

transforme ■ se



JAVA

AULA 6 - Orientação a Objeto - Parte 2



Polimorfismo

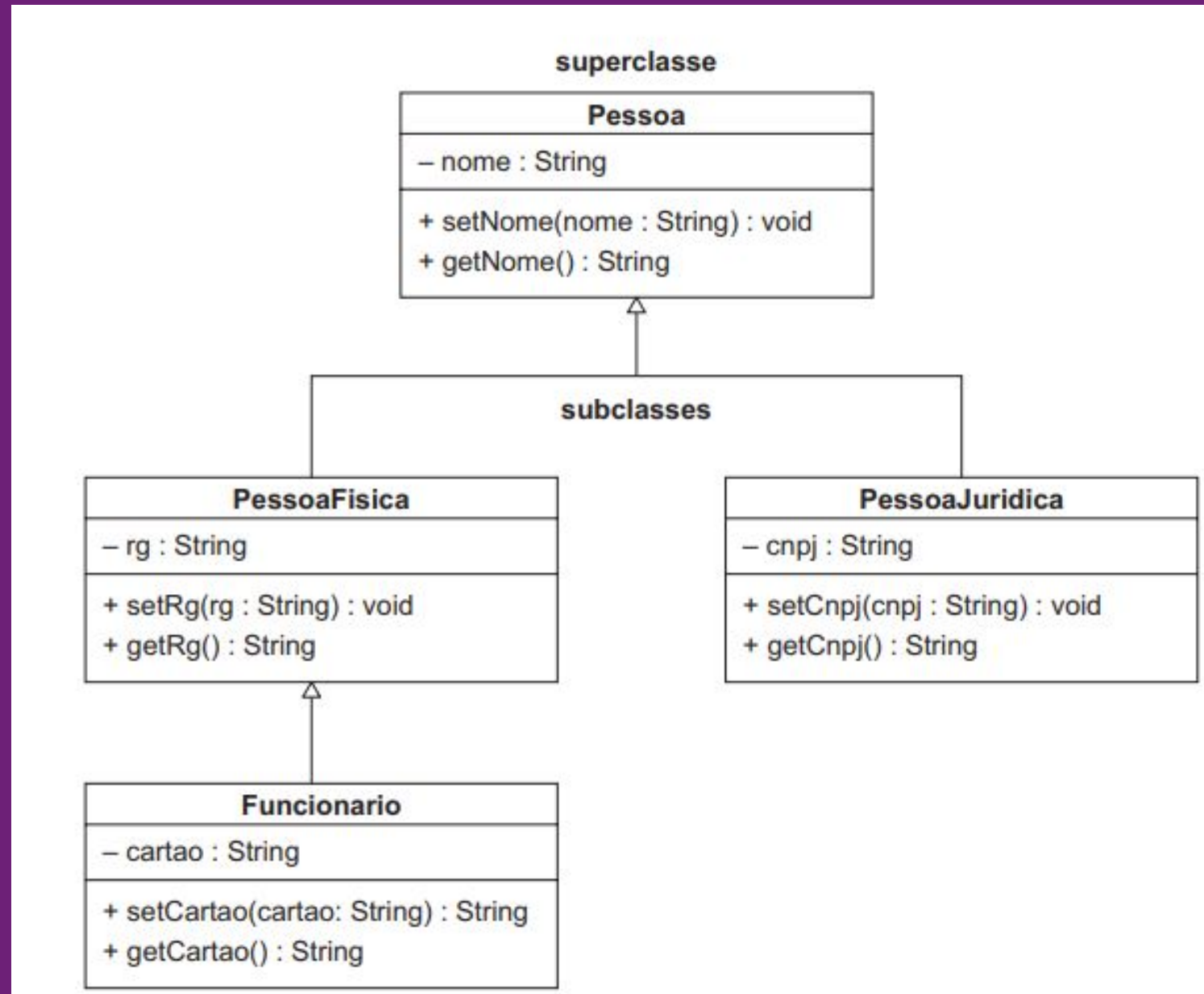


Polimorfismo

Por meio das técnicas de encapsulamento e herança, novas funcionalidades podem ser adicionadas às classes. Enquanto a herança é um poderoso mecanismo de especialização, o polimorfismo oferece um mecanismo para a generalização.

