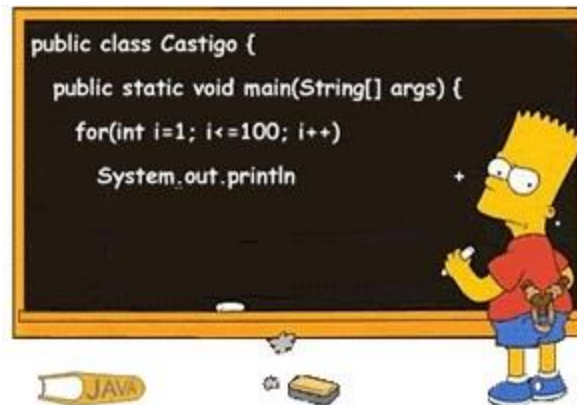




# Programação Orientada a Objetos

## Unidade 1 – Conceitos OOP



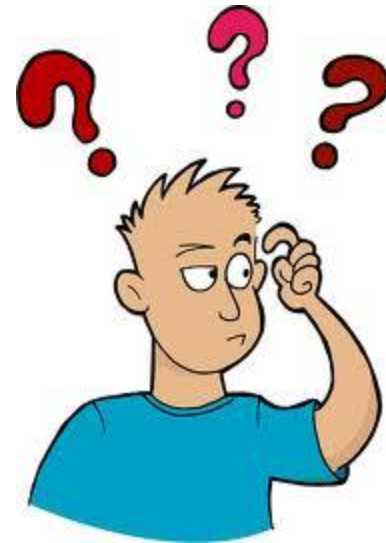


# Bibliografia

- Beginning Java 2 – Ivor Horton – 1999 WROX
- Java2 – The Complete Reference – 7<sup>th</sup> Edition – Herbert Schildt – Oracle Press
- **Core Java Fundamentals – Horstmann / Cornell – PTR- Volumes 1 e 2 – 8<sup>th</sup> Edition**
- Inside the Java 2 – Virtual Machine Venners – McGrawHill
- Understanding Object-Oriented Programming with JAVA – Timothy Budd – Addison Wesley
- Head First Java, 2<sup>nd</sup> Edition by Kathy Sierra and Bert Bates
- Effective Java, 2<sup>nd</sup> Edition by Joshua Bloch
- Thinking in Java (4<sup>th</sup> Edition) by Bruce Eckel
- **Java How to Program - 9<sup>th</sup> Edition by Paul Deitel and Harvey Deitel**

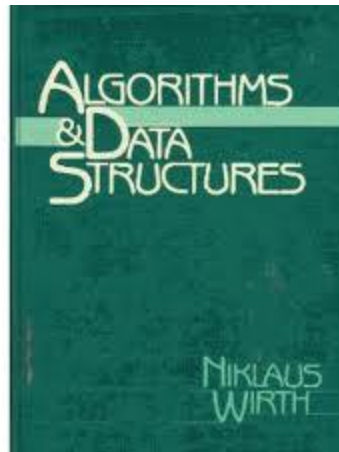


# Como era a programação nos anos 70 ?



# Programação Estruturada

- ⊕ Focada no processo de **transformação** de **dados**;
- ⊕ Após os **procedimentos** serem definidos, os próximos passos consistiam em se determinar as formas de armazenamento dos dados;
- ⊕ **Niklaus Wirth**, projetista do Pascal, escreveu o famoso livro “**Algorithms + Data Structures = Programs**”. (Observe que Algorithms vem antes de Data Structures...).



O que mudou no paradigma OOP ?



# Paradigma OOP

- ⊕ **OOP** inverte a ordem;
- ⊕ Primeiro foco nos **dados** e em seguida nos algoritmos que operam sobre os dados;
- ⊕ A implementação da **funcionalidade** é, em geral, **escondida** dos usuários;
- ⊕ Este conceito é chamado **Encapsulamento**;





OOP exige uma nova forma de pensar...



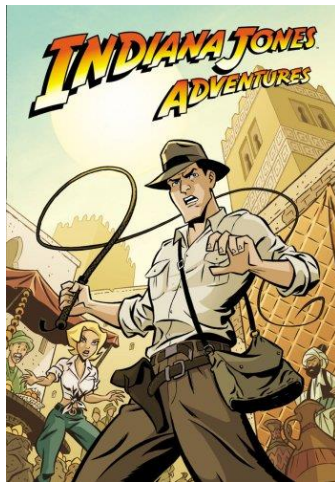
Java é totalmente orientado a objetos...







Para ser produtivo em Java, é necessário conhecer o paradigma OOP...

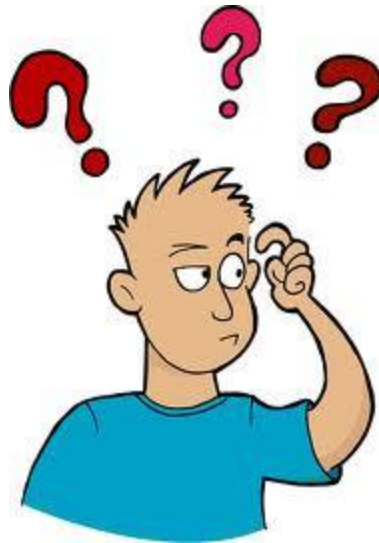




Afinal, o que é um programa OOP?



Antes, uma outra questão ...



Para que serve um programa OOP ?





para resolver um problema...





Imagine uma empresa com um determinado problema...



A empresa pode contratar um funcionário (**objeto**) ...



O funcionário irá resolver o problema  
desempenhando uma função...







Para a empresa o mais importante é **resolver** o problema...

Os **detalhes** de como o funcionário desempenha a função não são tão importantes...





Os funcionários da empresa operam como uma comunidade, trocando mensagens para a solução do problema...





# Como um programa OOP resolve um problema ?





Da mesma forma que em uma empresa!  
Com o emprego de objetos que executam ações (operações)...





Afinal, o que é um programa OOP ?



# Um programa OOP

- É um conjunto de **objetos** que trocam **mensagens** para, ao final do processamento, resolver o problema do usuário;
- Cada objeto tem uma **funcionalidade** que é exposta aos usuários (**interface**) e a implementação é, em geral, escondida dos mesmos (**encapsulamento**).



# Objetos java

- Podem ser obtidos de uma prateleira (**biblioteca**) com objetos prontos;
- Podem também ser construídos internamente no programa.



# Como se cria um objeto num programa Java?

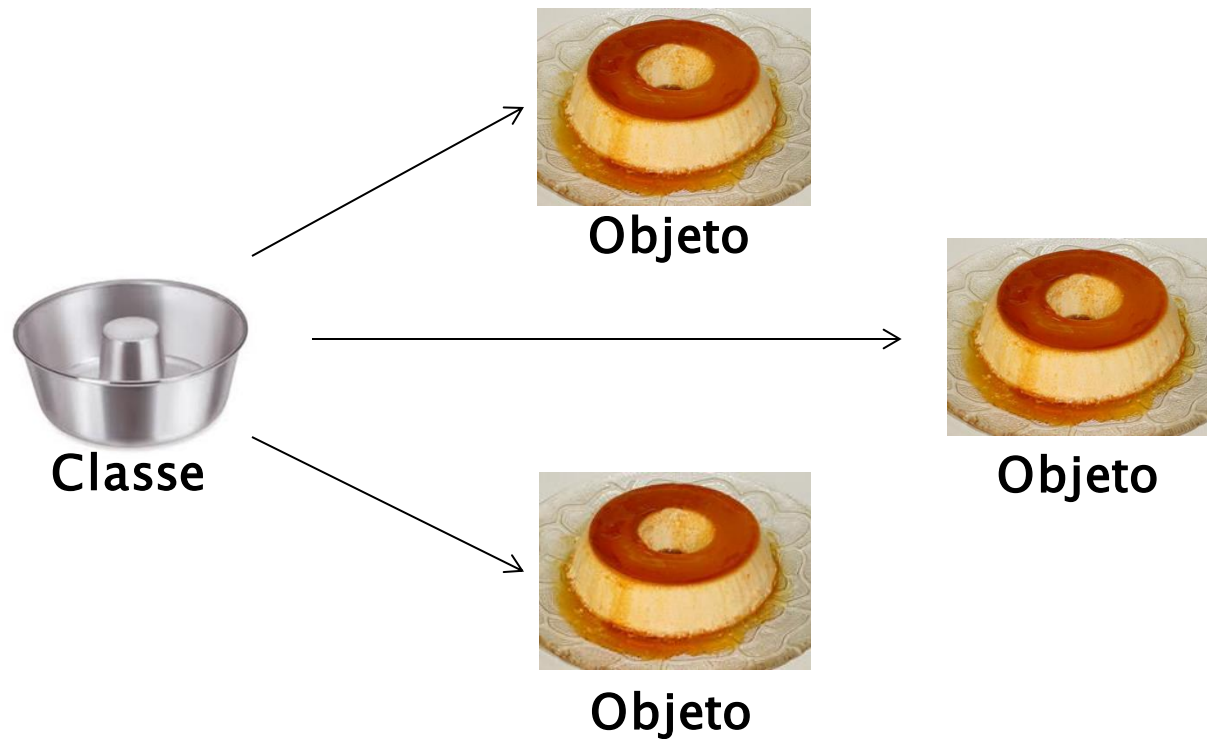




# Por meio de classes...



Uma classe é um modelo, uma especificação, um molde, a partir do qual criam-se objetos...



