

Programação Orientada a Objetos

CES-22

Prof. Paulo André Castro

pauloac@ita.br

Sala 110 – Prédio da Computação

www.comp.ita.br/~pauloac

IEC - ITA

Objetivos Gerais da Matéria

- Entender e aprender a utilizar os conceitos de classe, objeto, herança, polimorfismo, encapsulamento e demais conceitos de OO para programar e organizar software
- Aprender a utilizar a linguagem Java de forma eficiente e produtiva
- Familiarizar-se com o desenvolvimento de Interface gráficas e programação orientada a eventos

Estrutura do Curso

- Cap. 1 Introdução
 - Principais Conceitos
 - Java e C++
 - Introdução a Linguagem Java
- Cap. 2. Herança
- Cap. 3. Interfaces e Inner Classes
- Cap.4. Graphics Programming e Event Handling
- Cap. 5. Exception, Assertions, Logging and Debugging
- Cap. 6. Introdução a Padrões de Projeto
- Cap. 7 Programação Genérica e Coleções
- Cap. 8. Multithreading

Avaliação

- Por bimestre:
 - 1 prova
 - 2 a 3 labs.
 - Eventualmente listas de exercício
- Um exame final
- As listas de exercícios e os Labs. são individuais

Regras

- Permitido
 - Trocar idéia com algum colega (sem olhar o código) sobre como o lab deve ser feito.
 - Um aluno que já terminou o lab sentar junto para ajudar a encontrar um bug.
 - Dois alunos sentarem e bolarem a solução antes de programarem.

Regras 2

- Proibido
 - Olhar ou copiar o lab de outro aluno ou de anos anteriores. Seja cópia total ou parcial
 - Sentar junto com um amigo com dificuldades e fazer o lab ou parte do lab para ele.
 - Dois alunos fazerem o código do lab juntos.

Bibliografia Básica

Horstmann, Cay and Cornell, Gary. Core Java Volume I – Fundamentals. 9th. Edition. Prentice Hall 2012.

Horstmann, Cay and Cornell, Gary. Core Java Volume II– Advanced Features. 9th. Edition. Prentice Hall 2012.

Stroustrup, Bjarne. The C++ Programming Language,. Addison Wesley. 1997.

Deitel,H. Java Como Programar. 6a. Edição. Prentice Hall. 2005.

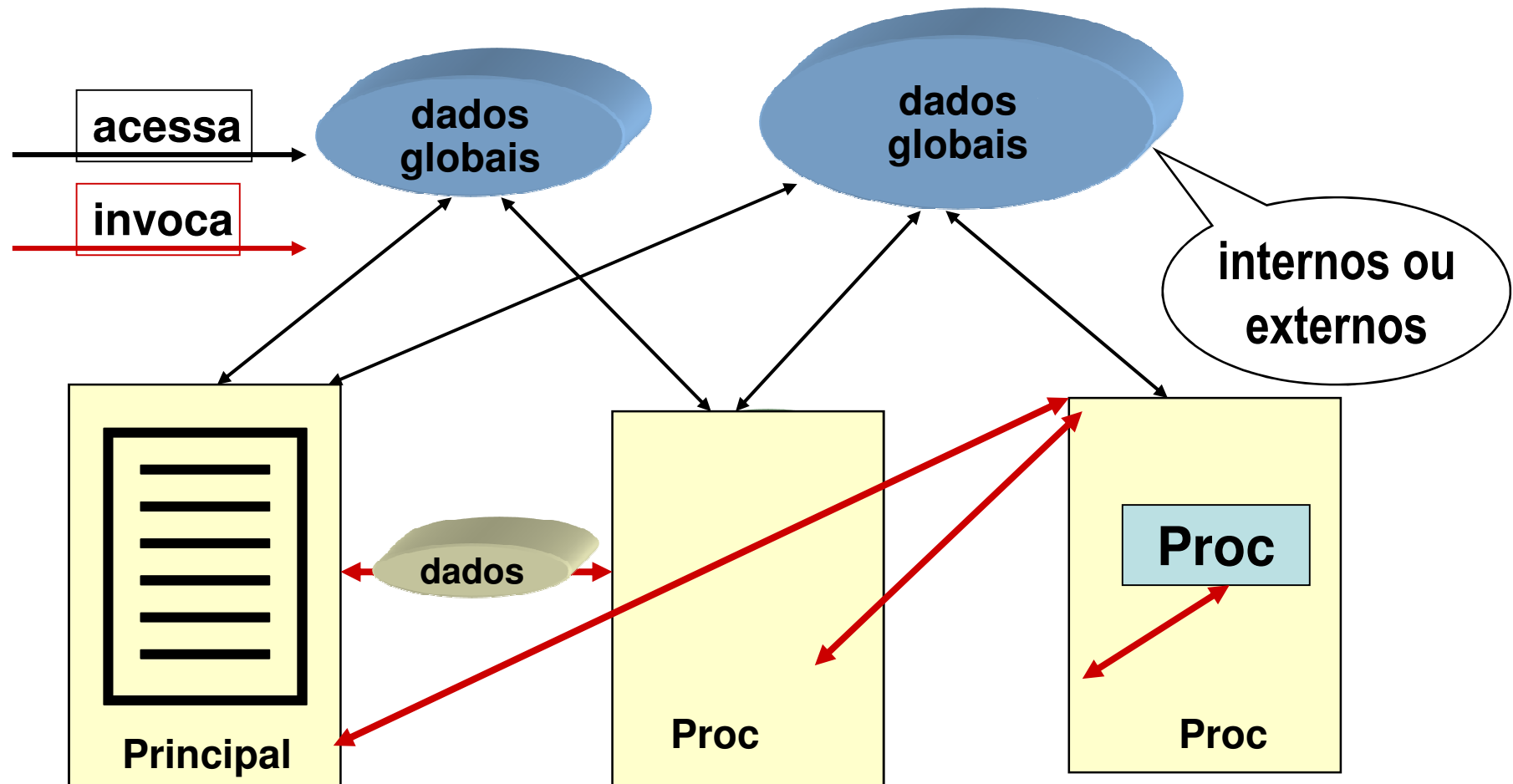
Bibliografia Suplementar

- **Barker, J. *Beginning Java Objects: From Concepts to Code*. New York: Springer-Verlag New York, 2nd Edition, 2005.**
- **Arnold, K.; Gosling, J. *The Java Programming Language*. Reading: Addison Wesley, 1996.**
- **Niemayer, P.; Peck, J. *Exploring Java*. Sebastopol: O'Reilly, 1997.**