Ao analisar a hierarquia de classes na próxima figura de forma inversa, isto é, da classe de nível inferior para a classe de nível superior, percebe-se a generalização das classes relacionadas, ou seja, cada classe da hierarquia pode assumir o mesmo comportamento da superclasse, sendo tratada como tal. Caso a superclasse modifique sua estrutura, igualmente todas as classes da hierarquia são afetadas. Com isso nota-se que uma classe da hierarquia pode assumir diferentes formas (funcionalidades) de acordo com as classes de nível superior

Polimorfismo



Polimorfismo

O polimorfismo representa uma técnica avançada de programação, e seu uso pode gerar economia de recursos computacionais. A ideia geral do polimorfismo é que um objeto de uma determinada classe mais genérica (a superclasse) possa assumir diferentes comportamentos, gerando objetos distintos, dependendo de certas condições.

Na prática quer dizer que um mesmo objeto pode executar métodos diferentes, dependendo do momento de sua criação. Como um mesmo objeto pode ser gerado a partir de classes diferentes e classes diferentes possuem métodos distintos, o objeto criado pode ter comportamentos variados, dependendo da classe a partir da qual ele foi criado.


Considere o diagrama de classes da figura anterior um objeto pode ser declarado como do tipo Pessoa e ser criado, em tempo de execução, como do tipo PessoaFisica, PessoaJuridica ou Funcionário.

Polimorfismo

O uso do polimorfismo pressupõe duas condições: a existência de herança entre as classes e a redefinição de métodos em todas as classes. Todas as classes devem possuir métodos com a mesma assinatura (nome e parâmetros), porém com funcionalidades diferentes. Esse mecanismo de redefinição de métodos entre superclasses e subclasses é conhecido como *overriding*, diferente do mecanismo de sobrecarga (*overloading*) de métodos que ocorre em uma mesma classe.

Vamos criar um exemplo de polimorfismo aproveitando o exemplo de herança apontado no diagrama de classes anterior. Siga os procedimentos:

1. Ao final da classe Pessoa, adicione um método chamado `mostraClasse` da forma a seguir:



```
1 public void mostraClasse(){
2     System.out.println("Classe: " + this.getClass().getSimpleName());
3 }
```

Polimorfismo

2. Agora, crie uma classe PessoaPolimorfa que vai ter um método main e um switch case que a depender da opção selecionada, vai instanciar uma classe ou outra como veremos a seguir:

```
1 public class PessoaPolimorfa {
2
3     public static void main(String[] args) {
4         Pessoa pessoa = null;
5
6         int opcao = Integer.parseInt(JOptionPane.showInputDialog("Escolha uma opção de 1 a 4"));
7
8         switch(opcao) {
9             case 1: pessoa = new Pessoa();
10                break;
11
12             case 2: pessoa = new PessoaFisica();
13                break;
14
15             case 3: pessoa = new PessoaJuridica();
16                break;
17
18             case 4: pessoa = new Funcionario();
19                break;
20
21             default: JOptionPane.showMessageDialog(null, "Tipo desconhecido");
22                System.exit(0);
23         }
24
25         pessoa.mostraClasse();
26     }
27
28 }
```


Polimorfismo

O resultado disso, no console, será o nome da classe instanciada de acordo com a opção que o usuário passou:

```
Classe: PessoaJuridica
```

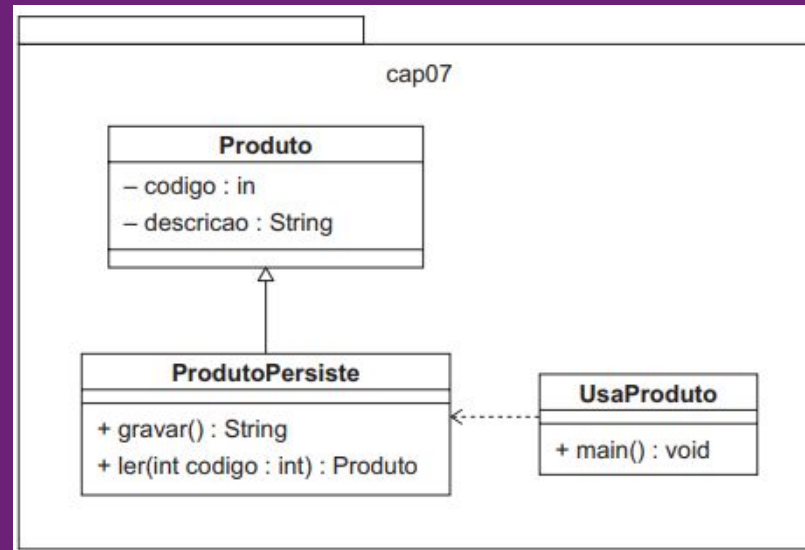
Gravação e leitura de objetos



Gravação e leitura de objetos

O conteúdo das variáveis de instância dos objetos de uma classe pode ser armazenado e recuperado mediante sua gravação em arquivo. Apesar de o método mais usual de armazenamento de dados do mercado ser por meio de tabelas, usadas no modelo relacional de banco de dados, neste momento vamos deixar essa questão de lado e aprender a salvar o conteúdo de nossos objetos em um arquivo.

