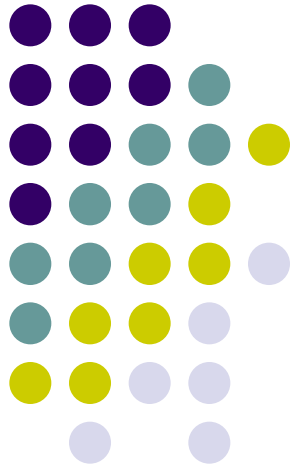
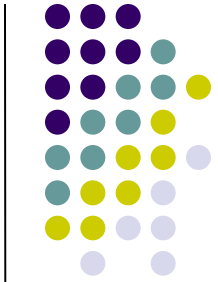
COM220

Aula 6: Programação em Java
Conceitos básicos
Parte I

Prof. Laércio Baldochi

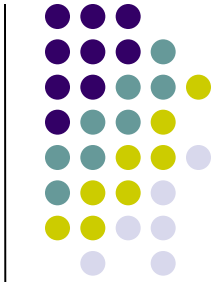


POO & Java



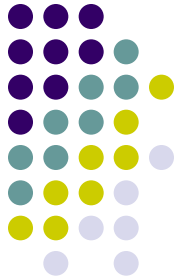
- Analogia: Computador =
  + Motherboard
  + Memória RAM
  + HD
  + Placa de vídeo
  + Monitor
  + Teclado
  + Gabinete
  + Fonte

POO & Java



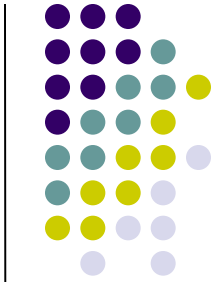
- Componentes podem ser comprados isoladamente e conectados formando um computador
- O funcionamento de cada componente é extremamente complicado e varia drasticamente de um componente para outro
  - Para montar um computador basta saber a função de cada componente e como tais componentes interagem entre si

POO & Java



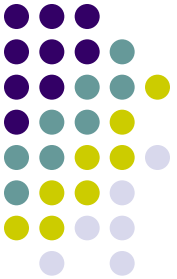
- O que este exemplo tem a ver com POO?
  - Tudo!
    - Um programa orientado a objeto nada mais é que uma coleção de componentes (objetos) que interagem entre si para a resolução de um problema
    - Assim como no exemplo do computador, para usar estes objetos basta saber como eles funcionam e quais são suas interfaces
      - Métodos, atributos

POO & Java



- POO provê uma série de conceitos que facilitam a criação e a manipulação de objetos. Entre eles está o conceito de classes e instâncias
- Classe
  - Modelo para vários objetos com características similares. Agrupa todas as características e funcionalidades de um conjunto particular de objetos

Um modelo



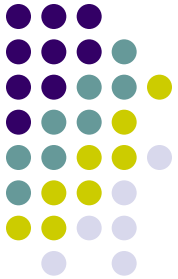
Lampada
- estadoDaLampada
- acende( ) - apaga( ) - mostraEstado( )

Modelo lâmpada

O pseudocódigo

```
Modelo lampada //  representa uma lâmpada em uso
Inicio do modelo
   dado do estadoDaLampada; // indica se a lâmpada está ligada ou não
   operacao acende( )       // acende a lâmpada
      inicio
            estadoDaLampada = aceso;
      fim

   operacao mostraEstado( )       // mostra estado da lâmpada
      inicio
            se (estadoDaLampada == aceso)
                        imprime “A lâmpada está acesa”;
            senão
                        imprime “A lâmpada está apagada”;
      fim
fim do modelo
```

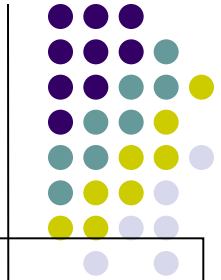
Outro modelo



Data
- dia - mes - ano
- inicializaData(d, m, a) - dataEValida(d, m, a) - mostraData( )