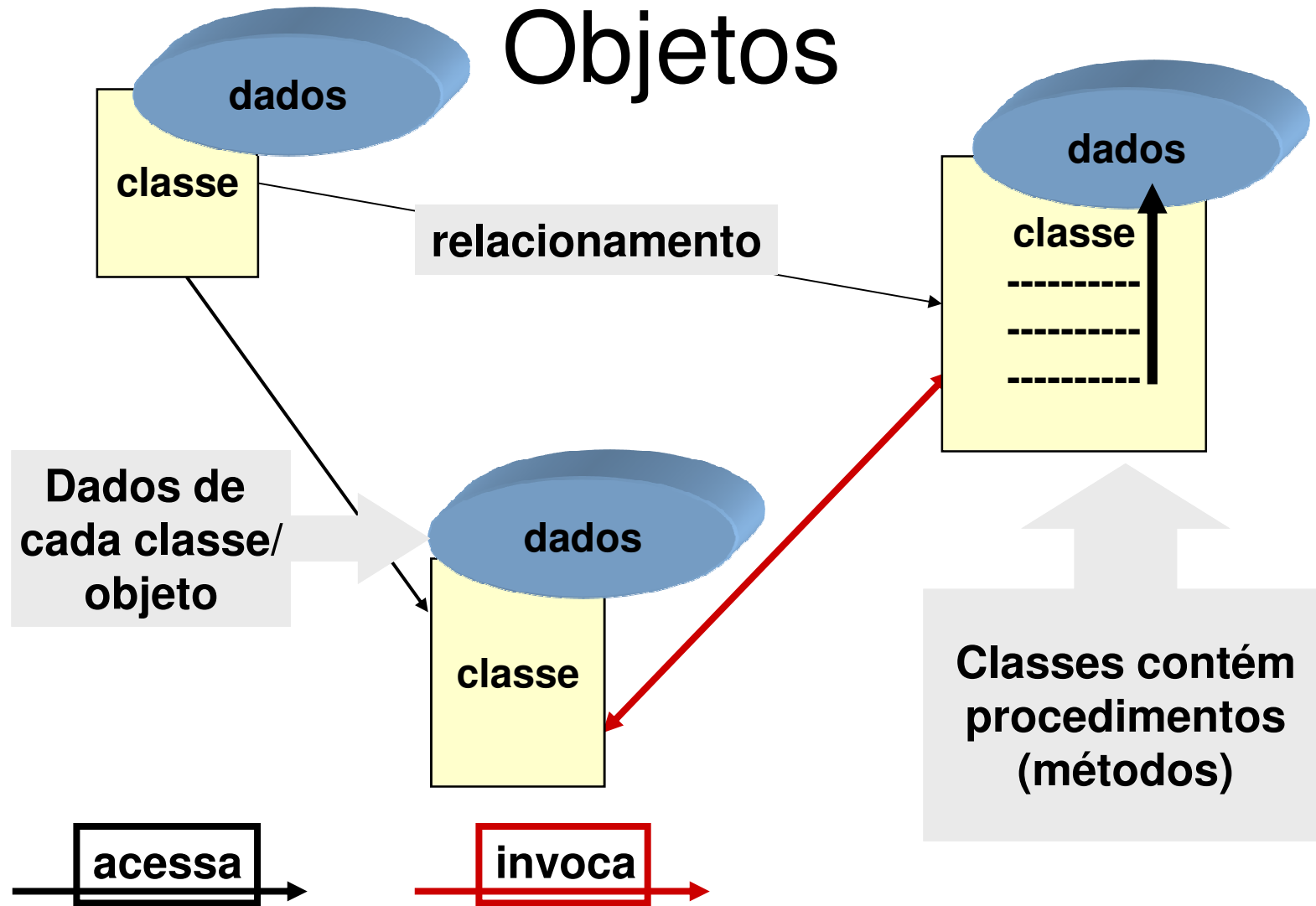
Introdução

- Programação Estruturada x Programação Orientada a Objetos
 - Modelagem com base no conceito de módulo ou sub-programa
 - Modelagem com base no conceito de classe e seus relacionamentos
- Linguagens Orientadas a Objetos
 - Simula, SmallTalk
 - C++, C#, (VB?), etc.
 - Java

Programação Estrurada



Programação Orientada a Objetos



Sumário de Hoje

- Introdução
- **Conceitos Básicos**
 - Nomenclatura básica em OO
 - Variáveis e Instâncias
 - Métodos
 - Encapsulamento
 - Herança e Polimorfismo
- Introdução a linguagem Java

Conceitos básicos de OO

- Classe: um categoria de entidades (“coisas”)
 - Corresponde a um tipo, ou coleção, ou conjunto de entidades afins
- Objeto: Uma entidade com existência física que pertence a um determinado conjunto de entidades afins (classe)

Exemplos de Classe e Objeto

- Classes:
 - Carro, Avião, Pessoa
- Objetos:
 - Carro: Porsche 910 Placa XXXX
 - Avião: Boeing 737-300 Prefixo: PY-XXX
 - Pessoa: José da Silva CPF: XXXXXXXX

Classes x Tipos de Dados

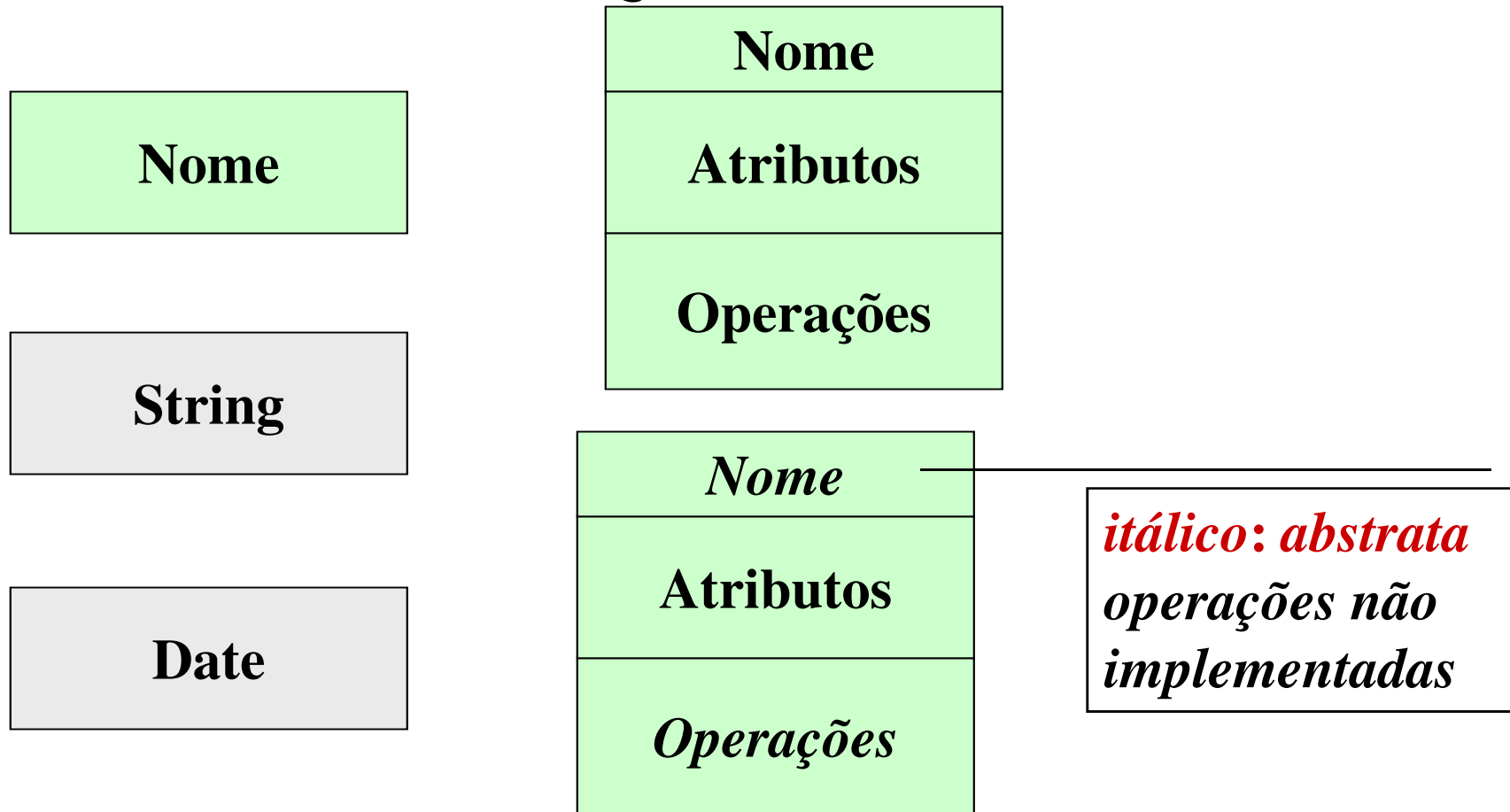
- Uma classe é um **tipo definido pelo usuário** que contém uma estrutura de dados e um conjunto de operações que atuam sobre estes dados
- Analogamente, é o mesmo que o tipo inteiro significa para as variáveis declaradas como inteiros: **acesso a valores através de operações**
- A classe encapsula dados e operações e **controla o acesso a estas propriedades**



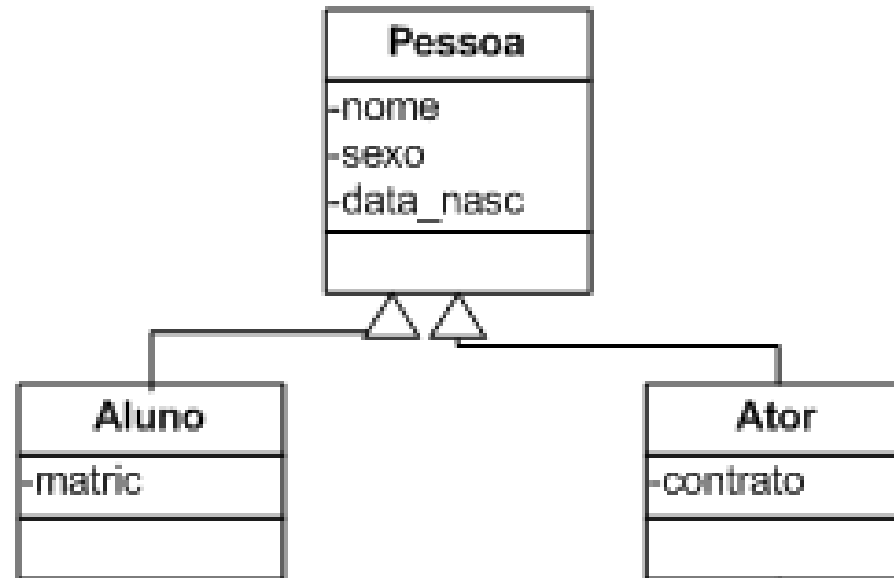
UML

- Unified Modeling Language
 - Linguagem Unificada de Modelagem
 - Linguagem Visual de Modelagem Orientada a Objetos
 - Referência: The Unified Modeling Language User Guide; G. Booch, J. Rumbaugh, I. Jacobson. ACM Press. 2000

