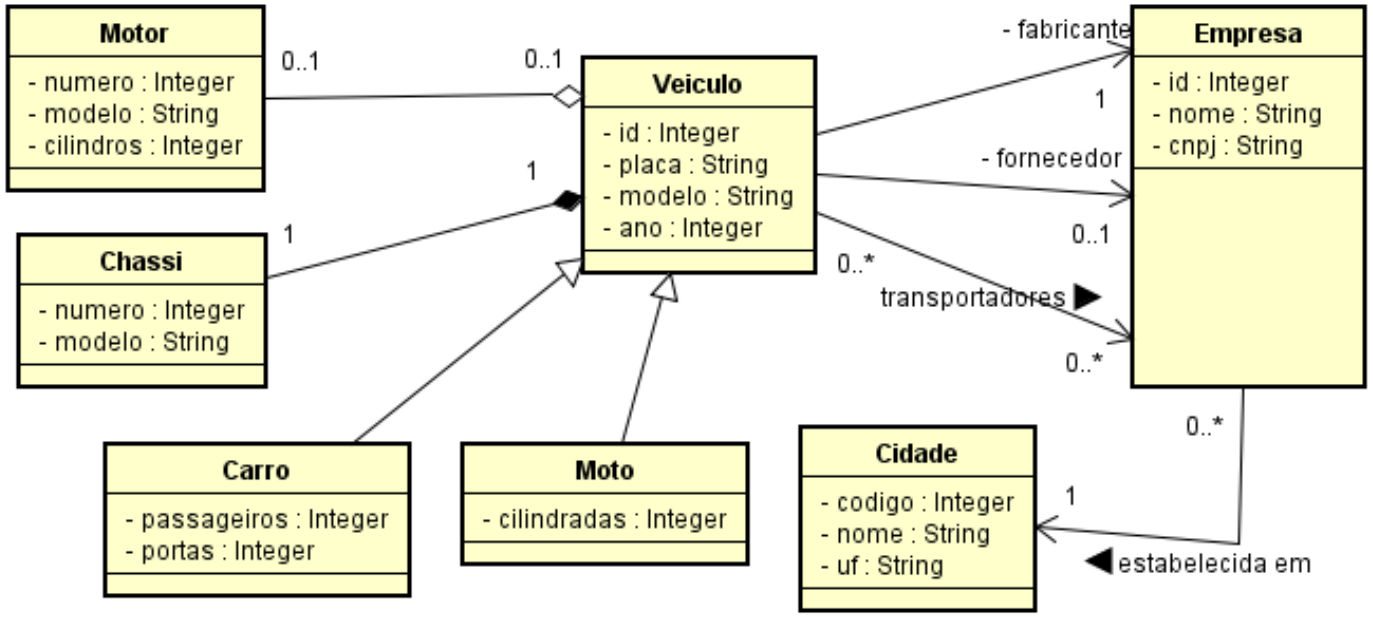
PROGRAMAÇÃO ORIENTADA A OBJETOS
Implementando relacionamentos entre classes e objetos
Prof. Jaqson Dalbosco

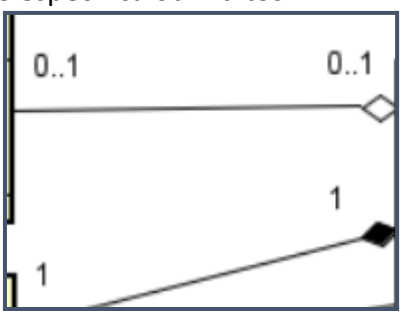
Os softwares orientados a objetos geralmente necessitam controlar diversos tipos diferentes de objetos, levando a definição de inúmeras classes para sua definição. Normalmente estes objetos precisarão ter relacionamentos estabelecidos entre eles, de forma semelhante ao que ocorre em um projeto de banco de dados.
Existem quatro tipos de relacionamentos que podem ser estabelecidos entre classes e objetos, sendo: Herança, agregação, agregação por composição e associação.
Segue um exemplo de como poderia ficar uma definição de classes para objetos de domínio para um estudo de caso simplificado de um software OO para uma montadora de veículos fictícia.