Modelo data

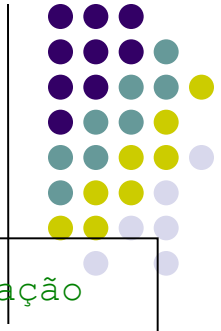
O pseudocódigo



```
Modelo Data
Inicio do modelo
  dado dia, mes, ano; // componentes da data
// inicializa simultaneamente todos os dados para esta operação
  operacao inicializaData(umDia, umMes, umAno)//argumentos para esta operação
     inicio
       se dataEValida(umDia, umMes, umAno) //repassa os argumentos a operação
         inicio
           dia = umDia;
           mes = umMes;
           ano = umAno;
         fim
       senão
         inicio
           dia = 0;
           mes = 0;
           ano = 0;
          fim
      fim
```

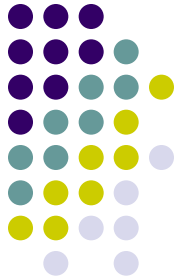
Modelo data (cont)

O pseudocódigo



```
  operacao dataEValida(umDia, umMes, umAno) // argumentos para a operação
    inicio
      // Se a data passada for válida, retorna verdadeiro
      se ((dia >= 1) e (dia <= 31) e (mes >= 1) e ( mes <= 12))
         retorna verdadeiro;
      senão  // senão retorna falso
         retorna falso;
    fim

  operacao mostraData( ) // mostra a data imprimindo valores de seus dados
    inicio
       imprime dia;
       imprime "/";
       imprime mes;
       imprime "/";
       imprime ano;
    fim
Fim do modelo
```

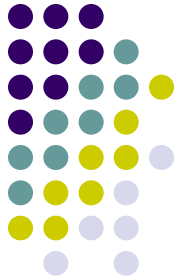
POO & Java



- Então...
  - Classe define um modelo de dados e as operações sobre esses dados

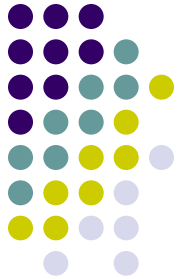
- Operação de instanciação (operador new)
  - Cria representações concretas do modelo abstrato
    - Objetos

POO & Java



- Biblioteca de classes da linguagem Java
  - classes prontas para serem usadas
  - definem objetos usados nas tarefas básicas de programação (Arrays, funções matemáticas, Strings)
  - provêem também facilidades para realização de tarefas mais complexas
    - programação em rede
    - criação de gráficos

POO & Java

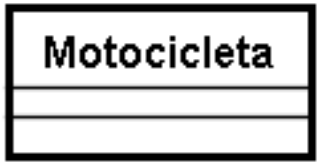


- Biblioteca de classes da linguagem Java
  - Para se escrever programas simples, basta criar uma única classe e usar as classes definidas na biblioteca de classes Java
  - Para se escrever programas mais complexos é necessário criar um conjunto de classes e definir as interações entre elas

POO & Java

- Criando uma classe (para uma aplicação Java)

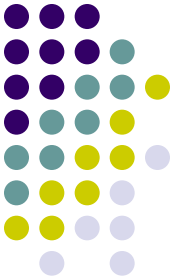
```
public class Motocicleta {

}
```



Criando uma classe

- Inserindo atributos (variáveis de instância), que servem para caracterizar os objetos criados a partir desta classe

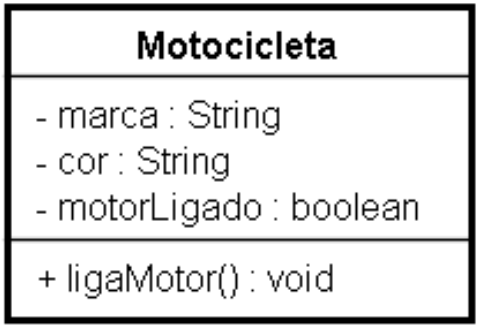
```
public class Motocicleta {

    private String marca;
    private String cor;
    private boolean motorLigado;
}
```