# Instâncias

- ◆ Um objeto é uma instância de uma classe;



Classe

Instância



Objeto



# OOP em Java

- ◆ Todo código escrito em Java está dentro de uma **classe**;
- ◆ A plataforma disponibiliza centenas de classes por meio de **API**;



- ◆ Você ainda pode criar classes para descrever objetos do domínio do problema de suas aplicações.



# Documentação API – Java

<http://docs.oracle.com/javase/8/docs/api/>



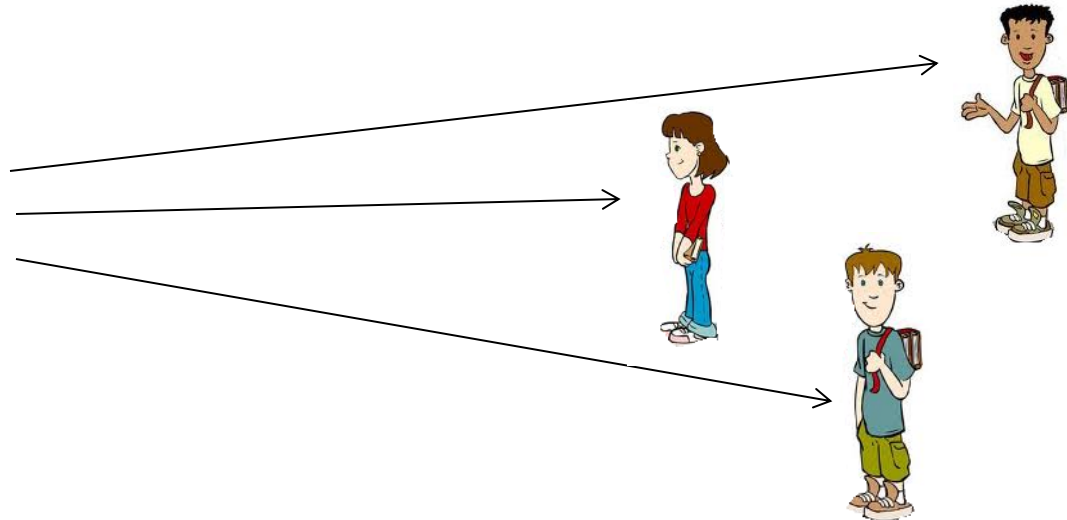
# Objetos

- ◆ Os dados (propriedades, **atributos**) de um objeto são os campos instância;
- ◆ Os procedimentos que operam os dados do objeto são os métodos;
- ◆ Um objeto específico (instância de uma classe) tem seus campos instância (valores) particulares e isto os tornam distintos de outros objetos (individualidade);
- ◆ O conjunto de valores de um objeto específico constitui o seu estado.

## Estudante

Nome
Endereço
E-mail
Notas
Imprime()
Calc_Media()
Imp_Faltas()

## Classe

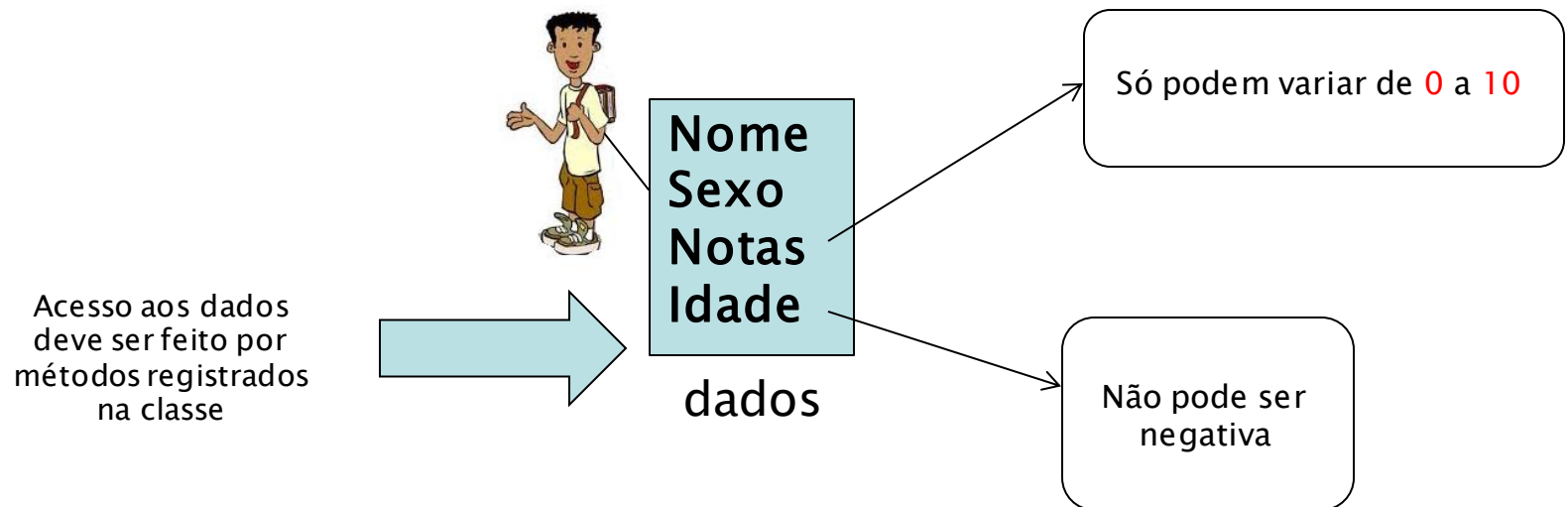


# O que significa Encapsulamento ?



# Encapsulamento

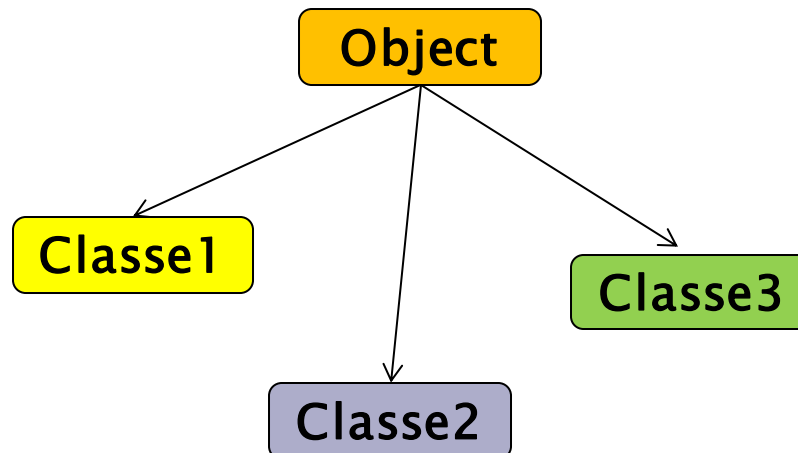
- Programas devem interagir somente com os dados do objeto por meio de métodos definidos na classe;
- Isto define uma “Caixa-preta” para acesso aos dados o qual é a chave para **reusabilidade** e **confiabilidade**.





# A mãe de todas as classes...

- **Classes** podem ser construídas por extensão de outras classes (**Herança**);
- Na plataforma **Java**, há uma super-classe chamada **Object** no qual todas as outras classes são extensões desta (**HERANÇA**).



# Características dos objetos

- **Comportamento** (Operações, **métodos** ou ações)
- **Estado** (Dados)
- **Identidade** (individualidade)



# Trabalhando com objetos

- ◆ Primeiro devemos **construí-los** (criá-los);
- ◆ Em seguida, estabelecemos o seu **estado** inicial;
- ◆ Feito isso, podemos aplicar **métodos** a estes objetos.



# Como um objeto é construído ?



# Construção de Objetos

- ◆ Em Java, usamos construtores para criar novas instâncias;
- ◆ Um construtor é um **método** especial cujo **propósito** é **criar** e **inicializar** objetos;
- ◆ Construtores têm sempre o mesmo nome da classe;
- ◆ Para criarmos objetos, combinamos o construtor com o operador **new**.





# Construtores

- ◆ Uma classe pode ter **mais** de um construtor;
- ◆ Um construtor pode ter zero, um ou mais parâmetros;
- ◆ Um construtor **nunca** retorna valor;
- ◆ Sempre é chamado pelo operador **new**.



# Construtores

- ◆ Há uma importante diferença entre construtores e outros métodos;
- ◆ Um construtor pode somente ser chamado por meio do operador **new**;
- ◆ Você **não** pode aplicar um construtor em um objeto existente para resetar os campos instância do objeto;
- ◆ Exemplo:

```
x.Estudante("André", 10, "Rua Santos, 34", 8.5); // Erro !!!
```



# Construtor – Exemplo

```
package qualit;  
  
import java.util.Date;  
  
public class ConstructObject {  
  
    public static void main(String[] args) {  
        Date x = new Date();  
        System.out.println(new Date());  
        System.out.println(new Date().toString());  
    }  
}
```





# Construtor default

◆ É um construtor sem parâmetros. Por exemplo:

```
public Estudante ( ) {  
  
}
```



# Construtor default



- ◆ Se você escrever uma classe sem construtor, então o compilador irá providenciar um construtor default;
- ◆ O construtor default **seta** todas as variáveis instância com valores default. Dados numéricos com zero, dados booleanos com false e variáveis objetos (referências) com null;
- ◆ Se uma classe fornece ao menos um construtor mas não fornece o default, então será ilegal construir-se objetos sem parâmetros de construção.