Para entender a gravação e leitura de objetos, vamos criar três classes.



Gravação e leitura de objetos



```
1  public class Produto implements Serializable{
2
3      private int codigo;
4      private String descricao;
5
6      public int getCodigo() {
7          return codigo;
8      }
9      public void setCodigo(int codigo) {
10         this.codigo = codigo;
11     }
12     public String getDescricao() {
13         return descricao;
14     }
15     public void setDescricao(String descricao) {
16         this.descricao = descricao;
17     }
18
19 }
```

Gravação e leitura de objetos

```
1 public class ProdutoPersiste extends Produto {
2
3     private static final String MSG_SUCESSO = "Produto armazenado com sucesso!";
4     private static final String CAMINHO_ARQUIVO = "C:/arquivos/Produto";
5
6     public String gravar() {
7         try {
8             FileOutputStream arquivo = new FileOutputStream(CAMINHO_ARQUIVO + this.getCodigo());
9             ObjectOutputStream stream = new ObjectOutputStream(arquivo);
10            stream.writeObject(this);
11            stream.flush();
12        }
13        catch(Exception erro) {
14            return "Falha na gravação\n" + erro.toString();
15        }
16
17        return MSG_SUCESSO;
18    }
19
20    public static Produto ler(int codigo) {
21        try {
22            FileInputStream arquivo = new FileInputStream(CAMINHO_ARQUIVO + codigo);
23            ObjectInputStream stream = new ObjectInputStream(arquivo);
24            return (Produto) stream.readObject();
25        }
26        catch (Exception erro) {
27            System.out.println("Falha na leitura\n" + erro.toString());
28            return null;
29        }
30    }
31
32 }
```

Gravação e leitura de objetos



```
1  public class App {  
2      public static void main(String[] args) throws Exception {  
3  
4          ProdutoPersiste produto = new ProdutoPersiste();  
5          produto.setCodigo(2);  
6          produto.setDescricao("Sabonete El Senador");  
7          System.out.println(produto.gravar());  
8  
9          Produto p = ProdutoPersiste.ler(2);  
10         System.out.println(p.getCodigo());  
11         System.out.println(p.getDescricao());  
12     }  
13 }
```