Criando uma classe

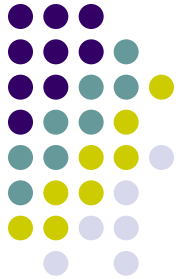
- Acrescentando comportamento
  - Através da criação de métodos é possível determinar o que se pode fazer com cada objeto da classe criada.

```
private void ligaMotor() {
    if(motorLigado == true)
      System.out.println("O motor já está ligado!");
    else {
      motorLigado = true;
      System.out.println("Motor acaba de ser ligado!");
    }
}
```




```
public class Motocicleta {
    private String marca;
    private String cor;
    private boolean motorLigado;

    private void ligaMotor() {
        if(motorLigado == true)
          System.out.println("O motor já está ligado!");
        else {
          motorLigado = true;
          System.out.println("Motor acaba de ser ligado!");
          }
    }
}
```

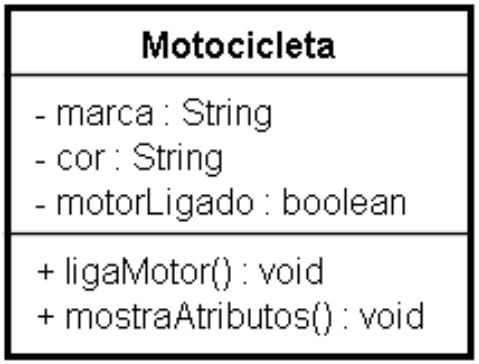
Criando uma classe



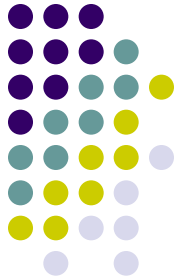
- Esta classe já é capaz de caracterizar seus objetos (através de atributos) e já possui um comportamento (método) especificado
  - Este comportamento tem como função mudar o estado de um objeto através da modificação do valor de um de seus atributos
- É possível especificar métodos que realizam ações com um objeto sem mudar seu estado. Exemplo: método mostraAtributos()

Criando uma classe

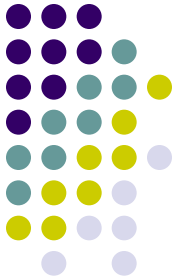
```
private void mostraAtributos()
{
    System.out.println("Esta  motocicleta  é  uma " + marca + " " + cor);
    if (motorLigado == true)
        System.out.println("Seu motor está ligado!");
    else
        System.out.println("Seu motor está desligado!");
}
```



Criando uma classe

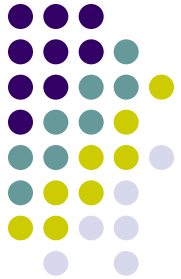


- Já criamos os atributos e comportamentos de nossa classe.
- Uma vez que a mesma será uma aplicação, falta criar um método chamado main()
  - Ponto de início do processamento de uma aplicação Java



```
public static void main (String args[]) {
    Motocicleta m = new Motocicleta();
    m.marca = "Suzuki";
    m.cor = "Vermelha";
    m.mostraAtributos();
    System.out.println("--------------");
    m.ligaMotor();
    System.out.println("--------------");
    m.mostraAtributos();
    System.out.println("--------------");
    m.ligaMotor();
}
```