# Overloading

- ◆ Uma classe pode ter diversos construtores;
- ◆ **Overloading** ocorre quando diversos métodos têm o mesmo nome, mas **diferentes parâmetros**;
- ◆ Exemplo:

```
public Estudante ( ) {  
    ...  
}
```



```
public Estudante ( String Nome, double NotaExame) {  
    ...  
}
```



# Construtor chamando Construtor

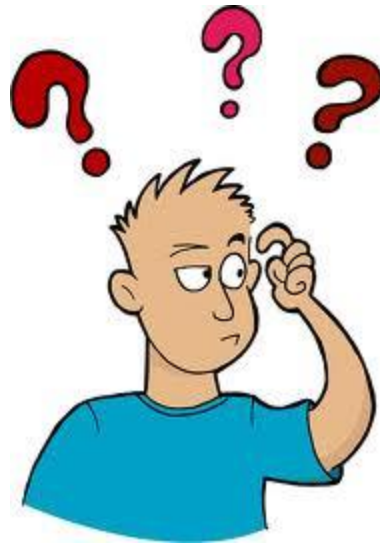
- ◆ Se o primeiro comando de um construtor tiver a forma **this(...)**, então o construtor estará chamando outro construtor da mesma classe.



```
public Estudante( double  NotaExame)  {  
  
    this ("Estudante padrao", ...); // chamada construtor  
  
}
```



# Objetos e Variáveis Objeto são a mesma coisa ?



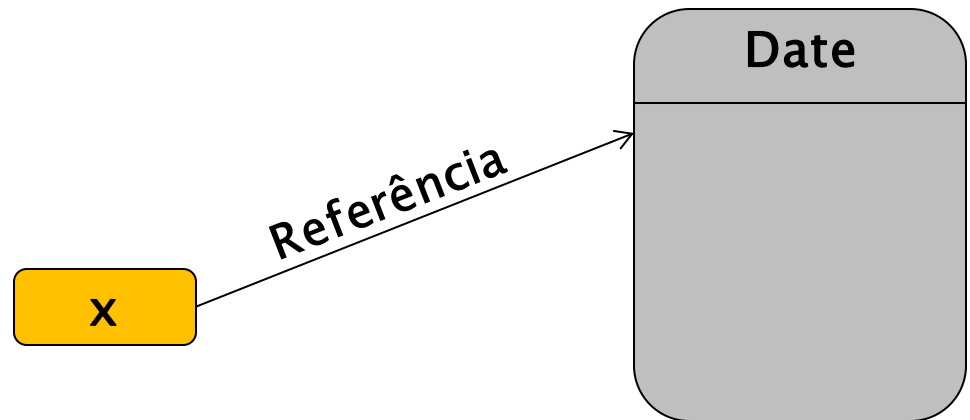
# Cuidado...

```
Date x;           // x ainda não aponta para nenhum objeto...
```

```
String S = x.toString() ;           // erro de compilação !!!
```

x é uma variável que pode fazer referência à um objeto do tipo **Date** !

x não é um objeto!!!



# Exercício

Criar uma classe Java chamada **Aluno** para modelar estudantes. A classe deve possuir os seguintes atributos de dados (propriedades):

nome – Nome do estudante - (tipo String)  
codmat - Código de Matrícula – (tipo int)  
cpf – (tipo String)  
sexo – (tipo char)  
notaP1 – (float)  
notaP2 – (float)  
notaP3 – (float)

A classe deve conter métodos para construir objetos e um método **ImprimeAluno()** que irá imprimir os dados do estudante. Adicionalmente a classe deve conter um método chamado **ImprimeSexo()** que irá imprimir “Masculino” se o sexo for ‘M’ e “Feminino” se o sexo for ‘F’.

A função **ImprimeSexo()** também imprime o nome do estudante.

Codificar também a função **MediaAluno()** que retorna a média aritmética das duas maiores notas dentre as notas P1, P2 e P3. (Exemplo: notas 2, 6 e 8 => considerar para a média as notas 6 e 8. )

Finalmente, codificar a função **Resultado()** que retorna “Aprovado” se a média for  $\geq 6.0$  ou “Reprovado” se a média for inferior a 6.0.

A classe deve ser criada dentro de um package chamado **uscs**.



```
package uscs;  
  
import java.util.Arrays;  
  
public class Aluno {  
  
    public String nome;  
    public int codmat;  
    public String cpf;  
    public char sexo;  
    public float notaP1;  
    public float notaP2;  
    public float notaP3;  
}
```





```
public Aluno(String nome, int codmat, String cpf, char sexo,
              float notaP1, float notaP2, float notaP3) {
```

```
    this.nome = nome;
    this.codmat = codmat;
    this.cpf = cpf;
    this.sexo = sexo;
    this.notaP1 = nota_P1;
    this.notaP2 = nota_P2;
    this.notaP3 = nota_P3;
```

```
}
```


```
public Aluno() {
```

```
    this.nome = null;
    this.codmat = 0;
    this.cpf = null;
    this.sexo = ' ';
    this.notaP1 = 0.0F;
    this.notaP2 = 0.0F;
    this.notaP3 = 0.0F;
```

```
}
```



```
public void imprimeAluno() {  
  
    System.out.println (  
  
        "Nome: " + this.nome + "\n" +  
        "Código de matrícula: " + this.codmat + "\n" +  
        "CPF: " + this.cpf + "\n" +  
        "Sexo: " + this.sexo + "\n" +  
        "Nota P1: " + this.notaP1 + "\n" +  
        "Nota P2: " + this.notaP2 + "\n" +  
        "Nota P3: " + this.notaP3 + "\n"  
  
    );  
}
```



```
public void imprimeSexo() {  
  
    System.out.println("Nome: " + this.nome);  
    if (this.sexo == 'M')  
        System.out.println ("Masculino");  
    else {  
        if (this.sexo == 'F')  
            System.out.println ("Feminino");  
        else  
            System.out.println ("Sexo inválido...");  
    }  
}
```



```
public float MediaAluno() {  
  
    float[] tabNotas = new float[3];  
  
    tab_notas[0] = this.notaP1;  
    tab_notas[1] = this.notaP2;  
    tab_notas[2] = this.notaP3;  
  
    Arrays.sort(tabNotas);  
  
    return (tabNotas[1] + tabNotas[2] ) /2;  
}  
  
public void imprimeMedia() {  
  
    System.out.println ( "Média : " + this.MediaAluno());  
}
```



```
public String Resultado() {  
  
    if (this.MediaAluno() >= 6.0F)  
        return ("Aprovado...");  
    else return ("Reprovado...");  
}  
  
}
```



# Exercício

Criar uma classe Java TesteAluno que possui um método main() para instanciar estudantes.

Criar um objeto referenciado pela variável X1 por meio do construtor com os parâmetros: nome = “Paulo”, codmat = 55123, cpf=”800912345-12”, sexo = ‘M’, notaP1 = 7.0, notaP2=6.0 e notaP3 = 8.0.

Criar um segundo objeto referenciado pela variável X2 por meio do construtor com os parâmetros: nome = “Ana”, codmat = 991239, cpf=”500876123-15”, sexo = ‘F’, notaP1 = 2.0, notaP2=6.0 e notaP3 = 9.0.

Para cada objeto chamar a função ImprimeAluno() para imprimir os dados dos dois objetos criados, e as funções MediaAluno() e Resultado().

Executar para cada objeto criado a função ImprimeSexo().

A classe deve ser criada dentro de um package chamado uscs.



```
package uscs;

public class TesteAluno {

    public static void main(String[] args) {

        Aluno X1 = new Aluno("Paulo", 55123, "800912345-12", 'M',
                               7.0F, 6.0F, 8.0F);

        Aluno X2 = new Aluno("Ana", 991239, "500876123-15", 'F',
                               2.0F, 6.0F, 9.0F);

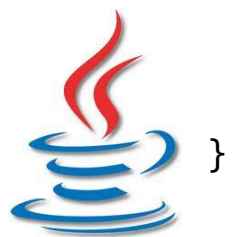
        System.out.println( "\n\n-----" ) ;

        X1.imprimeAluno();
        X1.imprimeMedia();
        X1.imprimeSexo();

        System.out.println( "\n\n-----" ) ;

        X2.imprimeAluno();
        X2.imprimeMedia();
        X2.imprimeSexo();

    }
}
```



# Exercício

Criar uma classe Java chamada Curso para modelar cursos. A classe deve possuir os seguintes atributos de dados (propriedades):

- cod\_curso – Código do Curso – (tipo int)
- desc\_curso – Descrição do Curso – (tipo String)
- duracao\_semestre – Total de semestres do curso (tipo int)

A classe deve conter métodos para construir objetos (um construtor com parâmetros e outro default) e um método Imprime\_Curso() que irá imprimir os dados do curso.

