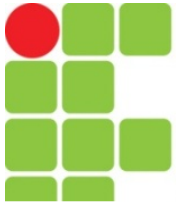


Instituto Federal de Santa Catarina – IFSC
Campus São José

Programação Orientada a Objetos

Paradigmas de Programação

Prof. Francisco de Assis S. Santos, Dr.

São José, 2015.

Paradigmas de Programação: Programação sequencial

- A solução para o problema se da através da execução sequencial de instruções, uma apos a outra;
- Faz uso de desvios incondicionais (GOTO e JUMP);
- Apresenta uma solução rápida para problemas de pequeno porte;
- Não e ideal para problemas de grande porte;
- Dificuldade em organizar o código e o uso de desvios incondicionais pode-se tornar um transtorno;
- Exemplos: Assembly, Basic.

Paradigmas de Programação: Programação estruturada

- Consiste em dividir o problema em partes menores e então apresentar soluções para essas pequenas partes;
- Dividir para conquistar!
- Esta fundamentada sobre as estruturas de sequência, decisão e repetição;
- Desvios condicionais são preferidos a desvios incondicionais;
- A solução de cada pequena parte do problema e feita em procedimentos (ou funções) e a solução de todo problema consiste na invocação destes procedimentos;
- Visa a reutilizacão de código;
- Exemplos: Pascal, C.

Paradigmas de Programação: Programação orientada a objetos

- Surgiu da idéia que todo sistema de software funcionasse como um ser vivo;
    - Cada célula do sistema poderia interagir com outras células, através do envio de mensagens e cada célula consistiria ainda em um sistema autônomo;
- Todo o sistema e visualizado como um conjunto de células interconectadas, denominadas objetos. Cada objeto possui uma tarefa específica e através da comunicação entre os objetos é possível realizar uma tarefa computacional completa;
- Tal paradigma e ideal para o desenvolvimento de software complexos;
    - Extensão do projeto de forma fácil e simplificada.
- Exemplos: Smalltalk, C++, Java, Python.

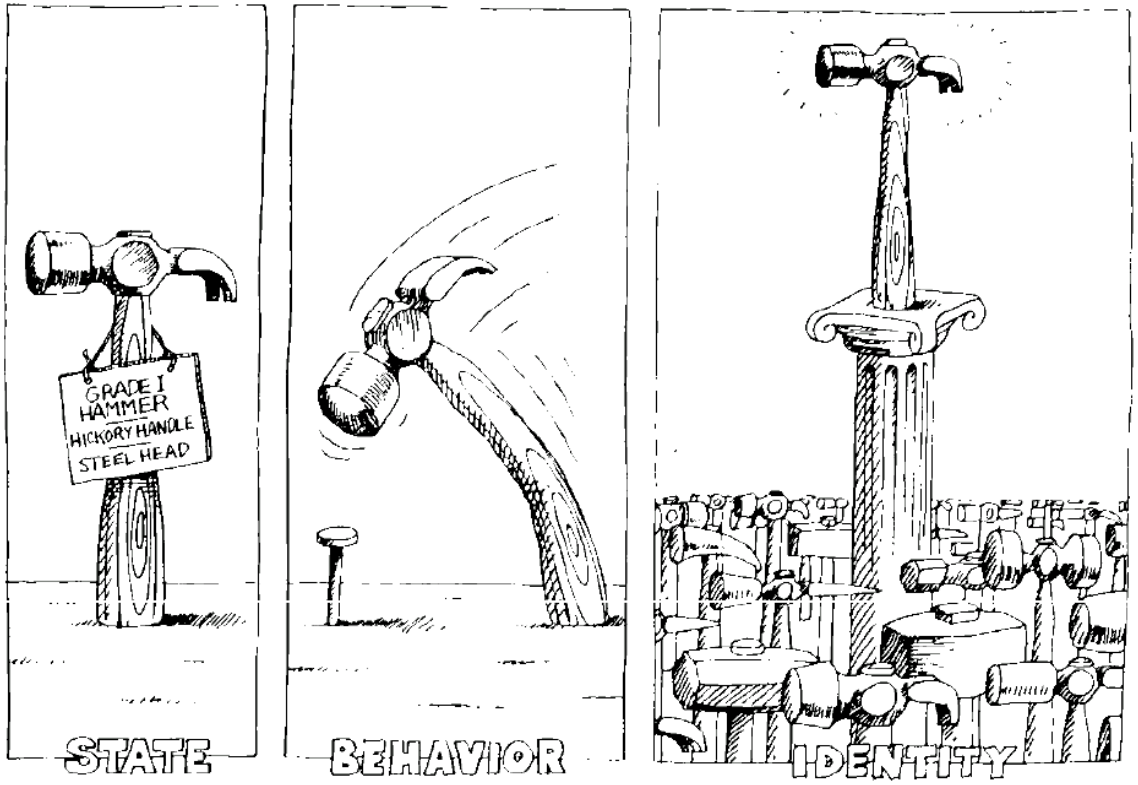
Programação Orientada a Objetos: Conceitos

A Orientação a Objetos fundamenta-se sobre 5 conceitos:

- Classes
- Objetos
- Mensagens
- Herança
- Polimorfismo

Programação Orientada a Objetos: Objetos

Um **objeto** e um item identificável e é composto por **estado** e por **comportamento**.



