# Programação Orientada a Objetos com Java

Prof. Júlio Machado
julio.machado@pucrs.br

# INTRODUÇÃO

# Plataforma Java

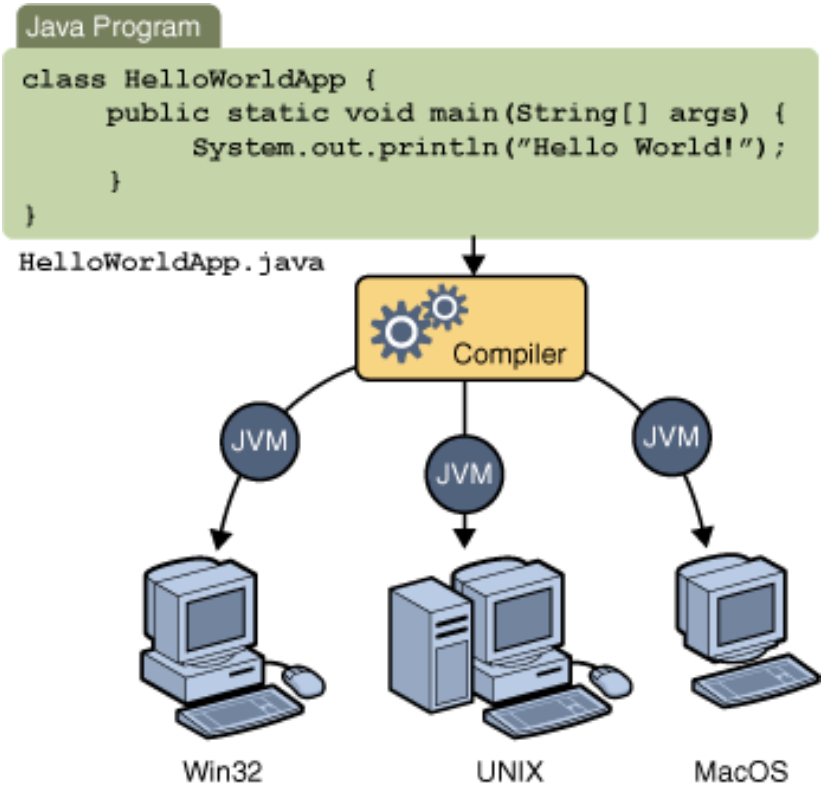
- Java é tanto uma linguagem de programação de alto nível quanto uma plataforma de desenvolvimento de sistemas
- Como linguagem, Java é orientada a objetos, independente de arquitetura (multiplataforma), portável, robusta, segura, interpretada, distribuída, etc

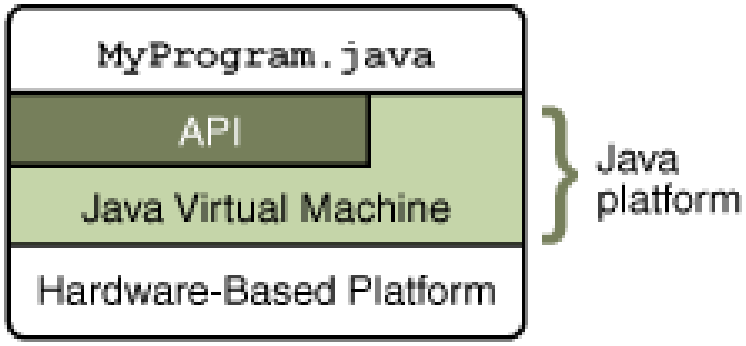
# Plataforma Java

- Java SE (Java Platform Standard Edition)
  - Desenvolvimento e execução de applets, aplicações stand-alone ou aplicações cliente
- Java EE (Java Platform Enterprise Edition)
  - Reúne um conjunto de tecnologias em uma arquitetura voltada para o desenvolvimento de aplicações servidoras
- Java ME (Java Platform Micro Edition)
  - Fornece um ambiente de execução otimizado e permite escrever programas cliente que são executados em pequenos dispositivos móveis (smart cards, telefones celulares, ...)

# Plataforma Java

- Compilador e máquina virtual disponíveis para vários sistemas operacionais



# Introdução à Programação Orientada a Objetos

- O que é um paradigma de programação?
  - É um padrão conceitual que orienta soluções de projeto e implementação
  - Paradigmas explicam como os elementos que compõem um programa são organizados e como interagem entre si
    - Exs.: procedural, funcional, orientado a objetos

# Orientação a Objetos

- É baseada na modelagem de objetos do mundo real
- O que é um objeto?
  - Uma entidade que você pode reconhecer
  - Uma abstração de um objeto do mundo real
  - Uma estrutura composta de dados e operações sobre esses dados

# Objetos

- Cada objeto possui características (atributos) e comportamento (operações)
  - Ex.: lâmpada
    - características: ligada (sim/não), potência, voltagem
    - comportamento: ligar, desligar, queimar

# Objetos

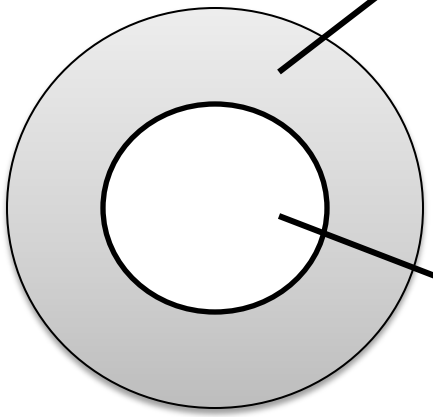
- Um programa orientado a objetos é estruturado como uma comunidade de objetos que interagem entre si
  - Cada objeto tem um papel a cumprir
  - Cada objeto oferece um serviço ou realiza uma ação que é usada por outros objetos
  - Ex.: um objeto Lustre interage com diversos objetos Lâmpada

# Classes

- A classe é a definição formal dos atributos e métodos que compõem os objetos
- Objetos são instâncias de uma classe

# Encapsulamento

- Encapsular é esconder como as coisas funcionam por trás de uma interface externa
  - Interface são as operações que o objeto fornece para os demais objetos
  - É um dos conceitos básicos da Orientação a Objetos
- A ideia é de uma “caixa preta”:
  - Não é necessário saber os detalhes de funcionamento interno do objeto, mas sim como utilizá-lo
- Ex.: caixa automático
  - Como ele é implementado internamente?
  - Utilizamos através de operações bem conhecidas



