Instituto de Computação

Orientação a Objetos (em Java)

Viviane Torres da Silva
viviane.silva@ic.uff.br

http://www.ic.uff.br/~viviane.silva/2010.2/es1

Programação Estruturada

- Composição dos Programas
  - Um programa é composto por um conjunto de rotinas
  - A funcionalidade do programa é separada em rotinas
  - Os dados do programa são variáveis locais ou globais

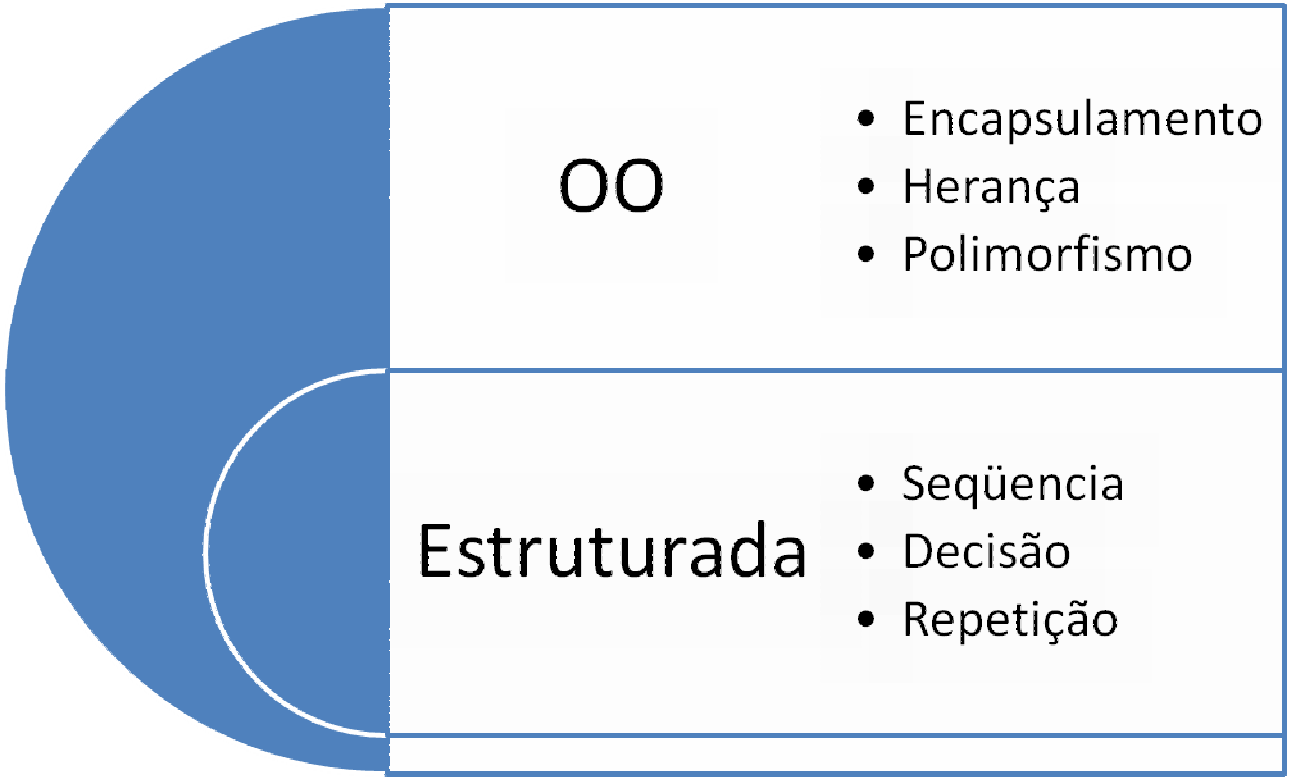
- Fluxo de Execução
  - O programa tem início em uma rotina principal
  - A rotina principal chama outras rotinas
  - Estas rotinas podem chamar outras rotinas, sucessivamente
  - Ao fim de uma rotina, o programa retorna para a chamadora

Programação Orientada a Objetos

- Composição do programa
  - A funcionalidade do programa é agrupada em objetos
  - Os dados do programa são agrupados em objetos
  - Os objetos agrupam dados e funções correlacionados

- Fluxo de Execução
  - Similar ao anterior
  - Os objetos colaboram entre si para a solução dos objetivos
  - A colaboração se realiza através de chamadas de rotinas

Programação Estruturada x OO



Objetos

- Definição
  - Um objeto é a representação computacional de um elemento ou processo do mundo real
  - Pode representar uma entidade concreta (computador, mesa, cadeira) ou entidade conceitual (estratégia de um jogo, política de escalonamento de requisições)
  - Cada objeto possui suas características e seu comportamento

- Exemplos de Objetos

cadeira	*mesa*	*caneta*	*lápis*
carro	*piloto*	*venda*	*mercadoria*
cliente	*aula*	*programa*	*computador*
aluno	*avião*		

Características de Objetos

- Definição
  - Uma característica descreve uma propriedade de um objeto, ou seja, algum elemento que descreva o objeto.
  - Cada característica é chamada de **atributo** do objeto

