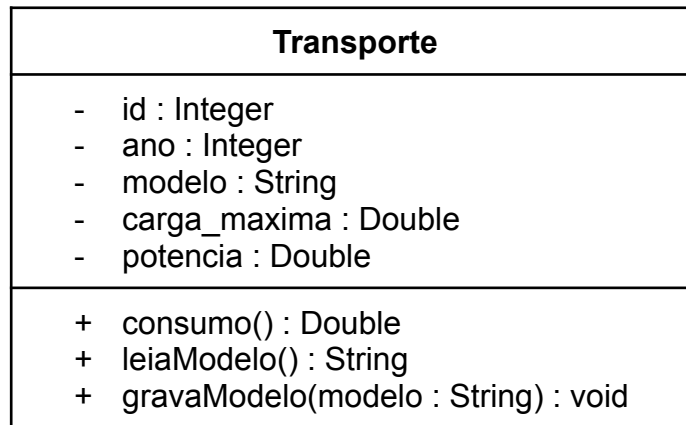


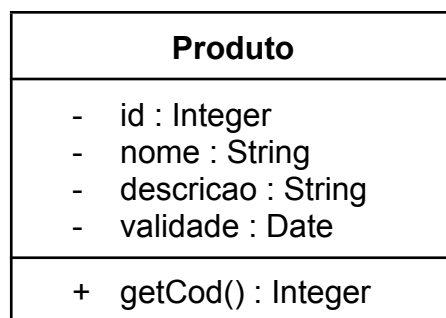
LISTA DE EXERCÍCIOS - ORIENTAÇÃO A OBJETOS COM JAVA

1. Implemente a classe abaixo:



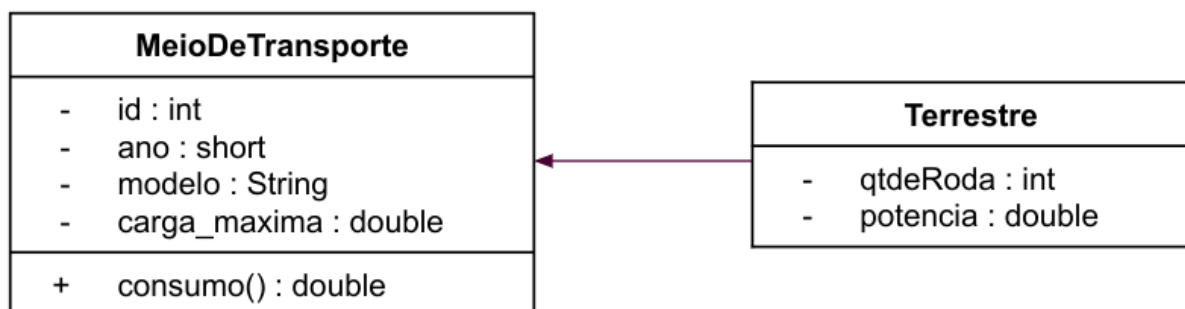
O método **consumo()** deve retornar **a potência do motor x carga máxima (kg) x 100**. Adicione nas classes o construtor padrão e um construtor que receba todos os parâmetros para inicializar os dados de um transporte.