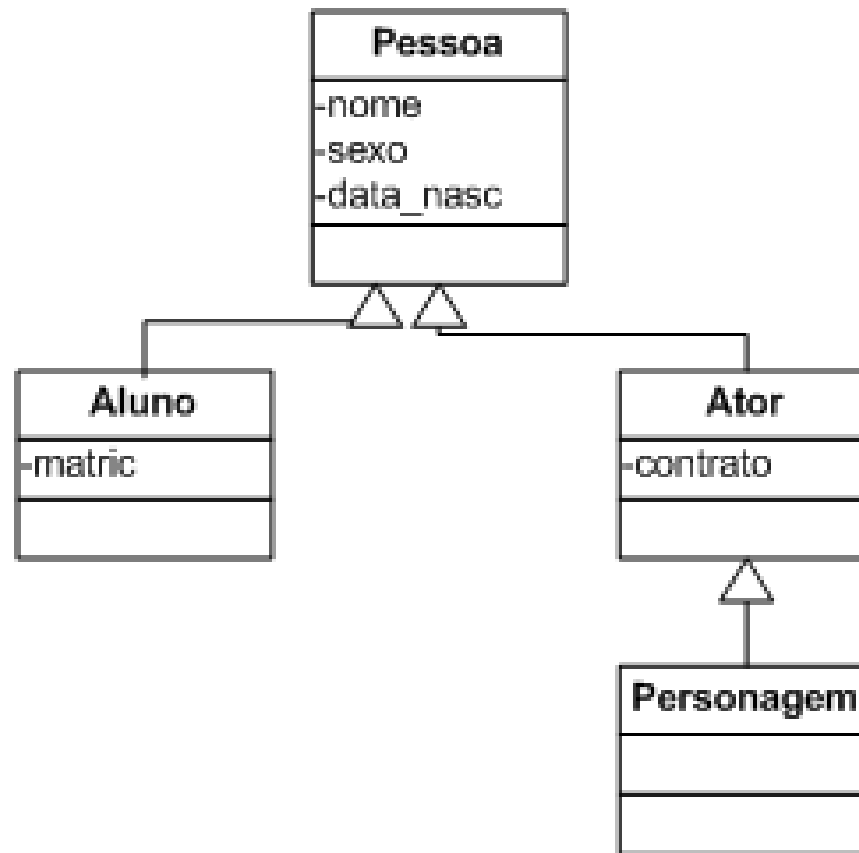
UML: notações de classes



Exemplo em UML - Herança



Exemplo em UML – Está correto ?



Exemplo de Orientação a Objetos - Java

- Classe

```
public class Pessoa {  
    private int idade;  
    private boolean sexo; // Verdadeiro para mulheres  
    private boolean ehResponsavel() {  
        if(idade>21 )  
            return true;  
        if(idade>18 && !sexo)  
            return true;  
        else  
            return false;  
    }  
    public Pessoa(String nome, int id, boolean sex) {Nome=nome; idade=id; sexo=sex; }  
}  
}.....
```

Mais em Orientação a Objetos

- Herança
 - Uma classe pode utilizar métodos e atributos de outras classes sem a necessidade de re-escrever ou copiar nada através do mecanismo de Herança
 - ```
public class Funcionario extends Pessoa {
 double salario;
 int id;
 public double getSalario() {
 return salario;
 }
}
```

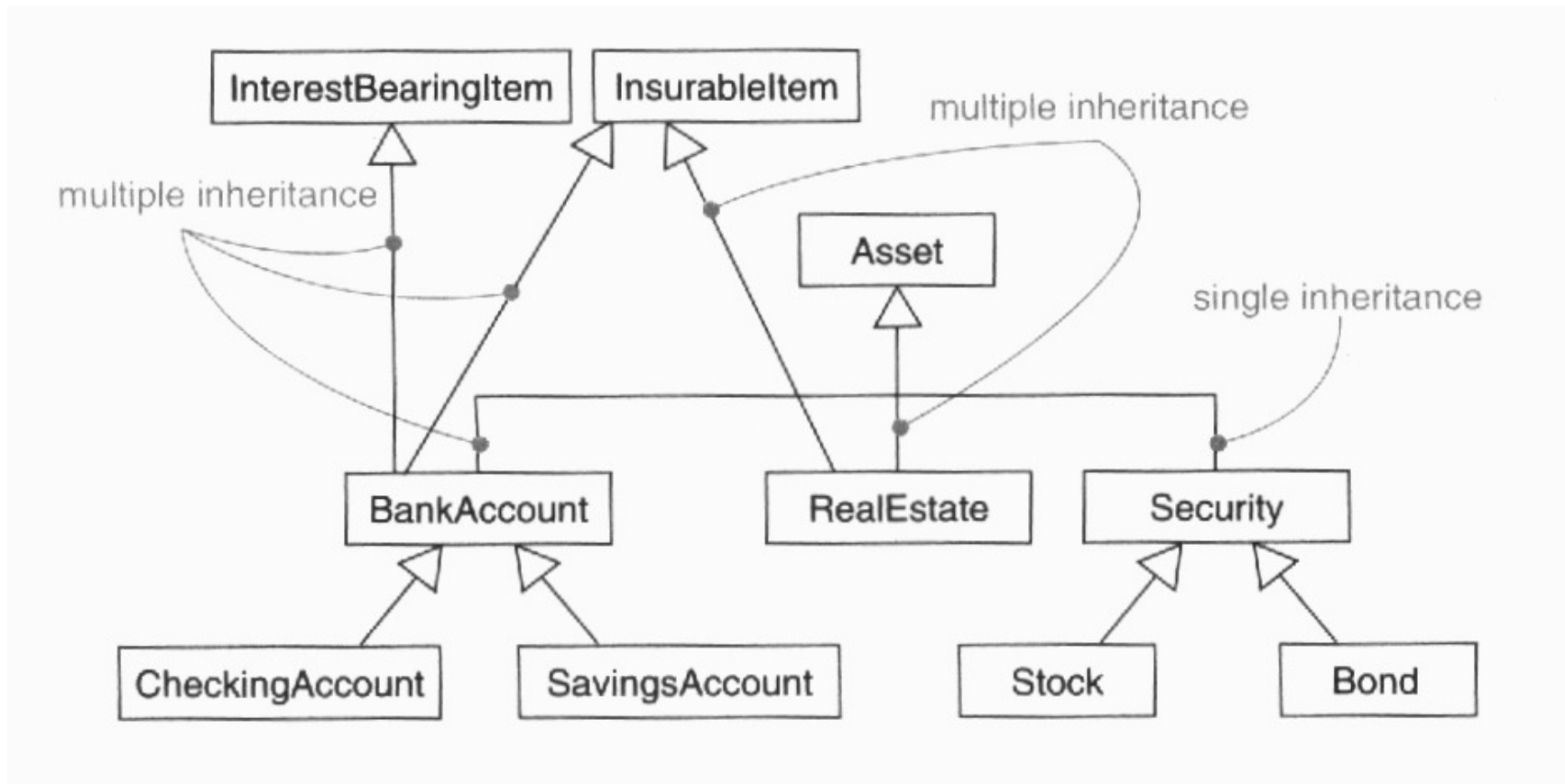
# Classes e sub-classes

- Classe, classe-pai, super-classe, classe base:
  - Carro
  - Motor
  - Avião
- Sub-classe, classe-filha, classe derivada
  - Carro: Porsche 910
  - Motor: Ford 16V
  - Avião: Boeing 737

# Relação de Herança

- O filho herda todas as características do pai
  - Comportamento: funções
  - Atributos: valores
- Em linguagens OO, geralmente há meios de restringir o que será ou não herdado