Criar uma classe Java chamada Disciplina para modelar disciplinas. A classe deve possuir os seguintes atributos de dados (propriedades):

- cod\_disciplina – Código da Disciplina – (tipo int)
- desc\_disciplina – Descrição da Disciplina – (tipo String)

A classe deve conter métodos para construir objetos (um construtor com parâmetros e outro default) e um método Imprime\_Disciplina() que irá imprimir os dados da disciplina.





```
package uscs;
```



```
public class Curso {
```

```
    int cod_curso;
```

```
    String desc_curso;
```

```
    int duracao_semestre;
```

```
    public Curso() {
```

```
        cod_curso = 0;
```

```
        desc_curso = null;
```

```
        duracao_semestre = 0;
```

```
    }
```

```
    public Curso(int c, String d, int du){
```

```
        cod_curso = c;
```

```
        desc_curso = d;
```

```
        duracao_semestre = du;
```

```
    }
```

```
    public void Imprime_Curso() {
```

```
        System.out.println("----- Funcao Imprime_Curso() -----");
```

```
        System.out.println("Codigo do curso: " + cod_curso);
```

```
        System.out.println("Descricao do curso: " + desc_curso);
```

```
        System.out.println("Duracao do curso em semestres: " + duracao_semestre);
```

```
        System.out.println("-----\n");
```

```
    }
```

```
}
```



```
package uscs;
```

```
public class Disciplina {
```

```
    int cod_disciplina;
```

```
    String desc_disciplina;
```

```
    public Disciplina() {
```

```
        cod_disciplina = 0;
```

```
        desc_disciplina = null;
```

```
    }
```

```
    public Disciplina(int c , String d) {
```

```
        cod_disciplina = c;
```

```
        desc_disciplina = d;
```

```
    }
```

```
    public void Imprime_Disciplina() {
```

```
        System.out.println("---- Funcao Imprime_Disciplina ----");
```

```
        System.out.println("Codigo da disciplina: " + cod_disciplina);
```

```
        System.out.println("Descricao da disciplina: " + desc_disciplina);
```

```
        System.out.println("-----\n");
```

```
    }
```

```
}
```



# Exercício



4. Criar uma classe Java chamada Professor para modelar professores. A classe deve possuir os seguintes atributos de dados (propriedades):

cod\_professor – Código do Professor – (tipo int)

nome\_prof – Nome do Professor – (tipo String)

email\_prof – Email do Professor – (tipo String)

fone\_prof – Fone do Professor – (tipo String)

A classe deve conter métodos para construir objetos (um construtor com parâmetros e outro default) e um método Imprime\_Professor() que irá imprimir os dados do professor.



package uscs;

public class Professor {

int cod\_prof;

String nome\_prof;

String email\_prof;

String fone\_prof;

public Professor() {

cod\_prof = 0;

nome\_prof = null;

email\_prof = null;

fone\_prof = null;

}

public Professor(int c, String n, String e, String f) {

cod\_prof = c;

nome\_prof = n;

email\_prof = e;

fone\_prof = f;

}

public void Imprime\_Professor() {

System.out.println("---- Funcao Imprime\_Professor() ----");

System.out.println("Codido do Professor: " + cod\_prof);

System.out.println("Nome do Professor: " + nome\_prof);

System.out.println("Email do Professor: " + email\_prof);

System.out.println("Fone do Professor: " + fone\_prof);

System.out.println("-----\n");

}

}



# Exercício

5. Criar uma classe Java chamada Teste\_Escola com um método **main()** para instanciar objetos.

Criar um array de cursos e armazenar em cada posição os seguintes cursos:  
(1, Ciência da Computação,8);(2,Sistemas de Informação,8); (3, Redes,5);  
(4,Sistemas para Internet,5); (5, Gestão de TI,5)

Criar um array de professores e armazenar em cada posição os seguintes professores:

(10,Paulo,pg@uol.com.br, 4330-4578); (15,Alfredo,am@uol.com.br, 5578-9812);  
(20,Marcos,ms@uol.com.br, 4125-9976); (25,Ana,an@bol.com.br, 5678-1145)

Criar um array de disciplinas e armazenar em cada posição as seguintes disciplinas:

(100,Estatística); (110,Algoritmos); (120,Redes); (130,Álgebra)

Escrever uma rotina que leia os arrays e imprima os dados de todos os objetos contidos em cada array.

Obs. As duas classes devem ser criadas dentro de um package chamado uscs.



```
package uscs;
```

```
public class Teste_Escola {
```

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

```
        Curso[] tab_curso = new Curso[5];
```

```
        tab_curso[0] = new Curso(1, "Ciência da Computação",8);
```

```
        tab_curso[1] = new Curso(2, "Sistemas de Informação",8);
```

```
        tab_curso[2] = new Curso(3, "Redes de Computadores", 5);
```

```
        tab_curso[3] = new Curso(4, "Sistemas para Internet", 5);
```

```
        tab_curso[4] = new Curso(5, "Gestão de TI ", 5);
```

```
        for (int i=0; i < tab_curso.length; i++) {
```

```
            tab_curso[i].Imprime_Curso();
```

```
        }
```



```
Professor[] tab_prof= new Professor[4];  
    tab_prof[0] = new Professor(10,"Paulo","pg@uol.com.br", "4330-4578");  
    tab_prof[1] = new Professor(15,"Alfredo","am@uol.com.br", "5578-9812");  
    tab_prof[2] = new Professor(20,"Marcos","ms@uol.com.br", "4125-9976");  
    tab_prof[3] = new Professor(25,"Ana","an@bol.com.br", "5678-1145");  
  
    for (int i=0; i <tab_prof.length; i++) {  
        tab_prof[i].Imprime_Professor();  
    }
```



```
Disciplina[] tab_disciplina = new Disciplina[4];  
    tab_disciplina[0] = new Disciplina(100, "Estatística");  
    tab_disciplina[1] = new Disciplina(110, "Algoritmos");  
    tab_disciplina[2] = new Disciplina(120, "Redes");  
    tab_disciplina[3] = new Disciplina(130, "Álgebra");  
  
    for (int i=0; i < tab_disciplina.length; i++) {  
        tab_disciplina[i].Imprime_Disciplina();  
    }  
}
```





# null

- ◆ Uma variável pode receber o valor **null**, indicando que **não** faz referência à um objeto.



```
import java.util.Date;

public class Const_Object {

    public static void main(String[] args) {

        Date x = null;

        String S = x.toString(); // Runtime error...

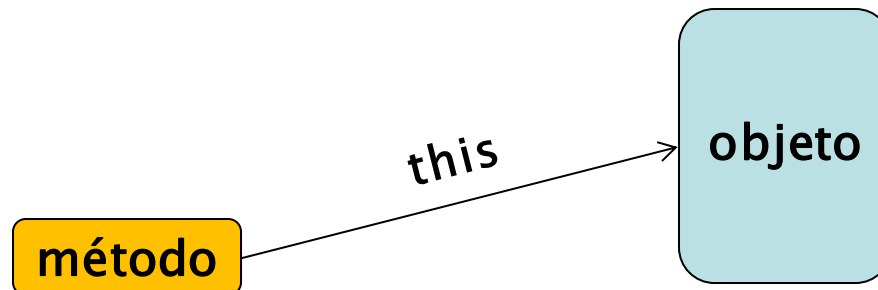
    }

}
```

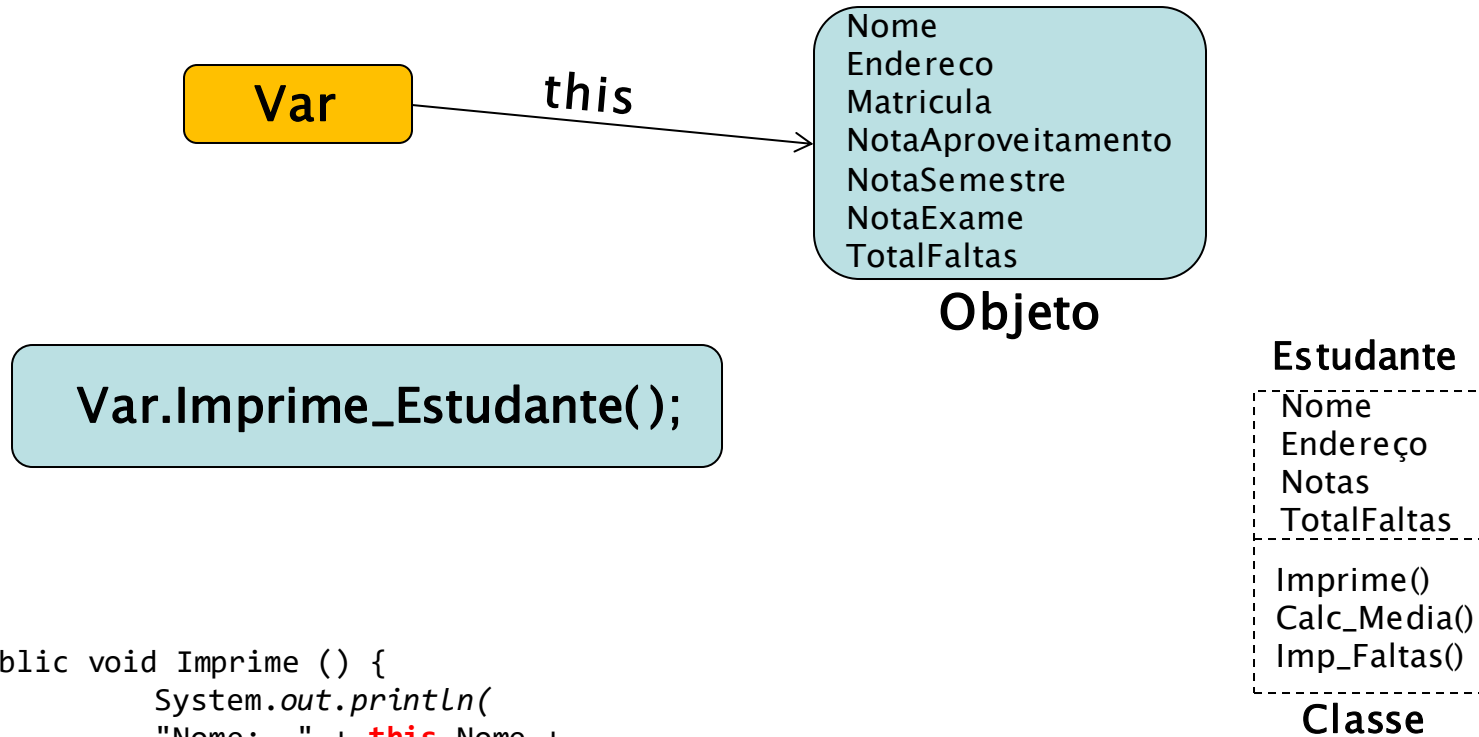


# Parâmetro implícito

- ◆ Métodos **instância** operam nos objetos e acessam seus dados instância;
- ◆ Implicitamente, métodos instância estão acessando o objeto e seus dados;
- ◆ O parâmetro implícito não aparece na declaração do método;
- ◆ Para que este parâmetro seja explicitado, podemos usar a keyword **this**.