Num diagrama de classes da UML, além da especificação das classes, também é possível definir características adicionais para cada tipo de relacionamento, aumentando a capacidade de representação do modelo. Seguem detalhes que podem ser estabelecidos para cada relacionamento.

Navegabilidade: indica a direção do relacionamento de associação. Qual objeto irá referenciar o outro.

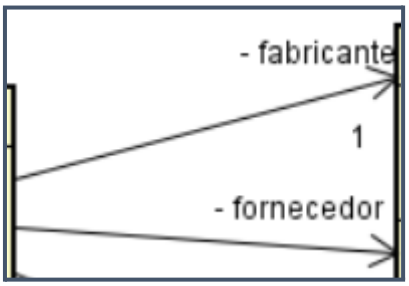
Multiplicidade (cardinalidade): indica a quantidade de objetos que poderão ser relacionados na referência, podendo ser nenhum, uma quantidade específica ou muitos.



Nome da relação: para dar melhor legibilidade e semântica ao relacionamento definido pelo modelo.

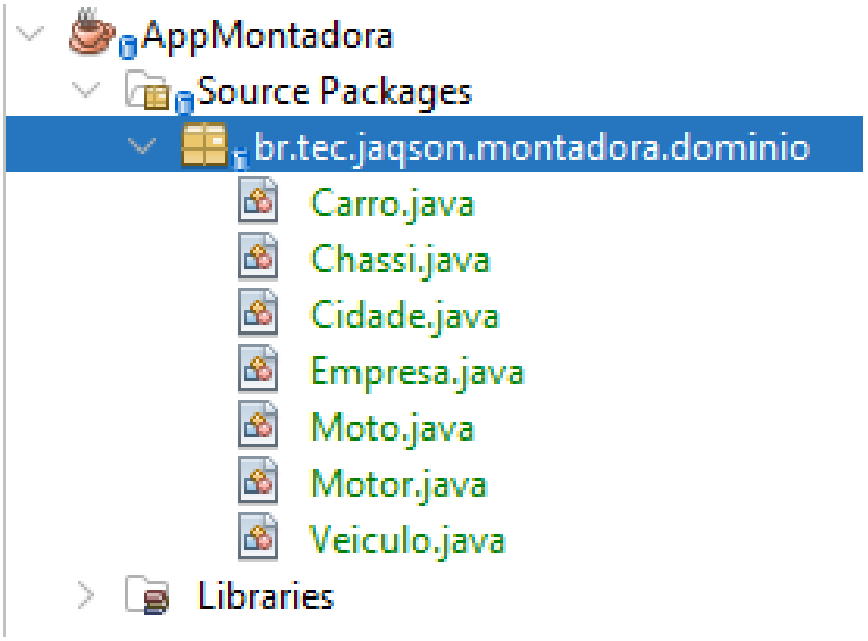
PROGRAMAÇÃO ORIENTADA A OBJETOS
Implementando relacionamentos entre classes e objetos
Prof. Jaqson Dalbosco

Papel: para definir o papel do(s) objeto(s) referenciado(s) pelo relacionamento. Pode ser usado como nome do atributo para a referência na implementação.



Para por em prática os conceitos aqui abordados, criar um projeto de Aplicativo Java (ou outra linguagem) e nele, implementar todas as classes para os objetos de domínio previstos no diagrama. Criar uma classe em cada arquivo. O arquivo deve ter o mesmo nome da classe e visibilidade public. Na sequência os exemplos de implementações em Java.

Como fica a estrutura do pacote de domínio com os arquivos para definição das classes.



O foco desse material não é na implementação da classe como um todo, mas sim em como implementar nas classes os relacionamentos definidos pelo modelo. Deve-se aplicar os conhecimentos já vistos anteriormente quanto à definição de classes e aqui agregar o conhecimento sobre como implementar cada tipo de relacionamento. Na sequência é descrito um exemplo de implementação para cada tipo de relacionamento.