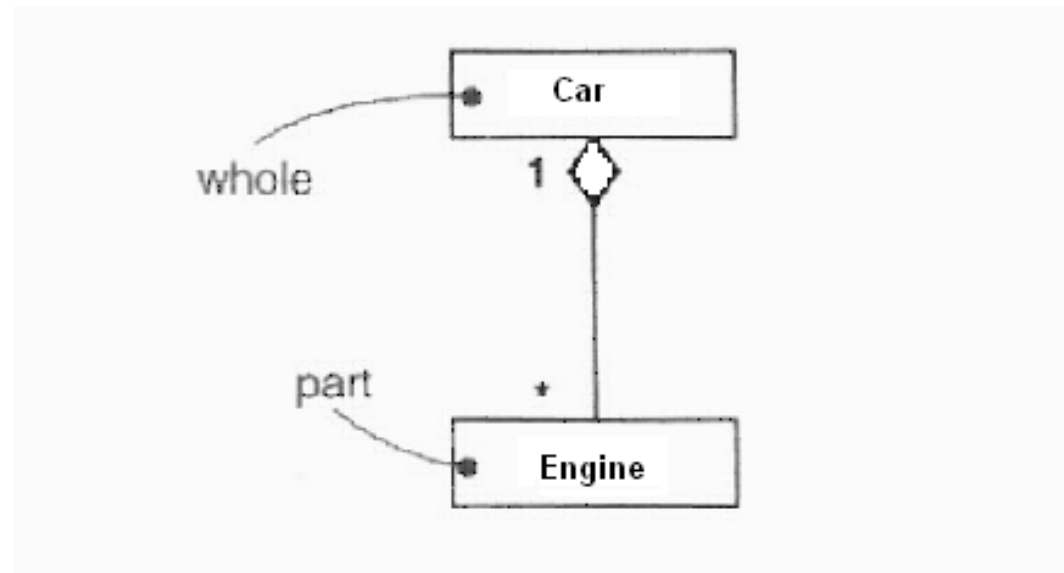
# Exemplo de Herança Múltipla



# Relação Agregação (Todo-Parte)

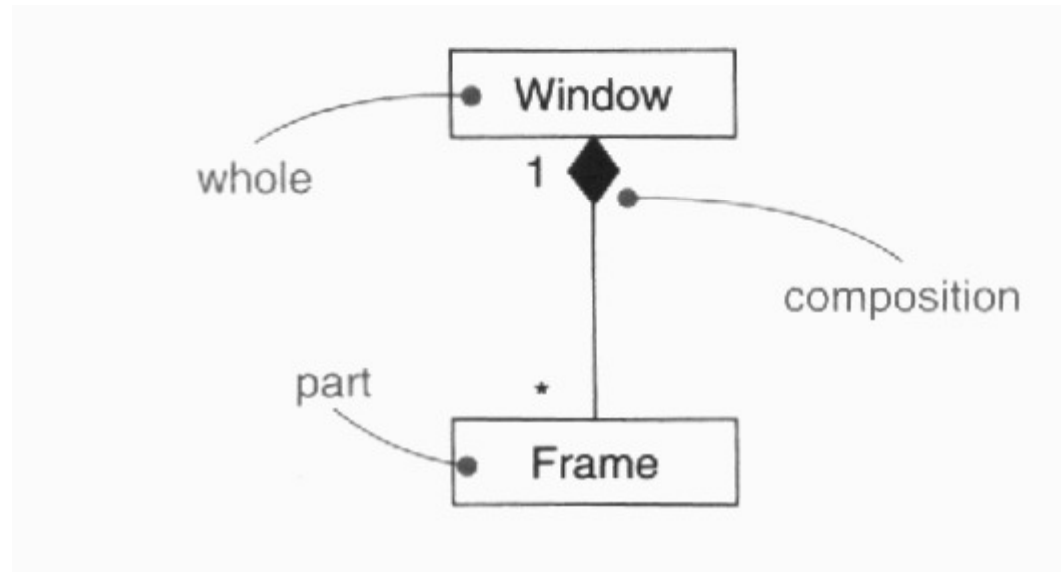
- Uma classe é composta por uma ou mais classes:
- Exemplos:
  - Carro e Motor
  - Avião e Motor
  - Pessoa e Braço

# Aggregação em UML

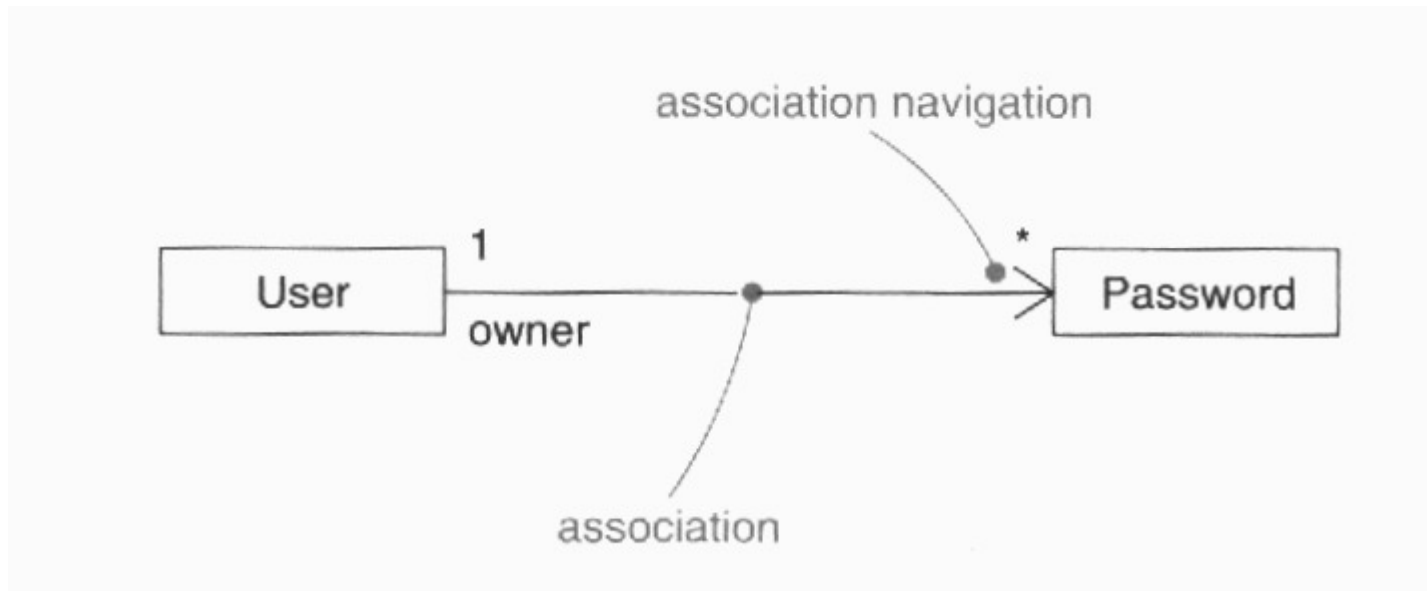


# Composição: Tipo especial de Agregação

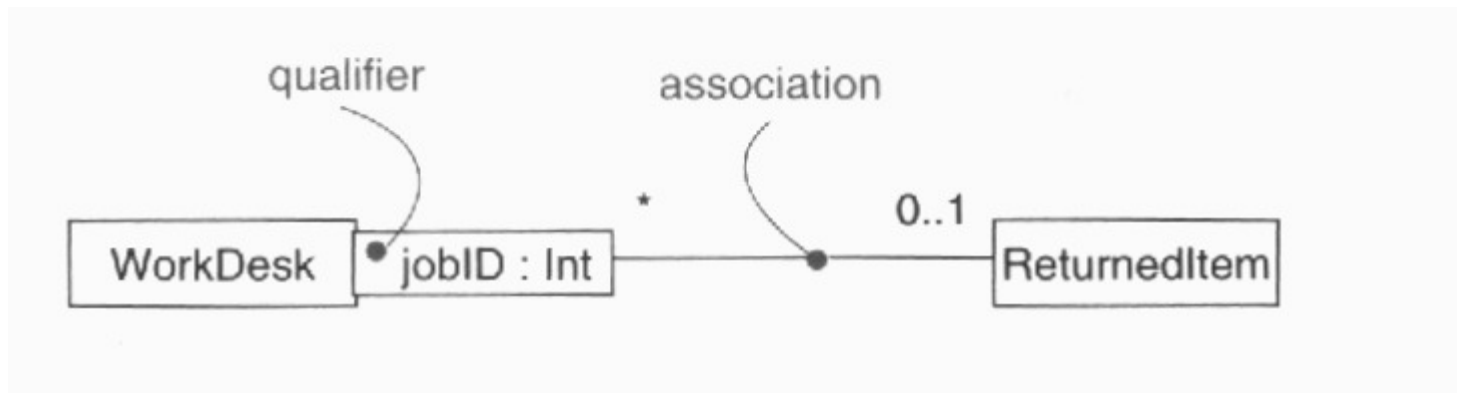
Parte pertence a apenas um todo e tem tempo de vida igual ao todo



# Outras Relações: Associação



# Outras Relações: Associação com Qualificação



# Métodos e Atributos

- Classes podem ter vários métodos e atributos
  - Método: define um comportamento de uma classe
  - Atributo: define uma informação a ser mantida por cada instância de uma classe
- Escopo
  - Escopo de Classe:
    - Ex. Boeing 737 atributo: número de motores
  - Escopo de Objeto (“Instância”)
    - Ex. Boeing 737 atributo: número de assentos

# Encapsulamento

- Encapsulamento: É a capacidade de “esconder” parte do código e dos dados do restante do programa
- Pode-se definir um grau de visibilidade aos métodos e atributos de cada Classe.
- Há vários graus de visibilidade mas todas as linguagens implementam pelo menos os seguintes:
  - Público: Todos podem acessar (ler e escrever)
  - Privado: Apenas a própria classe pode acessar.