# Keyword this



```
public void Imprime () {  
    System.out.println(  
        "Nome: " + this.Nome +  
        "Endereco: " + this.Endereco +  
        "Matricula: " + this.Matricula +  
        "Nota de Aproveitamento: " + this.Nota_Aproveitamento +  
        "Nota do Semestre: " + this.Nota_Semestre +  
        "Nota do Exame: " + this.Nota_Exame +  
        "Total de Faltas: " + this.Total_faltas);  
}
```



# Definindo classes

## Variáveis Instância



- ◆ Uma classe modela objetos;
- ◆ Estes objetos têm atributos de dados;
- ◆ Na classe, os atributos são definidos por variáveis;
- ◆ Variáveis instância são variáveis associadas aos atributos de um objeto;
- ◆ Cada instância da classe (objeto) contém sua própria cópia destas variáveis instância;
- ◆ Estas variáveis permitem a diferenciação entre os objetos criados num programa, dando-lhes sua individualidade.



# Variáveis Instância

```
public class Estudante {
```

```
    String    Nome;  
    String    Endereco;  
    int       Matricula;  
    double    Nota_Aproveitamento;  
    double    Nota_Semestre;  
    double    Nota_Exame;  
    int       Total_faltas;
```



Para cada objeto instanciado, estas variáveis serão preenchidas com os valores particulares do objeto criado...



# Definindo classes

## Variáveis de Classe

- ◆ São variáveis associadas com a classe;
- ◆ Há somente uma cópia de cada uma destas variáveis independentemente do número de objetos criados pelo programa;
- ◆ Os valores associados a estas variáveis existem na memória, mesmo se nenhum objeto ainda tiver sido criado pelo programa;
- ◆ Ou seja, estas variáveis são independentes da existência de objetos.



# Definindo classes

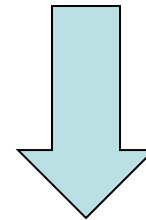
## Variáveis de Classe

- ◆ Se o valor de uma variável de classe é modificado, então o novo valor estará disponibilizado para todos os objetos da classe;
- ◆ Uma variável de classe deve ser definida com a keyword **static** precedendo o nome.



# Variáveis de Classe

```
public class PedidoCompra{  
  
    String                nomeCliente;  
  
    String                enderecoCliente;  
  
    static double         taxaDolar = 1.85;  
  
    ...  
}
```



A taxa do dolar é compartilhada entre todos os objetos da classe. Esta informação existe mesmo se nenhum pedido de compra for instanciado.



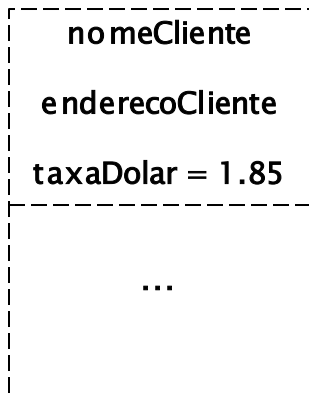


# Variáveis de Classe

nomeCliente = YZX

enderecoCliente = Av. Tao, 35

**PedidoCompra**



**PEDIDO 1**

**PEDIDO 2**

nomeCliente = ABC

enderecoCliente = Rua X, 10

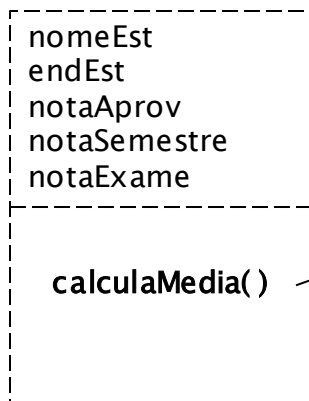
**taxaDolar=1.85**



# Métodos Instância

- ◆ São os **métodos** que somente podem ser executados quando **houver alguma instância (objeto) criado**;
- ◆ Portanto, estão associados à um objeto em particular;
- ◆ Se **não** houver objetos, estes métodos **não** podem ser executados;

## Estudante



O método **calculaMedia()** somente pode ser executado com os dados de um Estudante em particular. Do contrário, não haverá notas para o cálculo da média...



# Métodos de Classe

- ◆ São métodos que pertencem a classe;
- ◆ Podem ser executados mesmo quando não existirem objetos;
- ◆ São declarados através da keyword **static**.



# Método `main()`

- ⊕ Antes de uma aplicação iniciar, certamente não existem objetos;
- ⊕ Assim, para iniciarmos a aplicação será necessário iniciar um método de classe;
- ⊕ Este método é o método **`main()`**.
- ⊕ Portanto, **`main()`** é sempre um método declarado como **`static`**.



# Cuidado...

- ◆ Variável **static** não significa que o dado é constante. (A taxa do dolar pode variar...).
- ◆ Para definir o dado como constante:

```
static final double taxaDolar = 1.85;
```



# Definindo classes

```
package qualit;  
  
public class Estudante {  
    String Nome;  
    String Endereco;  
    int Matricula;  
    double Nota_Aproveitamento;  
    double Nota_Semestre;  
    double Nota_Exame;  
    int Total_faltas;  
}
```



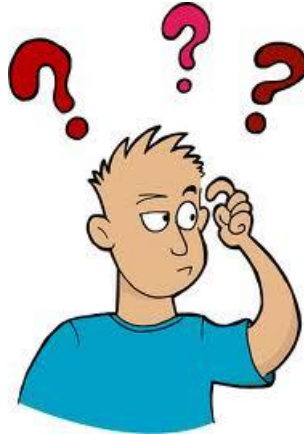
```
public Estudante(String nome,  
    String endereco,  
    int matricula,  
    double nota_Aproveitamento,  
    double nota_Semestre,  
    double nota_Exame,  
    int total_faltas) {  
  
    Nome = nome;  
    Endereco = endereco;  
    Matricula = matricula;  
    Nota_Aproveitamento = nota_Aproveitamento;  
    Nota_Semestre = nota_Semestre;  
    Nota_Exame = nota_Exame;  
    Total_faltas = total_faltas;  
}
```



```
public double Calcula_Media() {  
    return (this.Nota_Aproveitamento + this.Nota_Semestre)/2.0;  
}  
  
public void Imprime_Estudante() {  
    System.out.println(  
        "Nome: " + this.Nome +  
        "Endereco: " + this.Endereco +  
        "Matricula: " + this.Matricula +  
        "Nota de Aproveitamento: " + this.Nota_Aproveitamento +  
        "Nota do Semestre: " + this.Nota_Semestre +  
        "Nota do Exame: " + this.Nota_Exame +  
        "Total de Faltas: " + this.Total_faltas);  
}  
}
```





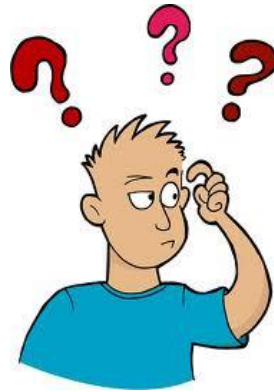


Quantos métodos tem a classe Estudante ?

Quantas variáveis instância tem a classe Estudante ?



# Eu consigo executar a classe Estudante ?



# Executando a classe Estudante...

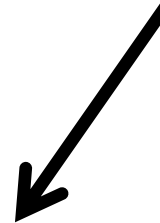
```
package qualit;  
  
public class Test_Estudante {  
  
    public static void main(String[] args) {  
  
        Estudante A = new Estudante("Paulo",  
                                     "Rua Brasil, 10 ",  
                                     55670,  
                                     5.7 ,  
                                     5.9 ,  
                                     8.0 ,  
                                     6 );  
  
        A.Imprime_Estudante();  
        System.out.println(A.Calcula_Media());  
  
    }  
}
```



# Como altero a nota do estudante?

```
package qualit;  
  
public class Test_Estudante {  
  
    public static void main(String[] args) {  
  
        Estudante A = new Estudante("Paulo",  
                                     "Rua Brasil, 10 ",  
                                     55670,  
                                     5.7 ,  
                                     5.9 ,  
                                     8.0 ,  
                                     6 );  
  
        A.Nota_Aproveitamento = 9.9;  
  
        A.Imprime_Estudante();  
        System.out.println(A.Calcula_Media());  
  
    }  
}
```

**Alteração  
feita  
diretamente  
nos dados  
do objeto...**



**A.Nota\_Aproveitamento = 9.9;**



# Que princípio foi violado ? ? ?



# Princípio do Encapsulamento....



# Sem encapsulamento...

- ◆ Os dados do objeto poderiam ter estado inválido. Por exemplo, a nota do estudante não pode ser negativa. Mas o programa abaixo é **legal**...

```
package qualif;
```

```
public class Test_Estudante {
```