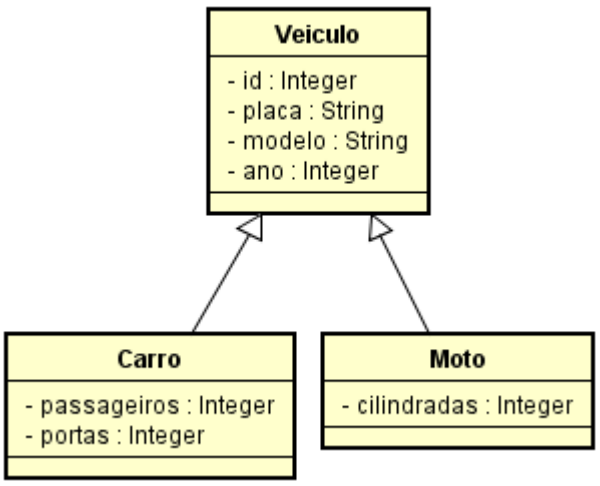
1. Implementando Herança

O relacionamento de herança define que uma **classe é uma especialização de outra classe mais genérica**. Segue o princípio de generalização/especialização, semelhante ao que é feito em processo de normalização de banco de dados.

PROGRAMAÇÃO ORIENTADA A OBJETOS
Implementando relacionamentos entre classes e objetos
Prof. Jaqson Dalbosco

Na herança, **a classe genérica é denominada superclasse**, e possui características comuns às outras classes mais **específicas / especializadas, que são denominadas como subclasses**.

Ao instanciar um objeto a partir de uma subclasse, este terá as características (atributos e métodos) de ambas as classes, da subclasse e da superclasse.



A representação do relacionamento no diagrama se dá pela ligação entre as classes com o **triângulo** aberto sempre no **lado da superclasse**.

Para a implementação de herança no Java utiliza-se a palavra **"extends" na definição das subclasses** de forma a declarar a sua extensão de uma superclasse.
Na herança não há navegabilidade e multiplicidade. Sempre uma subclasse herda características de uma superclasse.

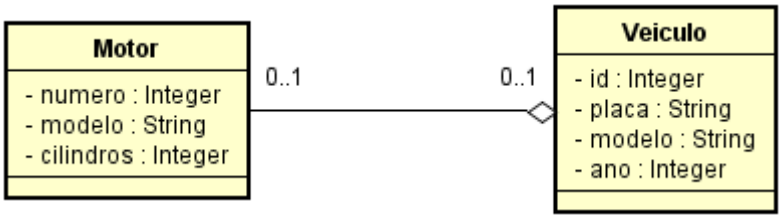
A seguir o código que ilustra como fica a implementação da herança. Cada classe deve ser definida em um arquivo separado, com o mesmo nome da classe e visibilidade public. Definem-se todos os atributos de cada classe, conforme consta no diagrama. Nas subclasses deve-se usar **extends** para a superclasse. Dessa forma todas as características (atributos e métodos) da superclasse serão herdadas pela sub-classe.
Como fica o código:

```
public class Veiculo {
    private Integer id;
    private String placa;
    private String modelo
    private Integer ano;
```

```
public class Carro extends Veiculo{
    private Integer passageiros;
    private Integer portas;

public class Moto  extends Veiculo{
    private Integer cilindradas;
```

2. Implementando Agregação



O relacionamento de agregação é utilizado quando se aplica o princípio de **"todo-parte"**. Ou seja, quando a formação de um objeto usa o outro como parte. A partir da agregação passamos a ver como um único objeto todo. Nesse caso é importante identificar quem será o **objeto todo** e quem será o **objeto parte**.
Na agregação, tanto o objeto todo, quanto o objeto parte podem existir sem que estejam no relacionamento, ou seja, **não há dependência** de um objeto para o outro. No exemplo, um motor pode existir, mesmo não estando no carro, assim como o carro pode existir, mesmo ainda estando sem o motor.