Programação Orientada a Objetos: Objetos

- **Estado**
  - O estado de um objeto representa as características deste;
  - Um carro possui como características uma cor, modelo, potência, velocidade atual, marcha atual, etc.
- **Comportamento**
  - Representa as operações (métodos) que este objeto e capaz de executar;
  - Um carro pode trocar de marcha, acelerar, frear, etc.

Identificar os estados e comportamentos de objetos do mundo real é um grande passo para se começar a pensar em termos de programação orientada a objetos

Objetos: Domínio do problema

Olhe ao redor e escolha dois objetos. Para estes responda:

- Quais os possíveis estados que este objeto pode assumir?
- Quais os possíveis comportamentos que este objeto pode ter?

- É possível notar diferentes níveis de complexidade de cada objeto
- Por exemplo: lâmpada *versus* computador
- E possível notar que alguns objetos podem conter outros objetos
- Um computador possui um disco rígido, este ultimo por sua vez também e um objeto.

Objetos em Sistemas Computacionais

- **Objetos de software são semelhantes aos objetos reais**
  Um objeto armazena seu estado em atributos e seu comportamento se da através de operações (métodos);

- Em Java, os métodos de um objeto são invocados para realizar uma determinada computação e potencialmente para modificar os atributos deste objeto.

programador: Qual a tua velocidade atual?
objeto carro: 20 km/hora
programador: Diminua a velocidade em 10%
objeto carro: Ok

Máquinas de vender bilhetes – uma visão externa

Explorando o comportamento de uma máquina simples de vender bilhetes:

As máquinas fornecem bilhetes a um preço fixo.

- Como é que o preço é determinado?
- Como o ‘dinheiro’ é inserido em uma máquina?
- Como uma máquina controla o dinheiro que é inserido?

Estrutura de Classe Básica

```
public class TicketMachine
{
    A parte interna da classe omitida.
}
```

O invólucro externo de TicketMachine

```
classe pública NomeDaClasse
{
    Campos
    Construtores
    Métodos
}
```

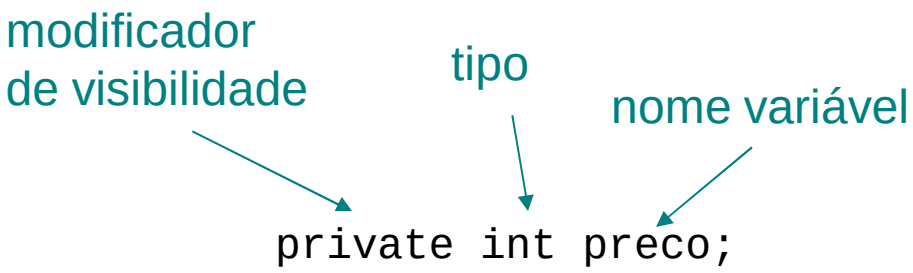
O conteúdo de uma classe

Campos

- Os campos armazenam dados para um objeto.
- Os campos são também conhecidos como variáveis de instância.
- Os campos definem o estado de um objeto.

```
public class TicketMachine
{
    private int preco;
    private int saldo;
    private int total;

    Detalhes adicionais são
    omitidos.
}
```



Métodos de acesso

Os métodos implementam o comportamento de objetos.

Os métodos de acesso fornecem informações sobre um objeto.