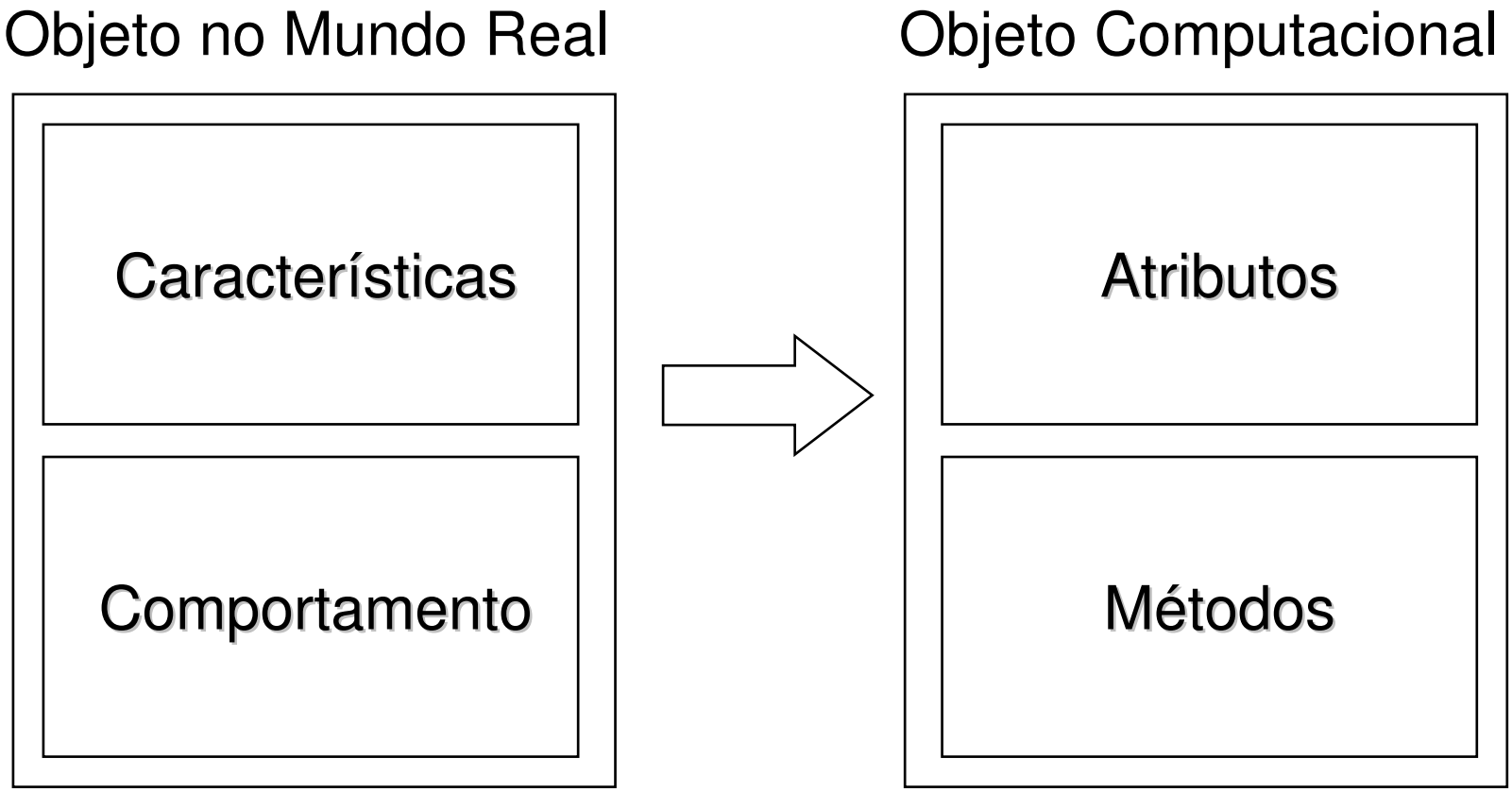
- Exemplo de características do objeto **carro**
  - Cor
  - Marca
  - Ano de fabricação
  - Tipo de combustível

Comportamento de Objetos

- Definição
  - Um comportamento representa uma ação ou resposta de um objeto a uma ação do mundo real
  - Cada comportamento é chamado de **método** do objeto

- Exemplos de comportamento para o objeto **carro**
  - Mudar velocidade
  - Parar
  - Mudar marcha

Mapeamento de Objetos



Classes

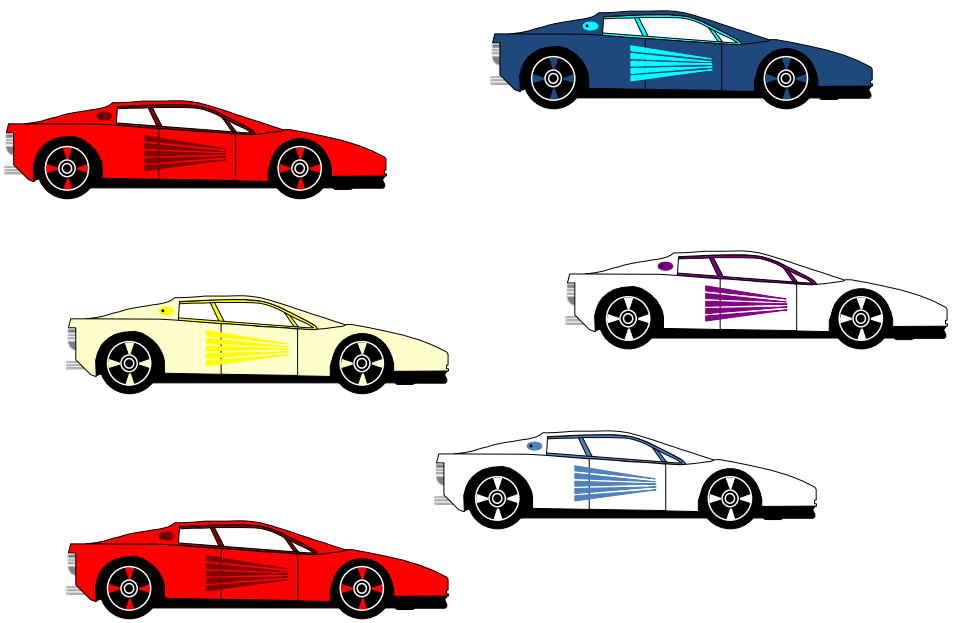
- Objetos com as mesmas características e o mesmo comportamento são originários de uma mesma **classe**
- Uma classe representa uma fôrma para a criação de objetos similares
- Cada objeto é uma **instância** de uma classe
- Cada instância = objeto possui seus próprios valores para as características = atributos