PROGRAMAÇÃO ORIENTADA A OBJETOS
Implementando relacionamentos entre classes e objetos
Prof. Jaqson Dalbosco

No exemplo, ao usar um motor (parte) para montar um carro (todo), passamos a ver um veículo como resultado.
No diagrama o **losango aberto** deve ficar no **lado da classe para o objeto todo (**chamada de **classe todo ou classe agregada)**. Ao implementar é adicionado um atributo na classe do objeto todo para referenciar sua(s) parte(s), adicionando atributo de referência para a **classe parte**.
Se a multiplicidade da referência na classe parte for 0 ou 1, defini-se o atributo como sendo do tipo da classe parte.
Se a multiplicidade pode ser maior que um, define-se o atributo como uma coleção de objetos do tipo da parte.

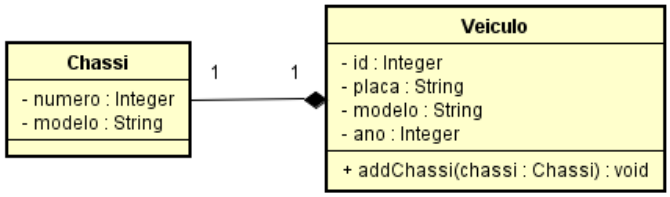
```
public class Veiculo {
    private Integer id;
    private String placa;
    private String modelo;
    private Integer ano;
    private Motor motor;
```

A classe parte não sofre nenhuma alteração!

Deve-se observar a multiplicidade máxima do atributo a ser implementado. Quando a multiplicidade máxima for superior a 1, deve-se utilizar uma coleção para definir o atributo, permitindo assim que tenha mais de uma instância da parte agregada. O atributo "motor" tem multiplicidade máxima 1 (um). Deve ser implementado como do tipo Motor, podendo referenciar uma instância de motor.
Se atributo "motor" tivesse multiplicidade máxima * (muitos). Deveria ser implementado como uma coleção, como abaixo.
private List<Motor> motores.
Essa forma de implementar considerando a multiplicidade, também se aplica aos relacionamentos de associação ou composição.

3. Implementando Agregação por Composição

O relacionamento de agregação por composição, ou somente composição, também se aplica ao princípio de **"todo-parte"**. A **diferença da composição para a agregação** é que **há dependência entre os objetos** quanto à sua existência e integridade. O objeto **parte sempre fica dependente** do todo quanto a sua existência e **o todo pode apresentar dependência** da parte. Exemplo de um prédio e apartamento, se for modelado deverá ser usado relacionamento de composição, pois um apartamento só pode existir como parte de um prédio e um prédio só existe se tiver as partes "apartamentos".
No diagrama o **losango fechado** deve ficar **no lado da classe para o objeto todo** (chamada de **classe todo ou classe agregada)**. Ao implementar é adicionado um atributo na classe do objeto todo para referenciar sua(s) parte(s) e um atributo na classe do(s) objeto(s) parte para referenciar o seu todo.
No estudo de caso adotado aqui, um veículo só passa a existir se houver um chassi, o qual é montado juntamente com a fabricação do veículo.



Para a implementação da composição **será definido um atributo em cada classe do relacionamento**, devido a dependência dos objetos envolvidos nesse tipo de relacionamento.

```
public class Veiculo {
    private Integer id;
    private String placa;
    private String modelo;
    private Integer ano;
    private Motor motor;
    private Chassi chassi;
```

Uma instância de veículo terá o atributo chassi que servirá para manter referência da instância de chassi utilizado na sua montagem.

PROGRAMAÇÃO ORIENTADA A OBJETOS
Implementando relacionamentos entre classes e objetos
Prof. Jaqson Dalbosco

A classe todo deverá ter um atributo para referenciar a(s) parte(s) e a classe parte deverá ter um atributo para referenciar o todo.
Implementa-se como um relacionamento de navegabilidade bidirecional, ou seja, ambas as classes envolvidas receberão atributos, referenciando uma à outra, pois nesse conceito de relacionamento o todo deve conhecer sua(s) parte(s) e cada parte deve conhecer a que todo pertence.

```
public class Chassi {
    private Integer numero;
    private String modelo;
    private Veiculo veiculo;
```