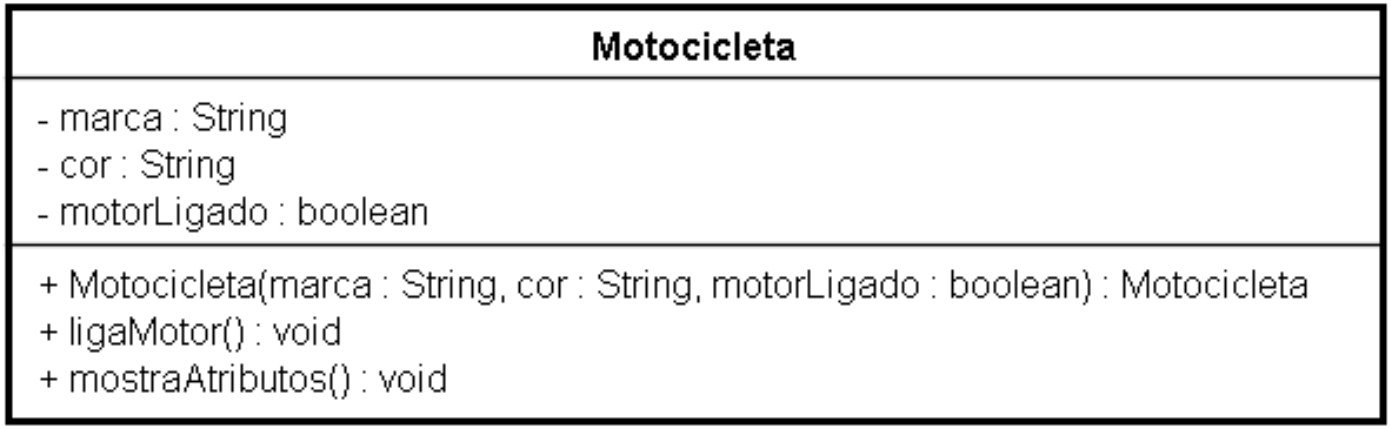
Métodos construtores

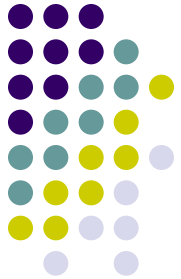


- A maioria das classes Java apresentam os chamados métodos construtores
  - Possuem o mesmo nome da classe
  - São usados para construir instâncias das classes
    - Em geral, já fazem o ajuste das variáveis de instância

Métodos construtores

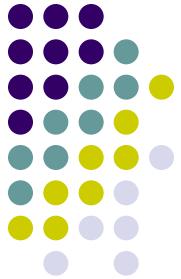
```
//construtor para a Motocicleta
public Motocicleta(String pMarca, String pCor, boolean pMotorLigado){
    marca = pMarca;
    cor = pCor;
    motorLigado = pMotorLigado;
}
```





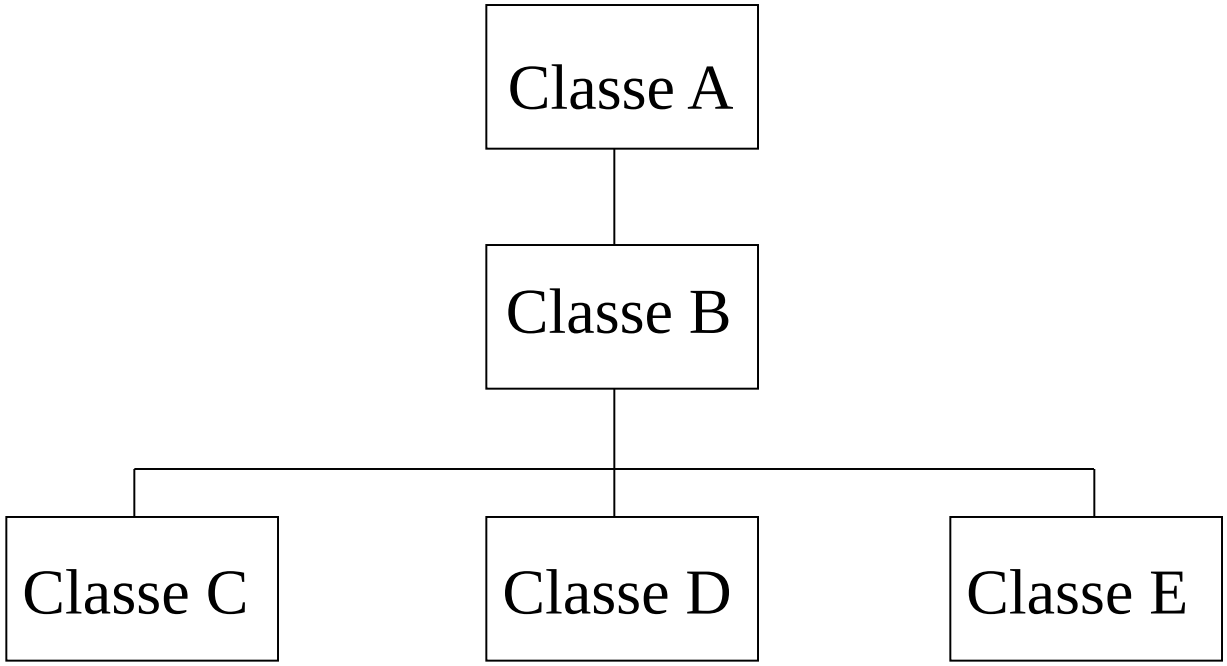
```
public static void main (String args[]) {
    Motocicleta m = new Motocicleta("Suzuki", "vermelha",
    falso);
    //m.marca = "Suzuki";
    //m.cor = "Vermelha";
    m.mostraAtributos();
    System.out.println("-------------");
    m.ligaMotor();
    System.out.println("-------------");
    m.mostraAtributos();
    System.out.println("-------------");
    m.ligaMotor();
}
```

