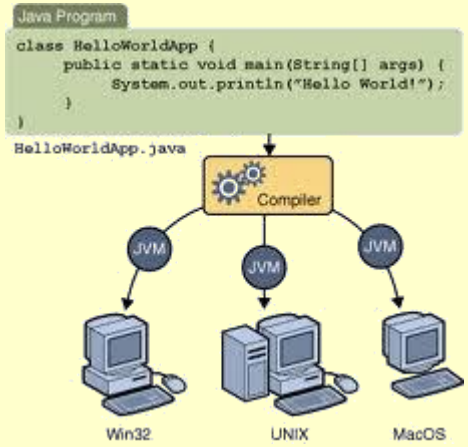
# Pacotes, Classes Comuns e Visibilidade em Java

Werther



# Pacotes (*Packages*)

- É uma coleção nomeada de classes
- Servem para agrupar classes relacionadas
- Pode existir pacotes e subpacotes
- A plataforma java define pacotes padrões
  - "**java**", "**javax**" e outros
    - Subpacotes "***java.lang***", "***java.util***", "***java.io***", "***java.net***", etc.
  - O nome completo da classe a ser referenciada tem que especificar o pacote
    - Ex.: ***java.lang.String***, ***java.io.File***, ***java.util.ArrayList***, ***java.lang.Math***;
- Definindo pacote para a classe:
  - No começo da classe (primeira definição)
    - Nome geralmente em minúscula
  - Diretiva ***package***

# Pacotes, classes e visibilidade

## Pacotes

- Dentro de um mesmo pacote, uma classe pode referenciar a outra pelo seu nome simples
  - Fora do pacote é necessário utilizar o nome completo
    - **ATENÇÃO**: Isto vale para qualquer classe java
- Diretiva ***import***
  - Especifica classes e/ou pacotes de classes
  - Permite que se referencia apenas pelo nome simples
  - Deve estar após a diretiva ***package***
- O pacote ***java.lang*** é comum a todos
  - Suas classes podem ser referenciadas apenas pelo nome simples, não necessitam do ***import***

# Pacotes, classes e visibilidade

## Pacotes

- Exemplos:

```
package figuras;

public class Circulo extends Figura {
  public static final double PI = 3.14159;
  protected double raio;
  ... ... // etc.
}
-----------------------------------------------------------------
import figuras.*; // poderia ser, por exemplo, import figuras.Circulo e
                  //  import figuras.Retangulo
                  // OBS.: * serve como coringa

public class Exercicio {
    public static void main(String[] args) {
      Circulo c = new Circulo(1.0);
      Retangulo r = new Retangulo(2.0,3.0);
      ... ...
    }
}
```

# Classes Comuns (pacote java.lang)

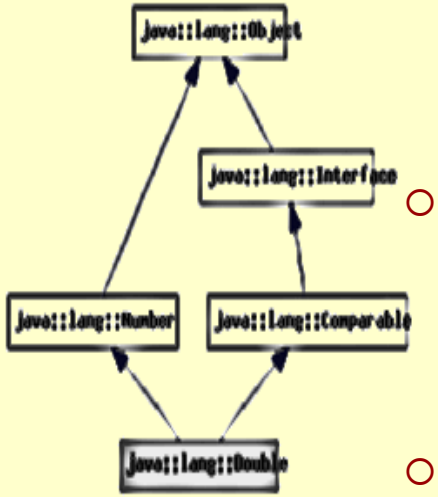
- Classes mais utilizadas

| Class | Description |
|---|---|
| Boolean | The Boolean class wraps a value of the primitive type boolean in an object. |
| Byte | The Byte class wraps a value of primitive type byte in an object. |
| Character | The Character class wraps a value of the primitive type char in an object. |
| Double | The Double class wraps a value of the primitive type double in an object. |
| Float | The Float class wraps a value of primitive type float in an object. |
| Integer | The Integer class wraps a value of the primitive type int in an object. |
| Long | The Long class wraps a value of the primitive type long in an object. |
| Math | The class Math contains methods for performing basic numeric operations such as the elementary exponential, logarithm, square root, and trigonometric functions. |
| Short | The Short class wraps a value of primitive type short in an object. |
| String | The String class represents character strings. |
| System | The System class contains several useful class fields and methods. |
| Thread | A thread is a thread of execution in a program. |
| Throwable | The Throwable class is the superclass of all errors and exceptions in the Java language. |

# A Classe Integer (*java.lang.Integer*)


- Define um valor do tipo primitivo **int** em um objeto.
  - Um objeto do tipo Integer contém um único campo cujo tipo é **int**.
- Fornece vários métodos para converter um **int** em um **String** e um **String** para um **int**, assim como outras constantes e métodos úteis ao lidar com um **int**
- Alguns exemplos de utilização:
  - `Integer x = new Integer("10");`
  - `Integer y = new Integer(5);`
  - `Integer z = Integer.valueOf("71");`
  - `int i = Integer.parseInt("23");`
  - `int j = z.intValue();`
  - `long k = z.longValue();`

