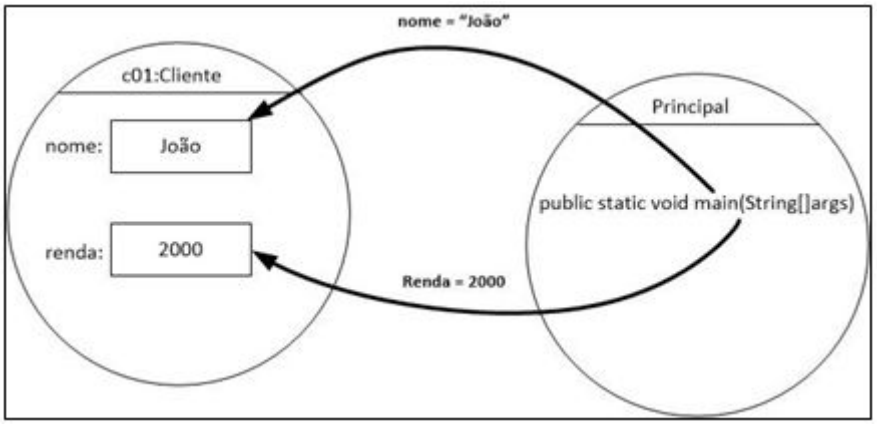
Reforçando:

Public (público): Atributos e métodos definidos como públicos podem ser visualizados, acessados e quando possível, modificados, por qualquer outra classe ou objeto do sistema. É o nível de acesso mais permissivo existente na linguagem Java. Por definição, todas as classes criadas também devem ser definidas como públicas. No desenvolvimento do diagrama de classes, quando uma informação precisa ser identificada como pública, a mesma é precedida por um sinal de adição (**+**).

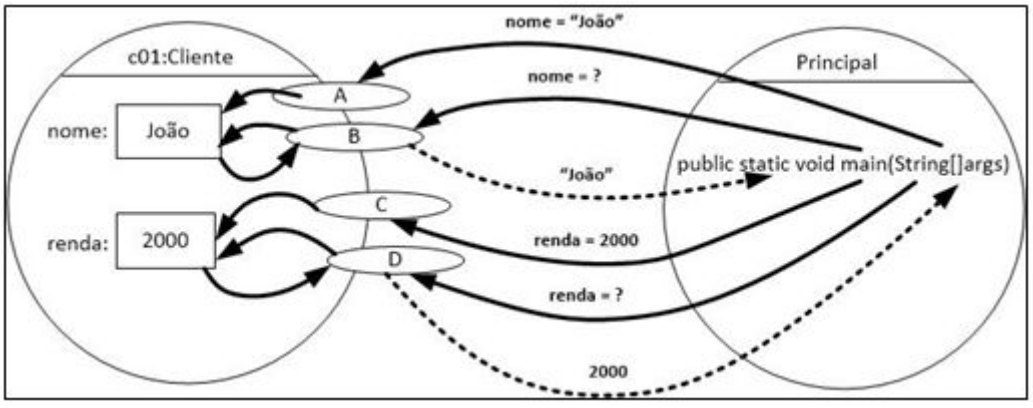
Private (privado): Ao contrário do nível público, atributos e métodos definidos como privados somente podem ser acessados pela própria classe em que estes são definidos. Ou seja, se uma outra classe ou objeto do sistema tentar acessar ou modificar uma informação que está definida como privada, haverá um erro de compilação e a operação não poderá ser executada. No desenvolvimento do diagrama de classes, quando uma informação precisa ser identificada como privada, a mesma é precedida por um sinal de subtração (**-**).

Para a exemplificação do encapsulamento e visibilidade, vamos observar a **imagem do UML** abaixo serão apresentados cenários onde a aplicação de modificadores de acesso interferirá na forma com que um algoritmo é desenvolvido com a seguinte classe:





Já se definirmos a visibilidade como privada (“private”), esta só pode ser acessada dentro da própria classe. * Para representarmos a visibilidade de atributos/métodos no diagrama UML, usamos os símbolos + para public e – para private.