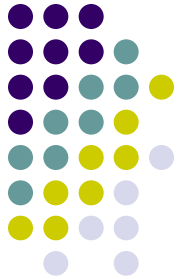
Herança



- Conceito básico da POO através do qual pode-se definir uma classe (classe filha ou subclasse) a partir de uma ou mais classes já existentes (classes pai ou superclasses)
- Segundo este conceito, uma subclasse herda atributos e comportamentos de sua(s) classe(s) pai

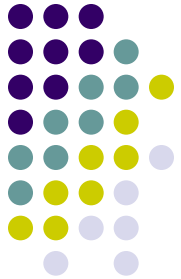
Herança





Herança

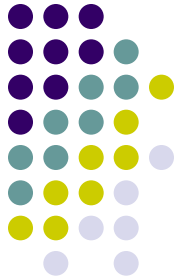
- Toda classe existente na linguagem Java é subclasse de alguma classe
- No topo da hierarquia de classes da linguagem Java está a classe *Object*, sendo que toda classe herda desta superclasse
- Quando se define uma classe como a Motocicleta, que não especifica explicitamente uma classe pai, subentende-se que esta classe herda da classe *Object*



Herança

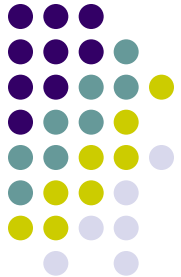
- Quando se define uma subclasse, esta automaticamente herda da superclasse (e de suas superclasses) seus atributos e métodos, podendo usá-los diretamente sem que tenha que definí-los novamente
  - A subclasse estende a superclasse provendo novos atributos e comportamentos. Ex:
    - Árvore
      - Árvore frutífera

Herança



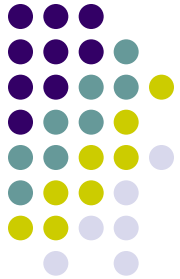
- Quando se cria um programa Java, está se estendo a hierarquia de classes já existente
- Se for necessário criar um conjunto grande de classes, é desejável que elas formem também uma hierarquia, de forma a modularizar as classes e fazer reaproveitamento de código
- Exemplo: Suponha que se queira criar a classe carro.

Herança



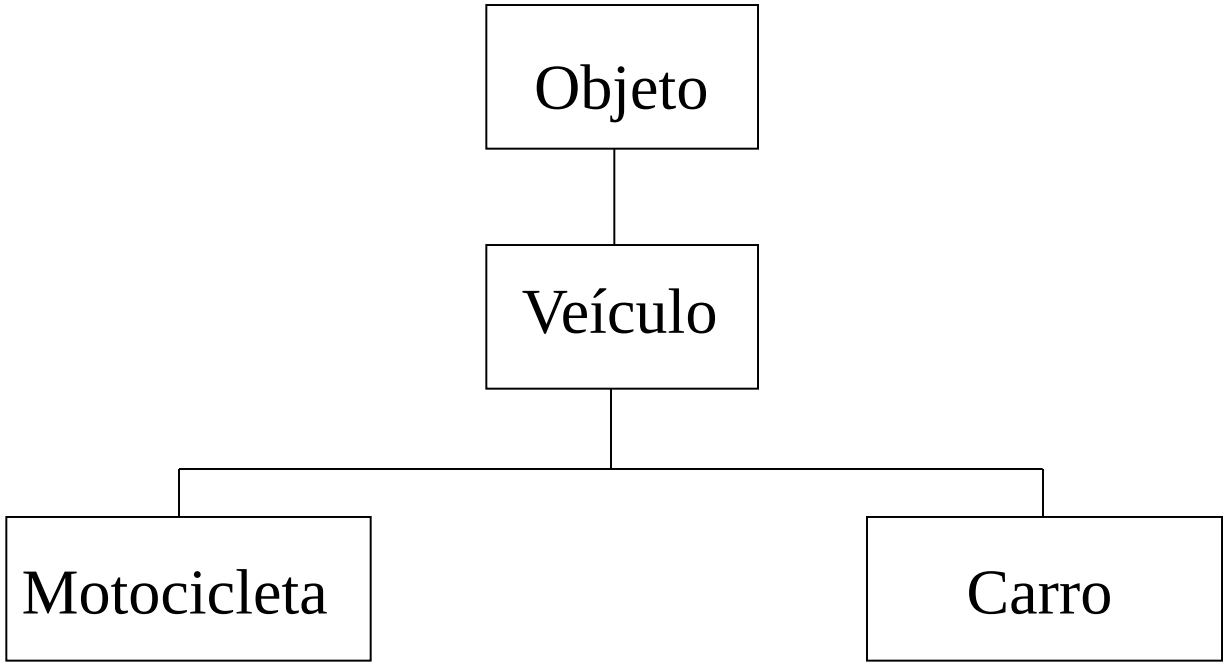
- Criação da classe carro
  - Carro: um carro possui uma série de coisas em comum com uma moto:
    - motor
    - rodas
    - velocímetro
    - marca/cor/modelo
  - Criar a classe carro copiando parte do código da classe Motocicleta.