2. Implemente a classe abaixo.



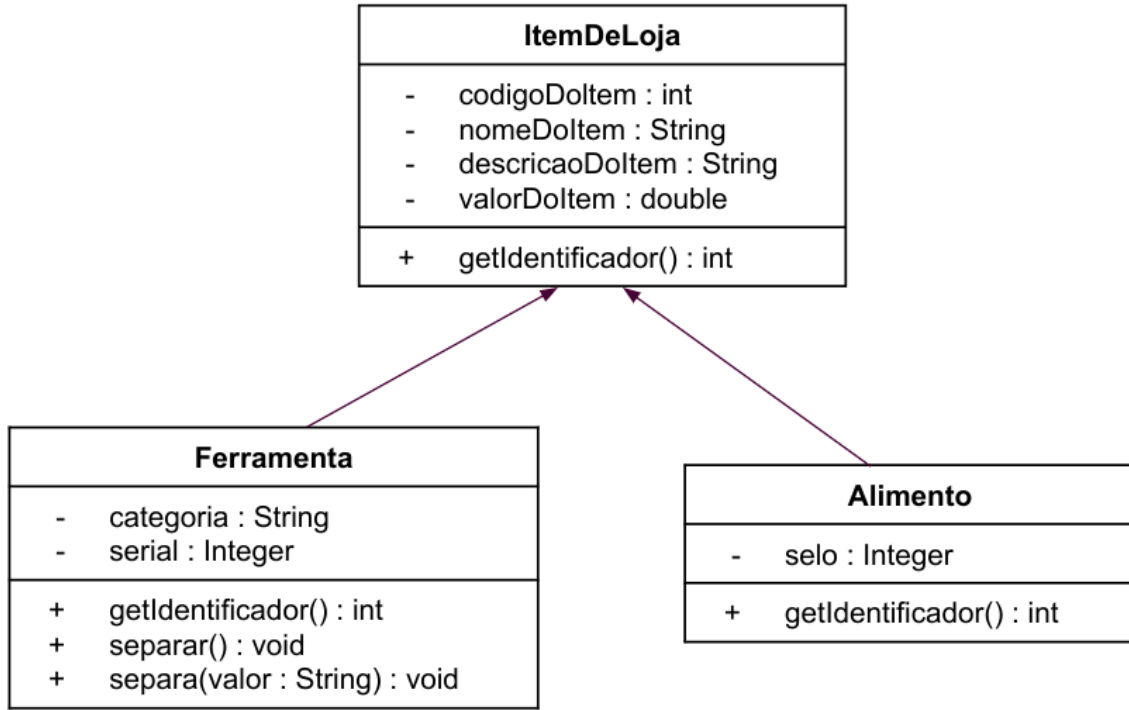
O método **getCod()** deve retornar o identificador da classe instanciada, ou seja, para um Produto deverá retornar o "id".

3. Implemente o modelo abaixo.



O método **consumo()** deve retornar o consumo médio do transporte. Para Terrestre, o consumo médio representa a **potência do motor x carga máxima (kg) x 100**.

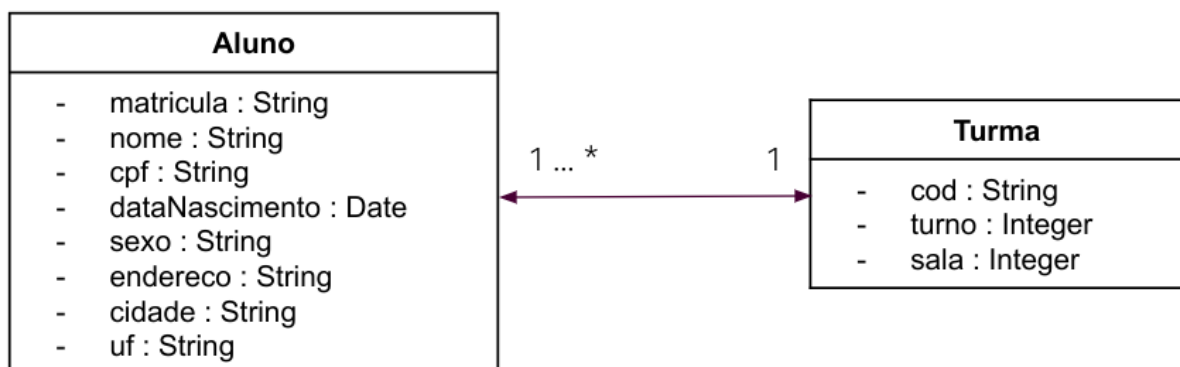
4. Implemente o modelo abaixo.