# A Classe Double (*java.lang.Integer*)




- Define um valor do tipo primitivo **double** em um objeto.
  - Um objeto do tipo Double contém um único campo cujo tipo é **double**.
- Fornece vários métodos para converter um **double** em um **String** e um **String** para um **double**, assim como outras constantes e métodos úteis ao lidar com um **double**
- Alguns exemplos de utilização:
  - `Double x = new Double("10.7");`
  - `Double y = new Double(5.1);`
  - `Double z = Double.valueOf("271");`
  - `double i = Double.parseDouble("23.09");`
  - `double j = z.doubleValue();`
  - `float k = z.floatValue();`

# A Classe String (*java.lang.String*)

- Simula o conceito e funcionamento de variáveis tipo *strings*, como conhecemos em outras linguagens
- Cada variável da classe String é um **objeto**
  - Possui atributos (incluindo seu próprio valor) e métodos
- Instanciando (declarando) um objeto da classe String:

```
String nome = new String("Maria");
```

  - ***new*** é utilizado para instanciar objetos em java

```
String nome = "Maria";
```

  - String é uma **classe especial** e pode ser instanciada como se fosse um tipo primitivo
    - Mas deve-se lembrar que é um objeto!

# A Classe String (*java.lang.String*)

- Algumas operações básicas:

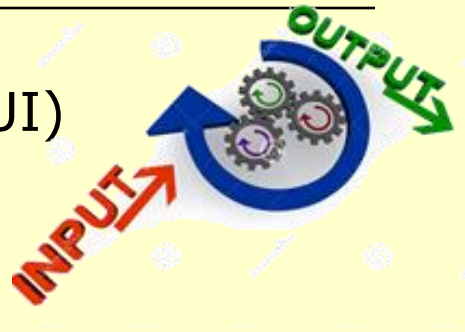
```
String s = "Missão"; // instanciando String s
s = s + " Impossível"; // concatenando com operador +
String t = s + 2; // converte para String e concatena
int compr = s.length(); // método retorna comprimento da string
String sub = s.substring(14); // retorna "vel"
sub = s.substring(7,10); // retorna "Imp"
sub = s.substring(0,2); // retorna "Mi"
char c = s.charAt(4);  // retorna o caractere 4 de s → "ã"
String caps = s.toUpperCase(); // retorna maísculas
String lower = s.toLowerCase(); // retorna minúsculas
boolean b1 = s.equals(t); // retorna false
boolean b2 = s.equalsIgnoreCase(caps); // retorna true
```

# A Classe Math (*java.lang.Math*)

- Define várias funções (métodos) que fornecem funções matemáticas para operações trigonométricas, logarítmicas, exponenciais e de arredondamentos, dentre outras
- Alguns exemplos:

```
double d = Math.toRadians(27); // convertendo 27º em radiano
d = Math.cos(d); // retorna cosseno de 27º
d = Math.sqrt(d); //raiz quadrada
d = Math.log(d); // logaritmo neperiano
d = Math.exp(d); // exponencial: e^d
d = Math.pow(3,d); // potenciação: 3^d
double teto = Math.ceil(d);  // arredonda para o teto inteiro
double piso = Math.floor(d); // arrendonda para o piso inteiro
long proximo = Math.round(d); // para o mais próximo inteiro
```

# Classes de Entradas e Saídas




- Interface básica de caracteres (não é GUI)
  - GUI – *Graphics User Interface*

- Saída em tela (1):
  - ***System.out.println(<ItemAExibir>)***
    - ***<ItemAExibir>*** → qualquer objeto ou tipo primitivo, ou ainda expressão válida que retorne resultado
      - Transforma tudo em String
      - Podem ser expressões concatenadas
      - Exemplos:

```
System.out.println("Oi");          System.out.println(2+3);
System.out.println("x = "+x);
System.out.println(s+i); // String s e int i
```

# Entradas e Saídas

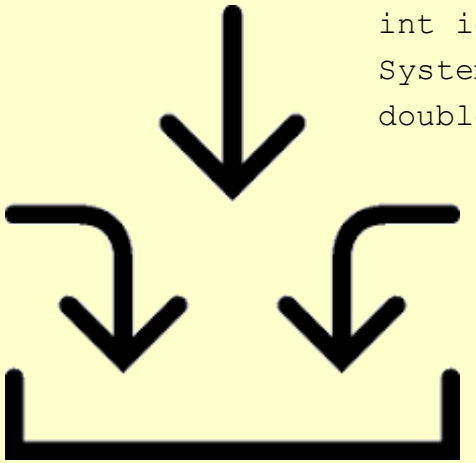
- Saída em tela (2):
  - ***System.out.printf(<formatação>,<valores>)***
    - Parâmetros → Mesma sintaxe do C
      - %d para inteiros;
      - %f para reais;
      - %s para strings; etc.
      - E ainda pode utilizar os limitadores de formatação
      - Exemplos:

```
System.out.printf("Idade de %s: %d",nome,idade); // Idade de Ana: 20
System.out.printf("Valor => %03d",5); // Valor => 005
System.out.printf("Salario de %,.2f ",sal); // Salario de 10.200,21
```