Métodos

•Interface pública declara as operações permitidas

Atributos

objeto

•Acesso aos atributos é feito através da chamada de um método

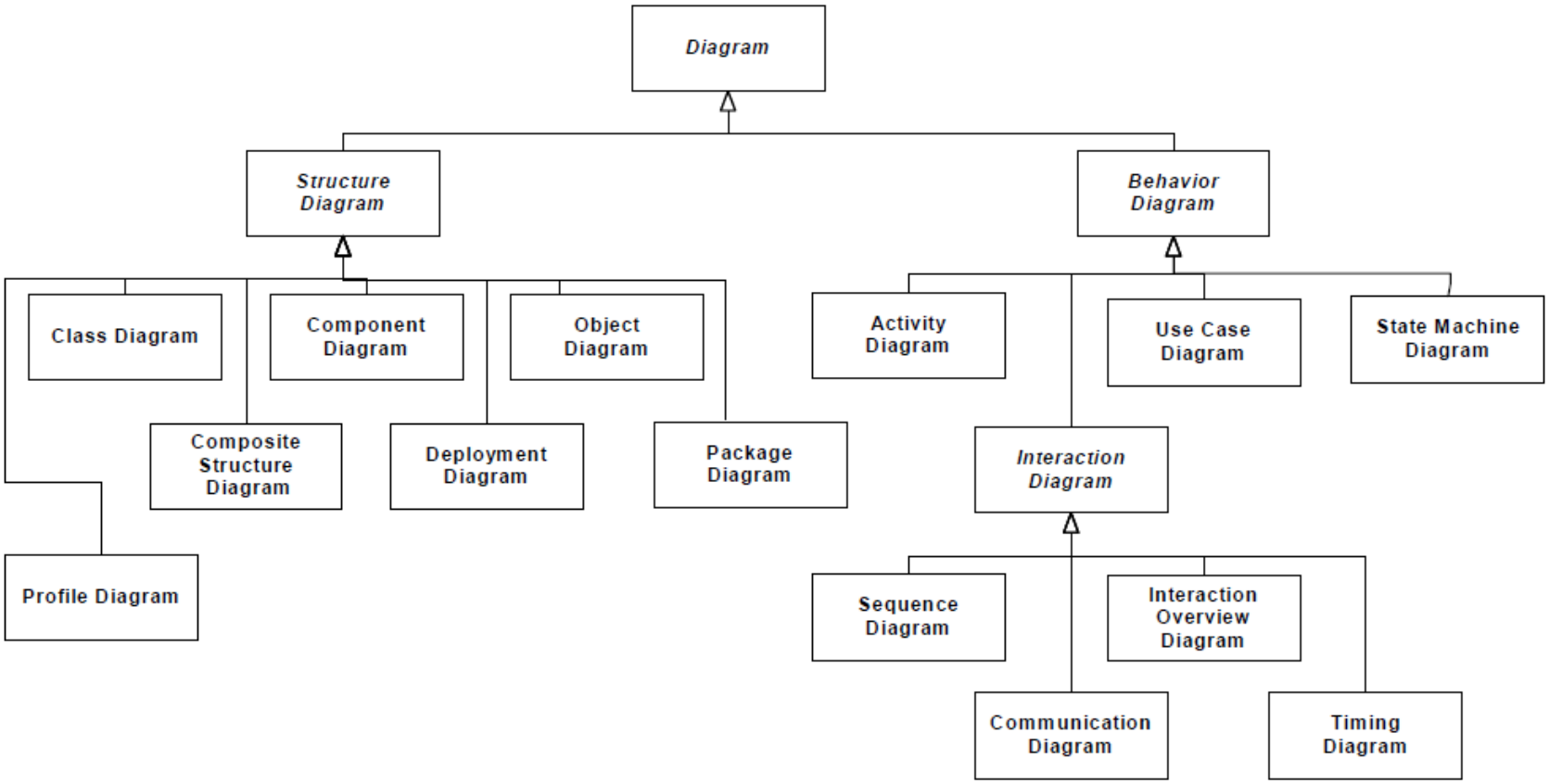
# Encapsulamento

- Alguns benefícios:
  - A implementação interna de um objeto pode mudar e o resto do sistema não é afetado (desde que a interface de acesso não mude)
  - Maior segurança ao proteger os atributos de um objeto de alterações indevidas por outros objetos
  - Maior independência entre os objetos, pois eles só precisam conhecer a interface externa definida

# Projetando Objetos

- De uma forma simples, o projeto orientado a objetos de um sistema pode ser dividido em três etapas:
  - Identificar as abstrações/entidades envolvidas no problema
  - Identificar o comportamento que cada uma destas entidades deve ser capaz de fornecer
  - Identificar os relacionamentos entre essas entidades
  - Identificar as estruturas de dados internas necessárias para implementar o comportamento e relacionamentos desejado

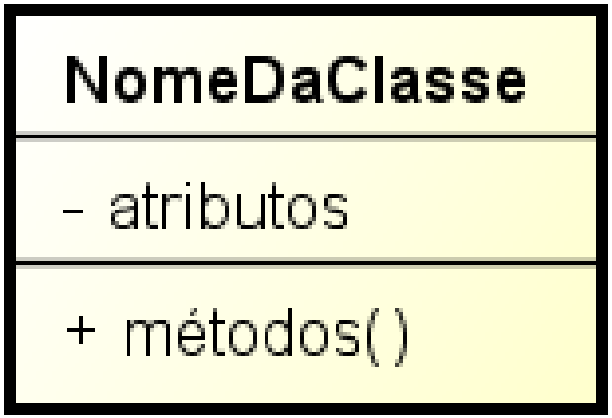
# Diagramas UML



# Diagrama de Classes UML

- Denota a estrutura estática do sistema
- Apresenta as classes e seu relacionamentos com outras classes

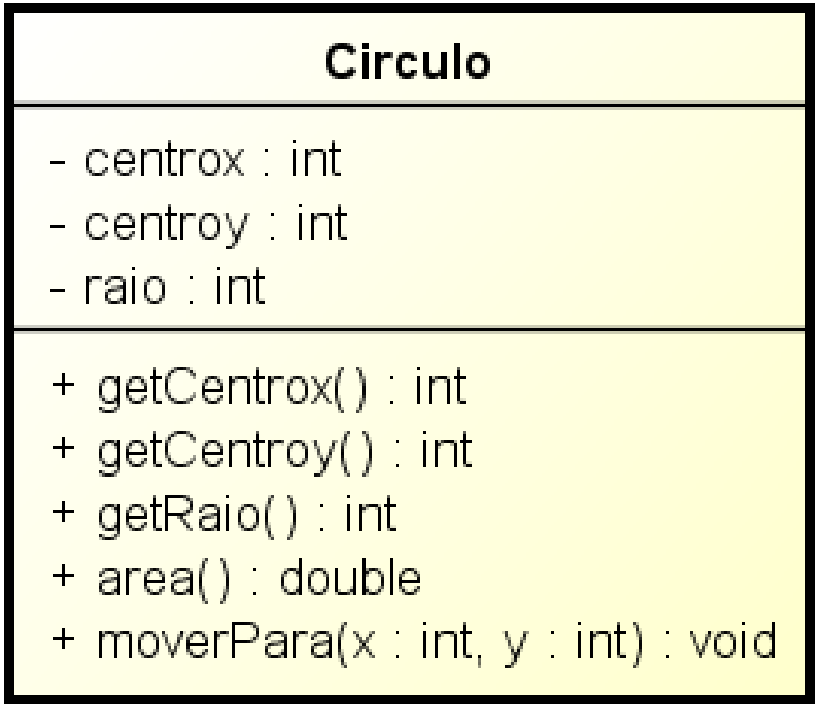
# Diagrama de Classes da UML



# Diagrama de Classes da UML

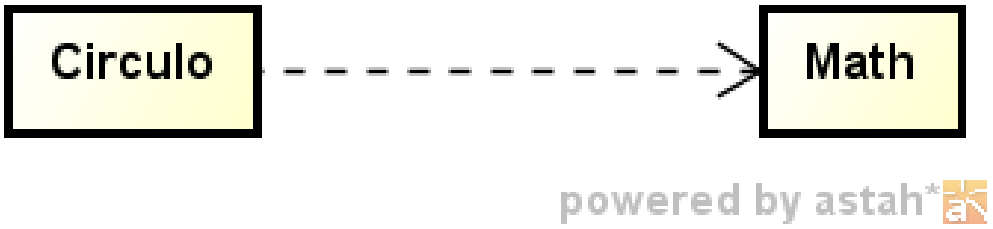
- Modificadores:
  - Público +
  - Privado -

# Diagrama de Classes UML



# Diagrama de Classes UML

- Relacionamento de dependência:
  - É um relacionamento que significa que um elemento necessita de outro elemento para sua especificação ou implementação
  - É um relacionamento “fornecedor-cliente”
    - Um objeto fornece algo que outro objeto utiliza



# Diagrama de Classes UML

- Relacionamento de associação:
  - É um relacionamento estrutural que descreve um conjunto de ligações, onde uma ligação é uma conexão entre objetos
  - Usualmente implementado através de atributos

# Diagrama de Classes UML

- Relacionamento de associação:
  - Navegabilidade da associação
    - Bidirecional


    - Unidirecional


# Diagrama de Classes UML

- Relacionamento de associação:
  - Multiplicidade da associação
    - Especifica-se o menor e o maior valor
    - Formato Menor..Maior
    - Valores mais utilizados
      - Menor: 0 (opcional), 1 (obrigatório)
      - Maior: 1 (somente um), * (vários)

# Diagrama de Classes UML

- Relacionamento de associação:
  - Multiplicidade da associação
    - Cliente tem uma única conta (1..1 ou 1)


    - Cliente pode ter ou não uma conta


    - Cliente tem várias contas, mas no mínimo uma


    - Cliente tem várias contas, mas não é obrigatório (0..* ou *)

# Resumo

- Objeto
  - Unidade básica de orientação a objetos. Um objeto é uma entidade que tem atributos, comportamento e identidade. Objetos são membros de uma classe e os atributos e métodos de um objeto são definidos pela classe.
- Classe
  - Uma classe é uma descrição de um conjunto de objetos. Este conjunto de objetos compartilha atributos e comportamento em comum. Uma definição de classe descreve todos os atributos dos objetos membros da classe, bem como os métodos que implementam o comportamento destes membros.