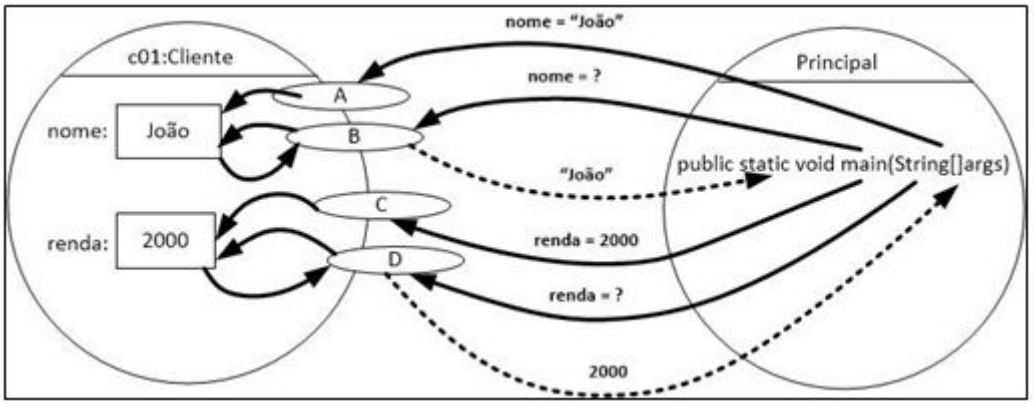
Quando nosso atributo possui a visibilidade pública o acesso a ele é feito de forma direta, ou seja, no Main digitamos o nome do objeto seguido de um ponto, seguido do nome do atributo. Já quando alterarmos a visibilidade para privada, o acesso se torna diferenciado, pois o mesmo não fica visível diretamente pelas outras classes.

Métodos acessores

Todo o atributo que conter **visibilidade private**, terá que possuir dois métodos especiais, **um método de acesso set para o caso de poder ser alterado, e um método de consulta get para o caso de poder ser consultado. Método set**

O set: é utilizado para que se consiga enviar uma informação para um atributo.
Exemplo: informar um nome que será guardado na variável-atributo nome.
O set se caracteriza por ser um método sem retorno, já que seu objetivo é simplesmente armazenar um dado num atributo, e obrigatoriamente deve conter argumento, pois precisa receber um valor externo para poder armazená-lo no atributo.

Método get: O get é utilizado para **consultar/obter o valor de um atributo**. **Sua função é retornar o valor de um atributo específico.** Portanto, sempre tem retorno, e não precisa ter argumentos.



Por convenção, os métodos responsáveis por modificar o estado interno de um objeto como fazem os métodos "A" e "C" no exemplo são denominados como métodos *set* e

métodos responsáveis por retornar o valor do estado interno de um objeto são denominados como métodos *get*.

Implementando os métodos gets e sets na classe

Aplicando os conceitos de modificadores de acesso e encapsulamento a classe Cliente é implementada conforme imagem abaixo.

Entendendo o código fonte

Logo na **linha 2** durante a declaração da **classe é informado que a mesma deve ser pública através do modificador de acesso public**, podendo ser acessada por qualquer outra classe do sistema.

```
2   public class Cliente {
3       private String nome;
4       private double renda;
5
6   public Cliente(String valorNome, double valorRenda){
7       nome = valorNome;
8       renda = valorRenda;
9   }
10
11  public void setNome(String valorNome) {
12      nome = valorNome;
13  }
14
15  public String getNome() {
16      return nome;
17  }
18
19  public void setRenda(double valorRenda){
20      renda = valorRenda;
21  }
22
23  public double getRenda() {
24      return renda;
25  }
26
27  public double calcularCredito() {
28      //implementação do método para calcular crédito
29      return 0;
30  }
31  }
```

As **linhas 3 e 4** apresentam na declaração dos atributos nome e renda o **modificador de acesso private,** ou seja, o acesso a estes atributos será de **exclusividade de classe Cliente e seus objetos.**

As **linhas 6, 11, 15, 19, 23 e 27** que declaram os métodos da classe **estão definidos com o modificador de acesso public, também podendo serem acessados por outras classes e objetos do sistema.**

Na **linha 11** está sendo implementado o **método *set* para o atributo "nome" da classe**. **Este método deve receber por parâmetro um valor String que será adicionado ao atributo "nome" das instâncias classe.** O método é definido como void por não haver a necessidade de se retornar informações após a inserção do nome do cliente no objeto.

Na **linha 15** está sendo implementado o **método *get* para o atributo "nome" da classe**. Este método **não recebe parâmetro nenhum, porém por necessitar retornar o valor do atributo este método deve possuir como tipo de dado de retorno o mesmo tipo de dado do atributo, sendo assim, o método neste caso é definido como de retorno String.**

As **linhas 18 e 23** representam os **métodos de *get* e *set* para o atributo "renda"**. De modo que as únicas modificações em relação aos métodos de acesso do atributo "nome" são os nomes e os tipos de dados dos parâmetros e retornos, de modo que neste caso utiliza-se o tipo double no lugar do tipo String.

O **método calcularCredito** retorna diretamente o valor zero por ser apenas um exemplo dos métodos acessores, não havendo ainda uma regra de negócio definida para o mesmo.