Classes

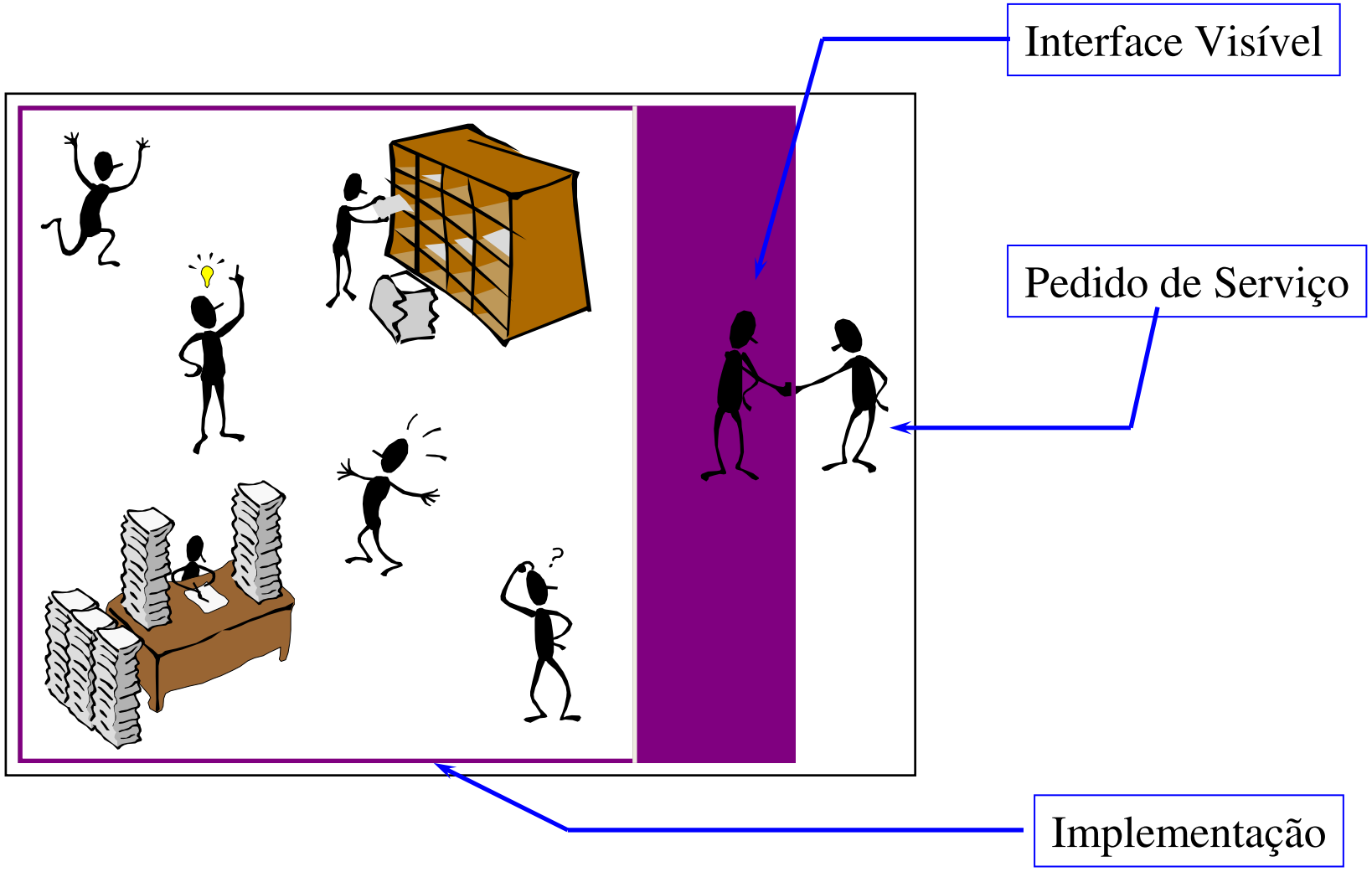
- Cada objeto é instância de uma única classe

- A classe é o bloco básico para a construção de programas OO

Carro
Cor Marca Ano Velocidade Marcha
Muda marcha Muda velocidade Para

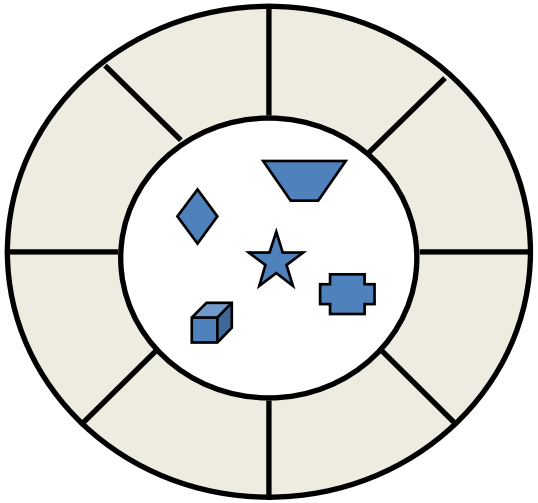


Encapsulamento



Encapsulamento

- Atributos e Métodos
  - Os métodos formam uma “cerca” em torno dos atributos
  - Os atributos não podem ser manipulados diretamente
  - Os atributos somente podem ser alterados ou consultados através dos métodos do objeto



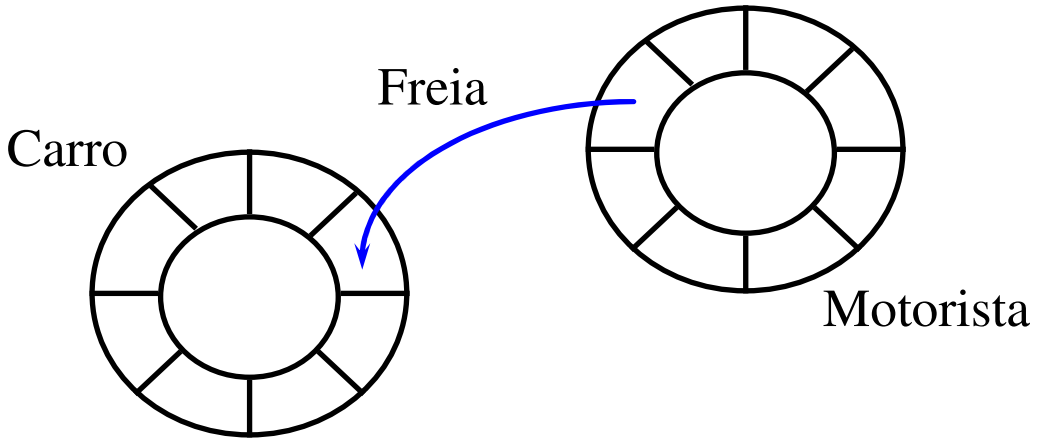
Benefícios do Encapsulamento

- Chamamos de clientes de um objeto X aos outros objetos que utilizam métodos de X

- Pelo encapsulamento:
  - Clientes de um objeto podem utilizar seus métodos sem conhecer os detalhes de sua implementação
  - A implementação de um objeto pode ser alterada sem o conhecimento de seus clientes, desde que a interface visível seja mantida