Herança

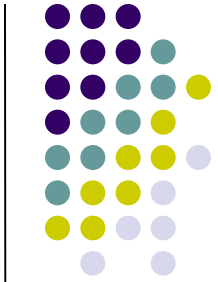


- Criação da classe carro
  - Copiando código você teria informação em duplicidade e estaria indo na contra mão da POO
  - Uma melhor solução, em concordância com os princípios da POO, seria criar uma superclasse para as classes *Motocicleta* e *Carro*, na qual pudesse ser colocada os atributos e comportamentos comuns às duas classes em questão

Herança

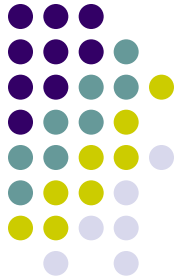


Em Java



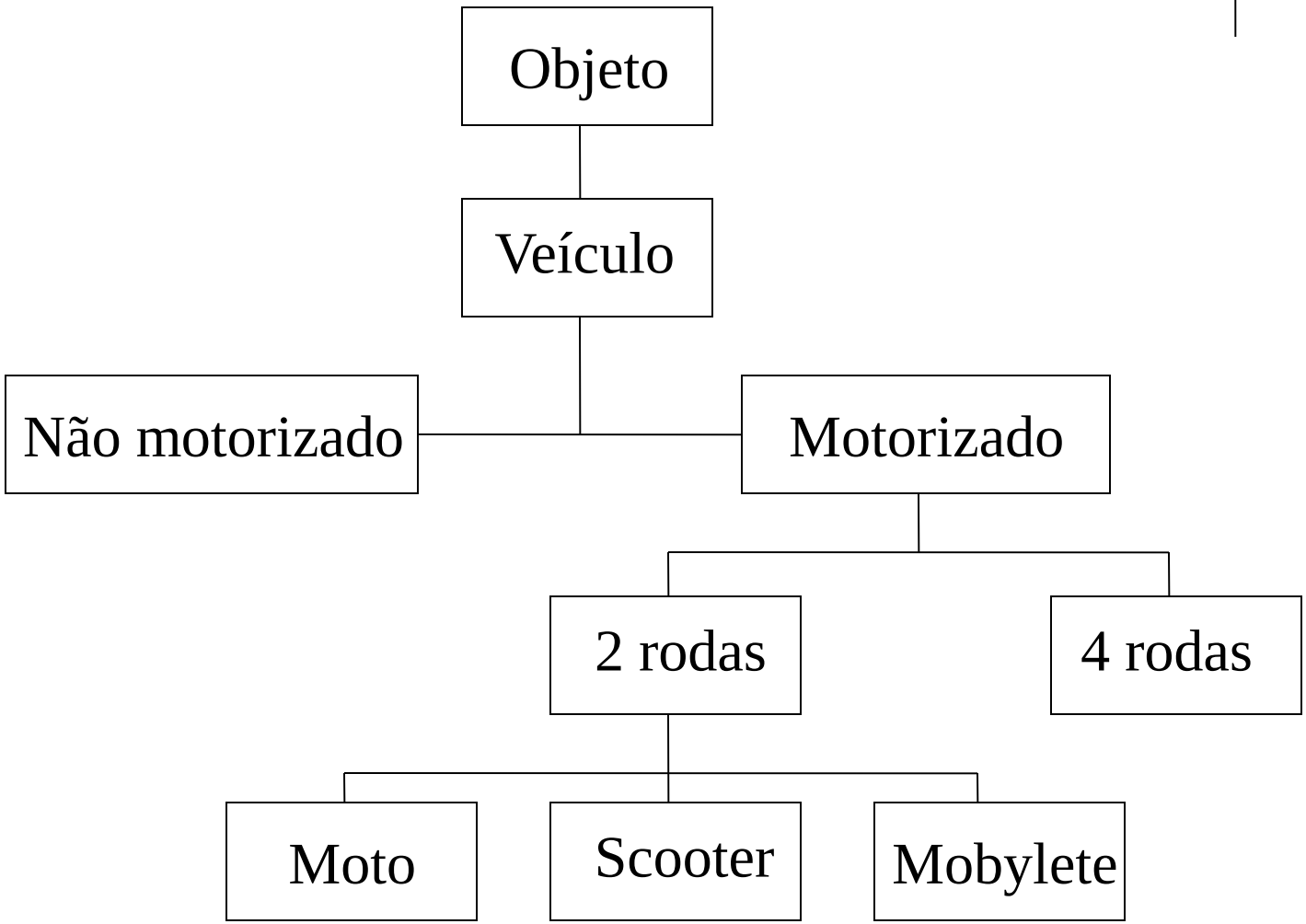
- public class Veiculo{ ... }
- public class Motocicleta extends Veiculo { ... }
- public class Carro extends Veiculo { ... }