Classes abstratas



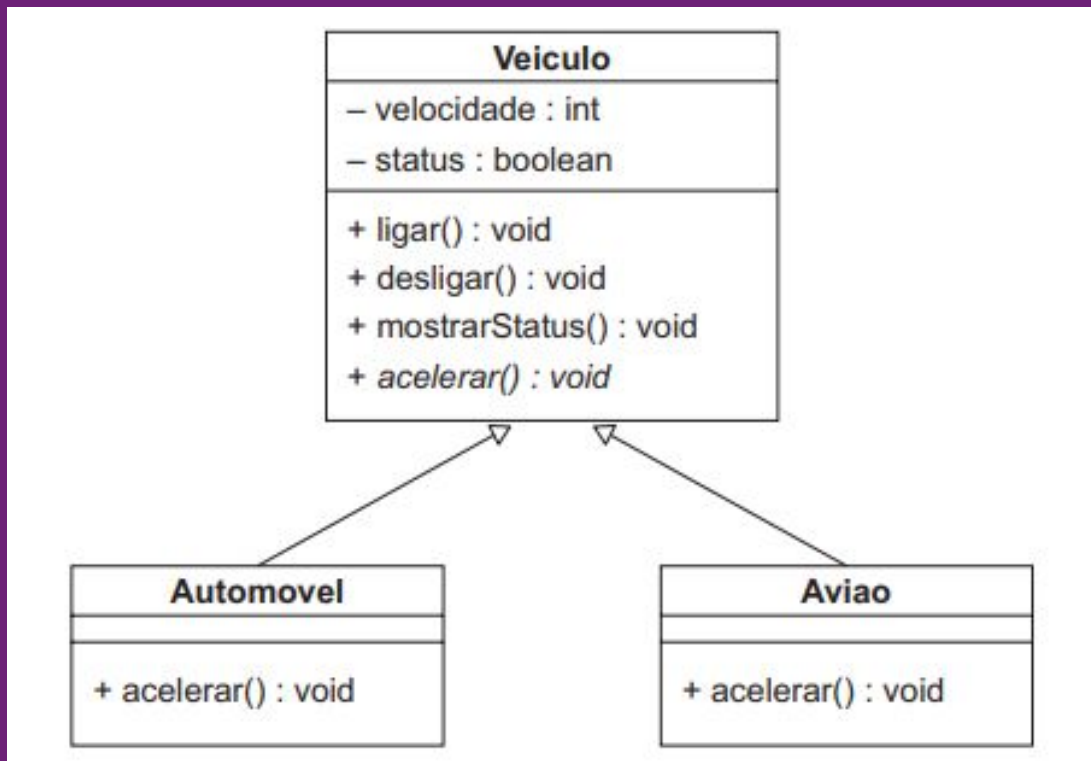
Classes abstratas

A classe abstrata é uma classe que NÃO PERMITE a geração de instâncias a partir dela, isto é, não permite que sejam criados objetos; ao contrário, uma classe concreta permite a geração de instâncias.

Bem, por que precisamos de uma classe que não permite a criação de objetos? Isso não parece estranho? Afinal, não é para isso que elaboramos classes? Bem, uma classe abstrata tem realmente essa característica, isto é, impedir que seja criado um objeto de seu tipo. Uma justificativa para seu uso é quando a classe serve apenas de base para a elaboração de outras classes.

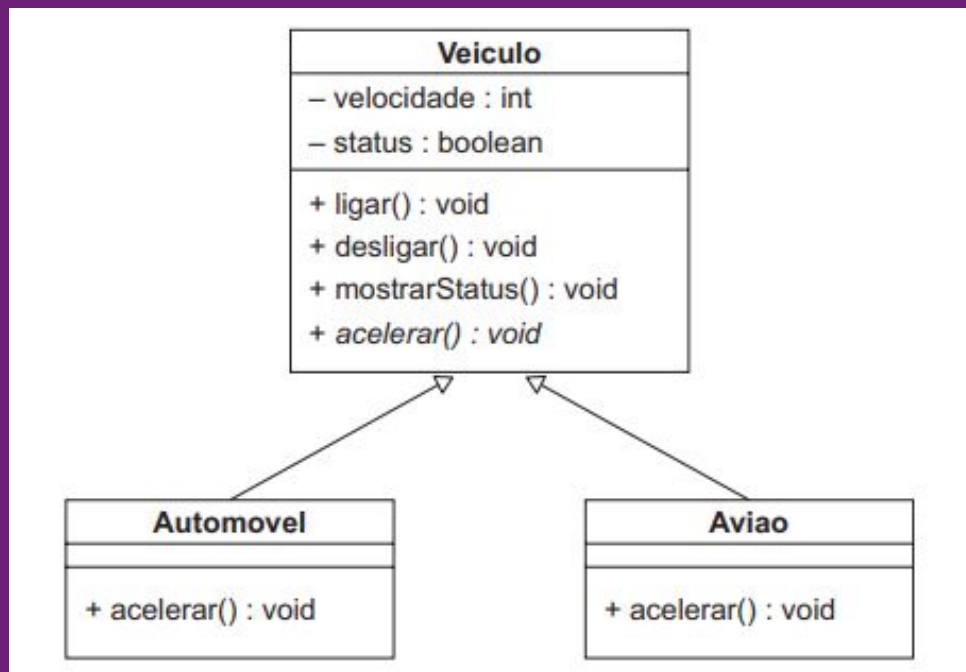
Classes abstratas

Uma classe abstrata pode ser usada também para definir um comportamento padrão para um grupo de outras classes. Vamos examinar essa característica de forma prática. Considere o diagrama de classes da figura:



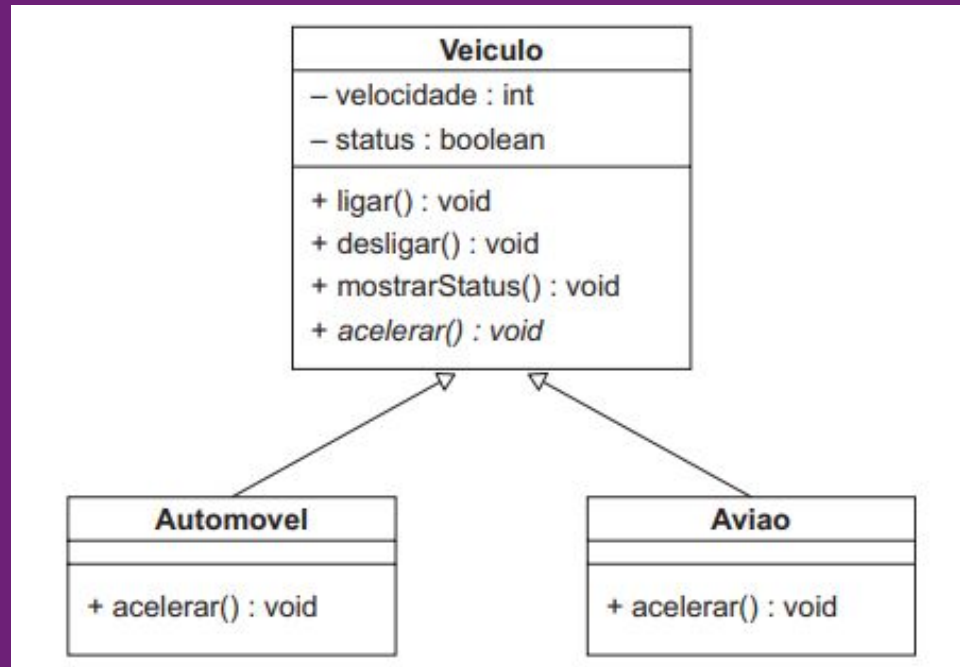
Classes abstratas

A classe Veiculo implementa (codifica) os métodos “ligar”, “desligar” e “mostrarStatus”, que são comuns às suas subclasses (Automovel e Aviao), isto é, funcionam do mesmo jeito para as duas classes, no entanto o método “acelerar” é apenas definido na classe Veiculo, ele não é implementado (não é codificado). A implementação fica a cargo das subclasses, já que acelerar um automóvel pode ser diferente de acelerar um avião.



Classes abstratas

Veja que a classe abstrata Veiculo apenas define o que as subclasses devem implementar. Reforçando: a classe abstrata Veiculo define um método abstrato chamado “acelerar”, que deve, obrigatoriamente, ser implementado em todas as subclasses de Veiculo. Veja pelo diagrama que as duas subclasses possuem também o método “acelerar”. Isso já não ocorre com os outros três métodos. As subclasses podem até realizar a implementação desses métodos, mas não é obrigatório, pois eles não são abstratos.