Mensagens

- Colaboração
  - Um programa OO é um conjunto de objetos que colaboram entre si para a solução de um problema
  - Objetos colaboram através de trocas de mensagens
  - A troca de mensagem representa a chamada de um método



Mensagens

- Um envio de mensagem sempre possui:
  - Um emissor
  - Um receptor
  - Um seletor de mensagens (nome do método chamado)
  - Parâmetros (opcionais)

- Uma mensagem pode retornar um valor

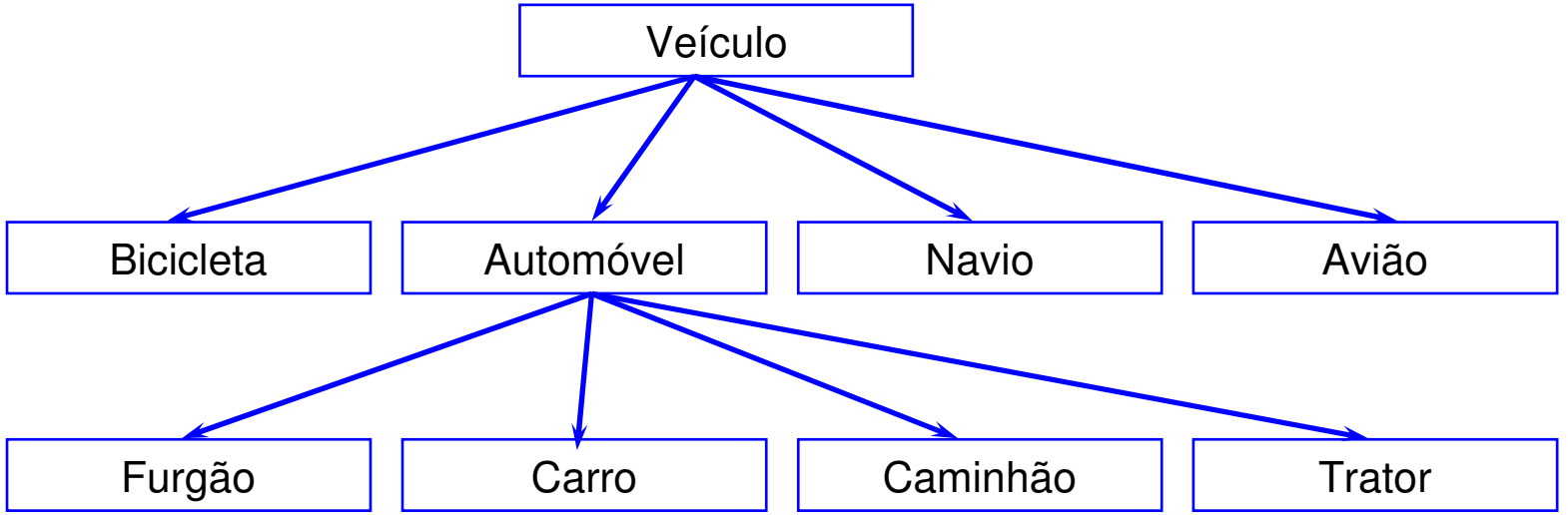
Métodos Especiais

- Criação de Objetos
  - A classe é responsável pela criação de seus objetos
  - Esta criação é realizada através de um método especial, chamado de **construtor**

- Eliminação de Objetos
  - A classe é responsável pela eliminação de seus objetos, quando eles não podem mais ser utilizados pelo sistema
  - Esta eliminação é realizada por um método especial, chamado de **destrutor**

Herança

- Classes são organizadas em estruturas hierárquicas
  - Uma classe pode herdar características e comportamento de outras classes
  - A classe que forneceu os elementos herdados é chamada de **superclasse**
  - A classe herdeira é chamada de **subclasse**
  - A subclasse herda todos os métodos e atributos de suas superclasses
  - A subclasse pode definir novos atributos e métodos específicos



Polimorfismo

- Uma subclasse pode redefinir um elemento herdado
  - Este mecanismo é chamado de **polimorfismo**
  - Normalmente se aplica sobre o comportamento herdado
  - O polimorfismo se realiza através da recodificação de um ou mais métodos herdados por uma subclasse

- Polimorfismo em Java
  - Todos os métodos herdados podem ser redefinidos em uma subclasse (exceto métodos *finais*)

Interfaces

- Extensão Java para complementar a herança simples
  - Uma interface define um protocolo
  - Classes de objetos podem suportar uma ou mais interfaces
  - Suportar uma interface implica em implementar o protocolo

- Protocolo
  - Um protocolo é composto de um conjunto de métodos
  - Os métodos do protocolo são abstratos
  - Os métodos do protocolo devem ser codificados nas classes que implementam a interface

Interfaces

- Uma interface é um contrato assinado por uma classe
  - A interface define as responsabilidades da classe
  - As responsabilidades são mapeadas em métodos abstratos
  - A classe que implementa a interface implementa os métodos

- Métodos Abstratos (*abstract*)
  - Não possuem implementação
  - Apenas definem um protocolo
  - São implementados em subclasses

Orientação a Objetos em Java

Introdução

- Orientada a Objetos:
  - Java é “pura” e possui grande diversidade de bibliotecas de classes;
- Simples:
  - Java é mais simples que outras linguagens OO, como C++, e possui facilidades como “Garbage Collector”;
- Distribuída:
  - Suporta aplicações em rede, objetos distribuídos e threads;

Introdução

- Independente de Plataforma:
  - Java é interpretada, podendo rodar em qualquer plataforma (JVMs);
- Robusta:
  - Java suporta o tratamento de exceções;
- Performance:
  - Mais rápida que linguagens script;
  - Passível de compilação just-in-time;

Introdução

- Java reforça bons padrões de programação:
  - Orientação a Objetos;
  - Reutilização;
  - Utilização de componentes de software (*JavaBeans*);
  - Desenvolvimento de componentes de software (EJB);
  - Modelo de comentário (*JavaDoc*);
  - Geração semi-automática de documentação;

Introdução (programa exemplo)

- Passos para criar uma aplicação Java:
  - Edição do código fonte;
  - Compilação;
  - Execução via interpretador;

```
// Meu primeiro programa Java
class AloMundo
{
        public static void main(String[] args)
        {
                System.out.println("Alô Mundo!");
        }
}
```