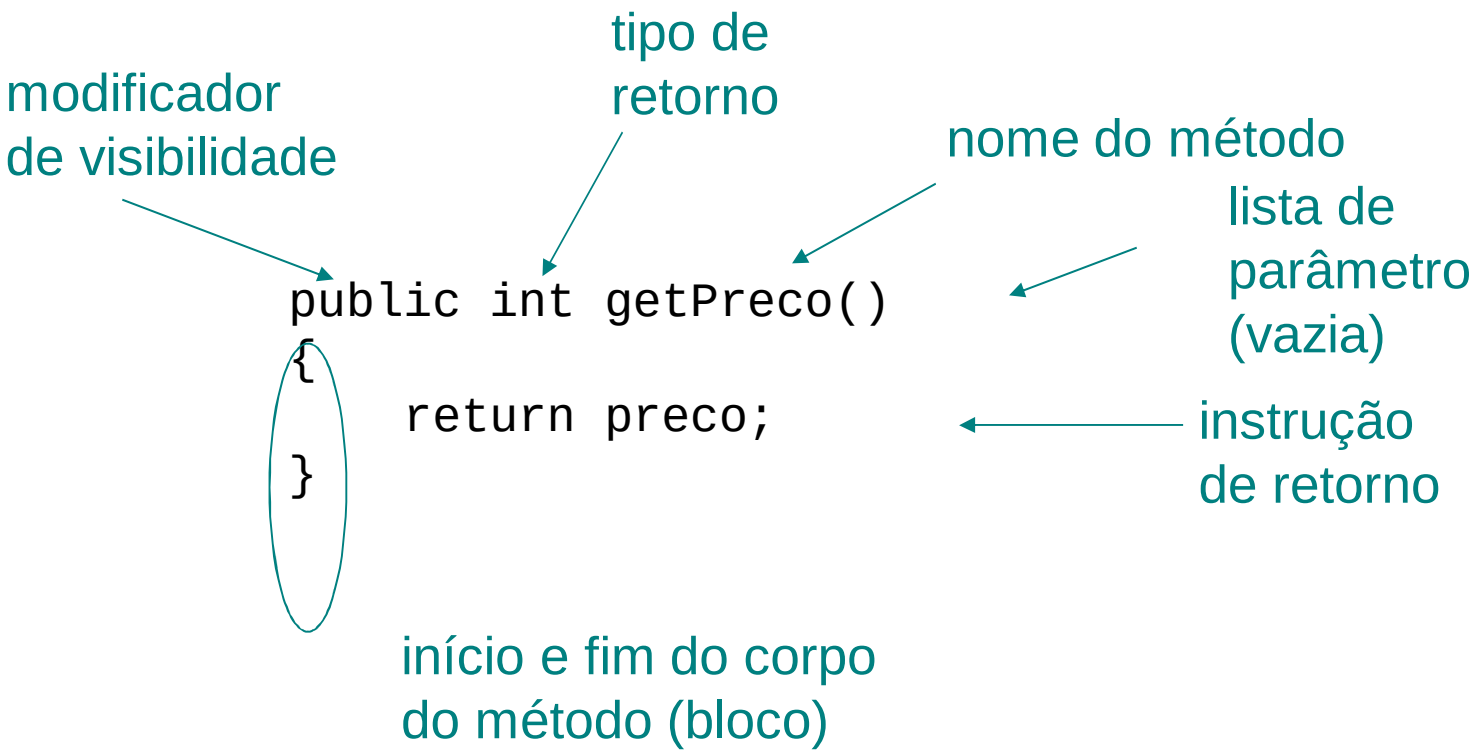
Os métodos têm uma estrutura constituída por um cabeçalho e um corpo.

O cabeçalho define o método da *assinatura*:

```
public int getPreco()
```

O corpo inclui instruções do método.

Métodos de Acesso



Teste

```
public class Machine
{          int
private price;

public Machine()
{
   price = 300   ;
}

public int getPrice   ()
{
   return Price;
}
}
```

- O que está errado aqui?

(há cinco erros!)

Métodos Modificadores

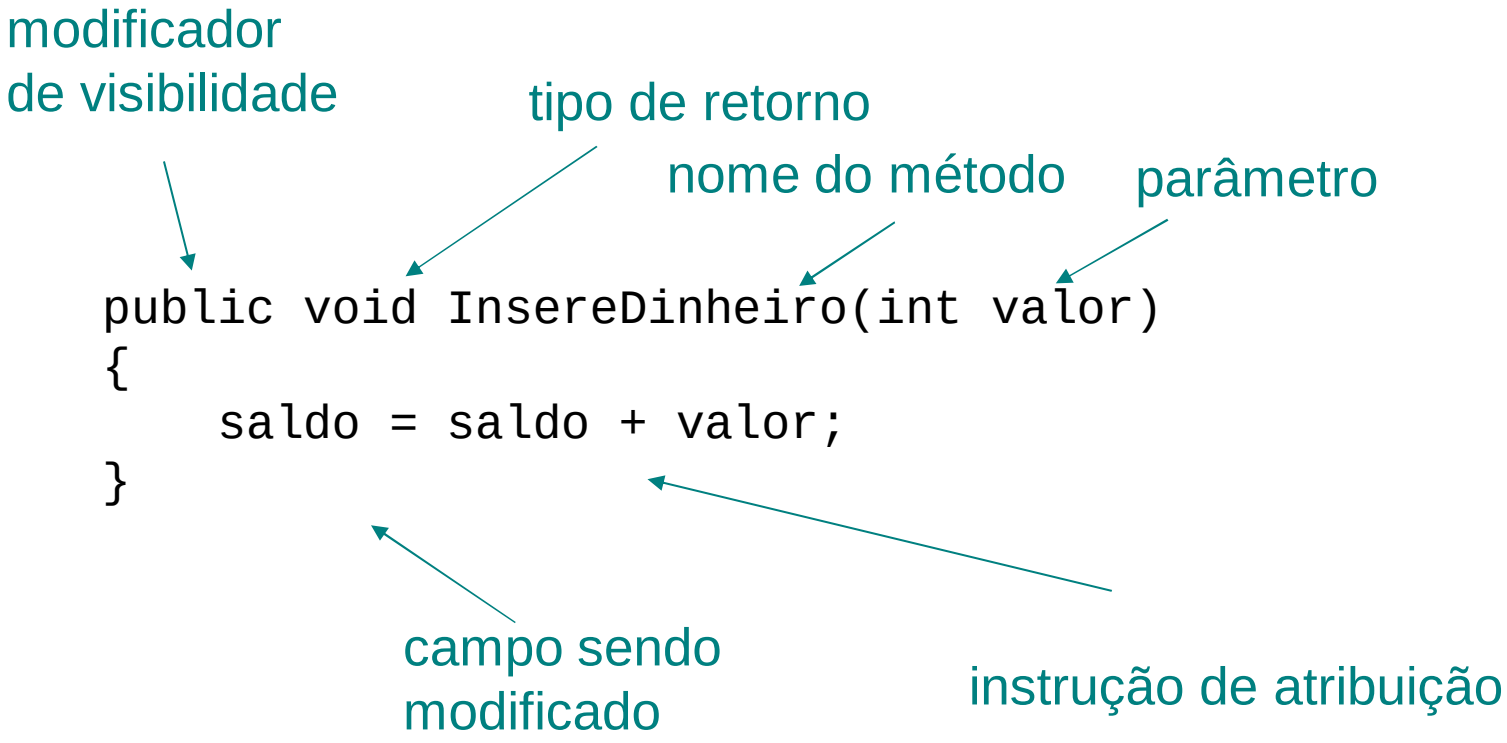
São usados para *modificar* o estado de um objeto.