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

```
    Estudante A = new Estudante("Paulo",  
                                "Rua Brasil, 10 ", 55670, 5.7 , 5.9 , 8.0 , 6 );
```

```
    A.Nota_Aproveitamento = -1.5
```

```
    A.Imprime_Estudante();
```

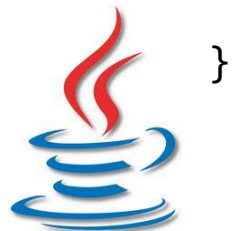
```
    System.out.println(A.Calcula_Media());
```

```
}
```

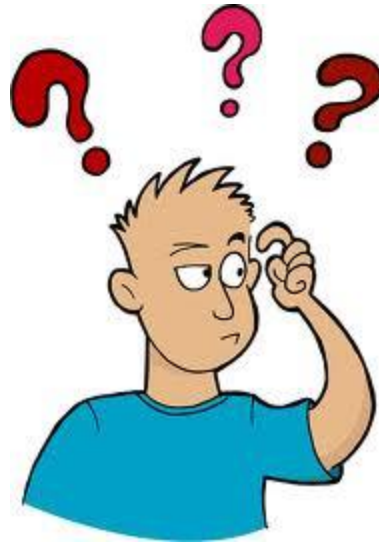
```
}
```



**Nota  
Inválida...**



Então, como implementar o  
Encapsulamento ? ? ?





# Keyword private

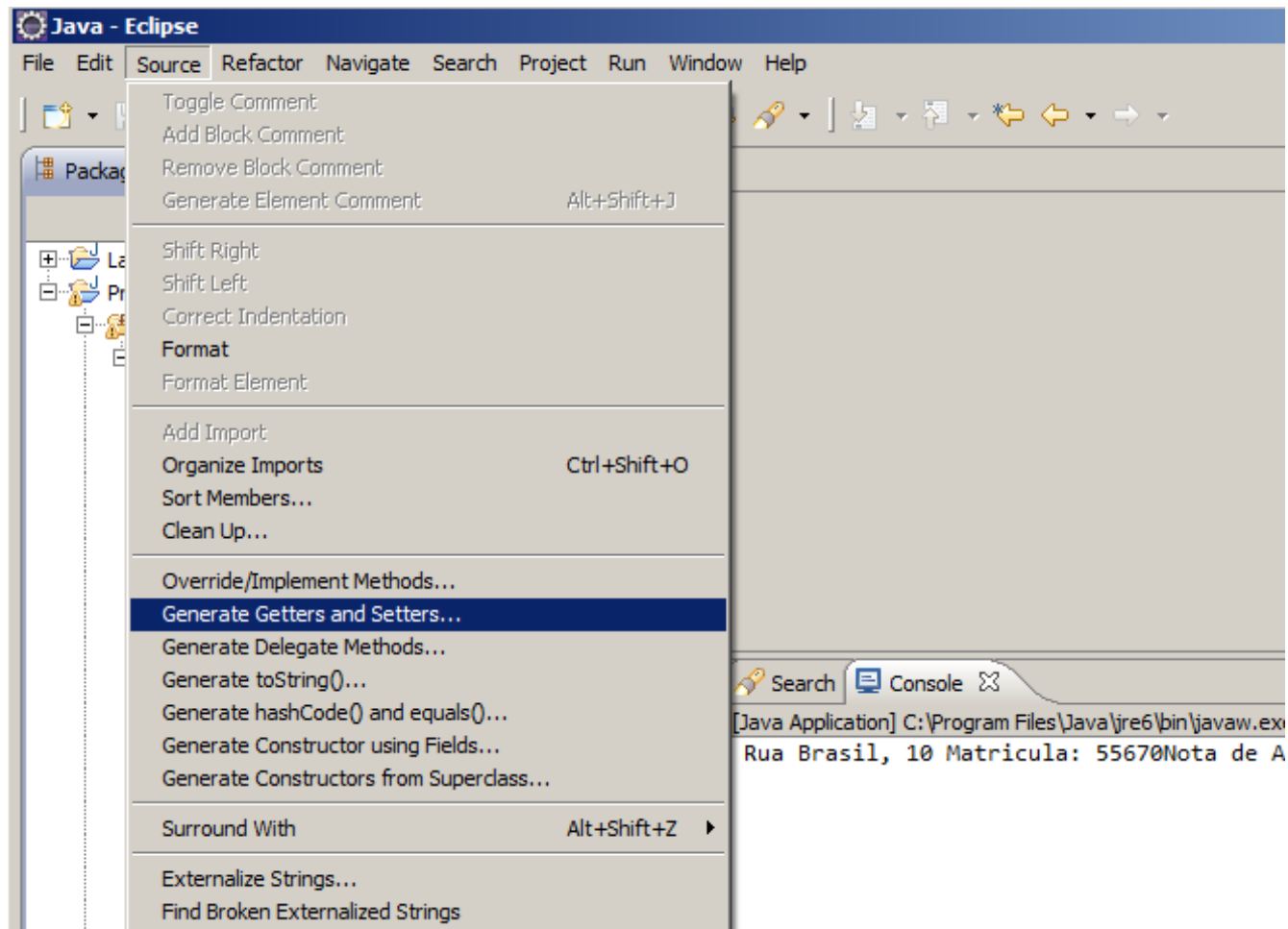
```
package qualif;  
  
public class Estudante {  
  
    private String Nome;  
    private String Endereco;  
    private int Matricula;  
    private double Nota_Aproveitamento;  
    private double Nota_Semestre;  
    private double Nota_Exame;  
    private int Total_faltas;  
}
```

- ◆ A keyword **private** assegura que os **únicos métodos** que podem acessar **diretamente** os campos instância do objeto são os **métodos da própria classe**.



# Getters e Setters

- ◆ São métodos auxiliares que provêm a interface aos dados do objeto;
- ◆ A IDE Eclipse automaticamente gera os métodos **Getters** e **Setters**.



# Getters e Setters

**Generate Getters and Setters**

Select getters and setters to create:

<input checked="" type="checkbox"/>	<input type="checkbox"/>	Endereco
<input checked="" type="checkbox"/>	<input type="checkbox"/>	Matricula
<input checked="" type="checkbox"/>	<input type="checkbox"/>	Nome
<input checked="" type="checkbox"/>	<input type="checkbox"/>	Nota_Aproveitamento
<input checked="" type="checkbox"/>	<input type="checkbox"/>	Nota_Exame
<input checked="" type="checkbox"/>	<input type="checkbox"/>	Nota_Semestre
<input checked="" type="checkbox"/>	<input type="checkbox"/>	Total_faltas

Select All  
Deselect All  
Select Getters  
Select Setters

☐ Allow setters for final fields (remove 'final' modifier from fields if necessary)

Insertion point:  
After 'Estudante(String, String, int, double, double, double, int)'

Sort by:  
Fields in getter/setter pairs

Access modifier  
☒ public ☐ protected ☐ default ☐ private  
☐ final ☐ synchronized

☐ Generate method comments

The format of the getters/setters may be configured on the [Code Templates](#) preference page.

i 14 of 14 selected.

OK Cancel



# Getters e Setters



```
public String getNome() {  
    return Nome;  
}
```

```
public void setNome(String nome) {  
    Nome = nome;  
}
```

```
public String getEndereco() {  
    return Endereco;  
}
```

```
public void setEndereco(String endereco) {  
    Endereco = endereco;  
}
```



# Alteração da nota do estudante

```
package qualif;  
  
public class Test_Estudante {  
  
    public static void main(String[] args) {  
  
        Estudante A = new Estudante("Paulo",  
                                     "Rua Brasil, 10 ", 55670, 5.7 , 5.9 , 8.0 , 6 );  
  
        A.setNota_Aproveitamento(9.9);  
  
        A.Imprime_Estudante();  
        System.out.println(A.Calcula_Media());  
  
    }  
}
```

A função `setNota_Aproveitamento()` deverá  
prover o código de consistência do campo...



# Keyword private

- ◆ Recomenda-se que todos os campos instância sejam definidos com **private** para atender ao conceito de **encapsulamento**;
- ◆ **Métodos** em geral são declarados com **public**. No entanto, pode-se definir métodos **private** em certas circunstâncias. Em algumas situações, pode-se quebrar o código em métodos separados para auxiliar funcionalidades (**helper**).