# Resumo

- Orientação a objetos
  - Um paradigma de programação que usa abstração com objetos, classes encapsuladas e comunicação por mensagens, hierarquia de classes e polimorfismo.
- Abstração
  - Um modelo de um conceito ou objeto do mundo real.
- Encapsulamento
  - Processo de esconder os detalhes internos de um objeto do mundo externo.

# Resumo

- Comportamento
  - Atividade de um objeto que é vista do ponto de vista do mundo externo. Inclui como um objeto responde a mensagens alterando seu estado interno ou retornando informação sobre seu estado interno.
- Método
  - Uma operação ou serviço executado sobre o objeto, declarado como parte da estrutura da classe. Métodos são usados para implementar o comportamento do objeto.
- Estado
  - Reflete os valores correntes de todos os atributos de um objeto e são o resultado do comportamento do objeto ao longo do tempo.
- Atributo
  - Usado para armazenar o estado de um objeto. Pode ser simples como uma variável escalar (int, char, double, ou boolean) ou pode ser uma estrutura complexa tal como outro objeto.

# PROGRAMAÇÃO COM JAVA

# Estrutura de um Programa

- Um programa Java é um conjunto composto por uma ou mais classes
- Tipicamente, cada classe é implementada em um arquivo fonte separado, sendo que o arquivo deve ter o mesmo nome da classe.
  - Ex.: a classe Lampada deve estar definida no arquivo Lampada.java
- Em geral, os arquivos que compõem um programa java devem estar no mesmo diretório

# Biblioteca de Classes (API)

- Application Programming Interface
- É uma coleção de classes, normalmente provendo uma série de facilidades que podem ser usadas em programas
- Classes são agrupadas em conjuntos chamados packages
  - Exs:
  - java.lang: inclui classes básicas, manipulação de arrays e strings. Este pacote é carregado automaticamente pelo programa
  - java.io: operações de input e output
  - java.util: classes diversas para manipulação de dados

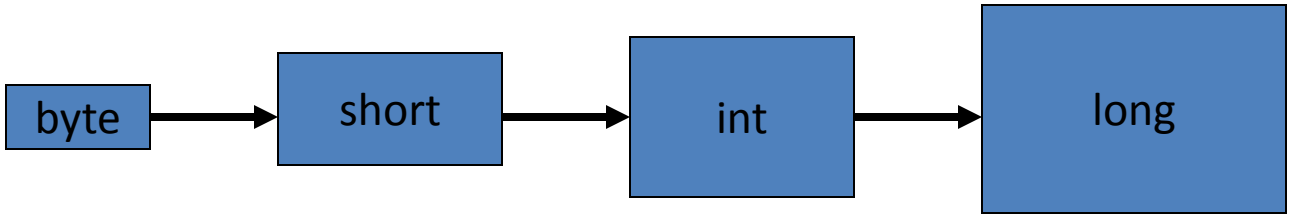
# Tipos de Dados Básicos

- Tipos de dados primitivos
  - inteiros: byte (8 bits), short (16), int (32), long (64)
    - 1 (decimal) , 07 (octal), 0xff (hexadecimal), 1L(long)
  - reais: float (32), double (64)
    - 3.0F (float), 4.02E23 (double), 3.0 (double)
  - caractere: char (16)
    - ‘a’, ‘\141’, ‘\u0061’, ‘\n’
  - booleano: boolean (8)
    - true, false

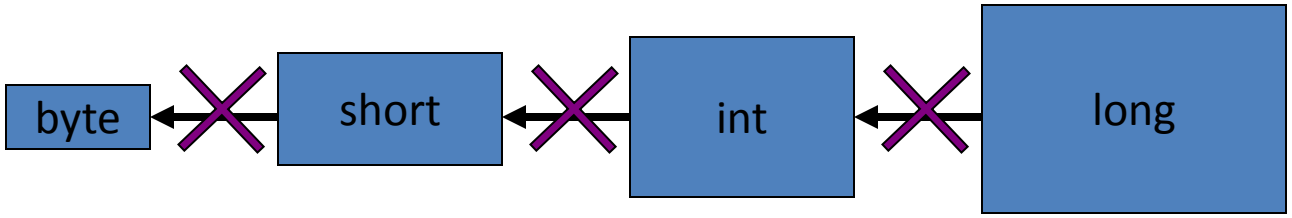
# Tipos de Dados Básicos

- Em Java, tem-se dois tipos de conversão de valores:
  - conversão para um tipo maior
    - automática
  - conversão para um tipo menor (chamada de *casting*)
    - não é automática

```
int x=1;
long y=x;
```




```
long y=1;
int x=y;
```
Erro!!!



# Tipos de Dados Básicos

- Para converter de um tipo para um tipo menor, precisamos referenciar de forma explícita.
  - *(tipo Java) expressão;*
  - Ex.:
    - ```
      long y = 1;
      int x = (int)y;
      ```
    - ```
      byte b1=1, b2=2, b3;
      b3 = (byte)(b1 + b2);
      ```
  - Cuidado! Ao somar dois valores *byte* iguais a 100, o resultado é o *int* 200. Ao realizar o *cast* para *byte*, o resultado é convertido para -56, o equivalente ao padrão de bits armazenados.

# Operadores

- Operadores básicos:
  - aritméticos: +, -, *, /, % (resto da divisão)
  - relacionais: >, >=, <, <=
  - igualdade: ==, !=
  - lógicos: &&, & (and), ||, | (or), ^ (xor), ! (not)
  - atribuição: =, +=, -=, *=, /=, %=
  - incremento, decremento: ++, --

# Operadores

- A maioria dos operador aritméticos resultam em *int* ou *long*
  - Quando utilizamos valores *byte* e *short*, eles são convertidos para *int* antes da operação
  - Da mesma forma, se um dos operandos for *long*, os outros são convertidos para *long* antes da operação
  - Ex.:
    - 10 + 10 o resultado é *int*
    - 10L + 10 o resultado é *long*

# Operadores

- Cuidado:
  - O resultado da operação de divisão em Java depende do tipo dos operandos
    - Tipo inteiro: o resultado é a divisão inteira

```
int resultado = 10/4 //igual a 2
```

    - Tipo ponto flutuante: o resultado é a divisão decimal

```
float resultado = 10f/4f //igual a 2.5
```

# Funções Matemáticas

- Funções matemáticas (classe *Math*):
  - sqrt(x): cálculo da raiz quadrada de x (x é do tipo double)
  - abs(x): valor absoluto de x (x pode ser float, int, long)
  - cos(x): coseno trigonométrico de x (x em radianos)
  - exp(x): método exponencial $e^x$
  - pow(x,y): x elevado a potência y ($x^y$)
- Exemplo:

```
double raio;
raio = Math.sqrt(area/Math.PI);
```