Utilizando os objetos de uma classe encapsulada

Com a **classe Cliente** modificada em seus níveis de acesso e com a presença dos **métodos acessores *get* e *set***, caso seja solicitado a leitura dos dados de um cliente e uma posterior modificação de algum dado, a classe principal será desenvolvida de acordo com a imagem abaixo:

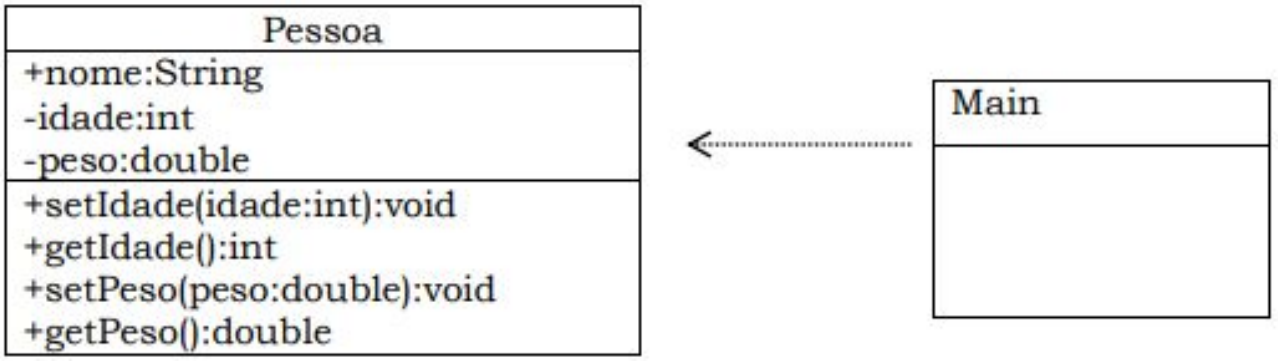
```
1  import java.util.Scanner;
2  public class Principal {
3      public static void main(String[] args) {
4          String nome;
5          double renda;
6
7          Scanner teclado = new Scanner(System.in);
8
9          System.out.print("\fNome: ");
10         nome = teclado.nextLine();
11
12         System.out.print("Renda: ");
13         renda = teclado.nextDouble();
14
15         Cliente c01 = new Cliente(nome, renda);
16         System.out.println("O cliente " + c01.getNome() + "possui uma renda de " + c01.getRenda());
17         c01.setRenda(4500);
18
19         System.out.println("O cliente " + c01.getNome() + "possui uma renda de " + c01.getRenda());
20     }
21 }
```

Entendendo o código fonte

A linha 16 do código demonstra a utilização dos **métodos *get*** para se apresentar o estado interno do objeto **c01 referente às informações de nome e renda.**
Na sequência foi invocado o método ***set*Renda na linha 17 solicitando a modificação da renda para o valor de 4500.**
Na **linha 19 o estado interno do objeto é reimpresso na tela para demonstrar que houve modificação na renda do cliente.**
O resultado da execução do algoritmo pode ser visto na abaixo onde inicialmente se informa uma renda de 2000 para a cliente "Ana" e na sequência sua renda é modificada e apresentada ao usuário.

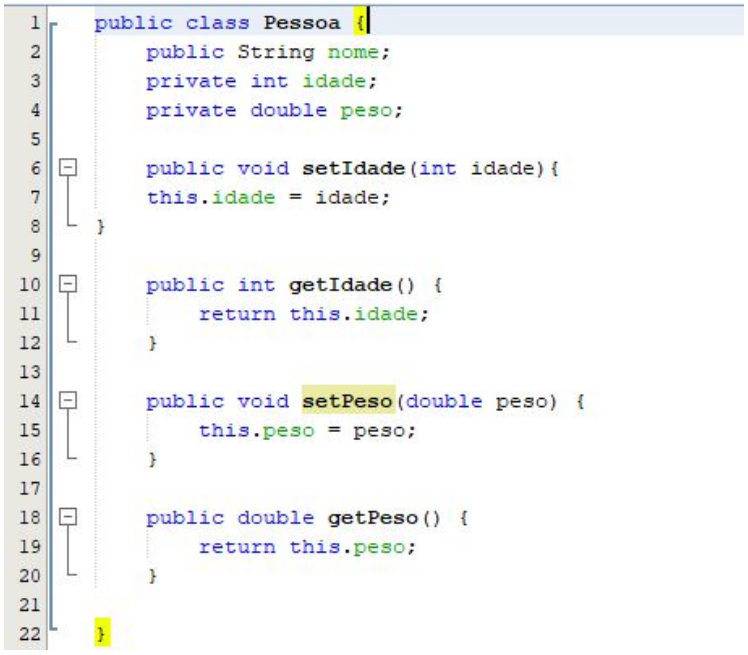
```
run:
Nome: Ana
Renda: 2000
O cliente Anapossui uma renda de 2000.0
O cliente Anapossui uma renda de 4500.0
CONSTRUÍDO COM SUCESSO (tempo total: 9 segundos)
```