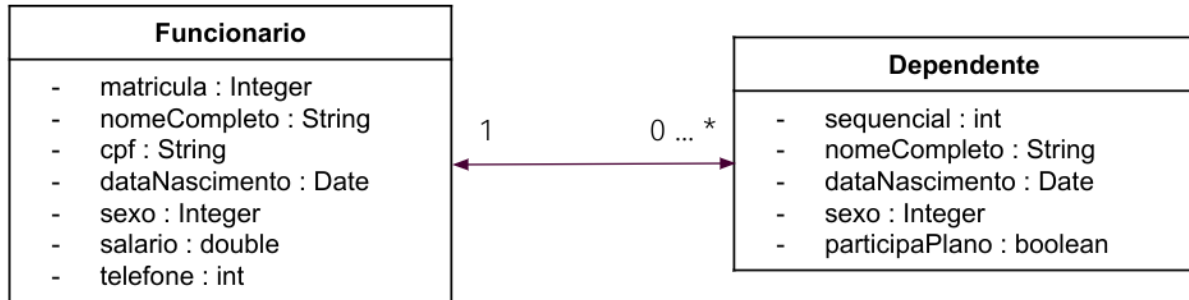
O método **getIdentificador()** deve retornar o identificador da classe Instanciada, ou seja, para um **ItemDeLoja** deverá retornar o “codigoDoItem”, para um **Alimento** o “selo” e para uma **Ferramenta** o “serial”.

O método **separar()** deve registrar a categoria em que foi armazenado o produto. Se a informação não for passada, o registro será feito na categoria “Outros”.

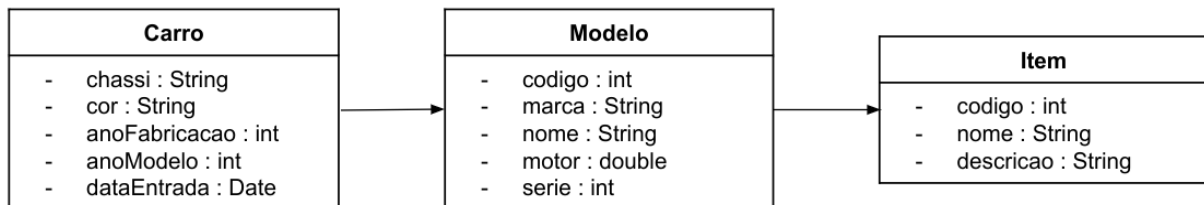
5. Implemente o modelo abaixo.



6. Implemente o modelo abaixo.



7. Implemente o modelo abaixo.



8. Sobre as características da Orientação a Objetos em Java, assinale verdadeiro (V) ou falso (F):

	Java possui menos de 9 tipos primitivos.
	Não é possível comparar tipos primitivos utilizando o operador “==”.
	Atributos de interface são sempre <i>final</i> .
	Palavras reservadas do Java são aquelas que não podem ser usadas para nomear membros de classes.
	Em Java toda variável, sem exceção, deve ser declarada.
	As subclasses podem adicionar membros próprios.
	Variáveis polimórficas podem referenciar objetos de uma classe (subclasse) da superclasse declarada.
	Em métodos, o modificador <i>abstract</i> obriga que suas subclasses não abstratas implementem o método.
	Um método definido como <i>final</i> pode ser sobreposto apenas por uma classe descendente.
	Classe com modificador <i>final</i> só pode ser especializada por herança.

	O modificador <i>final</i> estabelece que um atributo não pode ter seu valor modificado.
	A visibilidade definida por <i>public</i> permite que um determinado atributo seja acessível a partir de quaisquer métodos, objetos e classes.
	O modificador <i>protected</i> não restringe acesso oriundo de outro pacote.
	Os atributos e métodos privados de uma classe são acessíveis apenas nos métodos da própria classe.
	A estrutura <i>switch</i> aceita qualquer tipo de dado primitivo do Java.
	Encapsulamento consiste em proteger os atributos de acessos e modificações não controladas, centralizando o gerenciamento e a validação dos dados antes de serem armazenados pelos objetos.
	O encapsulamento permite definir o grau de visibilidade dos atributos de uma classe, estabelecendo restrições e permissões por métodos ao sistema.
	Não se implementa o encapsulamento em interface.
	A implementação de uma interface obriga a classe a implementar todos os métodos definidos, a não ser que a classe seja definida como abstrata, podendo assim deixar a implementação para as suas subclasses não abstratas.
	O polimorfismo de sobrecarga pode ser utilizado para distinguir, em uma determinada classe, dois métodos com o mesmo nome, mas com parâmetros diferentes.
	Polimorfismo é o conceito que define que mais vários métodos, com o mesmo nome, podem implementar diferentes formas de executar, dependendo de como ele é acionado.
	<i>Override</i> é um tipo de Polimorfismo que só ocorre em caso de herança.
	Na herança, todos os atributos são herdados, inclusive os privados.
	Em Java as subclasses herdam atributos e métodos da classe <i>Object</i> .
	Uma classe Java pode herdar de uma única classe na herança simples, e de várias na herança múltipla.
	Além de herdar entidades de sua classe-pai, uma classe derivada pode modificar métodos herdados, inclusive podendo até acrescentar novas entidades, sem afetar a estrutura da classe que a originou.