# Classe String

- String
  - É uma classe e não tipo primitivo
  - Representa um grupo de caracteres
    - Codificação Unicode UTF-16
  - É uma classe de objetos imutáveis
    - Uma vez inicializado, o valor da string jamais é alterado
  - Declarados entre aspas duplas
    - `String nome = “Júlio”;`

# Classe String

- Operadores
  - concatenação: +
    - `String nomeCompleto = nome + " " + "Machado”;`
  - comparação: equals
    - ```
      String str1 = "texto";
      String str2 = "txt";
      if(str1.equals(str2)){} //compara conteúdo
      ```
    - ```
      String str1 = "texto";
      String str2 = "txt";
      if (str1 == str2){} //compara endereço
      ```

# Classe String

- Métodos úteis
  - Tamanho:
    - Método *length()*
    - ```
      String texto1 = "Início";
      System.out.println(texto1.length());
      --> 6
      ```
  - Caractere em uma posição:
    - Método *charAt(posição)*
    - O primeiro caractere está na posição 0
    - ```
      char c = texto1.charAt(1);
      --> n
      ```
  - Substrings:
    - Método *substring(início,fim)*
    - ```
      String texto1 = "Início";
      String sub = texto1.substring(1,3)
      --> ní
      ```
      ```

# Classe String

- Conversão
  - Java converte outros tipos para strings
    - int idade = 25;
      String nomeIdade = nome + " " + idade;
  - Como converter tipos primitivos para strings?
    - Métodos *String.valueOf()*, *Integer.toString()*, *Double.toString()*
    - São métodos de classe
    - ```
      String sete = String.valueOf(7);
      String umPontozero =
      Double.toString(1.0);
      ```
      ```

# Classe String

- Conversão
  - Como converter strings para tipos primitivos?
    - Métodos *Integer.valueOf()*, *Double.valueOf()*
    - São métodos de classe
    - ```
      int sete = Integer.valueOf(“7”);
      double umPontozero =
      Double.valueOf(“1.0”);
      ```
      ```

# Enumeração

- Um tipo de enumeração (ou tipo enumerado) é um tipo para qual os valores são conhecidos quando o tipo é definido
- Exemplos:
  - Naipes, dias da semana, meses do ano

# Enumeração

- Declaração
  - Palavra-chave *enum*
  - Identificador da enumeração
  - Lista de constantes da enumeração entre chaves e separadas por vírgula
- Exemplo:

```
enum Naipe {PAUS,OUROS,COPAS,ESPADAS}
```

# Enumeração

- Uso
  - Enumerações são seguras quanto ao tipo
    - Somente os valores declarados e null
  - Declara-se uma variável do tipo da enumeração
  - É possível utilizar comparação via ==
  - Pode ser utilizado com comando switch
- Exemplo:

```
Naipe n = Naipe.OUROS;
if(n == Naipe.OUROS)...
switch(n){
  case PAUS : ...
  ...
}
```

# Comandos - Declaração

- Variáveis:
  - int valor1, valor2 = 123;
    - Com inicialização
  - double taxa, percentual;
    - Sem inicialização
    - Variáveis locais não são inicializadas automaticamente
    - Atributos são inicializados automaticamente
- Constantes:
  - final double PI = 3.1415;
    - Modificador *final*

# Comandos – Condicional IF

- *if (condição) {*
  *comandos;*
  *}*

- *if (condição) {*
  *comandos;*
  *} else {*
  *comandos;*
  *}*

- *if (condição) {*
  *comandos;*
  *} else if (condição) {*
  *comandos;*
  *} else {*
  *comandos;*
  *}*

# Comandos – Condicional IF

```
if (i % 2 == 0) {
   System.out.println(“Par”);
} else {
   System.out.println(“Ímpar”);
}

if (vel >= 25) {
   if (vel > 65) {
       System.out.println(“maior 65”);
   } else {
       System.out.println(“entre 25 e 65”);
   }
} else {
   System.out.println(“menor 25”);
}
```

# Comandos – Condicional SWITCH

- Utilizado para cobrir múltiplas escolhas sobre valores alternativos de variáveis *int*, *byte*, *short*, *long* ,*char, enumeration*

- ```
  switch (expressão) {
      case constante1:
          comandos;
          break;
      ...
      default:
          comandos;
  }
  ```
  ```

# Comandos – Condicional SWITCH

```
switch (menuItem){
   case 0:
        System.out.println(“zero”);
        break;
   case 1:
        System.out.println(“um”);
        break;
   default:
        System.out.println(“inválido”);
}

switch (nota){
   case ‘A’:
   case ‘B’:
   case ‘C’:
        System.out.println(“Passou”);
        break;
   default:
        System.out.println(“Reprovou”);
}
```

# Comandos – Repetição FOR

- *for (inicialização; terminação; incremento) {*
  *comandos;*
  *}*

```
int soma = 0;
for (int i=1; i<=3; i++) {
  soma +=i;
}
System.out.println("Soma "+soma);
```

# Comandos – Repetição WHILE

- *while (condição) {*
  *comandos;*
  *}*

```
int i = 0;
while (i<10) {
  System.out.println("i= "+i);
  i++;
}
```

# Comandos – Repetição DO WHILE

- *do {*
  *comandos;*
  *} while (condição);*

```
int i = 0;
do {
  System.out.println("i= "+i);
  i++;
} while (i<10);
```

# Comandos - Repetição

- Controle de Loops
  - *break*:
    - Termina o comando de repetição
  - *continue*:
    - Abandona a iteração atual da repetição e passa para a próxima iteração
  - Ex.:

```
soma = 0;
for (i=0; i < 5; i++)
{
 if (i == 3)  continue;
 soma += i;
}
```

```
soma = 0;
for (i=0; i < 5; i++)
{
 if (i == 3)  break;
 soma += i;
}
```

# CLASSES E OBJETOS

# Classes

- Definições de classes incluem (geralmente):
  - modificador de acesso
  - palavra-chave *class*
  - nome da classe
  - corpo classe
    - atributos
    - métodos
    - construtores

# Classes

- Modificadores de acesso
  - Permitem definir o encapsulamento de atributos e métodos
  - Dois modificadores principais:
    - ***private***: visível apenas para objetos da própria classe
    - ***public***: visível para quaisquer objetos

# Classes

- Recomendações
  - A menos que hajam razões fortes, os atributos de uma classe devem ser definidos como *private* (encapsulamento) e os métodos que são chamados de fora da classe devem ser *public* (interface de acesso ao comportamento público)
  - Métodos que devem ser usados somente dentro da própria classe, devem ser especificados como *private* (comportamento privado)

# Classes

- Métodos get
  - Retornam o valor do estado atual de um objeto, uma vez que não é possível acessá-lo diretamente
- Métodos set
  - Permitem alterar o valor do estado atual do objeto
  - Estes métodos são chamados por alguns autores de mutantes (mutator methods*)

* David J. Barnes, Michael Kölling. Objects First with Java: A Practical Introduction using BlueJ. Prentice Hall / Pearson Education, 2003

# Exemplo: classe Professor

```
class Professor
{
  private String nome;
  private int matricula;
  private int cargaHoraria;
  ...
}
```

- Atributos estão encapsulados!!!
- Apenas métodos da própria classe Professor podem acessar os atributos

# Exemplo: classe Professor

- Métodos:

```
...
public void setNome(String n) {
    nome = n;
}
public String getNome() {
    return nome;
}
public void setMatricula(int m) {
    matricula = n;
}
public int getMatricula() {
   return matricula;
}
...
```

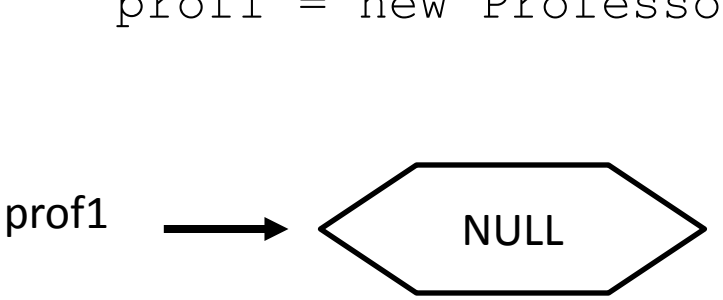
# Exemplo: classe Professor

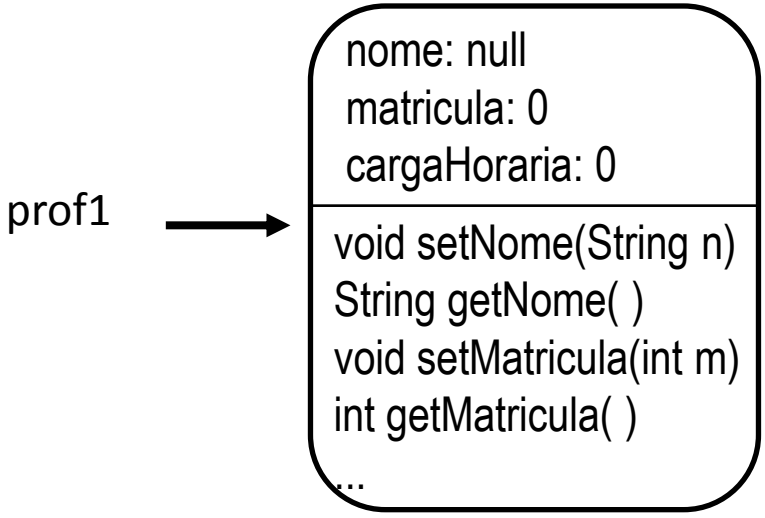
```
...
public void setCargaHoraria(int c) {
    cargaHoraria = c;
}
public int getCargaHoraria() {
    return cargaHoraria;
}
public float getCargaHorariaMensal() {
    return (cargaHoraria * 4.5F);
}
```

# Objetos

- Instanciação
  - Um objeto depois de criado, conterá todos os atributos e métodos descritos em sua classe
  - Para instanciar um objeto em Java utilizamos o operador *new*
  - Ex.:

```
Professor prof1;
prof1 = new Professor();
```



# Objetos

- Quando o operador *new* é usado é “alocada” memória
- Quando um objeto não é mais necessário, devolve-se o(s) recurso(s) para o sistema
- Java realiza a coleta de lixo automática da memória (*garbage collector*)
- Quando um objeto não é mais utilizado, ele é marcado para coleta de lixo

# Programa

- Como executar um programa em Java?
  - Um programa é composto de várias classes e objetos
  - Como indicar por onde o programa começa?
  - Em Java temos um método especial que o interpretador assume como o início do programa: *main*.
    - *public static void main (String args[ ])*

# Programa

```
public static void main (String
  args[]){
  Professor prof1, prof2;
  prof1 = new Professor();
  prof1.setNome(“Júlio”);
  prof1.setMatricula(1234);
  prof1.setCargaHoraria(14);
  System.out.println(prof1.getCarga
  HorariaMensal());
}
```

# Escopo de Variáveis

- O escopo de uma variável informa onde ela pode ser utilizada.
- Ex.:

```
1: public class VerificaEscopo{
2:  private int escopoA;
3:  public void metodo(int escopoB){
4:    int escopoC;
5:  }
6:  private int escopoD;
7: }
```

# Escopo de Variáveis

- Ex.:

```
1: public class VerificaEscopo{
2:  private int escopoA;
3:  public void metodo(int escopoB){
4:    int escopoC;
5:  }
6:  private int escopoD;
7: }
```

- No exemplo
  - escopoA e escopoD são atributos de instância do objeto e seu escopo vale a partir da linha 1
  - escopoB e escopoC são variáveis locais cujo escopo é válido somente dentro do método

# Escopo de Variáveis

- Variáveis locais podem ser declaradas a qualquer momento dentro de um método
  - Ex.:

```
for (int i=1; i<5; i++){
 int j = 0;
 //i e j só valem aqui dentro
}
System.out.println(i);//erro
```

# Inicialização de Variáveis

- Atributos de uma classe são inicializados com valores padrão:
  - `0 -> byte, short, int, long`
  - `0.0 -> float, double`
  - `false -> boolean`
  - `\u0000 -> char`
  - `null -> Object`

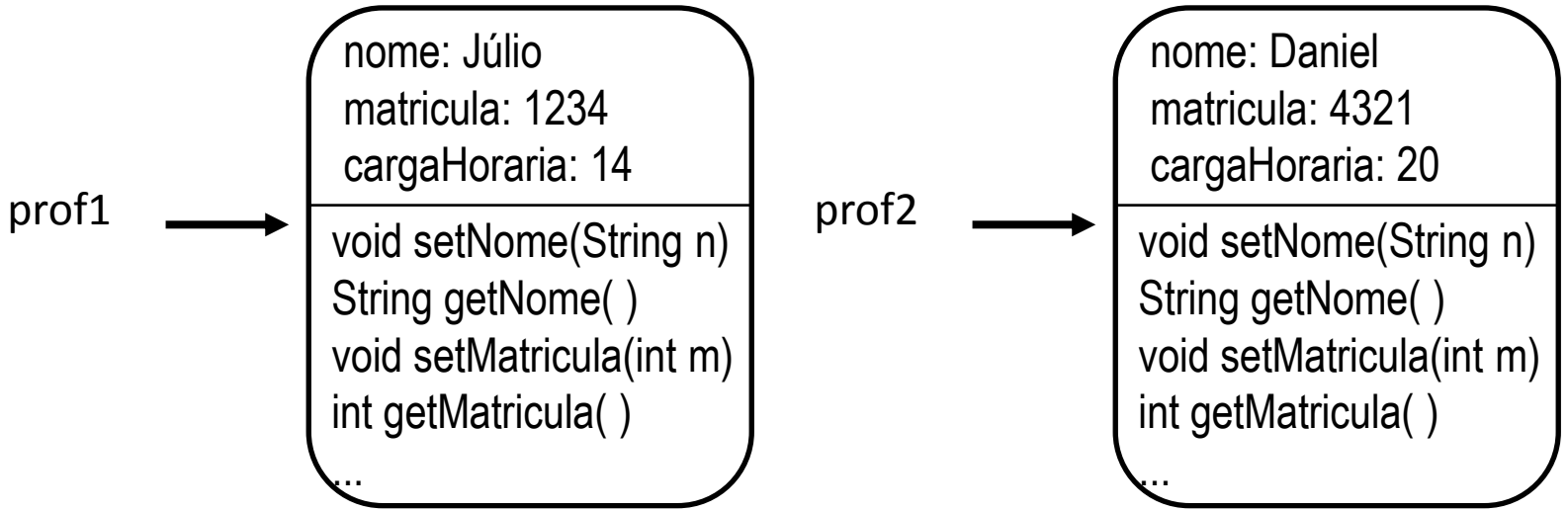
# Inicialização de Variáveis

- Variáveis locais declaradas dentro de método devem obrigatoriamente serem inicializadas antes de utilizadas
  - O compilador Java irá indicar se não inicializarmos as variáveis

# Referências

- Quando criamos um objeto em Java, mantemos uma referência para o objeto na memória
- Ex.:

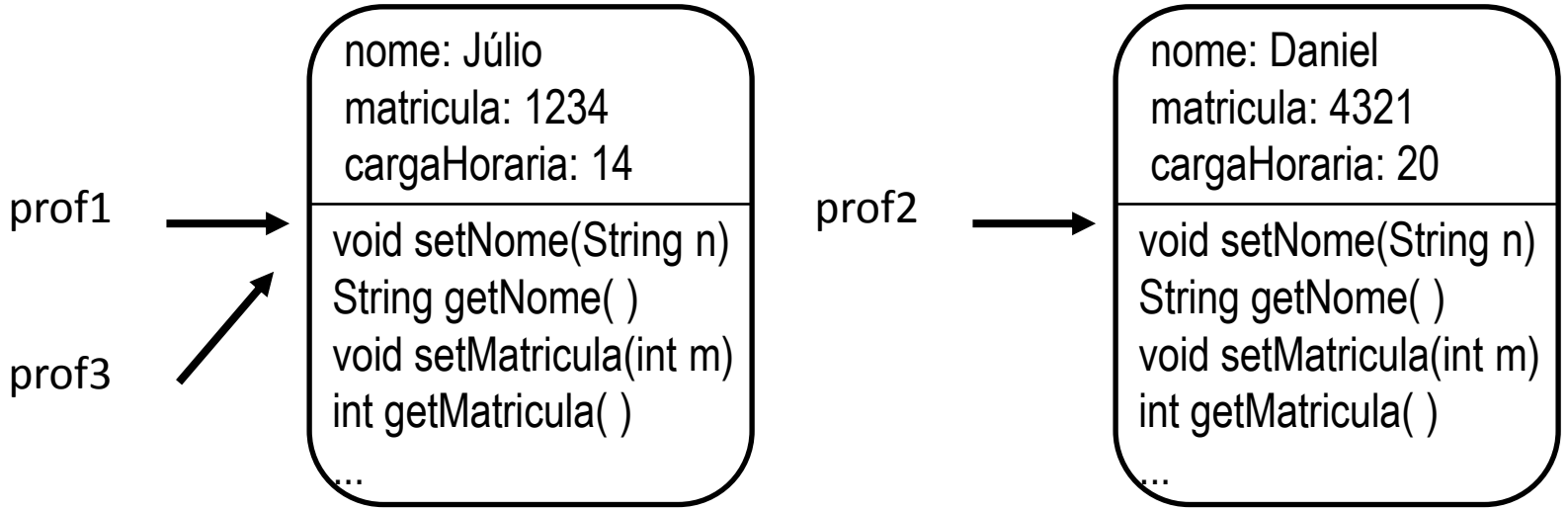
```
Professor prof1, prof2;
prof1 = new Professor();...
prof2 = new Professor();...
```



# Referências

- Ao atribuir prof1 ou prof2 a uma terceira variável, o que irá acontecer?
- Ex.:

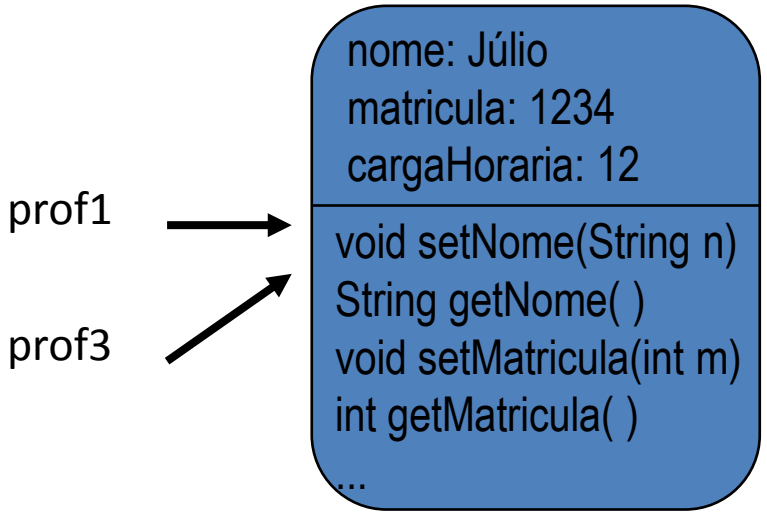
```
Professor prof1, prof2, prof3;
prof1 = new Professor();...
prof2 = new Professor();...
prof3 = prof1;
```



# Referências

- Se alteramos algum atributo do objeto referenciado por prof3, estaremos alterando também o referenciado por prof1!
- Ex.:

```
Professor prof1, prof2, prof3;
prof1 = new Professor();...
prof2 = new Professor();...
prof3 = prof1;
prof3.setCargaHoraria(12);
```



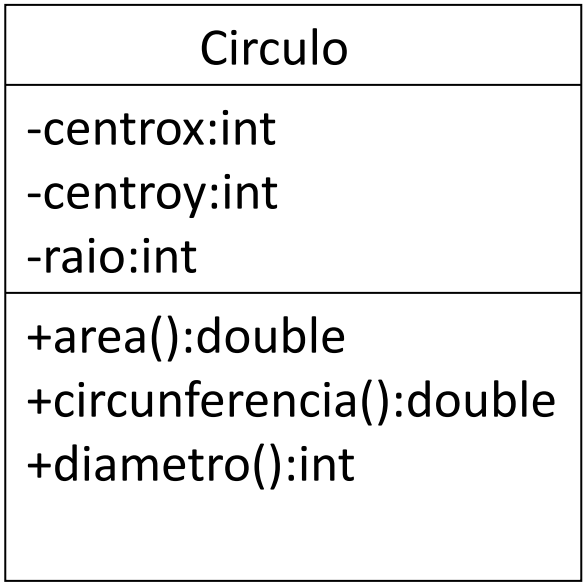
# Inicialização de Objetos

- Objetos:
  - Estado: definido pelos atributos declarados na classe
  - Comportamento: definido pelos métodos declarados na classe
- Quais valores os atributos do objeto possuem após a sua instanciação?
- Como definir o estado inicial do objeto?

# Inicialização de Objetos

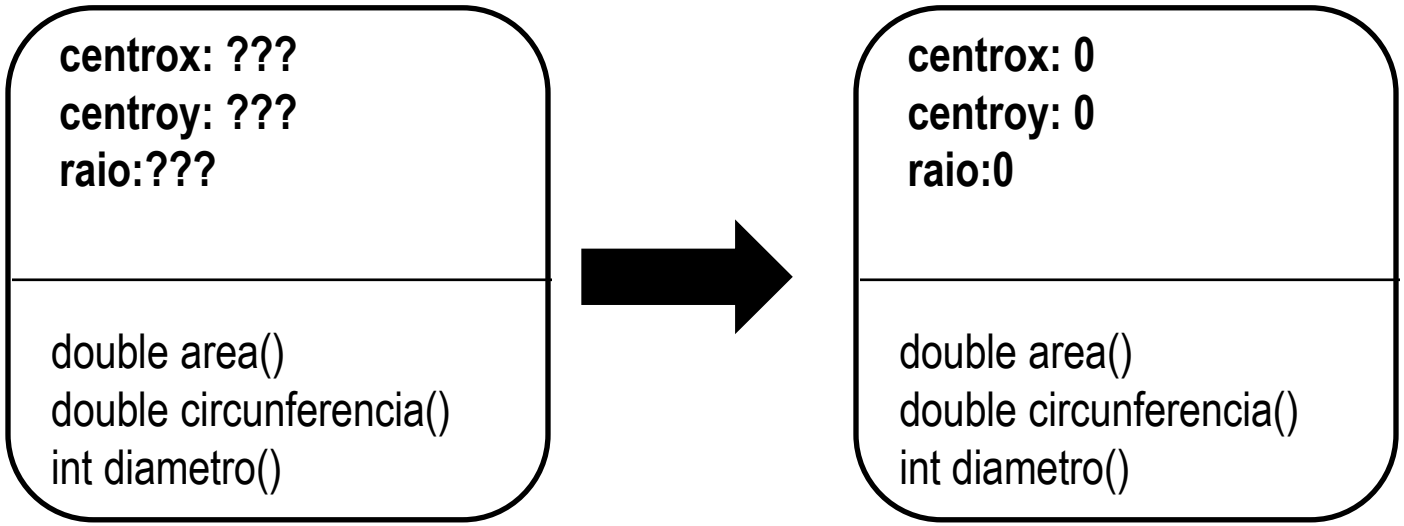
- Exemplo: classe Circulo