# Mais em Orientação a Objetos

- Alterando comportamento nas classes filhas. Sobrescrever método.
- ```
public class Casado extends Pessoa {  
    public boolean ehResponsavel() {  
        return true;  
    }  
    public Casado(String nome,int id,boolean sexo) {  
        super(nome,idade,sexo);  
    }  
}
```

Mais em Orientação a Objetos – Sobrecarga de Métodos

- Sobrecarga permite a existência de vários métodos de mesmo nome, porém com assinaturas levemente diferentes ou seja variando no número e tipo de argumentos e no valor de retorno

```
public Logaritmo {
```

```
.....
```

```
    public double log(double x) {  
        return Math.log(x);
```

```
    }
```

```
    public double log(double x, double b) {  
        return (Math.log(x)/Math.log(b));
```

```
    }
```

```
}
```

Métodos Abstratos

- Métodos abstratos não tem implementação, porém obrigam as classes filhas a realizarem esta implementação. Útil para criar padronizações para as classes derivadas
- Se uma classe tem um (ou mais) métodos abstratos torna-se uma classe abstrata e não pode ser instanciada
- Exemplo:
 - Avião e o método Mover

Herança e Interfaces

- Herança Múltipla: quando uma classe pode herdar métodos e atributos de várias classes
- Não existe herança múltipla em Java, para evitar erros e diminuir a complexidade da programação
- Uma classe pode herdar apenas de uma outra classe
- Todas as classes herdam da classe Object
- Classe totalmente abstrata: Interface

Exemplo - Java

```
class Pessoa{  
    public String nome;  
    public char sexo;  
    public Date dataNasc;  
    ----  
}
```

```
public class Ator extends Pessoa{  
    public String contrato;  
    /* campos herdados  
    public String nome;  
    public char sexo;  
    public Date dataNasc; */  
    ----  
}
```

```
public class Aluno extends Pessoa{  
    public long matric;  
    /* campos herdados  
    public String nome;  
    public char sexo;  
    public Date dataNasc; */  
    ----  
}
```

Interfaces

- Interface: Classe com todos os métodos abstratos e todos os atributos Constantes e de escopo de classe.
- Contrato(s) que uma classe deve respeitar. Isto é, um conjunto de métodos que a classe não pode deixar de implementar.
- Sintaxe em Java:

```
interface NomeDaInterface {  
    type attribute_Name;  
    return_type  
    methodName(type1 param1,type2 param2);  
}
```

Exemplos de Interfaces

```
interface Carro {  
    /* public abstract */ void mover();  
    /*public static final */ int NUM_RODAS=4;  
}  
  
interface Lancha {  
    /* public abstract */ void mover();  
    /*public static final */ int NUM_VELAS=0;  
}
```

Herança e Interfaces

- Uma classe em Java pode implementar várias interfaces. Sintaxe:

```
public class Class1 extends Class2 implements  
Interface1,Interface2,Interface3 {  
.....}
```

Herança Múltipla para Interfaces

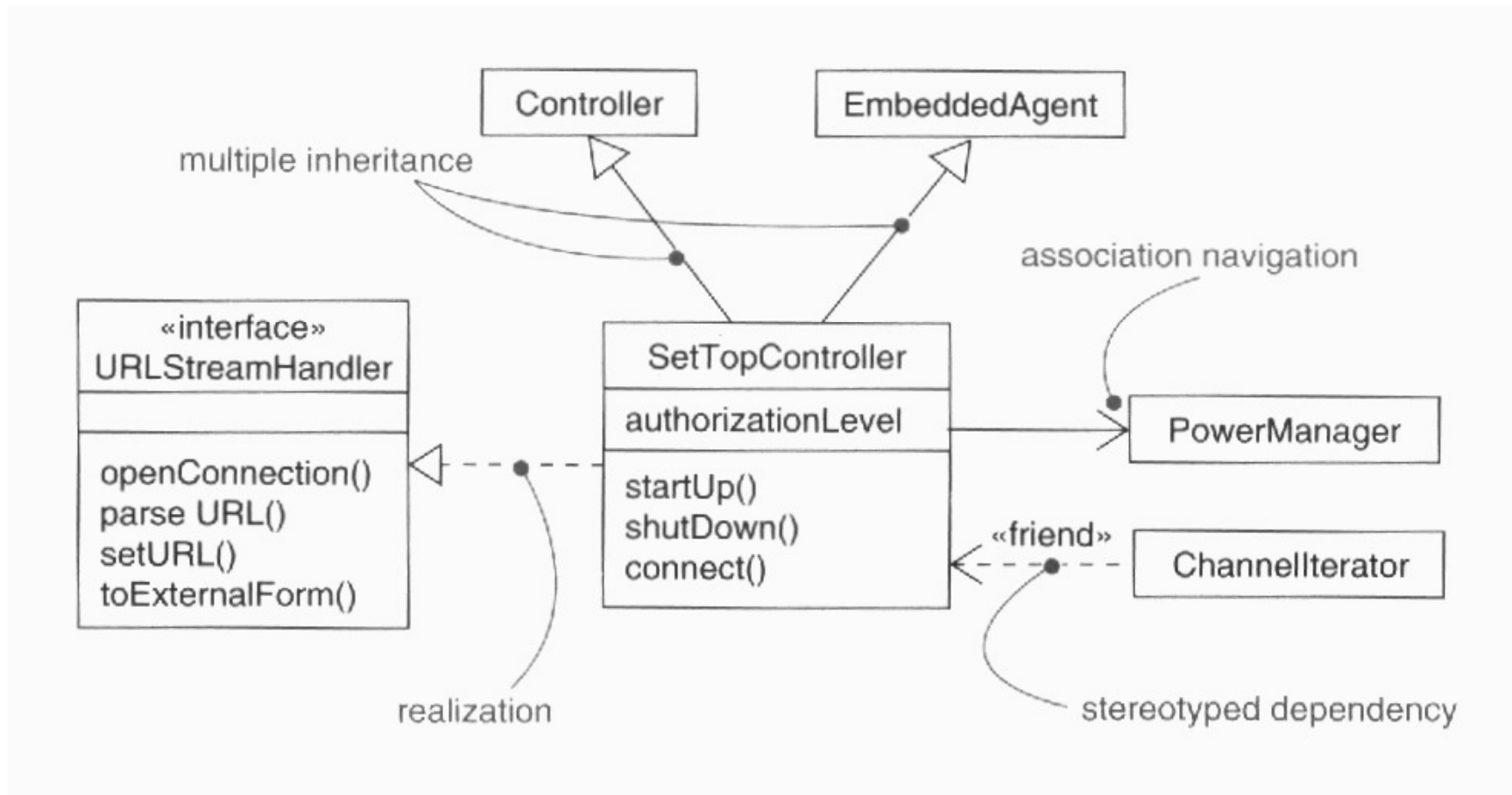
- Uma Interface em Java pode herdar de várias interfaces.

```
interface Carro extends Veiculo,Ativo {  
    /* public abstract */ void mover();  
    /*public static final */ int NUM_RODAS=4;  
}
```

Exemplo de Implementação de Interfaces

```
public class CarroAnfibio implements Carro,Lancha {  
    public void mover() {  
        if(isOnTheGround())  
            ....  
        else  
            ....  
    }  
}
```