9. Marque a afirmativa *incorreta*:

- (A) Encapsulamento representa a proteção das propriedades referentes a um determinado objeto.
- (B) O encapsulamento permite concentrar as regras de negócio da classe dentro dela mesma.
- (C) Encapsulamento representa a proteção das propriedades referentes a um determinado objeto.
- (D) O encapsulamento é obrigatório para os atributos de uma classe.
- (E) Um atributo encapsulado não pode ser acessado diretamente por outro objeto ou classe.
- (F) Os métodos de acesso do encapsulamento (get/set) não devem ser privados.

10. A habilidade de duas ou mais classes, derivadas da mesma superclasse, responderem à mesma solicitação cada qual à sua maneira, é o conceito de:

- (A) Simetria
- (B) Abstração
- (C) Polimorfismo**
- (D) Encapsulamento
- (E) Reutilização
- (F) Herança

11. A respeito das palavras reservadas do Java, associe a coluna da esquerda à coluna da direita.

(A) super	(1) Método sem retorno
(B) default	(2) Define constantes
(C) new	(3) Tipo de dado
(D) this	(4) Implementa herança
(E) throw	(5) Declara importação
(F) void	(6) Implementa uma interface
(G) int	(7) Classe
(H) implements	(8) Incremento
(I) ++	(9) Laço de repetição
(J) class	(10) Levanta uma exceção
(K) get	(11) Sem significador

(L) static	(12) Instancia a class
(M) public	(13) Encerra a execução do método
(N) return	(14) Define membro de classe
(O) String	(15) Definição de uma classe
(P) abstract	(16) Auto referência
(Q) do	(17) Referencia classe pai
(R) extends	(18) Define classe sem objeto
(S) final	(19) Else do switch
(T) import	(20) Modificador de visibilidade

12. Defina com suas palavras o que se pede:

- (A) Classe
- (B) Objeto

13. O que compõe a assinatura de um método? Exemplifique.

14. A partir da classe abaixo, extraia o seu padrão/modelo (superclasse/interface). Atente aos métodos herdados, explicitados pela anotação **@Override**.