São alcançados através da mudança de valor de um ou mais campos.

Em geral, contêm instruções de atribuição.

Normalmente, recebem parâmetros.

Métodos Modificadores



Imprimindo a partir de métodos

```
public void printTicket()
{
    // Simule a impressão de um bilhete.
    System.out.println("####################");
    System.out.println("# Ticket");
    System.out.println("# " + preco);
    System.out.println("####################");
    System.out.println();

    // Atualize o total coletado com o saldo.
    total = total + saldo;
    // Limpa o saldo.
    saldo = 0;
}
```

Refletindo sobre as Máquinas de ilhetes

- Sob vários aspectos, seu comportamento é inadequado:
    - Sem verificação sobre as quantias inseridas;
    - Sem restituições;
    - Sem verificação para uma inicialização correta.
- Como podemos melhorar?
    - Precisamos de um comportamento mais sofisticado.

Fazendo escolhas:

```
public void InsertDinheiro(int valor)
{
    if(valor > 0) {
        saldo= saldo+ valor;
    }
    else {
        System.out.println("Use um valor positivo: " +
                                valor);
    }
}
```

Restituição de Saldo

Uma variável local

```
public int RestituicaoSaldo()
{
    int ValorRestituicao;
    ValorRestituicao = saldo;
    saldo = 0;
    return ValorRestituicao;
}
```

Encapsulamento

• Processo de esconder todos os detalhes de um objeto que não contribuem para as suas características essenciais.

Ex: uma caixa preta

• A **interação entre objetos** se da através da troca de **mensagens**;

• O emissor da mensagem não precisa conhecer como o destinatário processara a mensagem, ao emissor só importa receber a resposta.

Exemplo: *System.out.println*("Ola mundo");

Mensagens são compostas por três partes

➢Objeto: *System.out*

➢Nome do método: *println*

➢Parâmetros: "Ola mundo"

Encapsulamento

• O emissor das mensagens precisa saber quais operações o destinatário e capaz de realizar ou quais informações o destinatário pode fornecer;

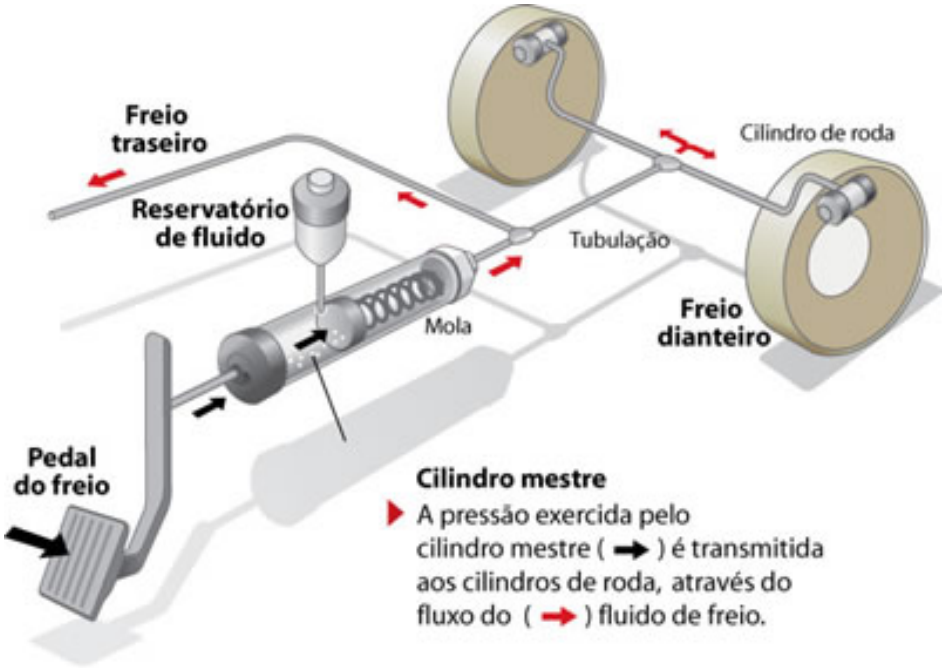
• A interface de um objeto corresponde ao que ele conhece e ao que ele sabe fazer, no entanto sem descrever como ele conhece ou faz;