Herança

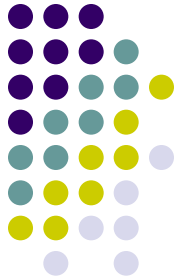


- No entanto, quando se está definindo uma hierarquia de classes deve-se pensar da forma mais ampla possível:
  - Veículo
    - Motorizados
      - carro, caminhão, trator, ônibus, motocicleta
    - Não motorizados
      - bicicleta, charrete, carro de boi

Herança

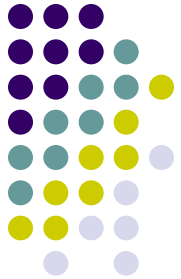

Objeto
Veículo
Não motorizado
Motorizado
2 rodas
4 rodas
Moto
Scooter
Mobylete

Herança



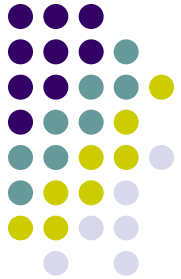
- O mecanismo de herança
  - Como instâncias de uma classe podem ter acesso imediato a variáveis e métodos definidos em classes hierarquicamente superiores?

Herança



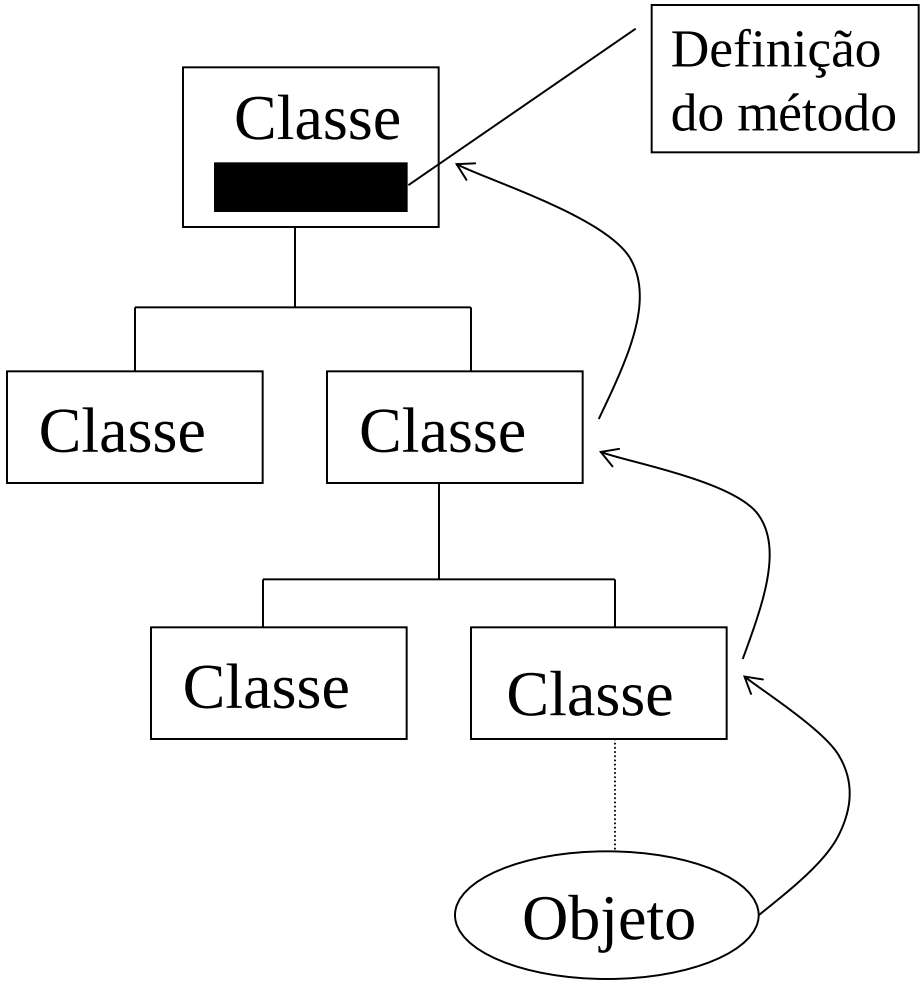
- O mecanismo de herança (cont)
  - Variáveis de instância
    - Quando se cria uma nova instância de uma classe, automaticamente é alocado memória para as variáveis de instância definidas na própria classe e para todas as variáveis de instância das classes hierarquicamente superiores.

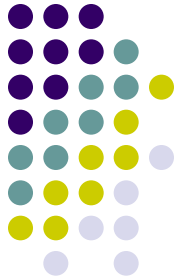
Herança



- O mecanismo de herança (cont)
  - Métodos
    - Similar ao funcionamento de variáveis
    - Um objeto tem acesso aos métodos de sua classe e de suas superclasses