Introdução (rotina principal)

- A rotina *main*
  - A rotina principal determina o início do programa Java;
  - Um programa pode ser composto por diversas classes com diferentes rotinas principais;
  - O método *main* possui o seguinte formato:

    public static void main(String[] args)
  - O parâmetro **args** indica os argumentos do programa;
  - Os argumentos são as palavras da linha de comando;

Introdução (pacotes)

- São utilizados para agregar classes relacionadas;
- O pacote de uma classe é indicado na declaração *package*;
- Esta declaração normalmente se encontra no início do arquivo onde a classe é declarada;

```
package acervo;

public class Livro {...}
```

- Se uma classe não declara seu pacote, o interpretador assume que a classe pertence a um pacote *default*;

Introdução (pacotes)

- Modificadores permitem que determinadas classes sejam visíveis apenas para outras classes do mesmo pacote

- Classes públicas:
  - São visíveis em todos os pacotes

- Classes sem modificador:
  - São visíveis apenas no pacote onde foram declaradas

- Para visualizar uma classe de outro pacote:
  - Utilizar **import** seguido do nome da classe

```
import acervo.Livro;

class Biblioteca {…}
```

Introdução (pacotes)

- API's Java:
  - As diversas API's Java organizam suas classes em pacotes;
  - Cada classe de cada API está em apenas um pacote;
  - Uma classe pode utilizar diretamente os serviços de outra classe do mesmo pacote;
  - Se uma classe deseja utilizar os serviços de uma classe de outro pacote, ela deve fazer sua importação;

Introdução (pacotes)

- ***Import***
  - A importação se realiza através da palavra-chave ***import***, seguida do nome das classes desejadas;
  - As importações são apresentadas antes da declaração da classe;

```
import java.util.*;                 // Importa todas as classes do pacote JAVA.UTIL
import java.swing.JFrame; // Importa a classe JAVA.SWING.JFRAME

public class QualquerClasse
{
        < métodos que utilizem as classes importadas >
}
```

Introdução (pacotes)

- Principais pacotes da API Java SE:
  - java.lang.*: pacote fundamental da linguagem;
  - java.math.*: rotinas matemáticas para números grandes;
  - java.net.*: acesso a rede;
  - java.sql.*: acesso a SGBD;
  - java.io.*: entrada e saída de dados;
  - java.util.*: classes complementares (vetores, tabelas, …);
  - javax.swing.*: interface com o usuário;
  - etc;

Classes em Java

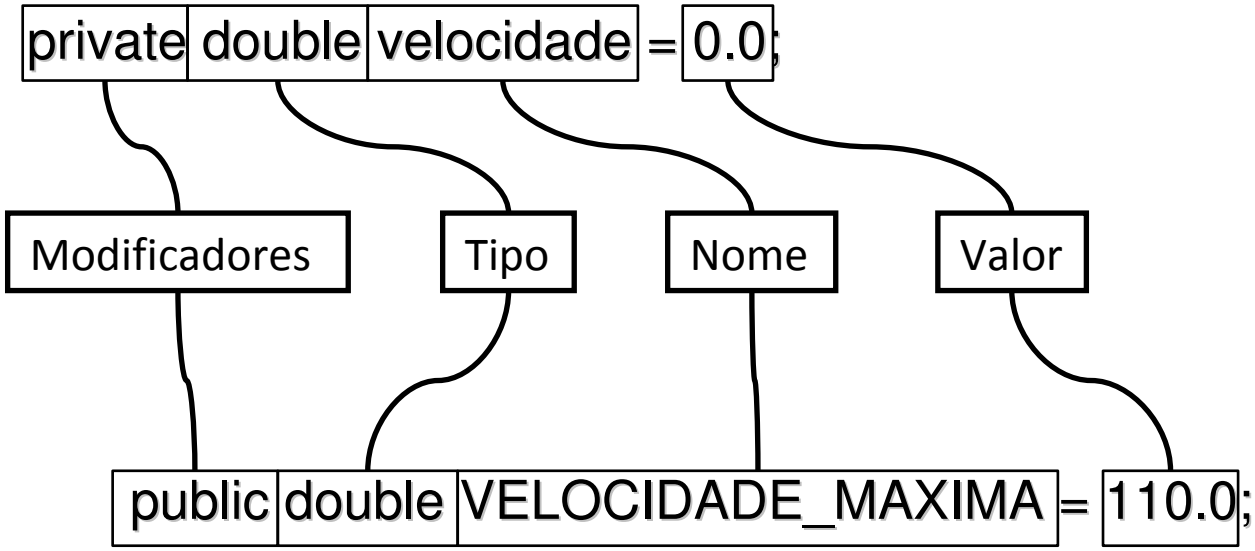
```
public class Ferrari extends Carro implements Taxavel
{
        <atributos da Ferrari>
        <métodos da Ferrari>
        <métodos redefinidos de Carro>
        <métodos da interface Taxavel>
}
```

Interface em Java

- A definição de uma interface é similar a de uma classe:
  - Utilizamos a palavra reservada *interface*;
  - A palavra reservada deve ser seguida do nome da interface;
  - Uma interface pode herdar de outras interfaces (*extends*);
  - A interface possui apenas métodos abstratos e constantes;

```
public interface Taxavel
{
        public final static int ANO_INICIO = 1996;
        abstract double calculaTaxa();
}
```

Atributos em Java (exemplos)



Atributos em Java (tipos)

- Os tipos da linguagem Java são utilizados:
  - Na declaração de atributos;
  - No tipo de retorno de um método;
  - Na lista de parâmetros de um método;
  - Nas variáveis locais de um método;
- A linguagem Java suporta:
  - Tipos primitivos;
  - Arrays;
  - Classes e interfaces;

Atributos em Java (tipos primitivos)

- Inteiros:

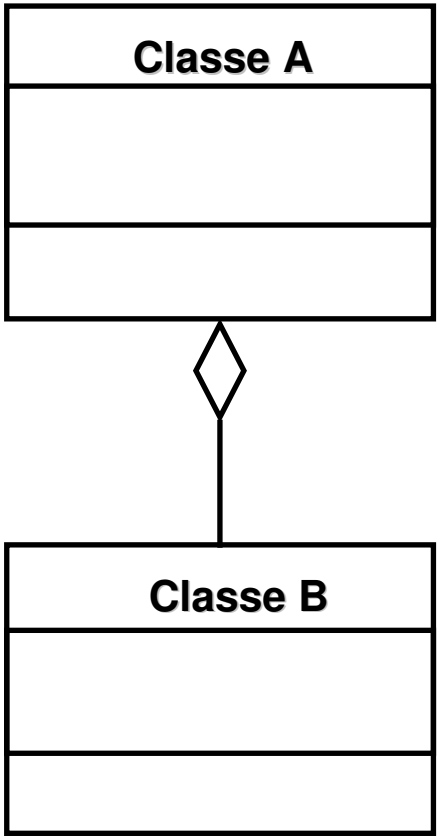
  byte: 8-bits; short: 16-bits;

  int: 32-bits; long: 64-bit;

- Números Reais:
  - float: precisão simples 32-bits (IEEE 754 SPFP);
  - double: precisão dupla 64-bits (IEEE 754 DPFP);

- Outros:
  - char: caractere 16-bit (Unicode);
  - boolean: pode receber dois valores (true ou false);

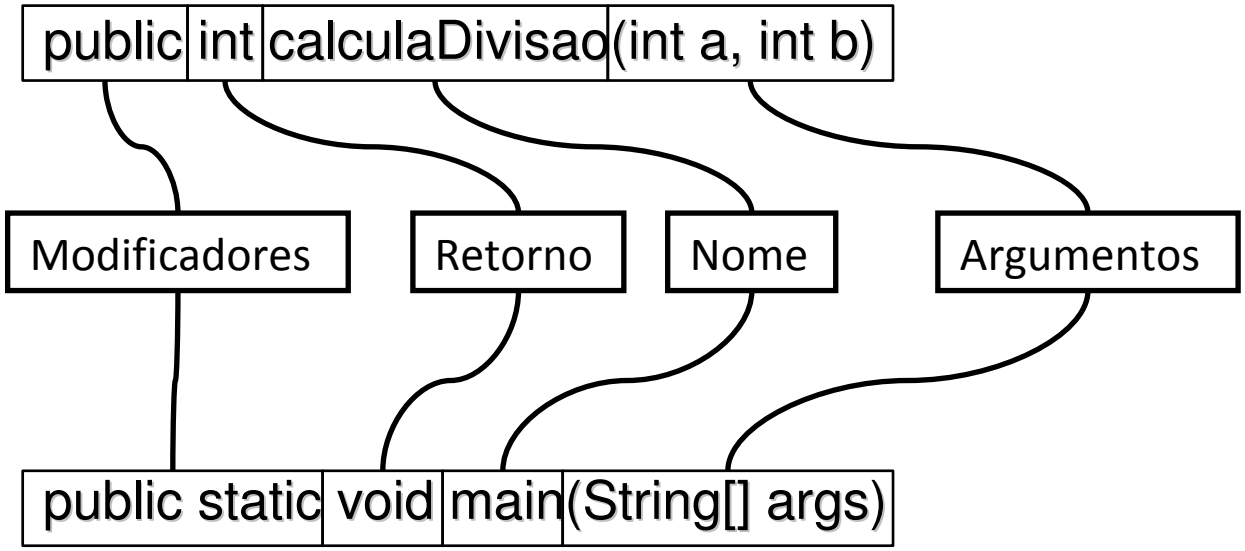
Atributos em Java (associações)




```
class A
{
        private B b;
        …
}

class B
{
        private A a;
        …
}
```

Métodos em Java (exemplos)



Código em Java (exemplos)

```
int contador, x, y;
double  valor;
...
contador = 0;
contador++;
valor = 20.0 * x + Math.exp(y);
...
(x > 10)
!(valor <= 0)
(x > 0)  && (x < 100)
(y == 1)  ||  (y == 2)
```

Declaração de Variáveis

Expressões de Cálculo

Expressões de Controle

Código em Java

- O código em Java é colocado no interior dos métodos;
- O código de um método Java:
  - Muito similar ao código de um método em C++;
  - Apresentado logo após o cabeçalho do método;
  - Delimitado por um par de chaves;
- Principais diferenças em relação a C++:
  - Não existe aritmética de ponteiros;
  - Objetos não precisam ser liberados (“*garbage collection*”);

Código em Java (fluxo)

- Blocos de Comandos;
- Decisões:
  - if-else;
  - switch-case;
- Repetições:
  - while;
  - do-while;
  - for;

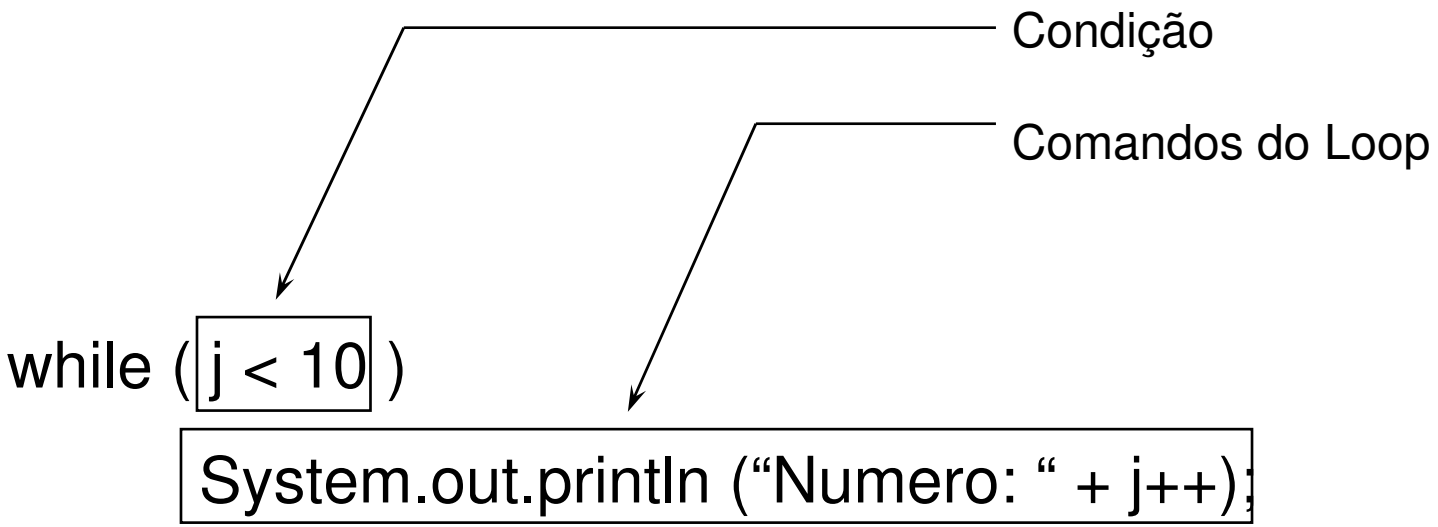
Código em Java (decisões – if-else)

```
if (x > 0) {
        x = x + 10;
        System.out.println (“x foi acrescido de 10”);
}


if (y < 10  &&  y > 0)
        System.out.println (“Y está entre 0 e 10”);
else
        System.out.println (“Y fora do intervalo 0-10”);
```