Uma instância de chassi terá o atributo veículo que servirá para manter referência da instância de veículo em que foi utilizado na montagem.
Geralmente na implementação desse tipo de relacionamento teremos na classe do objeto todo um método que permita adicionar sua(s) parte(s) dentro do seu escopo, pois a(s) parte(s) dependerão de estar em seu escopo para sua existência.
No exemplo seria o método addChassi, que seria o responsável por adicionar um chassi ao veículo.
Na implementação desse método precisa ser garantida a amarração da parte com o todo e do todo com a parte, ficando assim:

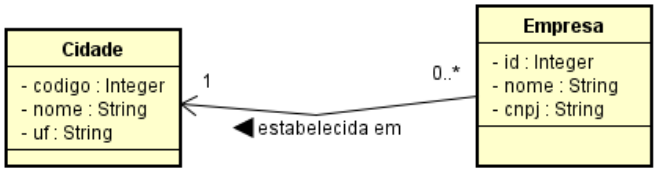
```
public void addChassi(Chassi chassi){
    this.chassi = chassi;
    chassi.setVeiculo(veiculo: this);
}
```

4. Implementando Associação

Associação é um relacionamento entre classes/objetos que não pode ser caracterizado como herança, nem como agregação e nem como composição, ou seja, não apresenta significado preciso e é aplicado para todos os demais tipos de relacionamento que podem ocorrer em um software onde não se aplica os outros tipos de relacionamento. Mais comum em aplicações voltadas para comércio e serviços. Exemplos: A cidade onde uma pessoa mora, o grupo de um produto, a marca de um equipamento etc.

No diagrama **a linha com seta** deve ficar **direcionada para a classe do objeto que será associado**. Ao implementar é adicionado um atributo na classe do objeto que terá a referência para o outro objeto associado.

Segue como fica a implementação das associações definidas no estudo de caso de exemplo.

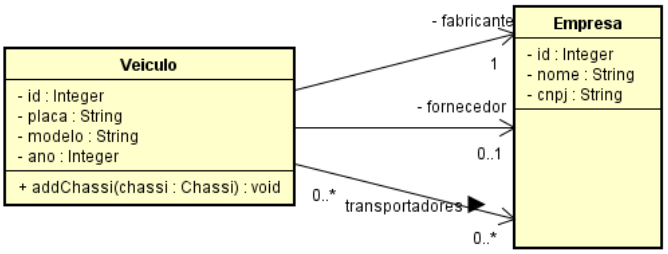


Ao implementar a associação **observa-se a navegabilidade estabelecida** (direção da seta). No exemplo, existe navegabilidade da classe Empresa para Cidade. Portanto, **a implementação é realizada adicionando o atributo para a navegabilidade (cidade).** O atributo cidade na classe Empresa permitirá que cada instância de empresa tenha um atributo para a referência da cidade onde está estabelecida.

A definição da classe fica como segue:

```
public class Empresa {
    private Integer id;
    private String nome;
    private String cnpj;
    private Cidade cidade;
```

Na outra situação de associação do exemplo, há três associações com papéis e multiplicidades diferentes para cada relação, como segue.



Neste caso, além dos cuidados descritos no exemplo de associação anterior, deve-se também atentar aos papéis ou nomes de relacionamento para entender o sentido para cada associação. Utilização estes nomes para definir os atributos na classe. Ficando o código da classe como segue.

```
public class Veiculo {
    private Integer id;
    private String placa;
    private String modelo;
    private Integer ano;
    private Motor motor;
    private Chassi chassi;
    private Empresa fabricante;
    private Empresa fornecedor;
    private List<Empresa> transportadores;
```

Na implementação do relacionamento para não há preocupação com a multiplicidade mínima, somente com a máxima. Quando a multiplicidade máxima para o atributo que está sendo implementado for 0 ou 1, o atributo será definido com o tipo da classe que está sendo associada. Se for superior a 1, utilização coleções para sua definição.
A implementação da multiplicidade mínima superior a 0 (obrigatoriedade) será realizada posteriormente utilizando-se de recursos específicos para este fim (validações). Neste momento basta definir as estruturas para que possam suprir as definições especificadas no diagrama. Estas definições aplicam-se também para implementações de agregação e composição.