Outras Relações: Realização



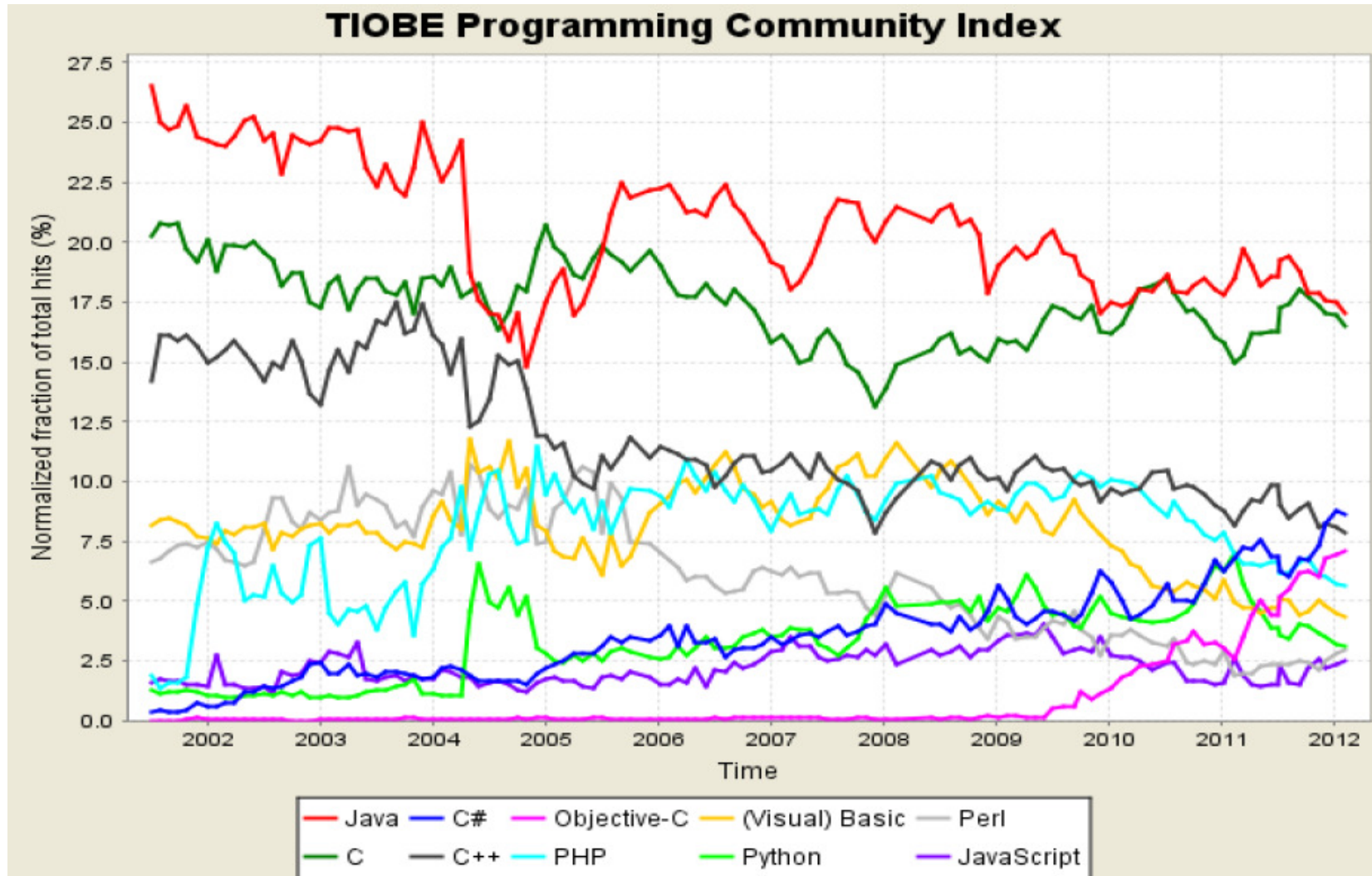
Resumo até agora

- Componentes de uma classe
 - Métodos
 - Atributos(variáveis)
 - Relações
- Relações entre Classes
 - Herança
 - Agregação
 - Associação
 - Realização(Classe/Interface)

Sumário

- Introdução
- Conceitos Básicos
 - Nomenclatura básica em OO
 - Variáveis e Instâncias
 - Métodos
 - Construtores
 - Herança e Polimorfismo
- **Introdução a linguagem Java**
 - Primeiros Programas
 - Fundamentos
 - Tratamento de Erros
 - Coleções

A linguagem mais Popular: Java



Java x C++

- **Similarities with C++**
 - User-defined classes can be used the same way as built-in types.
 - Basic syntax
- **Differences from C++**
 - Methods (member functions) are the only function type
 - Object is the topmost ancestor for all classes
 - All methods use the run-time, not compile-time, types (i.e. all Java methods are like C++ virtual functions)
 - The types of all objects are known at run-time
 - All objects are allocated on the heap (always safe to return objects from methods)
 - Single inheritance only

Criando Programas java

- **Name of file must match name of class**
 - It is case sensitive, even on Windows
- **Processing starts in main**
 - `public static void main(String[] args)`
 - Routines usually called “methods,” not “functions.”
- **Printing is done with System.out**
 - `System.out.println`, `System.out.print`
- **Compile with “javac”**
 - Open DOS window; work from there
 - Supply full case-sensitive file name (with file extension)
- **Execute with “java”**
 - Supply base class name (no file extension)

Exemplo

- **File: HelloWorld.java**

```
public class HelloWorld {  
    public static void main(String[] args) {  
        System.out.println("Hello, world.");  
    }  
}
```

- **Compiling**

```
DOS> javac HelloWorld.java
```

- **Executing**

```
DOS> java HelloWorld  
Hello, world.
```

Exemplo de Orientação a Objetos - Java

- **Classe**

```
public class Pessoa {
    private int idade; private String nome;
    private boolean sexo; // Verdadeiro para mulheres
    private boolean ehResponsavel() {
        if(idade>21 )
            return true;
        if(idade>18 && !sexo)
            return true;
        else
            return false;
    }
    public Pessoa(String n, int id, boolean sex)    {
        nome=n; idade=id; sexo=sex;
    }
    }.....
```

Criando Classes em Java

- Convenção de Nomes em Java
 - Classes iniciam com letras maiúsculas
 - Métodos, atributos e variáveis locais iniciam com minúsculas
- Declaração de Classes

```
public class MyClasse {  
.....  
}
```
- Criando Objetos a partir de Classes
 - Uso do “new”

Orientação a Objetos em Java

- Instâncias
 - `String teste=new String(60);`
 - `String teste2="Isto é um teste";`
 - `ServerSocket servidor= new ServerSocket(25);`
- Construtores e destrutores
 - Os construtores são similares ao C++
 - Não há destrutores em Java.

Exemplo de Orientação a Objetos - Java

- Objetos
 - Pessoa a=new Pessoa(“Joao”,25);
 - Pessoa b= new Pessoa(“Ana”,23);
 - Pessoa c=new Pessoa(“Carla”,28);
 - d=b;
- Atributos de uma classe
 - a.idade == ?
 - d.nome = ?

Formato de uma Definição de Classe em Java

```
modifier class Classname {  
  
    modifier data-type field1;  
    modifier data-type field2;  
    ...  
    modifier data-type fieldN;  
  
    modifier Return-Type methodName1(parameters) {  
        //statements  
    }  
  
    ...  
  
    modifier Return-Type methodName2(parameters) {  
        //statements  
    }  
}
```

Acessando variáveis de instância

- Use um ponto entre o nome da variável e o campo
 - `objectName.fieldName;`
- Por exemplo, usando a classe `Point` da biblioteca Java
 - `Point p=new Point(2,3); //criação de objeto p`
 - `int x2= p.x*p.x; // x2 é 4`
 - `int xPlusY=p.x+p.y; // xPlusY é 5`
 - `p.x=3;`
 - `x2=p.x* p.x; // x2 agora é 9`
- Dentro de um objeto, seus métodos podem acessar as variáveis de instância (e de Classe) sem utilizar o ponto

Exemplo – Orientado a Objetos?

```
class Ship1 {
    public double x, y, speed, direction;
    public String name;
}

public class Test1 {
    public static void main(String[] args) {
        Ship1 s1 = new Ship1();
        s1.x = 0.0;
        s1.y = 0.0;
        s1.speed = 1.0;
        s1.direction = 0.0;    // East
        s1.name = "Ship1";
        Ship1 s2 = new Ship1();
        s2.x = 0.0;
        s2.y = 0.0;
        s2.speed = 2.0;
        s2.direction = 135.0; // Northwest
        s2.name = "Ship2";
    }
}
```

Exemplos

```
...
s1.x = s1.x + s1.speed
      * Math.cos(s1.direction * Math.PI / 180.0);
s1.y = s1.y + s1.speed
      * Math.sin(s1.direction * Math.PI / 180.0);
s2.x = s2.x + s2.speed
      * Math.cos(s2.direction * Math.PI / 180.0);
s2.y = s2.y + s2.speed
      * Math.sin(s2.direction * Math.PI / 180.0);
System.out.println(s1.name + " is at ("
                  + s1.x + "," + s1.y + ").");
System.out.println(s2.name + " is at ("
                  + s2.x + "," + s2.y + ").");
}
}
```

Resultado

- **Compiling and Running:**

```
javac Test1.java  
java Test1
```

Output:

```
Ship1 is at (1,0) .  
Ship2 is at (-1.41421,1.41421) .
```

Problemas na Modelagem?

- Código específico a uma classe está escrito apenas nesta classe?