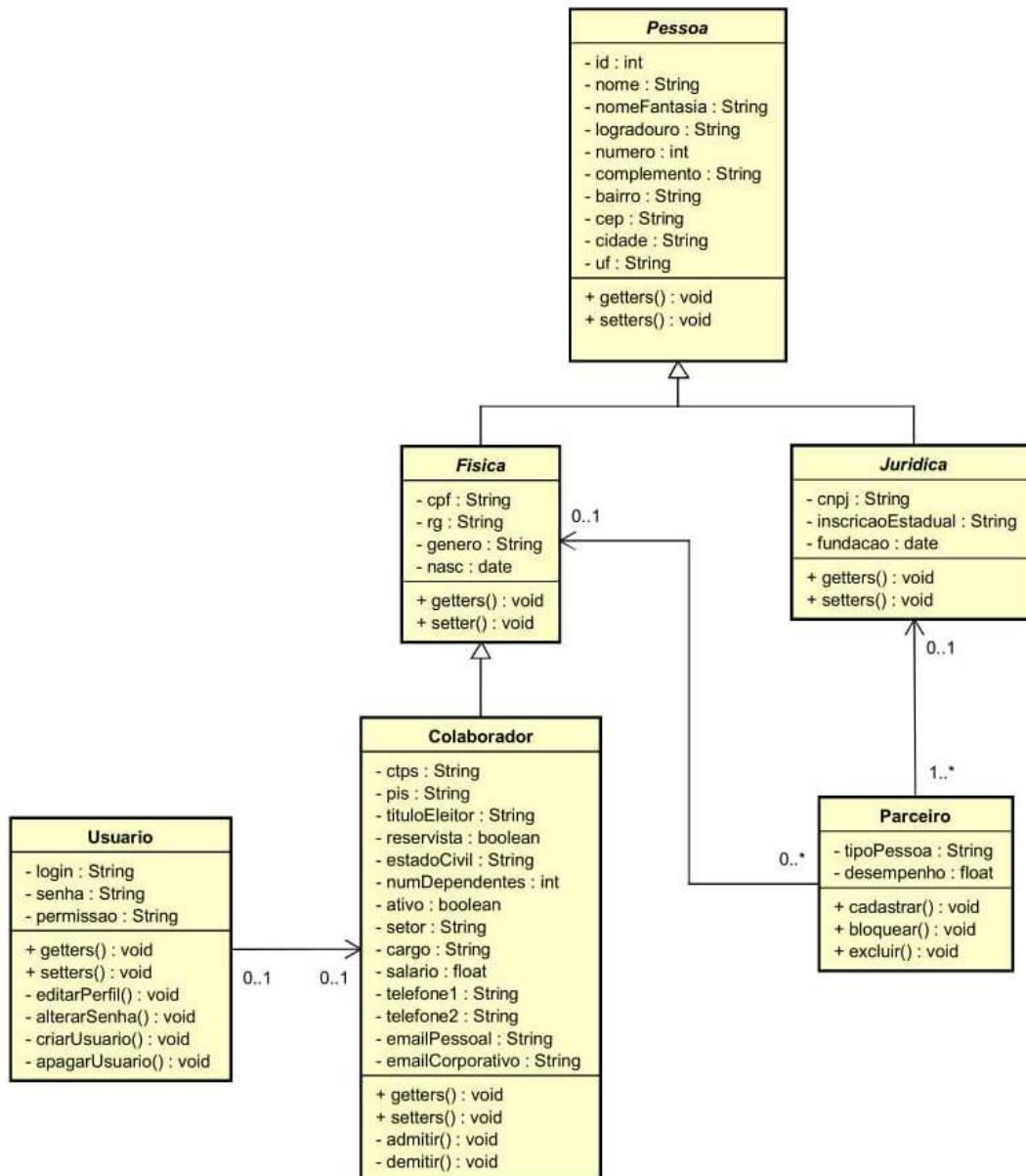
```
import java.util.*;

public class DadosImpl implements Dados {
    private ArrayList<Transporte> lista = new ArrayList<>();

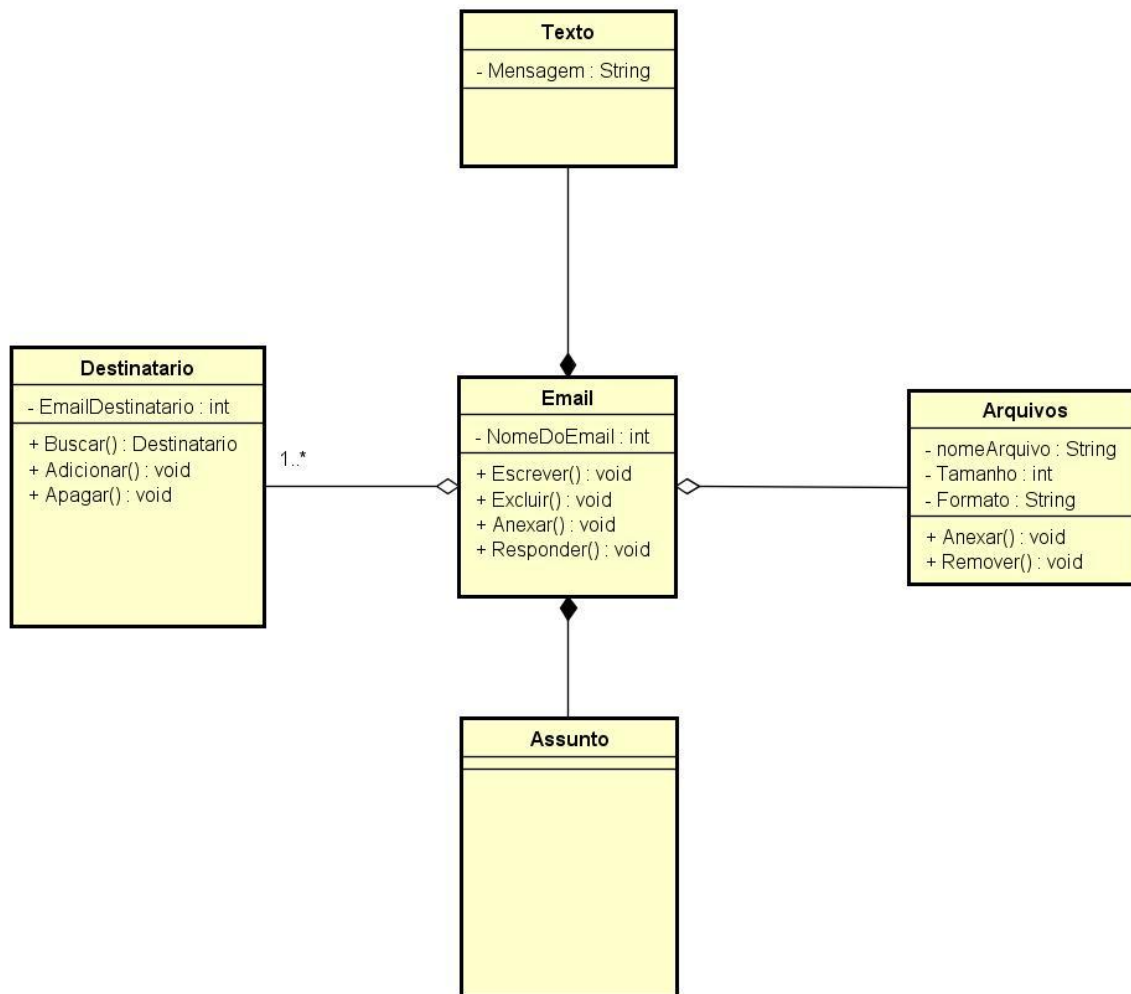
    @Override
    public void adicionar(Transporte t) throws Exception {
        lista.add(t);
    }

    @Override
    public void excluir(Transporte t) {
        lista.remove(t);
    }
}
```