```
public class Circulo {
    private int centrox;
    private int centroy;
    private int raio;
    public double area(){
        return (3.14 * raio * raio);
    }
    public double circunferencia(){
        return (2 * 3.14 * raio);
    }
    public int diametro(){
        return (2 * raio);
    }
}
```



# Inicialização de Objetos

```
Circulo circ = new Circulo();
```



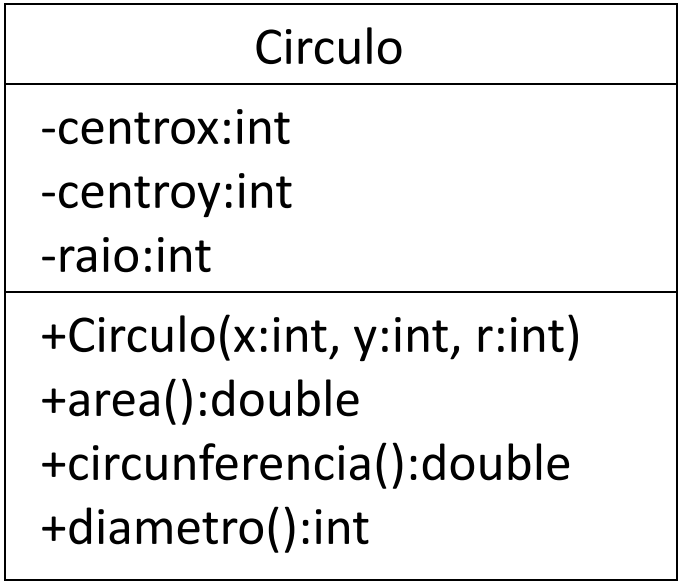
# Inicialização de Objetos

- Da forma como foi apresentada a classe Circulo, todos os objetos criados a partir dela terão seus atributos inicializados com valores padrão iguais a zero
- Como permitir que instâncias da classe Circulo possuam estados diferentes?
  - Adicionar à classe um método para inicializar os atributos com valores diferentes da inicialização padrão
  - Esse método é o construtor!

# Inicialização de Objetos

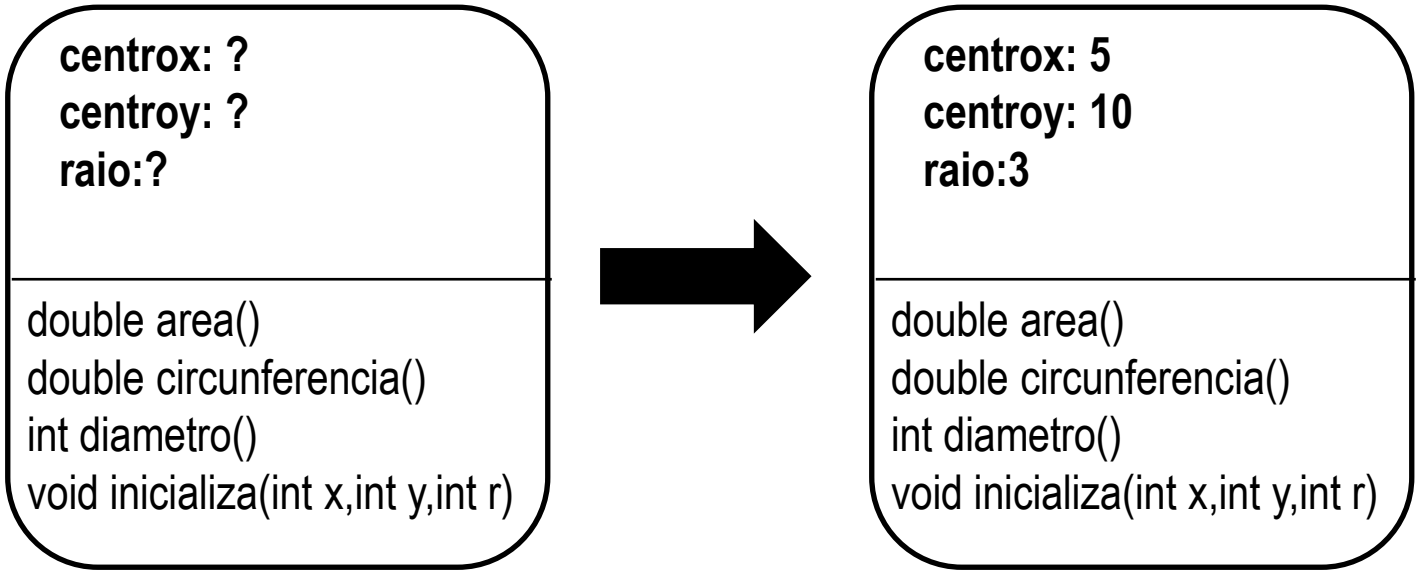
- Exemplo: classe Circulo

```
public class Circulo {
   private int centrox;
   private int centroy;
   private int raio;
   public Circulo(int x, int y, int r){
      centrox = x;
      centroy = y;
      raio = r;
   }
   ...
}
```



# Inicialização de Objetos

```
Circulo circ = new Circulo(5,10,3);
```



# Inicialização de Objetos

- Um construtor em Java:
  - Possui o mesmo nome da classe (respeitando maiúsculas e minúsculas)
  - Pode possuir ou não parâmetros
  - Não possui um tipo de retorno, nem mesmo void

```
<modificador_de_acesso> <nome_classe>(<parâmetros>){
   //corpo do construtor
}
```

# Inicialização de Objetos

- Se nenhum construtor é definido para uma determinada classe, Java irá definir um construtor padrão (chamado construtor *default*)
  - Não possui argumentos de entrada
  - Caso qualquer outro construtor seja definido na classe, Java não irá disponibilizar o construtor padrão

# Sobrecarga

- Chama-se de sobrecarga de métodos (*overloading*) o ato de criar diversos métodos com o mesmo nome que se diferenciam pela lista de argumentos (parâmetros)
  - Métodos são identificados pela sua assinatura: nome do método + lista de parâmetros
  - Métodos com mesmo nome, mas com tipo, quantidade ou ordenação de parâmetros diferentes, são considerados métodos diferentes

# Sobrecarga

- Cuidado!!!
  - Esses métodos possuem uma definição correta para sobrecarga?

```
public void soma(int n, double d)
public void soma(double d, int n)

public void soma(int n)
public void soma(int v)

public void soma(int n)
public double soma(int n)
```

# Sobrecarga

- Na API de Java, diversas classes utilizam a sobrecarga de métodos, por exemplo:
  - Classe String
    - valueOf (boolean b)
    - valueOf (char c)
    - valueOf (double d)
    - valueOf (float f)
    - valueOf (int i)
    - valueOf (long l)
    - retorna a representação em String do argumento recebido

# Sobrecarga de Construtores

- Usualmente é útil para uma classe possuir mais de um construtor a fim de oferecer diversas maneiras para instanciar e inicializar os objetos dessa classe
- Um construtor também pode sofrer o processo de sobrecarga

# Sobrecarga de Construtores

- Exemplo: classe Circulo
  - Deseja-se ter a capacidade de inicializar os atributos de um novo objeto de duas formas:
    - através de um construtor sem parâmetros, que cria um círculo padrão de centro (0,0) e raio 1,
    - e através de um construtor que recebe as informações de centro e raio para criar o círculo.

| Circulo |
| --- |
| -centrox:int<br>-centroy:int<br>-raio:int |
| +Circulo(x:int, y:int, r:int)<br>+Circulo()<br>+area():double<br>+circunferencia():double<br>+diametro():int |

# Sobrecarga de Construtores

```
public class Circulo {
   private int centrox;
   private int centroy;
   private int raio;
   public Circulo(int x, int y, int r){
      centrox = x;
      centroy = y;
      raio = r;
   }
   public Circulo() {
      centrox = 0;
      centroy = 0;
      raio = 1;
   }
   ...
}
```

# Sobrecarga de Construtores

- Testando a classe:

```
public class TesteCirculo {
  public static void main (String args[]) {
    Circulo circ1 = new Circulo();
    Circulo circ2 = new Circulo(1,2,4);
    System.out.println("Area circ1= " + circ1.area());
    System.out.println("Area circ2= " + circ2.area());
  }
}
```

# Sobrecarga de Construtores

- Observando mais de perto a implementação dos dois construtores da classe Circulo:
  - Nota-se que o segundo construtor (o construtor sem parâmetros) possui o mesmo código de inicialização do primeiro construtor (o construtor com três parâmetros)
- Repetir desnecessariamente código não é uma boa prática de programação
- Java permite compartilhar código entre os diversos construtores
  - Palavra-chave this()

# Sobrecarga de Construtores

```
public class Circulo {
   private int centrox;
   private int centroy;
   private int raio;
   public Circulo(int x, int y, int r){
      centrox = x;
      centroy = y;
      raio = r;
   }
   public Circulo() {
      this(0,0,1);
   }
   ...
}
```

# Atributos e Métodos de Classe

- Java permite declarar duas categorias distintas de atributos e métodos:
  - atributos de instância
  - atributos de classe
  - métodos de instância
  - métodos de classe

# Atributos de Classe

- Cada objeto de uma classe possui sua própria cópia de todos os atributos de instância da classe
- Em certos casos, entretanto, é interessante que apenas uma cópia de um atributo em particular seja compartilhada por todos os objetos de uma classe
- Exemplo: constantes da classe *Math*
  - As constantes matemáticas E e PI são armazenadas em um única cópia e então compartilhadas

# Atributos de Classe

```
public class TestaMath {
   public static void main(String args[]) {
      System.out.println(”PI = " + Math.PI);
      System.out.println(”E = " + Math.E);
   }
}
```

- Note que os atributos públicos não são acessados a partir de um objeto!
- Atributos acessados pelo nome da classe

# Atributos de Classe

- Atributos de Instância:
  - Cada objeto possui uma cópia particular com seus valores
  - Representam o estado de um objeto em particular
- Atributos de Classe:
  - Cada classe possui uma única cópia do atributo, independente do número de objetos instanciados a partir da classe
  - Objetos compartilham os atributos de classe
  - São declarados pela palavra-chave *static*
  - Invocação

  <nome_classe>.<nome_atributo_público>

# Atributos de Classe

- Exemplo: classe Circulo
  - Nos métodos de cálculo da área e circunferência, percebe-se a presença de um valor importante em cálculos geométricos que se repete para todas as instâncias
    - Esse valor é a constante Pi
  - Pode ser desejado manter somente uma cópia desse valor, com a aproximação desejada no número de suas casas decimais de uma forma consistente, impedindo que em um método seja utilizado o valor 3,14 e em outro 3,1415
  - PI será declarado como atributo de classe (*static*) e constante (*final*)

# Atributos de Classe

```
public class Circulo {
  public static final double PI = 3.14;
  private int centrox;
  private int centroy;
  private int raio;
  ...
  public double area() {
    return (PI * raio * raio);
  }
  public double circunferencia() {
    return (2 * PI * raio);
  }
  ...
}
```

# Inicialização de Atributos de Classe

- Convém destacar que a forma de inicialização dos atributos de classe é usualmente no momento de sua declaração, pois eles não pertencem às instâncias e portanto não dependem do construtor para serem inicializados
  - Se a inicialização com valores padrão for suficiente, não é necessário inicializar o atributo explicitamente

# Inicialização de Atributos de Classe

- Para inicializar atributos de classe que necessitam de uma forma mais complexa, Java fornece um bloco de inicialização estático
  - Não possui nome
  - Não possui tipo de retorno
  - Começa pela palavra-chave static, seguido de um bloco de código entre parênteses
  - Executa somente uma vez quando a classe é carregada em memória

```
public class UmaClasse {
  ...
  public static int atributo;
  static {
     //código para inicializar atributo
  }
}
```

# Métodos de Classe

- Em muitos exemplos de classes pode-se notar alguns métodos que não acessam nenhum atributo de uma instância
- Exemplo: funções trigonométricas da classe *Math*
  - Os métodos sin, cos e tan recebem o valor do ângulo (em radianos) por parâmetro e devolvem o seno, cosseno ou a tangente correspondente calculados unicamente a partir do valor recebido

# Métodos de Classe

```
public class Trigonometria {
  public static void main(String args[]) {
    System.out.println("Seno(45) = " +
  Math.sin(Math.PI/4));
    System.out.println("Coseno(45) = " +
  Math.cos(Math.PI/4));
    System.out.println("Tangente(45) = " +
  Math.tan(Math.PI/4));
  }
}
```

- Note que os métodos de cálculo não são executados sobre um objeto!
- Métodos acessados pelo nome da classe

# Métodos de Classe

- Métodos de Instância:
  - Fornecem o comportamento dos objetos instanciados a partir de uma classe
  - Trabalham sobre os atributos de instância de um objeto dessa classe
- Métodos de Classe:
  - Fornecem um comportamento que é independente da existência de objetos de uma classe
  - Pertencem à classe e são compartilhados por todas as instâncias da classe
  - Podem acessar os atributos de classe, mas não os atributos de instância diretamente
  - Indicados pela palavra-chave *static*
  - Invocação

<nome_classe>.<nome_método>(<parâmetros>)

# Métodos de Classe

- Exemplo: classe Circulo
  - O método *equacaoGeral* será acrescentado à classe Circulo
  - Seu propósito é, a partir dos valores de centro e raio de um círculo, obter a representação textual da chamada equação geral da circunferência

# Métodos de Classe

```
public class Circulo {
  ...
  public static String equacaoGeral(int x, int y, int r) {
    int a = -2 * x;
    int b = -2 * y;
    int c = (x*x) + (y*y) - (r*r);
    StringBuffer eq = new StringBuffer("x2 + y2");
    if (a > 0) {
      eq.append(" + ");
      eq.append(a);
      eq.append("x");
    }
    else if (a < 0) {
      eq.append(" ");
      eq.append(a);
      eq.append("x");
    }
```

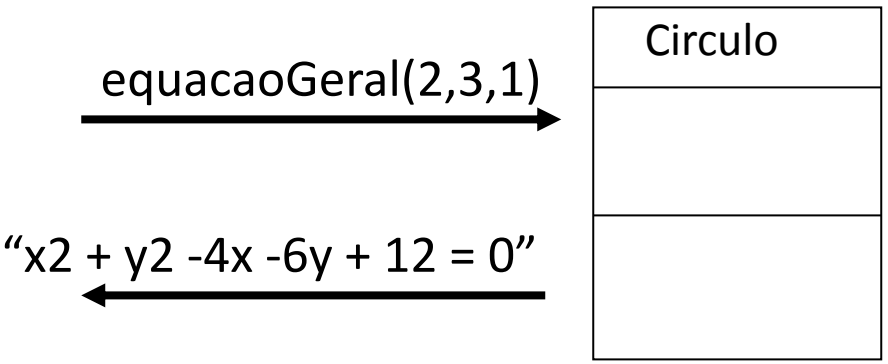
# Métodos de Classe

```
    if (b > 0) {
      eq.append(" + ");
      eq.append(b);
      eq.append("y");
    }
    else if (b < 0) {
      eq.append(" ");
      eq.append(b);
      eq.append("y");
    }
    if (c > 0) {
      eq.append(" + ");
      eq.append(c);
    }
    else if (c < 0) {
      eq.append(" ");
      eq.append(c);
    }
    eq.append(" = 0");
    return eq.toString();
  }
}
```

# Métodos de Classe

– Utilizando a nova definição em um exemplo:

```
String eq = Circulo.equacaoGeral(2,3,1);
```



# Recursos

- The Java Tutorial
  - http://download.oracle.com/javase/tutorial/index.html
- Java SE 6 API
  - http://download.oracle.com/javase/6/docs/api