Métodos: Exemplo

```
class Ship2 {
    public double x=0.0, y=0.0, speed=1.0, direction=0.0;
    public String name = "UnnamedShip";

    private double degreesToRadians(double degrees) {
        return(degrees * Math.PI / 180.0);
    }

    public void move() {
        double angle = degreesToRadians(direction);
        x = x + speed * Math.cos(angle);
        y = y + speed * Math.sin(angle);
    }

    public void printLocation() {
        System.out.println(name + " is at ("
            + x + ", " + y + ").");
    }
}
```

Exemplo (cont.)

```
public class Test2 {  
    public static void main(String[] args) {  
        Ship2 s1 = new Ship2();  
        s1.name = "Ship1";  
        Ship2 s2 = new Ship2();  
        s2.direction = 135.0; // Northwest  
        s2.speed = 2.0;  
        s2.name = "Ship2";  
        s1.move();  
        s2.move();  
        s1.printLocation();  
        s2.printLocation();  
    }  
}
```

- **Compiling and Running:**

```
javac Test2.java  
java Test2
```

- **Output:**

```
Ship1 is at (1,0).  
Ship2 is at (-1.41421,1.41421).
```

Métodos Especiais: Construtores

- Construtores são métodos de uma classe que tem o mesmo nome desta classe e são chamados quando da criação de um objeto desta classe.
 - São usados para “inicializar” um objeto
 - São chamados apenas através de um comando “new”
 - Java provê automaticamente um construtor sem parâmetros, caso nenhum construtor seja declarado explicitamente
 - Por isso, podemos utilizar: `Ship s1=new Ship();` Mesmo sem criar um construtor `Ship()`

Métodos Especiais: Construtores (2)

```
class Ship3 {  
    public double x, y, speed, direction;  
    public String name;  
  
    public Ship3(double x, double y,  
                double speed, double direction,  
                String name) {  
        this.x = x; // "this" differentiates instance vars  
        this.y = y; // from local vars.  
        this.speed = speed;  
        this.direction = direction;  
        this.name = name;  
    }  
  
    private double degreesToRadians(double degrees) {  
        return(degrees * Math.PI / 180.0);  
    }  
}
```

Métodos Especiais: Construtores (3)

```
public void move() {
    double angle = degreesToRadians(direction);
    x = x + speed * Math.cos(angle);
    y = y + speed * Math.sin(angle);
}
public void printLocation() {
    System.out.println(name + " is at ("
        + x + ", " + y + ").");
}
}

public class Test3 {
    public static void main(String[] args) {
        Ship3 s1 = new Ship3(0.0, 0.0, 1.0, 0.0, "Ship1");
        Ship3 s2 = new Ship3(0.0, 0.0, 2.0, 135.0, "Ship2");
        s1.move();
        s2.move();
        s1.printLocation();
        s2.printLocation();
    }
}
```

Métodos Especiais: Construtores (4)

- **Compiling and Running:**

```
javac Test3.java  
java Test3
```

- **Output:**

```
Ship1 is at (1,0) .  
Ship2 is at (-1.41421,1.41421) .
```

A variável especial: this

- The **this** object reference can be used inside any non-static method to refer to the current object
- The common uses of the **this** reference are:
 1. To pass a reference to the current object as a parameter to other methods

```
someMethod(this);
```

2. To resolve name conflicts
 - Using **this** permits the use of instance variables in methods that have local variables with the same name
- Note that it is only necessary to say **this.fieldName** when you have a local variable and a class field with the same name; otherwise just use **fieldName** with no **this**

Destrutores

- Não há destrutores em Java
 - Ao contrário de C++
- Garbage Collector
 - Coletor de lixo: Faz a limpeza (remoção da memória) de variáveis que não serão mais utilizadas pelo programa.
 - Pode ser chamado através de :
 - `System.gc();`

Polimorfismo

- Um mesmo comando enviado para objetos diferentes gera (ou pode gerar) ações diferentes.
- Exemplo:
 - Comando: Mover
 - Carro, Avião e Pessoa são subclasses de EntidadeMovel
- A chamada de uma mesma função pode levar a ser executadas funções diferentes
- Ex.: EntidadeMovel->mover();
 - //Pode resultar em Carro::mover(); Aviao::mover(); ou Pessoa::mover();

Exemplo de Orientação a Objetos – classe Pessoa

Arquivo: Pessoa.java

```
public class Pessoa {
    private int idade;
    private boolean sexo; // Verdadeiro para mulheres
    public boolean ehResponsavel() {
        if(idade>21 )
            return true;
        if(idade>18 && !sexo)
            return true;
        else
            return false;
    }
    public Pessoa(String nome, int id, boolean sex) {Nome=nome; idade=id;
        sexo=sex; }
    }
}
```

Exemplo de Orientação a Objetos – subclasse Casado

Arquivo: Casado.java

```
public class Casado extends Pessoa {  
    public boolean ehResponsavel() {  
        return true;  
    }  
    public Casado(String nome,int id,boolean sexo) {  
        super(nome,idade,sexo);  
    }  
}
```

Polimorfismo – Código Java

```
public class ExemploPessoa {
    public static void main(String[] args) {
        Casado casado=new Casado("Zé",17,false);
        Pessoa cidadao=new Pessoa("Maria",18,true);
        Pessoa[] trabalhador= new Pessoa[2];
        trabalhador[0]=casado;
        trabalhador[1]=cidadao;
        for(int i=0; i<2; i++) {
            String aux;
            if(trabalhador[i].ehResponsavel())
                aux.strcpy(trabalhador[i].Nome+"é responsável");
            else
                aux.strcpy(trabalhador[i].Nome+ "não é responsável.");
            System.out.println(aux);
        }
    }
}
```

Qual o resultado da execução?

Polimorfismo em C++

- Um mesmo comando enviado para objetos diferentes gera (ou pode gerar) ações diferentes.
- Exemplo:
 - Comando: Mover
 - Carro
 - Avião
 - Pessoa
- A chamada de uma mesma função pode levar a ser executadas funções diferentes
- Ex.: `EntidadeMovel->mover()`;
 - `//Pode resultar em Carro::mover(); Aviao::mover(); ou Pessoa::mover();`
- Em C++, deve-se definir este tipo de função como **virtual**

71

Exemplo de Orientação a Objetos – Pessoa e Casado

```
class Pessoa{
    public:
    char* nome;
    char sexo;
    int idade ;
    Pessoa( char* n,char s, int i) { nome=n; sexo=s; idade=i;
    } //uma função membro com corpo dentro da classe é inline
    virtual bool ehResponsavel();
};

bool Pessoa::ehResponsavel(){
    if(idade>18) return true; else return false;
}
```

Exemplo de Orientação a Objetos –Pessoa e Casado

```
class Casado : public Pessoa {  
public:  
    virtual bool ehResponsavel();  
    Casado( char* n,char s, int i) : Pessoa(n,s,i) { }  
};  
  
bool Casado::ehResponsavel() {  
    return true;  
}
```

Exemplo 1 – C++

```
void main() {
Pessoa pessoa("Alan",'m',20);
Casado pessoa_casada("Beth",'f',17);
Pessoa *p1=new Casado("Charlie",'m',17);
cout<<"\nNome:"<<pessoa.nome;
    if(pessoa.ehResponsavel())    cout<<" Legalmente responsável";
    else    cout<<" Não legalmente responsável";

cout<<"\nNome:"<<pessoa_casada.nome;
    if(pessoa_casada.ehResponsavel())    cout<<" Legalmente responsável";
    else    cout<<" Não legalmente responsável";

cout<<"\nNome:"<<p1->nome;
    if(p1->ehResponsavel())    cout<<" Legalmente responsável";
    else    cout<<" Não legalmente responsável";
//Resultado????
}
```

74

Resultado – Exemplo 1

- Nome:Alan Legalmente responsável
- Nome:Beth Legalmente responsável
- Nome:Charlie Legalmente responsável

Exemplo 2 – C++

```
void main() {
Pessoa pessoa("Alan",'m',20);
Casado pessoa_casada("Beth",'f',17);
Pessoa *p1=new Pessoa("Charlie",'m',17);
cout<<"\nNome:"<<pessoa.nome;
    if(pessoa.ehResponsavel())    cout<<" Legalmente responsável";
    else    cout<<" Não legalmente responsável";

cout<<"\nNome:"<<pessoa_casada.nome;
    if(pessoa_casada.ehResponsavel())    cout<<" Legalmente responsável";
    else    cout<<" Não legalmente responsável";

cout<<"\nNome:"<<p1->nome;
    if(p1->ehResponsavel())    cout<<" Legalmente responsável";
    else    cout<<" Não legalmente responsável";
//Resultado????
}
```

76

Resultado – Exemplo 2

- Nome:Alan Legalmente responsável
- Nome:Beth Legalmente responsável
- Nome:Charlie Não Legalmente responsável

Convenções e Boas Práticas em OO

Resumo

- Código referente exclusivamente a uma classe deve ficar dentro desta classe
- Uma classe deve ter o menor número possível de métodos públicos, mas deve ter pelo menos um método público
- É uma boa prática evitar variáveis públicas. Prefira utilizar métodos para acessar as variáveis. Os métodos get e set.
- Use métodos construtores para inicializar objetos