• Define as mensagens que ele esta apto a receber e responder.

Vantagem do encapsulamento:

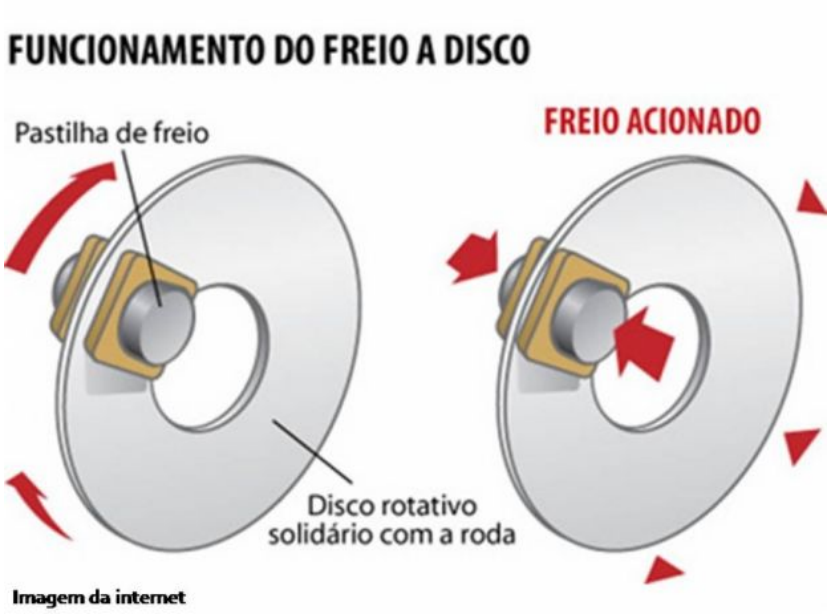
A implementação dentro de uma operação pode ser alterada sem que isso implique na alteração do código do objeto requisitante.

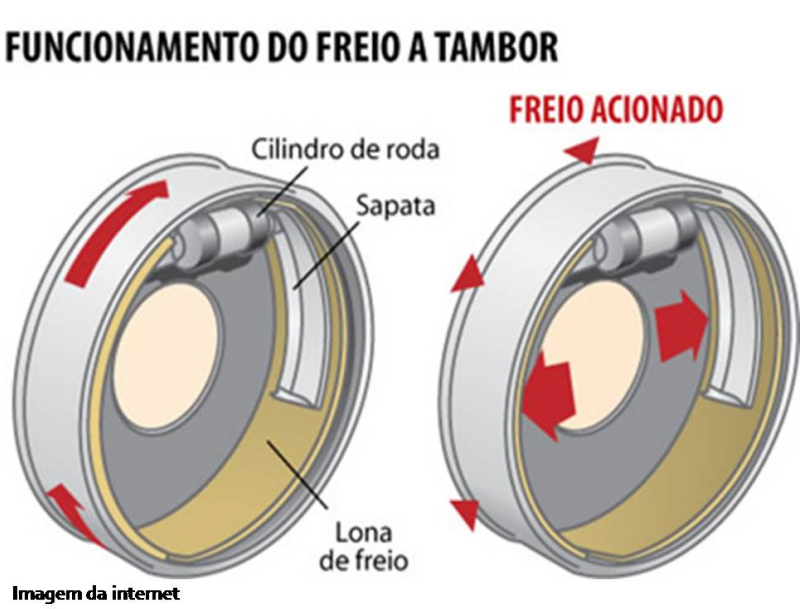
Encapsulamento: Exemplo de sistema de freio hidráulico



- Freios funcionam através de um sistema de pistões e mangueiras por onde circula o fluído de freio;
- Ao pisar no pedal de freio, aciona-se o cilindro mestre que ira pressurizar o fluído;
- Esse fluído transmite a pressão exercida no pedal ate as rodas, acionando o freio.