      - Este acesso é dinâmico, isto é, o método é procurado inicialmente na classe que criou o objeto. Se não for encontrado, busca-se o mesmo um nível acima na hierarquia de classe, e assim sucessivamente.

Herança (métodos)



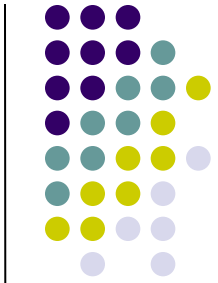


Exercício 1

- Elabore um programa contendo quatro classes:
  - Veículo
  - Carro
  - Motocicleta
  - TestaVeículo
- Considere os seguintes atributos
  - marca
  - cor
  - motorLigado (boolean)
  - estilo: trail, naked, custom
  - portaMalasCheio (boolean)
- Considere as seguintes operações
  - Liga/desliga motor
  - enche/esvazia porta malas
  - mostraAtributos

- A classe testa veículo deve instanciar um carro e uma moto. Deve-se ligar a moto e mostrar seus atributos. Em seguida, deve-se encher o porta malas do carro, ligá-lo e mostrar seus atributos
- Nota: Atributos e operações comuns devem ficar na classe de mais alto nível na hierarquia.

Exercício 2



- Elabore um programa contendo quatro classes:
  - Pessoa
  - Professor
  - Aluno
  - TestaPessoa
- Considere os seguintes atributos
  - Nome
  - CPF
  - Salario
  - Nota
- Considere as seguintes operações
  - AtribuiNota
  - ReajustaSalario
  - MostraAtributos

- A classe TestaPessoa deve instanciar um professor e dois alunos.
- Nota: Atributos em comum devem ficar na classe mais alta.