Convenções e Boas Práticas em OO

Resumo

- Classes devem iniciar com letras maiúsculas, métodos, atributos e variáveis com letras minúsculas
- Métodos devem ter um tipo de retorno ou “void”
- Acesse atributos através de `objectName.fieldName`
- Acesse métodos através de `objectName.methodName()`
- Métodos estáticos não precisam de instâncias da classe
- Construtores são métodos especiais sem tipo de retorno

Convenções e características em Java

Resumo

- A referência `this` aponta para o objeto atual
- Java faz seu próprio gerenciamento de memória e portanto não requer destrutores
- Java permite herança simples e o uso de interfaces
- As classes abstratas e interfaces não podem ser instanciadas
- Uma classe Java pode implementar várias interfaces

Fundamentos de Programação Java

- Comandos
- Sintaxe
- Estrutura
- Exemplos

Fundamentos de Programação Java

- Início e Fim de Blocos de Comandos
 - “{ “ e “}”
- comandos if, if-else, while, do-while, for e switch-case são idênticos ao C/ C++
- Comando break <label>;

Tipos básicos de Variáveis

Tipo	Tamanho
byte	1 bytes
short	2 bytes
int	4 bytes
long	8 bytes
float	4 bytes
double	8 bytes
char	2 bytes
boolean	1 bit

Variáveis

- Declaração
 - int inteiro; char letra;
 - byte apenasUmByte;
- Declaração e Inicialização
 - float saldoConta=0.0;
 - boolean serOuNaoSer=false;

Tipos Complexos

- `int[] arrayInteiro= new int[40];`
- `char sTexto[]= new char[60]; // use String's`
- **Classe String**
 - `String s=new String("isto e uma string");`
 - `String s2="isto e outra string";`
 - `String c=s+s2;`
 - `System.out.println("S="+s+"S2="+s2+"c="+c);`

Tipos Complexos

- **Arrays are accessed with []**
 - Array indices are zero-based
 - The argument to `main` is an array of strings that correspond to the command line arguments
 - `args[0]` returns first command-line argument
 - `args[1]` returns second command-line argument
 - Etc.
- **The length field gives the number of elements in an array**
 - Thus, `args.length` gives the number of command-line arguments
 - Unlike in C/C++, the name of the program is not inserted into the command-line arguments

Exemplo

- **File: ShowTwoArgs.java**

```
public class ShowTwoArgs {  
    public static void main(String[] args) {  
        System.out.println("First arg: " +  
                            args[0]);  
        System.out.println("Second arg: " +  
                            args[1]);  
    }  
}
```

Exemplo

- **Compiling**

```
DOS> javac ShowTwoArgs.java
```

- **Executing**

```
DOS> java ShowTwoArgs Hello World
```

```
First args Hello
```

```
Second arg: Class
```

```
DOS> java ShowTwoArgs
```

```
[Error message]
```

Comandos...

- **Comando** := *Comando_simples* | *Comando_composto* | *Comando_condicional* | *Comando_iterativo* | *Comando_de_seleção*
- **Comando_simples** := *Comando_de_atribuição* | *Comando_de_entrada* | *Comando_de_saída* | *Chamada_de_subprograma* | *Comando_vazio*
- **Comando_vazio** := ;
- **Comando_composto** := { *Comando* *Comando*
Comando }

Comandos...

- ***Comando_condicional*** := **if** (*Expressão*) *Comando_1* | **if** (*Expressão*) *Comando_1* **else** *Comando_2*
- ***Comando_iterativo*** := *Comando_while* / *Comando_do* / *Comando_for*

Comando de Seleção em Java

```
switch ( expressão ) {  
    case V11:  
    case V12:  
        .  
        .  
    case V1m: lista de comandos; break;  
    case V21:  
    case V22:  
        .  
        .  
    case V2n: lista de comandos; break;  
        .  
        .  
    case Vip: lista de comandos; break;  
    default: lista de comandos;  
}
```

Comandos...

- Comando atribuição condicional
 - Forma: $Expr1 ? Expr2 : Expr3$
 - Calcula-se $Expr1$;
 - Se o valor for **Verdadeiro**, calcula-se o valor de $Expr2$, que será o valor da expressão condicional;
 - Se o valor for **Falso**, calcula-se o valor de $Expr3$, que será o valor da expressão condicional.

Comandos de repetição

- **while**

```
while (continueTest) {  
    body;  
}
```

- **do**

```
do {  
    body;  
} while (continueTest);
```

- **for**

```
for(init; continueTest; updateOp) {  
    body;  
}
```

Comandos Iterativos...

- **Comandos for e while versus Comando do-while**
- **Comando *break***: Saída anormal de um comando iterativo ou comando de seleção mais interno.

```
while ( ..... ) {  
    .....  
    if ( ..... ) break;  
    if ( ..... ) continue;  
    .....  
}  
..... /* Proximo comando a ser executado depois do break */  
.....
```

- **Comando *continue***: encerra a iteração corrente e inicia a iteração seguinte.

Exemplo - while

```
public static void listNums1(int max) {  
    int i = 0;  
    while (i <= max) {  
        System.out.println("Number: " + i);  
        i++; // "++" means "add one"  
    }  
}
```

Exemplo – do/while

```
public static void listNums2(int max) {  
    int i = 0;  
    do {  
        System.out.println("Number: " + i);  
        i++;  
    } while (i <= max);  
        // ^ Don't forget semicolon  
}
```

Exemplo - for

```
public static void listNums3(int max) {  
    for(int i=0; i<max; i++) {  
        System.out.println("Number: " + i);  
    }  
}
```

Percorrendo todos os elementos de um array

```
public class LoopTest {
    public static void main(String[] args) {
        listNums1(5);
        listNums2(6);
        listNums3(7);
    }

    public static void listNums1(int max) {...}
    public static void listNums2(int max) {...}
    public static void listNums3(int max) {...}
}
```

Percorrendo todos os elementos de um array

```
public class ShowArgs {  
    public static void main(String[] args) {  
        for(int i=0; i<args.length; i++) {  
            System.out.println("Arg " + i +  
                               " is " +  
                               args[i]);  
        }  
    }  
}
```

Comando condicional - if

- **Single Option**

```
if (boolean-expression) {  
    statement;  
}
```

- **Multiple Options**

```
if (boolean-expression) {  
    statement1;  
} else {  
    statement2;  
}
```

Expressões Condicionais

- **==, !=**
 - Equality, inequality. In addition to comparing primitive types, `==` tests if two objects are identical (the same object), not just if they appear equal (have the same fields). More details when we introduce objects.
- **<, <=, >, >=**
 - Numeric less than, less than or equal to, greater than, greater than or equal to.
- **&&, ||**
 - Logical AND, OR. Both use short-circuit evaluation to more efficiently compute the results of complicated expressions.
- **!**
 - Logical negation.

Exemplos

```
public static int max2(int n1, int n2) {  
    if (n1 >= n2)  
        return(n1);  
    else  
        return(n2);  
}
```

Comparação de Strings

```
public static void main(String[] args) {  
    String match = "Test";  
    if (args.length == 0) {  
        System.out.println("No args");  
    } else if (args[0] == match) {  
        System.out.println("Match");  
    } else {  
        System.out.println("No match");  
    }  
}
```

- **Prints "No match" for *all* inputs**

– Fix:

```
if (args[0].equals(match))
```

Criando Arrays

- **Step 1: allocate an array of references:**
`type[] var = new type[size];`
- **Eg:**
`int[] values = new int[7];`
`Point[] points = new Point[someArray.length];`
- **Step 2: populate the array**
`points[0] = new Point(...);`
`points[1] = new Point(...);`
...
`Points[6] = new Point(...);`
- **If you fail to populate an entry**
 - Default value is 0 for numeric arrays
 - Default value is null for object arrays

Arrays de várias dimensões

```
int[][] twoD = new int[64][32];
```

```
String[][] cats = { { "Caesar", "blue-point" },  
                    { "Heather", "seal-point" },  
                    { "Ted", "red-point" } };
```

```
int[][] irregular = { { 1 },  
                      { 2, 3, 4 },  
                      { 5 },  
                      { 6, 7 } };
```

Exemplo

```
public class TriangleArray {
    public static void main(String[] args) {

        int[][] triangle = new int[10][];

        for(int i=0; i<triangle.length; i++) {
            triangle[i] = new int[i+1];
        }

        for (int i=0; i<triangle.length; i++) {
            for(int j=0; j<triangle[i].length; j++) {
                System.out.print(triangle[i][j]);
            }
            System.out.println();
        }
    }
}
```

Resultado

```
> java TriangleArray
```

```
0
```

```
00
```

```
000
```

```
0000
```

```
00000
```

```
000000
```

```
0000000
```

```
00000000
```

```
000000000
```

```
0000000000
```