5. Implementando Encapsulamento de atributos e métodos

Após todas as classes definidas conforme o diagrama, como nos exemplos anteriores, em cada classe realizar as seguintes implementações:

a) Padrão JavaBeans:

- Na definição da classe adicionar **implements Serializable**
- Adicionar método construtor sem argumentos: Botão direito > inserir código > Construtor > Gerar
- Sobrecarga de métodos construtores conforme achar necessário
- Fazerencapsulamento: atributos com visibilidade private e métodos de acesso get e set com visibilidade public: Botão direito > Refatorar > Encapsular campos > selecionar todos > Refatorar.

b) Sobrescrever/substituir métodos equals() e hashCode(): Botão direito > inserir código > equals e hashCode > selecionar atributo identificador > Gerar. Irá permitir identificar o objeto pelo(s) seu(s) atributo(s) identificador(es).
b) Sobrescrever/ substituir método toString(): Botão direito > inserir código > toString > selecionar atributo para descrever o objeto com uma string > Gerar. Adequar o código gerado. Será útil quando for preciso mostrar o objeto como um todo no formato de uma String.

A partir de definidas todas as classes e suas implementações para atender os relacionamentos, basta instanciar os objetos e estabelecer suas referências de relações, além de seus demais atributos (Integer, String, Float etc).
Exemplos:

```
public static void main(String[] args) {
    Empresa fabricante = new Empresa(id: 1, nome: "Empresa ABC", cnpj: "12.345.678/0001-12");
    Empresa fornecedor = new Empresa(id: 1, nome: "Empresa DEF", cnpj: "98.765.432/0001-98");
    Empresa transpUm = new Empresa(id: 1, nome: "Transportadora XXX", cnpj: "45.658.432/0001-98");
    Empresa transpDois = new Empresa(id: 1, nome: "Transportadora ZZZ", cnpj: "12.784.432/0001-98");
    Motor m = new Motor(numero: 1, modelo: "MOT654", cilindros: 4);
    Veiculo v = new Veiculo(id: 1, placa: "ASD1234", modelo: "Fusca Itamar",ano:1995);
    v.addChassi(new Chassi(numero: 1, modelo: "MOD1234"));
    v.setMotor(motor: m);
    v.setFabricante(fabricante);
    v.setFornecedor(fornecedor);
    v.setTransportadores(new ArrayList<>());
    v.getTransportadores().add(e: transpUm);
    v.getTransportadores().add(e: transpDois);
}
```

6. Implementar os métodos

Por fim, implementar os métodos para as operações em cada classe, conforme definidos no processo de abstração. Neste material os métodos não foram abordados, pois aqui o foco se deteve na implementação dos diferentes tipos de relacionamentos que podem ser estabelecidos entre objetos.

7. Exercício

Criar um projeto “AppMontadora” para o exemplo deste material e Implementar o que segue:

- Em pacote de domínio definir todas as classes com seus atributos e atendendo aos relacionamentos.
- Encapsular todos os atributos.
- Todas as classes devem ter métodos construtores.
- Todas as classes devem ter os métodos equals e hashCode substituídos de forma a usar atributo(s) como identificador(es) do objeto.
- Todos os objetos deverão ter um método toString substituído de forma a gerar uma representação dos dados do objeto em forma de uma String.
- Criar uma classe Java Principal e nela as implementações para:
    - Devem ter listas para motores, veículos, cidades e empresas
    - Instanciar 3 motores e colocar na lista
    - Instanciar 3 cidades e colocar na lista
    - Instanciar 3 empresas e colocar na lista
    - Instanciar 1 carro e 1 moto e colocar na lista
    - Percorrer a lista e mostrar um relatório de veículos mostrando os seguintes dados:

PROGRAMAÇÃO ORIENTADA A OBJETOS
Implementando relacionamentos entre classes e objetos
Prof. Jaqson Dalbosco

- Carro
  - placa, modelo, numero de passageiros, cilindros, número do chassi e nome do fabricante
- Moto
  - placa modelo, modelo do motor, nome do fornecedor