Encapsulamento: Exemplo de sistema de freio hidráulico



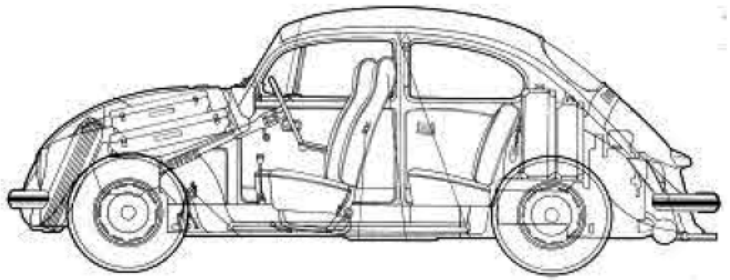


- Como você faz para frear um carro com o sistema de freio a tambor?
- Como você faz para frear um carro com o sistema de freio a disco?

Classes

- Classe e uma planta (projeto) que indica como os objetos deverão ser construídos
- Exemplo: Fusca

  Cada carro e construído com base em um mesmo projeto de engenharia e por consequência todos carros possuirão os mesmos componentes



Classe

Objetos

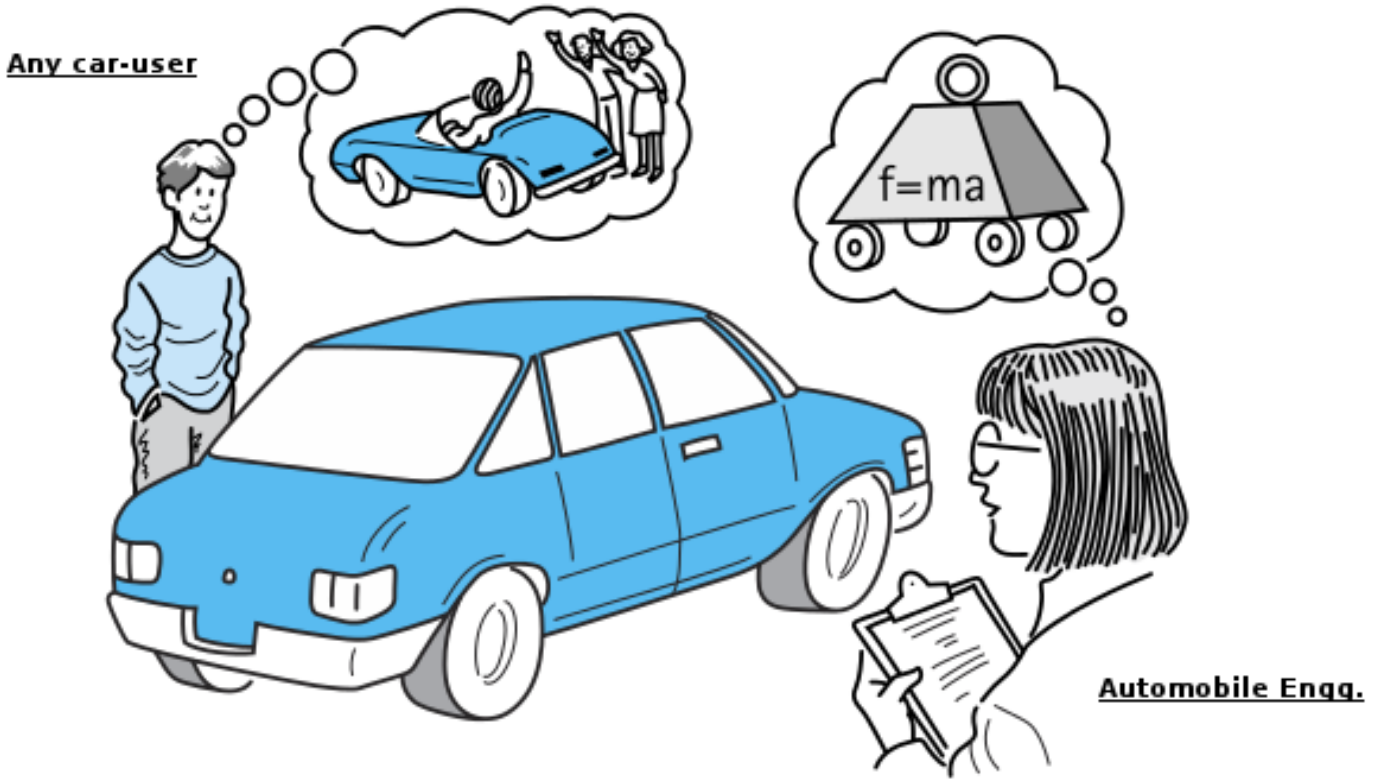
Classes: Exemplo

```
public class Carro{
   // atributos
   private double velocidade;
   private String marca;
   private String modelo;
   // metodos
   public void acelerar(double intensidade){
   ...
   }
   public void frear(double intensidade){
   ...
   }
   public String obterMarca(){
      return marca;
   }
   public void imprimirVelocidade(){
      System.out.println("Velocidade: " + velocidade);
   }
}
```

Abstração

- Trata-se do **processo mental** que nós seres humanos atemos aos aspectos mais relevantes de alguma coisa, ao mesmo tempo que ignoramos os aspectos menos importantes;
- Isso nos permite gerenciar a **complexidade de um objeto**, ao mesmo tempo que concentramos nossa atenção nas características essenciais do mesmo;
- Note que abstração e **dependente do contexto** sobre o qual este algo e analisado;
- O que e importante em um contexto pode não ser importante em outro.