Representação de Visibilidade no Diagrama UML - Outro exemplo



No exemplo acima, percebemos que o atributo **nome é de acesso público e os atributos idade e peso são de acessos privados**.
Para estes últimos, teremos dois métodos especiais, um setIdade e **getIdade e o setPeso e getPeso**. **O acesso a estes dados será por intermédio destes métodos, enquanto que o atributo nome, por ter sido definido como público, dispensa estes métodos já que o acesso a ele é direto.**

Observando a sintaxe acima, notamos que no **método setIdade** possui um argumento **int idade**, e no **método setPeso**, temos o argumento **double peso**. **Através destes argumentos que os dados serão passados aos atributos privados.**



```java
public class Pessoa {
    public String nome;
    private int idade;
    private double peso;

    public void setIdade(int idade){
    this.idade = idade;
}

    public int getIdade() {
        return this.idade;
    }

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

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

}
```

Agora vamos ver a classe principal ou main e ver como os acessos são feitos!!

```java
import java.util.Scanner;
public class Main {
    public static void main(String[] args) {
        Scanner ler = new Scanner(System.in);
        Pessoa p1 = new Pessoa();

        System.out.println("Digite seu nome: ");
        p1.nome = ler.nextLine();

        System.out.println("Digite sua idade: ");
        p1.setIdade(ler.nextInt());

        System.out.println("Digite seu peso: ");
        p1.setPeso(ler.nextDouble());

        System.out.println("Visualizando os dados: ");
        System.out.println("Nome: " + p1.nome);
        System.out.println("Idade: " + p1.getIdade());
        System.out.println("Peso: " + p1.getPeso());
    }
}
```

Observando a classe, vemos que ao ler o atributo nome, digitamos:
p1.nome = ler.nextLine();

```java
System.out.println("Digite seu nome: ");
p1.nome = ler.nextLine();
```

Ou seja, inserimos diretamente o nome no atributo da classe, já que foi definido como public na classe.

Já em idade e peso, o acesso foi através do **método set**:
p1.setIdade(ler.nextInt());

```java
System.out.println("Digite sua idade: ");
p1.setIdade(ler.nextInt());
```

Inserimos a idade como argumento para o método setIdade, este método por sua vez, insere o valor no atributo. O mesmo ocorre com o atributo peso.

```
System.out.println("Digite seu peso: ");
pl.setPeso(ler.nextDouble());
```

Ao visualizar os dados, também notamos uma diferença: O atributo nome foi visualizado de forma direta:

```
System.out.println("Visualizando os dados: ");
System.out.println("Nome: " + pl.nome);
```

Os atributos idade e peso foram acessados **através do get**: Ou seja, **o método get foi quem retornou a idade e peso, já que ambos os atributos, por serem privados, não possuem acesso a não ser através dos métodos.**

```
System.out.println("Idade: " + pl.getIdade());
System.out.println("Peso: " + pl.getPeso());
```

O método toString

É muito comum que existam situações em que se deseja exibir os dados presentes no estado interno de um objeto. Nestes casos também é comum que existam informações que não seja de interesse que sejam expostas, ou então é importante que estas informações sofram algum tipo de tratamento antes de serem disponibilizadas.

Um exemplo comum são os valores de tipo booleanos onde normalmente um texto é apresentado ao invés de "verdadeiro" ou "falso".

É comum na programação orientada a objetos que haja um método nas classes responsável por retornar os valores presentes no estado interno do objeto e que são de interesse de serem expostos e em um formato adequado. **Este método é comumente denominado de toString.**

Implementamos o **método toString** para retornar o objeto em formato de texto. **Ele simplifica a exibição dos atributos do objeto, convertendo o objeto para texto. Neste método, determinamos como os atributos devem ser exibidos.**

Sintaxe do método toString

Este método não pode ser criado de qualquer maneira. Ele possui uma sintaxe padrão, onde alteramos apenas o que vai no "return". O nome deve ser **toString**, **sempre deve retornar uma String e não possui argumentos.**

```
public String toString() {
    return this.dia + "/" + this.mes + "/" + this.ano;
}
```

Exemplo de classe com toString()

Data
-dia: int -mes: int -ano: int
+getDia():int +getMes():int +getAno():int +setDia(dia:int):void +setMes(mes:int):void +setAno(ano:int):void


+toString():String