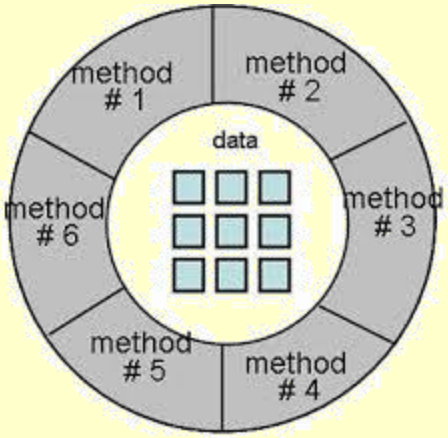
# Entradas e Saídas

- Entrada pelo teclado:
  - Classe **Scanner**. Exemplo:

```
Scanner tec = new Scanner(System.in); // ativa o leitor de teclado
System.out.print("Nome:"); // opcional, para informar o usuário
String nome = tec.nextLine(); // lê um campo String
System.out.print("Idade --> "); // opcional, para informar o usuário
int idade = tec.nextInt(); // lê um campo inteiro
System.out.print("Salario -> "); // opcional, para informar o usuário
double salario = tec.nextDouble(); // lê um campo double
```



# Encapsulamento de Dados




- Permite interação entre objetos sem um precisar ter conhecimento do funcionamento interno do outro
- Oculta detalhes de implementação interna da classe desenvolvida
- Fornece aos programadores que a utilizam conhecimento para saber:
  - As operações (métodos) que podem ser requisitadas
  - O que os métodos realizam (mas, não como)
    - Basta saber apenas a assinatura (nome+argumentos) dos métodos
- Protege e garante consistência a atributos e métodos
  - Bem como de acesso indevido

# Encapsulamento de Dados e Visibilidade

- Ocorre através de modificadores que definem controle e regras de acesso
- Devem vir antes do elemento (atributo ou método):
- ***public:*** indica que o elemento é visível (accessível) dentro e fora da classe;
  - Menos restritivo
- ***private:*** elemento visível apenas dentro da classe;
  - Mais restritivo
- ***protected:*** elemento visível dentro da classe, por todas as classes herdeiras (subclasses) **e** por todas as classes do mesmo pacote;
  - Restrição parcial
- ***package***: elemento visível por todas as classes do mesmo pacote.
  - Não é um modificador (não se declara) e sim uma regra de acesso
  - Se não vier com nenhum modificador descrito acima, o elemento é considerado como ***package***
    - ***Valor default***

# Encapsulamento de Dados

- Vantagens:
  - Evita, protege e/ou limita o acesso de fora à atributos/métodos da classe
    - O código original fica protegido!
  - Os atributos e métodos essenciais para a classe podem ficar ocultos
- Algumas boas práticas de programação em O.O.:
  - Para aumentar o encapsulamento da classe, declare todos atributos como privado (***private***) ou, no máximo, protegido (***protected***);
    - Evite atributos protegidos, a menos que seja necessária a sua visibilidade em subclasses;
  - Só deixe público (***public***) os métodos que fazem parte da interface da classe.
    - Interface da classe: métodos de acesso externo

# Exercício em Sala (1/2)

- Considere uma classe Colaborador com 3 atributos: nome, matricula e salario. Seu construtor recebe estes 3 parâmetros e ela possui todos os métodos acessores (***get*** e ***set***) para seus atributos.
- Existe um método público *aumentarSalario()* que aumenta em 5% o valor do salário.
- Faça duas subclasses de Colaborador: Gerente e Operador. Que utilizam, como base, o mesmo construtor da superclasse.
  - O método *getSalario()* da classe Gerente será sobrescrito, retornando o salário e mais 10% do seu valor a título de gratificação de chefia.
  - Para a classe Operador, o *getSalario()* também será sobrescrito e retornará o valor do salário adicionado de 30% a título de periculosidade.

**ATENÇÃO: Todas estas classes devem estar no mesmo pacote.**

***CONTINUA → →***

# Exercício em Sala (2/2)

- Escreva um programa em uma **classe dentro de um outro pacote** que será dividido seu processamento em 3 etapas:
  I. Vai receber valores de nome, matrícula e salário para 10 funcionários. Para cada funcionário, deve ser informado o seu cargo (se colaborador, gerente ou operador)
     - Todos os valores tem que ser informados EXCLUSIVAMENTE através do ***nextLine()*** da classe Scanner
       - Não utilize *nextDouble()*, *nextInt()* ou método similar
  II. Após a entrada de todos os 10, execute o método *aumentarSalario()* para todos
  III. Por último imprima cargo (se colaborador, gerente ou operador), a primeira letra do nome e o salário atual de todos esses 10 funcionários.