Abstração



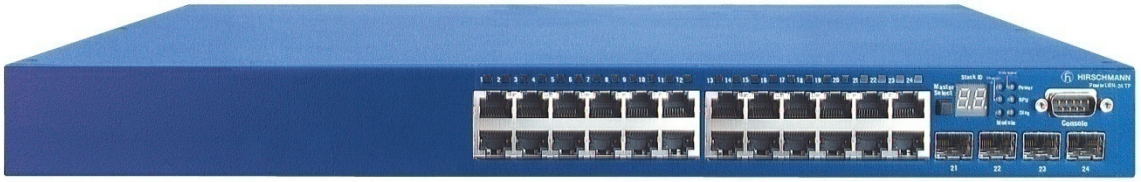
An abstraction includes the essential details relative to the perspective of the viewer

Exercícios

- A classe Contador possui um único atributo:
  - ValorAtual;
- A classe provê métodos para:
  - Atribuir um valor ao contador;
  - Incrementar o contador;
  - Obter o atual valor do contador.

Implemente este contador em Java.

Exercícios



A Figura acima apresenta um ativo de rede switch/comutador

• Pense em um contexto e realize o processo de abstração para coletar as informações essenciais deste objeto para o contexto escolhido;

• Implemente em Java a classe para o comutador e um aplicativo Java (classe *Java* com método *main*). Crie um objeto da classe Comutador e invoque alguns de seus métodos.

Modificadores de Acesso: *public* e *private*

- **Paradigma da POO**
  - Objetos interagem com objetos por meio da troca mensagens
  - A troca de mensagens ocorre por meio da invocação de métodos de objetos
- **Encapsulamento**
  - Emissor da mensagem não precisa saber como o resultado foi obtido, para este só importa o resultado
  - O emissor precisa conhecer quais operações o receptor sabe realizar ou quais informações o receptor pode fornecer
- **Modificadores de Acesso**
  - Indicam quais atributos e métodos de um objeto estarão visíveis aos demais objetos do sistema

Modificadores de Acesso: *public* e *private*

private Os membros de uma classe (atributos e métodos) definidos como privados só poderão ser acessados pelos demais métodos da própria classe

public Os membros de uma classe definidos como públicos poderão ser invocados por métodos de qualquer classe

Princípios da POO

- Geralmente **atributos** de uma classe **devem ser** declarados como **privados**
- **Métodos** geralmente devem ser públicos, porem ha casos que um método só interessa a própria classe e assim este deve ser privado
- Isto garante a **integridade** do **estado do objeto**, pois somente métodos da própria classe poderão alterá-lo

Modificadores de Acesso: *public* e *private*

private Os membros de uma classe (atributos e métodos) definidos como privados só poderão ser acessados pelos demais métodos da própria classe

public Os membros de uma classe definidos como públicos poderão ser invocados por métodos de qualquer classe

Princípios da POO

- Geralmente **atributos** de uma classe **devem ser** declarados como **privados**
- **Métodos** geralmente devem ser públicos, porem ha casos que um método só interessa a própria classe e assim este deve ser privado
- Isto garante a **integridade** do **estado do objeto**, pois somente métodos da própria classe poderão alterá-lo

```
public class CarroNaoIdeal{
   // atributos
   public float velocidade;
   // metodos
   public void definirVelocidade(float v){
      if (v <= 200){
         velocidade = v;
      } else velocidade = 0;
   }
   public void acelerar(float v){
      // o carro so' pode atingir 200km/h
      if ((velocidade + v) <= 200){
         velocidade += v;
      }else{
         velocidade = 200;
      }
   }
}
```

Modificadores de Acesso: *public* e *private*

```
public static void main(String args[]){
   CarroNaoIdeal fusca = new CarroNaoIdeal();

   // alterando a velocidade atraves dos metodos do objeto
   fusca.definirVelocidade(150);// velocidade = 150
   fusca.acelerar(400); // velocidade = 200

   // alterando diretamente o valor do atributo
   fusca.velocidade = 400;
}
```

```
public class CarroIdeal{
   // atributos
   private float velocidade;
   // metodos
   public void definirVelocidade(float v){
      if (v <= 200){
         velocidade = v;
      } else velocidade = 0;
   }
   public void acelerar(float v){
      // o carro so' pode atingir 200km/h
      if ((velocidade + v) <= 200){
         velocidade += v;
      }else{
         velocidade = 200;
      }
   }
}
```