Interfaces

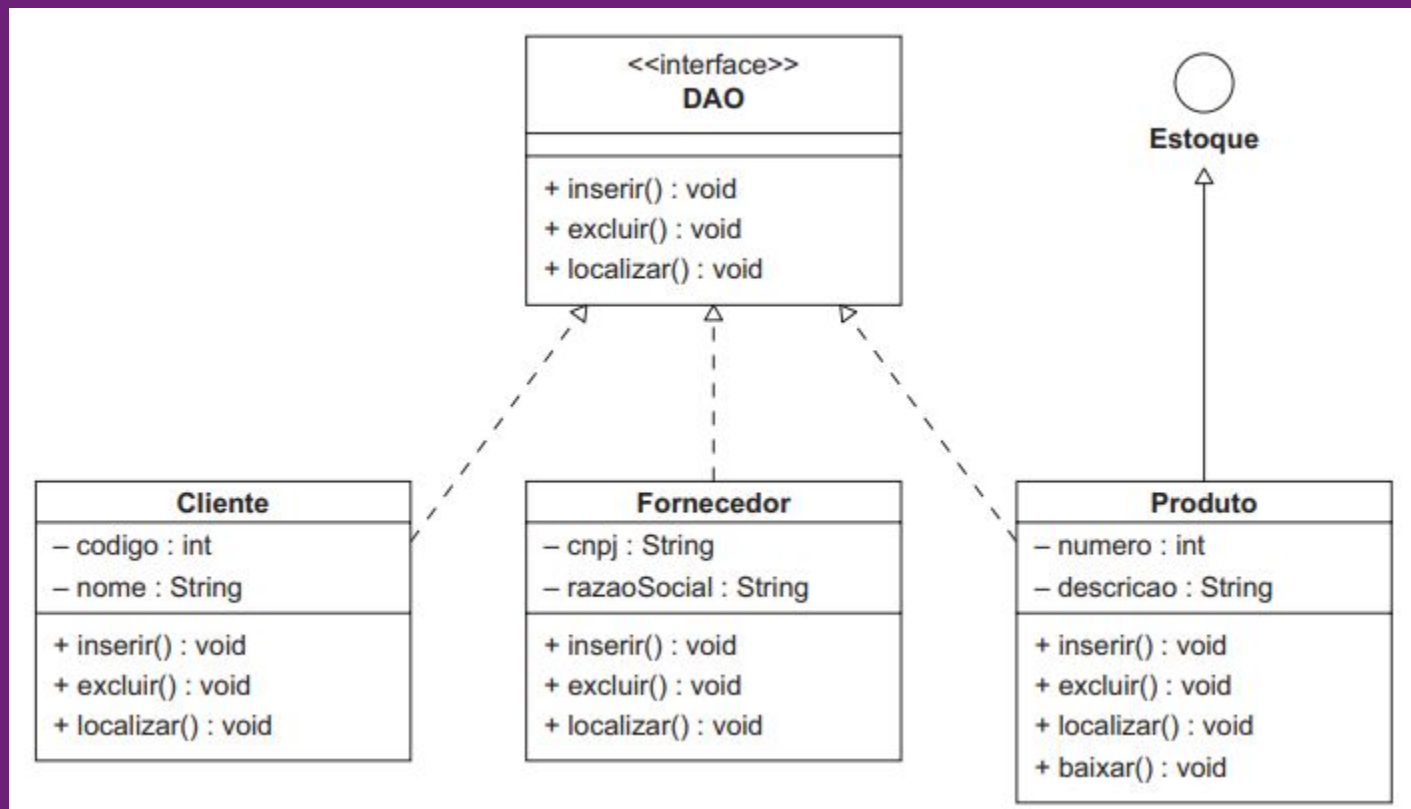


Interfaces

Costuma-se dizer que uma interface permite estabelecer um “contrato” entre as classes; funciona de maneira bastante similar a classes abstratas, porém não permite implementação de nenhum método, contendo apenas a especificação deste. A codificação de uma interface também é semelhante à de uma classe, no entanto a declaração de uma interface não se inicia com a palavra reservada `class` e sim `interface`.

Interfaces

Vamos elaborar um exemplo mais real para entendermos melhor.

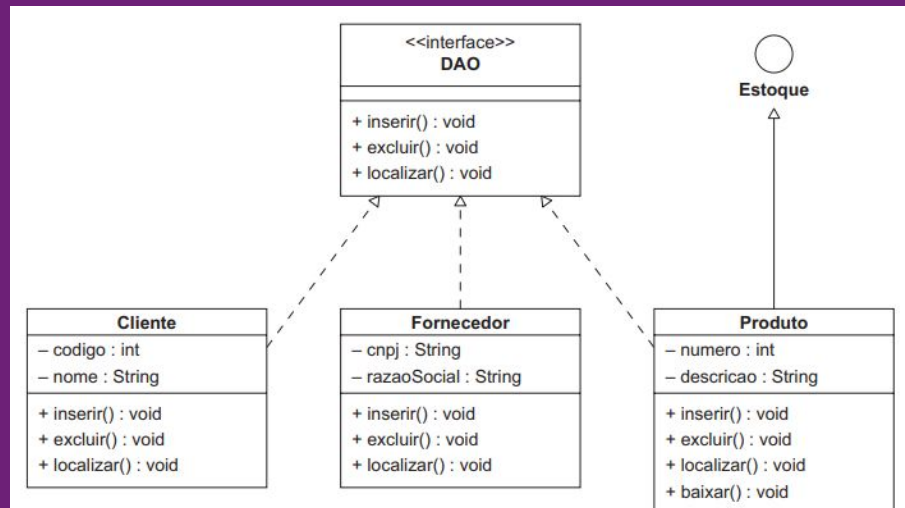


Interfaces

Esse diagrama representa o seguinte:

As classes Cliente e Fornecedor implementam a interface DAO, ou seja, codificam todos os métodos especificados na interface DAO, no caso os métodos “incluir”, “excluir” e “localizar”. Por esse motivo, cada uma das classes possui três métodos. As classes poderiam também implementar outros métodos não especificados pela interface DAO.

A classe Produto implementa a interface DAO (os métodos “incluir”, “excluir” e “localizar”) e a interface Estoque (o método “baixar”). Como dissemos anteriormente, apesar de a representação ser diferente, ambas são interfaces.



transforme ■ se

O conhecimento é o poder
de transformar o seu futuro.