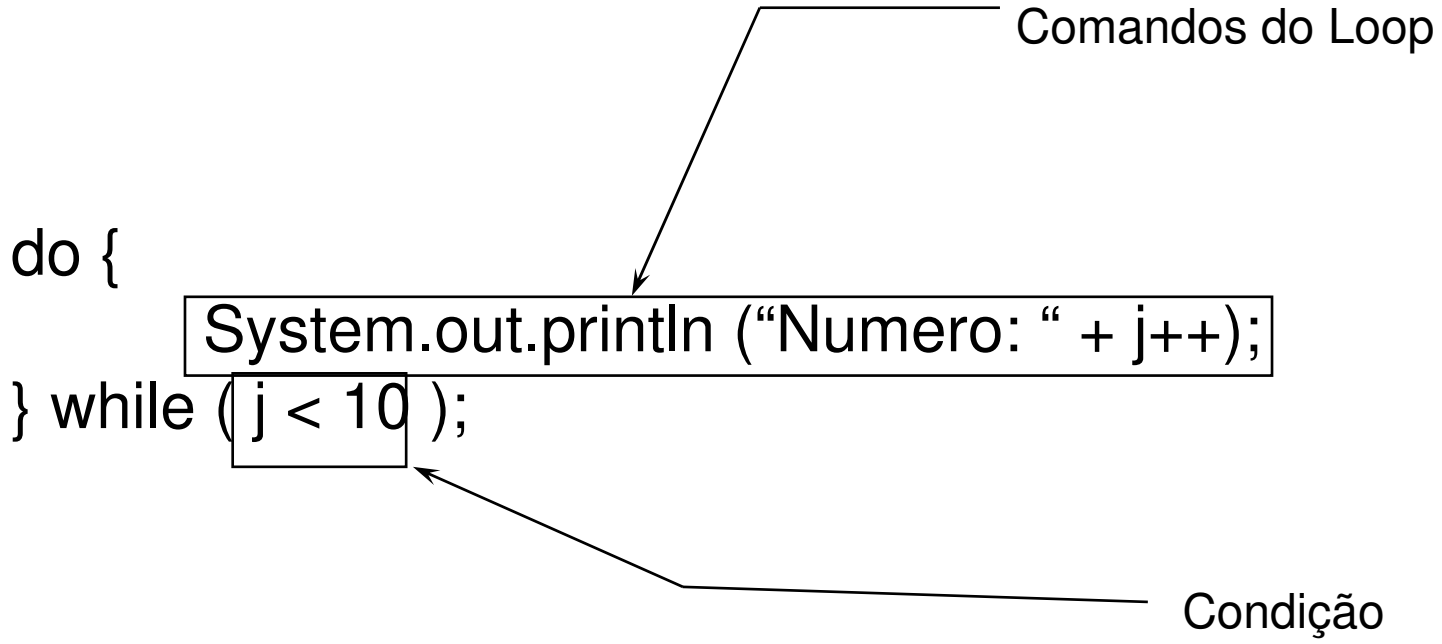
Código em Java (repetições – while)

- Executa um bloco de comandos enquanto uma condição for verdadeira. A condição é testada no início do loop.



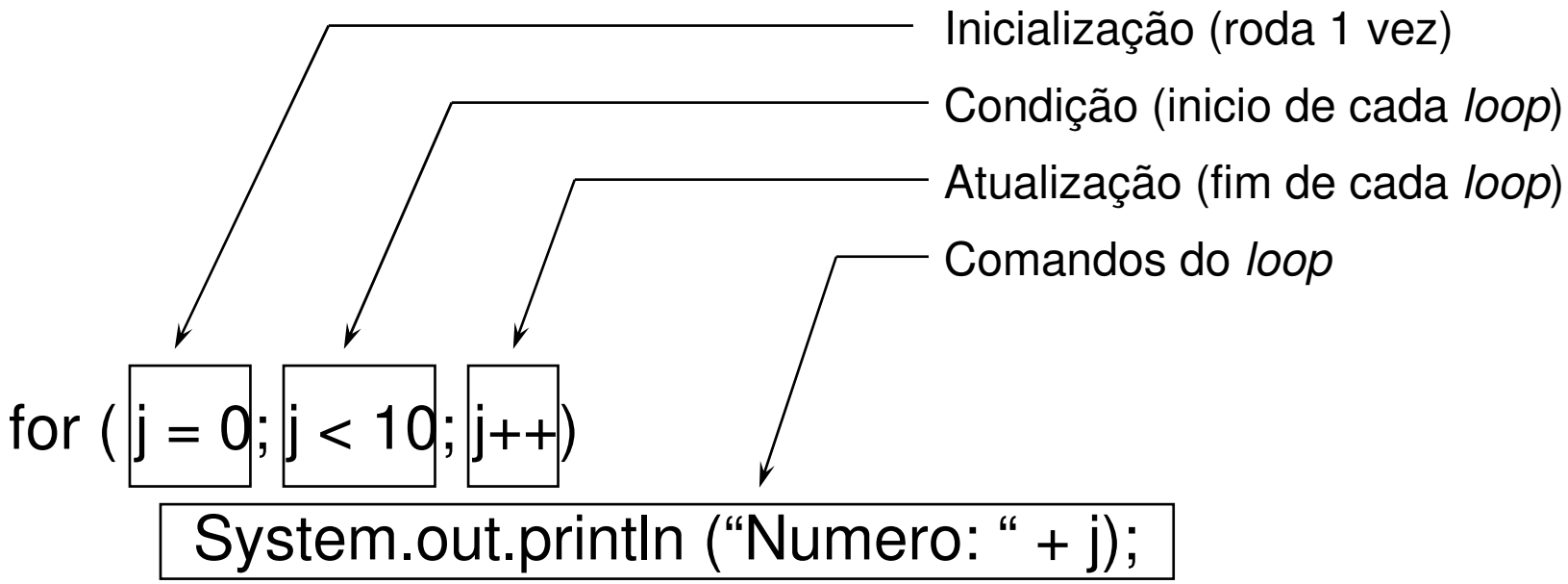
Código em Java (repetições – do-while)

- Executa um bloco de comandos enquanto uma condição for verdadeira. A condição é testada no fim do loop.