# Variáveis Locais

```
double media(double valor1, double valor2)  {  
  
    ➡ double resultado = (valor1 + valor2) / 2.0;  
  
    return resultado;  
}
```

- @ A variável **resultado** declarada no método **media( )** somente tem vida dentro do corpo do método;
- @ A variável **resultado** é criada cada vez que o método **media( )** for executado e é destruída quando a execução do método for encerrada;
- @ As variáveis locais **não** são inicializadas automaticamente e são aderentes às regras de escopo;



# Passagem de Parâmetros

- Ⓢ Em **Java**, qualquer passagem de parâmetros é feita por **valor**;
- Ⓢ Isso se aplica tanto para passagem de dados primitivos quanto para objetos;
- Ⓢ O método que recebe a informação recebe uma **cópia do dado** quando for tipo primitivo ou uma **cópia da referência** quando for um objeto;
- Ⓢ Para dados primitivos, quando um método modifica o valor de um parâmetro, na verdade ele modifica o valor da cópia que a ele é passada (valor original intacto);
- Ⓢ Para passagem de objetos, a referência é copiada e, **de forma indireta**, os dados do objeto são alterados.

```
public static void Func (double x) {  
    x = 10.5 + x; // operacao na cópia do dado  
}
```





# Passagem de Parâmetros



```
public class PassParam {  
  
    public static void main(String[] args) {  
  
        double x = 30. ;  
  
        System.out.println("X antes de chamar a função: " + x);  
  
        Dobro(x); // a função opera com uma cópia de x  
  
        System.out.println("X após a chamada da função: " + x);  
  
        System.out.println(x);  
    }  
  
    public static double Dobro ( double par) {  
  
        return (par * 2.) ;  
  
    }
```



# Passagem de Parâmetros

```
public class PassPar2 {  
  
    public static void main(String[] args) {  
        Estudante A = new Estudante("Paulo",  
            "Rua Brasil, 10 ", 55670, 5.7 , 5.9 , 8.0 , 6 );  
  
        MudaNota(A);  
  
        A.Imprime_Estudante();  
        System.out.println(A.Calcula_Media());  
  
    }  
  
    public static void MudaNota(Estudante x) {  
        x.setNota_Exame(10.);  
        x.setTotal_faltas(32);  
    }  
}
```



# Quais as formas de se inicializar uma variável ?



# Inicialização de Variáveis

- ◆ Setando um valor no **construtor**;
- ◆ Efetuando **atribuições** internas a métodos.



# Inicialização de Variáveis

- ◆ No entanto, há uma terceira forma de **inicialização** de variáveis;
- ◆ São os **BLOCOS de INICIALIZAÇÃO**.

```
{  
  
    . . .  
  
}
```



# Blocos de Inicialização

- ◆ São blocos de código escritos nas declarações de classe;
- ◆ Estes blocos são executados quando um objeto da classe é instanciado.
- ◆ O bloco de inicialização é primeiramente executado e em seguida executa-se o código do construtor;
- ◆ Este mecanismo não é usualmente empregado. Em geral, é mais prático escrever o código diretamente no construtor.

```
{  
    ... código  
}
```



```
package qualit;
```

```
public class Empregado {
```

```
    public Empregado(String nome, double salario, String funcao) {  
        this.nome = nome;  
        this.salario = salario;  
    }
```

```
    public Empregado() {  
        this.nome = "";  
        this.salario = 0.0;  
    }
```

```
    private static int proxId;
```

```
    private int codFunc;  
    private String nome;  
    private double salario;
```

```
// Codigo de inicializacao  
{  
    this.codFunc = proxId;  
    proxId++;  
}
```

## Blocos de Inicialização



O que ocorre quando um construtor é chamado ?





# Passos durante a chamada do Construtor

- ◆ Todos os campos de dados são **inicializados** com seus valores **default** (0, false ou null);
- ◆ Os blocos de **inicialização** são executados, na ordem em que ocorrem na declaração da classe;
- ◆ Se na primeira linha do construtor houver chamada de outro construtor, então o **corpo** do segundo construtor é chamado;
- ◆ O corpo do construtor é executado.



# Como inicializamos um campo **static** ?



# Inicialização de área **static**

- ◆ Por meio de **atribuição** do seu valor inicial ou por meio de bloco de inicialização .

Por exemplo: `private static int proxId=0;`

- ◆ Para bloco de inicialização , defina o código dentro de um bloco circundado pela keyword **static**.

Por exemplo: 

```
static {  
    Random val = new Random() ;  
    int proxId = val.nextInt(1000) ;  
}
```

- ◆ Obs. A inicialização **static** ocorre quando a **classe** é **inicialmente carregada**.



# Um Hello World diferente...



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



# Um Hello World diferente...



```
package USCS;

public class HelloWorldDiferente {
    //Bloco de Inicialização static
    //código que é executado logo que a classe é
    //carregada para a memória
    static {
        System.out.println("Hello World...antes");
    }

    //função main()
    public static void main(String[] args) {
        System.out.println("Hello World...depois");
    }
}
```



## Controlando acesso a membros de classes

- ◆ A acessibilidade de variáveis e métodos de classe é feita através de atributos de acesso;
- ◆ Você pode ter **quatro** possibilidades ao especificar um atributo de acesso para um membro de classe;

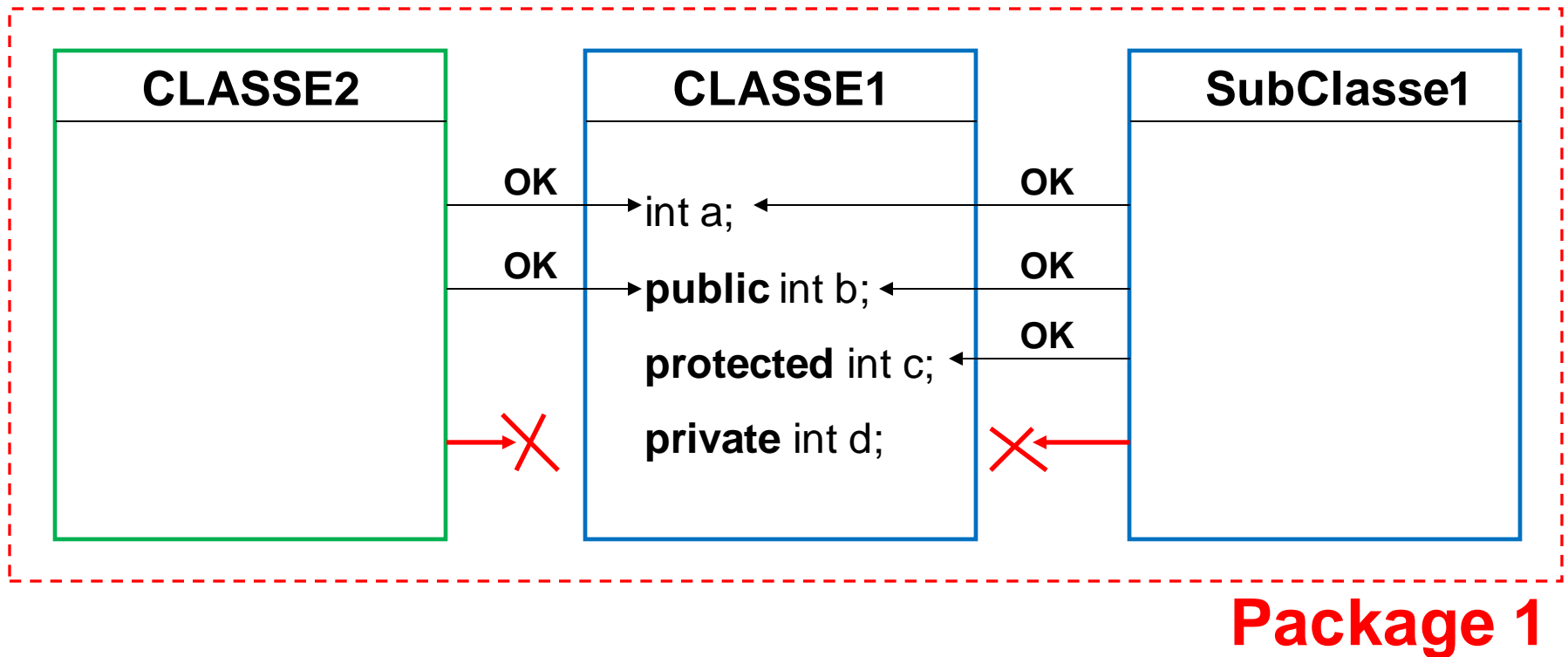


# Controlando acesso a membros de classes

<b>Atributo</b>	<b>Acesso Permitido</b>
<b>Nenhum atributo de acesso definido</b>	de qualquer classe dentro do mesmo package.
<b>public</b>	de qualquer classe de qualquer lugar.
<b>private</b>	nenhum acesso fora da classe.
<b>protected</b>	de qualquer classe dentro do package e de qualquer subclasse.



# Controlando acesso a membros de classes





# Destruição de Objetos

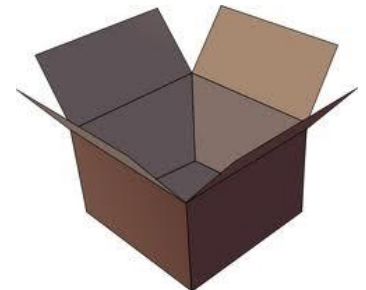


- ◆ **Java** faz regeneração (aproveitamento) de memória de forma automática;
- ◆ Isto é feito automaticamente pelo procedimento **Garbage Collector**;
- ◆ No entanto, você pode – a qualquer momento – adicionar o método **finalize()** antes do **Garbage Collector** proceder a limpeza de objetos em memória;
- ◆ Na **prática**, a ação do método **finalize()** é duvidosa, pois não se sabe ao certo o momento em que a máquina virtual efetua a chamada do **Garbage Collector**.

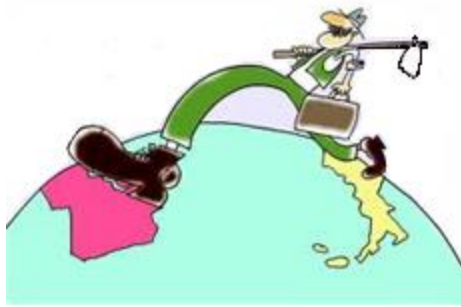


# Packages

- ❖ Java é orientado a **packages**;
- ❖ Java permite que suas **classes** sejam agrupadas em coleções chamadas **packages**;
- ❖ **Packages** são formas convenientes para organizar seu trabalho e separar seu código do código produzido por outros programadores;
- ❖ A **API Java** é distribuída em diversos packages: ***java.lang***, ***java.util***, ***java.net***, etc.
- ❖ Todos os **packages** standards **Java** estão dentro de packages hierárquicos ***java*** e ***javax***.