Modificadores de Acesso: *public* e *private*

```
public static void main(String args[]){
   CarroIdeal fusca = new CarroIdeal();

   // alterando a velocidade atraves dos metodos do objeto
   fusca.definirVelocidade(150);// velocidade = 150
   fusca.acelerar(400); // velocidade = 200

   // alterando diretamente o valor do atributo
   fusca.velocidade = 400; // ERRO ! nao ira' compilar
}
```

Valores Iniciais de Atributos

```
public class Pessoa{
   private String nome;
   private String cpf;
   private int anoNasc;

   public void imprimirDados(){
    System.out.println("Nome: " + nome);
    System.out.println("CPF: " + cpf);
    System.out.println("Ano: " + anoNasc);
   }
}// fim da classe
```

```
Pessoa p = new Pessoa();
p.imprimirDados();
```

- O que será impresso?

```
15  Nome:
16  CPF:
17  Ano: 0
```

Valores Iniciais de Atributos

- Em Java atributos de um objeto que não forem iniciados na criação deste objeto, receberão valores padrões
- Números ficam 0, *boolean* com *false* e referências de objetos com null

Recomenda-se iniciar os atributos de forma explícita

```
Pessoa p = new Pessoa();

p.definirNome("Joao");
p.definirCPF("123.456.789-00");
p.definirAno(1950);
```

Método Construtor

- Trata-se de um método especial cujo objetivo é iniciar com valores os atributos de um objeto
- O método possui o mesmo nome da classe e não possui tipo de retorno
- Uma classe pode conter métodos construtores sobrecarregados
- Ao criar um objeto o desenvolvedor indica qual construtor irá chamar

Método **construtor padrão** e aquele cuja de lista de parâmetros esta vazia. Toda classe Java possui um construtor padrão vazio implícito.

Método Construtor

```
public class Pessoa{
   private String nome, cpf;
   private int anoNasc;

   // metodo construtor padrao
   public Pessoa(){
      nome = ""; cpf = ""; anoNasc = 0;
   }

   // metodo construtor com 1 parametro
   public Pessoa(String no){
      nome = no; cpf = ""; anoNasc = 0;
   }

   // metodo construtor com 3 parametros
   public Pessoa(String no, String c, int a){
      nome = no; cpf = c; anoNasc = a;
   }
}// fim da classe
```

Método Construtor: Invocação

```
Pessoa a = new Pessoa();
Pessoa b = new Pessoa("Maria");
Pessoa c = new Pessoa("Maria","123.456.789-00",1959);
```

Exercícios

1 - Implemente em Java uma Classe para representar um número complexo x = (a; b)

- Crie métodos construtores que permitam iniciar os atributos dessa classe.
- Crie os métodos soma e subtração. Estes devem receber um numero Complexo como parâmetro e somá-lo ou subtraí-lo com o objeto em questão.
  - Soma: x + y = (a + c, b + d)
  - Subtração: x - y = (a – c, b - d)
- Crie um método para imprimir o número complexo na forma (a; b), sendo a a parte real e b a parte imaginaria
- Instancie dois objetos da classe Complexo e invoque alguns de seus métodos.

Exercícios

2 - Classe Data para representar uma data (dia, mês e ano)

Crie uma classe em Java para representar uma data

Escreva um programa Java, instancie um objeto da classe Data e invoque alguns de seus métodos

A classe deverá prover os seguintes métodos:

- Construtor padrão
- Construtor para iniciar todos os atributos da classe
- imprimir - Devera imprimir o valor no dispositivo de saída padrão. Ex: 02/09/2013
- imprimirPorExtenso - Devera imprimir o valor por extenso. Ex: dois de setembro de 2013.
- denirDia - recebe um valor com parâmetro e armazena na classe
- denirMes - recebe um valor com parâmetro e armazena na classe
- denirAno - recebe um valor com parâmetro e armazena na classe

Exercícios

3 - Crie uma classe em Java para representar um valor em Reais (moeda brasileira). Escreva um programa Java, instancie um objeto da classe Moeda e invoque alguns de seus métodos.

• O maior valor permitido e R$ 1.000, 00 e o menor é R$ -1.000, 00
Escreva um programa Java, instancie um objeto da classe Data e invoque alguns de seus métodos

• A classe devera prover os seguintes métodos:
• Construtor padrão
• Construtor para iniciar todos os atributos da classe
• imprimir - Devera imprimir o valor no dispositivo de saída padrão. Ex:
R$ 123,45
• imprimirPorExtenso - Devera imprimir o valor por extenso. Ex: Cento e vinte e três reais e quarenta e cinco centavos
• obterValor - retorna o valor armazenado
• denirValor - recebe um valor com parâmetro e armazena na classe

Referências

Notas de aula do Prof. Emerson Ribeiro de Mello