Código em Java (repetições – for)

- Executa um bloco de comandos enquanto uma condição for verdadeira. A condição é testada no início do *loop*;



Exercício

- Implementar a classe Carro abaixo no pacote veículo
  - Implementar os atributos e os métodos

- Criar a classe Principal no pacote default contendo o método main para criar objetos da classe Carro e chamar os seus métodos

Exercício 2

- Implementar uma interface Veículo que defina o comportamento de mudar de marcha, mudar velocidade parar e dar ré.
  - Faça a classe Carro implementar a interface Veículo

- Caso algum comportamento fosse igual em todos os veículos, seria mais apropriado implementar uma interface ou uma classe Veículo?

Código em Java (exceções)

- Conceito:
  - Exceções representam situações de erro, ocorridas durante a execução de um programa;
  - Exemplos de exceções são divisão por zero ou incapacidade de ler dados de um arquivo;
- Geradores de exceções:
  - Interpretador Java: quando percebe uma situação de erro padrão (divisão por zero, falha de segurança, …);
  - Métodos do programa: quando percebe uma situação de erro interna do programa (informação inválida, …);

Código em Java (exceções)

- A palavra reservada *throws*, seguida pela classe de exceção gerada, deve ser indicada no cabeçalho de um método que gere uma exceção;
- Os comandos *try-catch-finally* executam um código que pode gerar exceções de maneira segura, realizando o tratamento das exceções;

```
public int gravaRegistro () throws IOException
{
        …       // Código que gera a exceção
}
```

Código em Java (exceções)

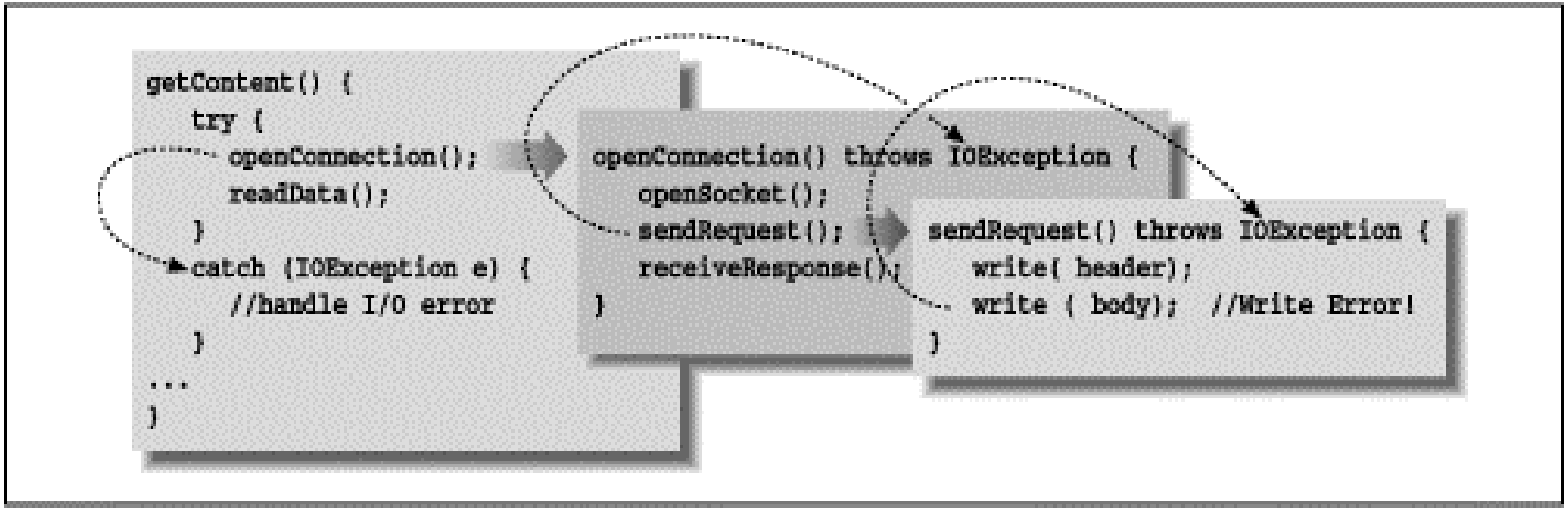
```
try {
        // Código que pode disparar exceções
}
catch (Excecao1 e) {
        // Código executado caso o código no bloco try dispare uma
exceção tipo Excecao1
}
...
catch (ExcecaoN e) {
        // Código executado caso o código no bloco try dispare uma
exceção tipo ExcecaoN
}
finally {
        // Código executado mesmo que tenha ocorrido uma exceção no
bloco try
}
```

Código em Java (exceções)

- Fluxo de execução do programa:
  - A ocorrência de uma exceção transfere o fluxo de execução para o primeiro catch que trate a exceção
  - A ordem dos tratadores de exceção é definida pela ordem com que os métodos foram chamados durante a execução
  - Após o tratamento da exceção no catch, o fluxo é transferido para o finally do mesmo grupo try-catch-finally
  - Na falta de um catch para o tratamento da exceção, o fluxo é transferido diretamente para o finally
  - Na falta de qualquer try-catch-finally, o interpretador Java realiza o tratamento da exceção

Código em Java (exceções)

- Exemplo de propagação de exceções



Referências

- Várias transparências foram produzidas por Leonardo Murta
  - http://www.ic.uff.br/~leomurta