# Importação de Classes

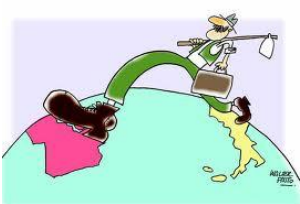


- ◆ Uma classe pode usar todas as classes de seu próprio **package** e todas as classes **public** de outros packages.
- ◆ Podemos acessar as classes públicas em outro **package** de duas formas:
  - A primeira é simplesmente adicionar o nome do package em frente ao nome da classe. Exemplo:

```
java.util.Date    var    =    new    java.util.Date();
```

- A segunda forma é por meio da diretiva **import** .





# Importação de Classes

```
package qualit;
```

```
public class HelloWorld {
```

```
    static {
```

```
        System.out.println("Hello World...");
```

```
    }
```

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

```
        java.lang.String x = new java.lang.String("xxx");
```

```
        java.lang.System.out.println(x);
```

```
    }
```

```
}
```





# A diretiva import

- ❖ Ao utilizar **import** não é necessário mais usar o caminho completo de uma classe existente em um determinado package;
- ❖ Pode-se importar uma classe específica ou todo o package;
- ❖ A diretiva **import** é codificada no topo do arquivo fonte e após o comando **package**.

❖ Exemplo:

```
import    java.util.*; //menos tedioso
```

ou

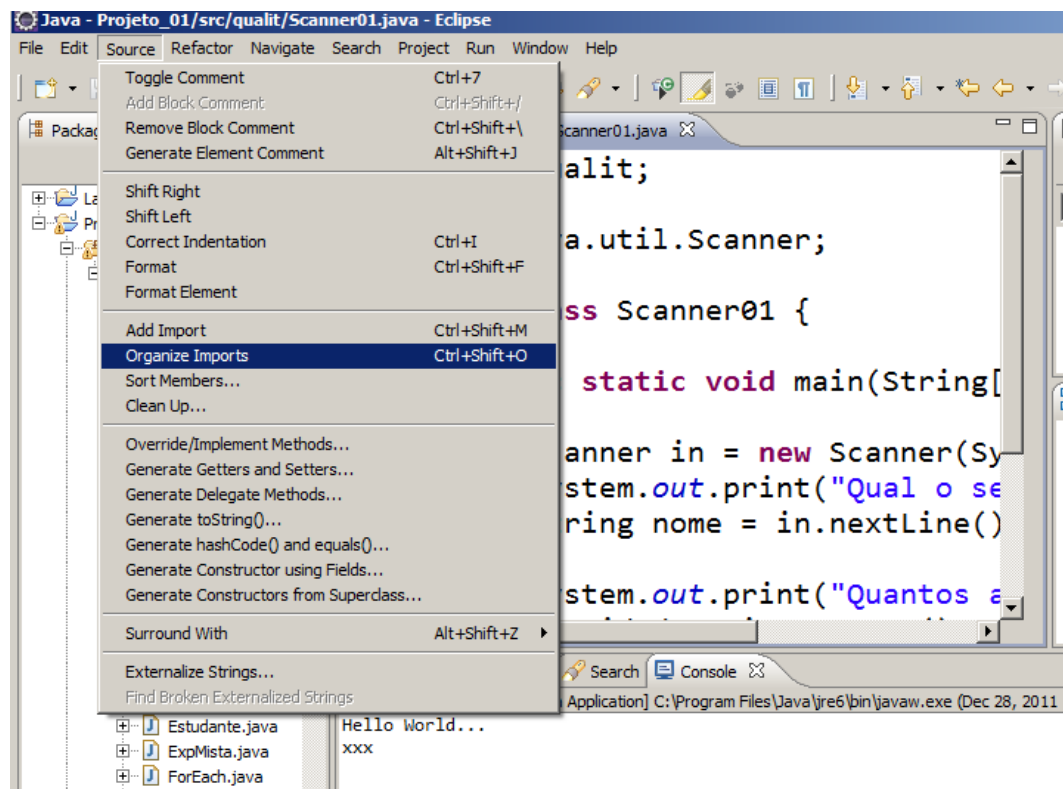
```
import    java.util.Date; //mais legível
```





# A diretiva import

- Em Eclipse, pode-se selecionar a opção **Source > Organize Imports** para se expandir a lista de classes da diretiva import.





# Observações – Import

- ◆ Não se pode usar *import java.\** ou *import java.\*.\** para se importar todas as classes com o prefixo **java**;
- ◆ Deve-se prestar atenção para classes que apresentem conflito de nomes. Por exemplo, os packages **java.util** e **java.sql** têm uma classe chamada **Date**.



```
package qualit;  
import java.util.*;  
import java.sql.*;
```

```
public class ConflitoPackages {
```

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

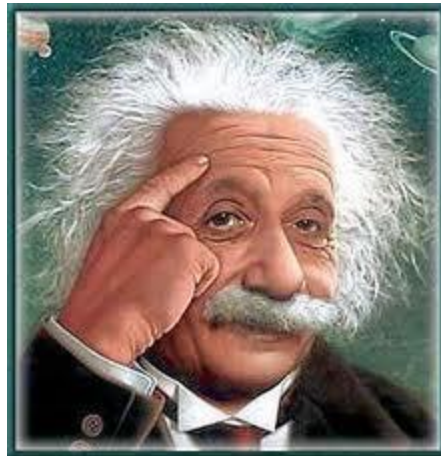
```
        Date x = new Date(); // the type Date is ambiguous
```

```
    }
```

```
}
```



# Como eliminar a ambiguidade?

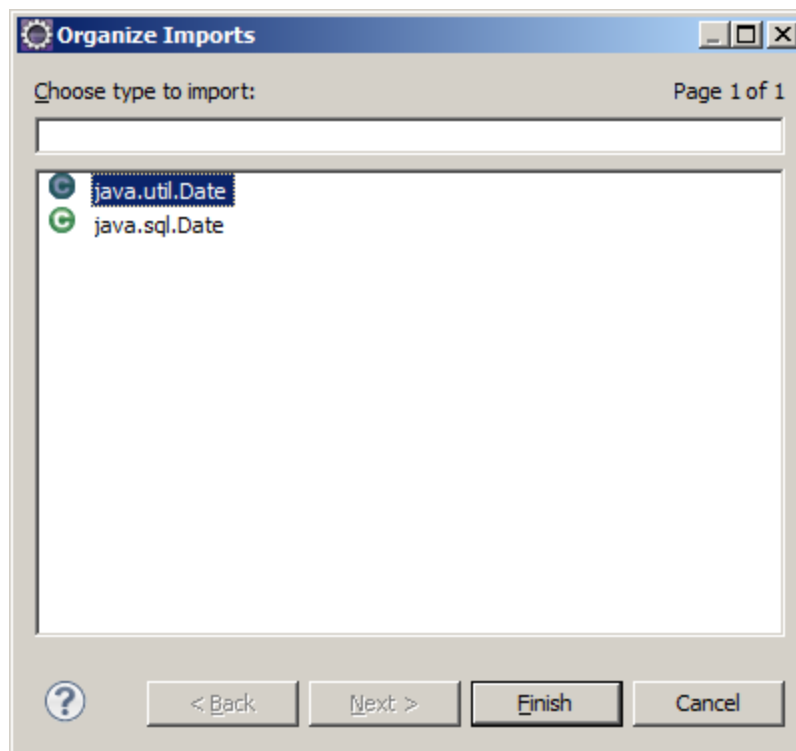






## Como eliminar a ambiguidade?

- ❖ Sob Eclipse, selecione a opção **Source > Organize Imports** e escolha qual o **package** apropriado para o programa.



# Como eliminar a ambiguidade?

- ◆ A localização das classes no package é uma atribuição do compilador.

```
package qualit;
```

```
import java.util.Date;
```

```
public class ConflitoPackages {
```

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

```
        Date x = new Date();
```

```
        System.out.println(x);
```

```
    }
```

```
}
```





# E se precisarmos das duas classes ?



# E se precisarmos das duas classes ?

- ◆ Deves-se escrever o nome da classe de forma **qualificada**;
- ◆ Os **bytecodes** nos arquivos de classes sempre usam **full package names** para se referirem às classes.

```
public class ConflitoPackages {  
  
    public static void main(String[] args) {  
  
        java.util.Date x = new java.util.Date();  
  
        java.sql.Date y = new java.sql.Date(0);  
  
        System.out.println(x);  
  
        System.out.println(y);  
  
    }  
}
```





# Static Imports

- ◆ A partir da versão 5.0, o comando `import` teve melhorias para permitir a importação de campos e métodos **static**, não somente classes.

```
package qualif;  
  
import static java.lang.System.*;  
  
public class ImportStatic01 {  
  
    public static void main(String[] args) {  
  
        out.println("Hello World...");  
  
    }  
  
}
```





# Static Imports

```
package qualif;
```

```
import static java.lang.Math.*;
```

```
public class ImportStatic02 {
```

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

```
        System.out.println(PI);
```

```
        System.out.println(sqrt(PI));
```

```
    }
```

```
}
```