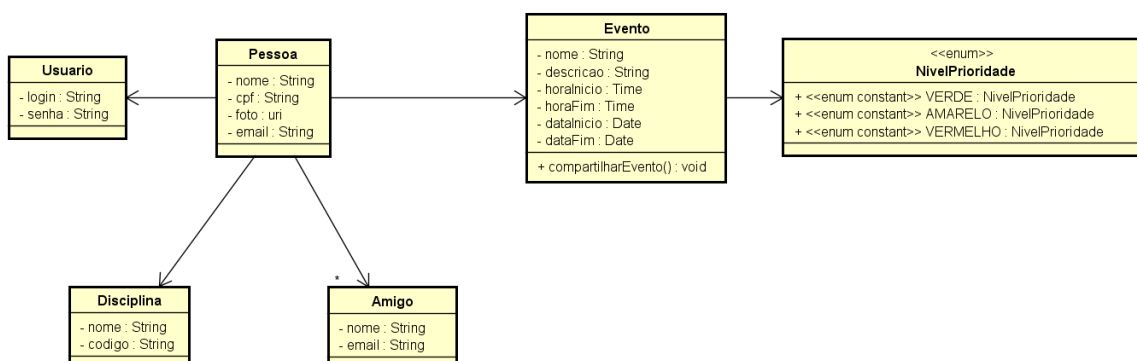
15. Implemente o modelo abaixo. Para os métodos, apenas a título de exercício, ao ser chamado, faça com que seja impresso o nome da classe e o nome do método.



16. Idem ao exercício anterior.



17. Idem ao exercício anterior.



18. Idem ao exercício anterior.

