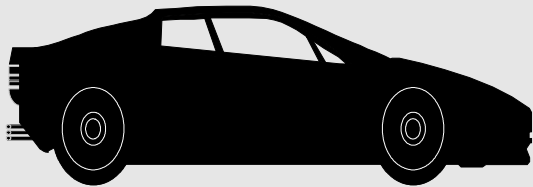


# *Orientação a Objetos em Java*

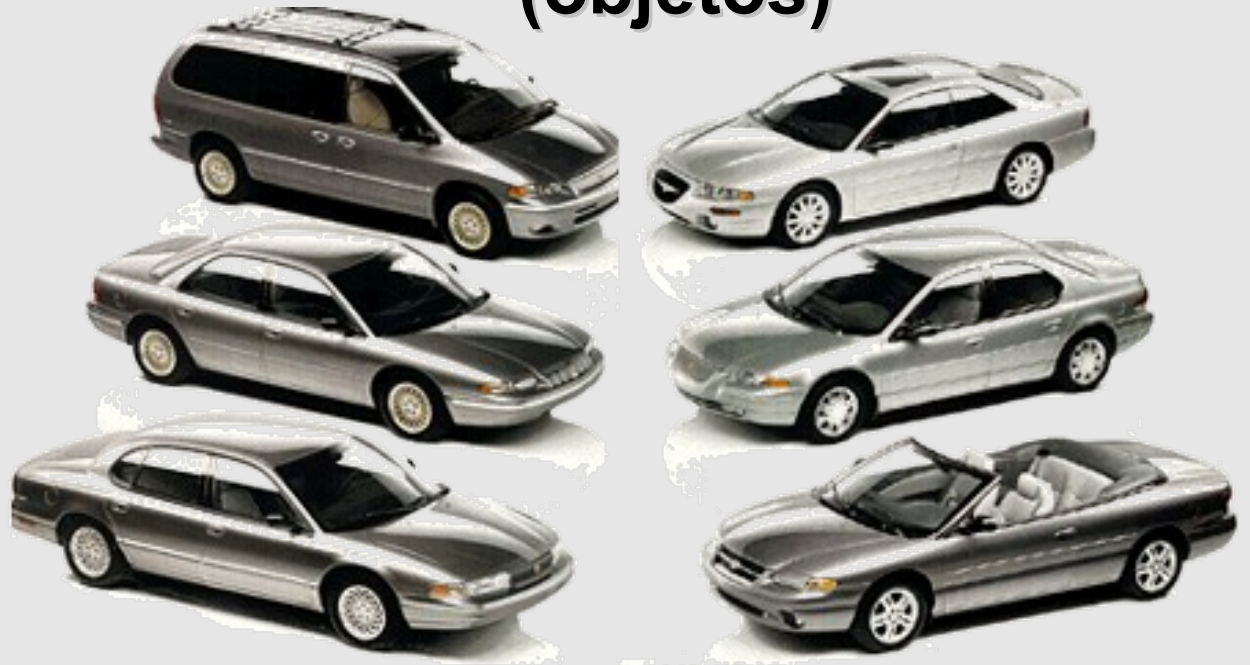
# Orientação a Objetos em Java

## Classes x Objetos



**Classe  
(abstração)**

**Instâncias da classe  
(objetos)**



# Orientação a Objetos em Java

## Como Encontrar Objetos?

- Sistema de software é um Modelo Operacional, baseado na interpretação do mundo.
  - Os objetos que compõem o software devem ser a representação dos objetos relevantes que constituem o mundo exterior.
  - Devem mapear os objetos reais em objetos computacionais e escrever programas que dão vida a estes objetos em um sistema computacional.
- Conclusão:
  - “os objetos estão por aí; é só pegá-los.”

# Orientação a Objetos em Java

## Como Descrever Objetos?

- Objetos são descritos a partir de classes
- Uma classe contém a definição das características que os objetos daquela classe terão
  - Definição dos Atributos
  - Definição dos Métodos

# Orientação a Objetos em Java

## Classe



- Classes definem um novo tipo de dado e como ele é implementado
- Define as características que os objetos daquela classe terão
- Definições:



### Atributos

- *são os dados, as variáveis relacionadas*



### Métodos

- *são as operações que os objetos daquela classe poderão executar*



- Sintaxe:

```
class NomeDaClasse {  
    // Atributos  
    // Métodos  
}
```

Nomes de classes começam com letra maiúscula e seguem este padrão (convenção)

# Orientação a Objetos em Java

## Classe – Exemplo Prático



- Definição da Classe Circulo

```
class Circulo {  
    double posX, posY;  
    double raio;  
  
    double getDiametro() {  
        return 2 * raio;  
    }  
  
    double getArea() {  
        return 3.14159 * raio * raio;  
    }  
}
```

Atributos posX, posY e raio

Método getDiametro()

Método getArea()

# Orientação a Objetos em Java

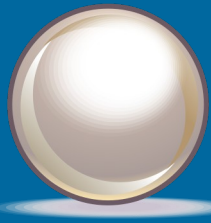
## Objeto



- Objetos são as instâncias de uma classe
- Os Objetos são criados em tempo de execução
  - Eles ocupam espaço na memória
  - Esse espaço é alocado quando um novo objeto de uma classe é criado
  - E liberado automaticamente quando o objeto não está mais em uso
- Um objeto é criado apenas através do operador `new`

# Orientação a Objetos em Java

## Objeto – Exemplo Prático



- Instanciando a Classe Circulo (i.e., criando um objeto)

```
class Principal {
```

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

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

```
        circ.posX = 7.5;
```

```
        circ.posY = 3.1;
```

```
        circ.raio = 2.5;
```

```
        System.out.println("O circulo c está em ("
            + circ.posX + ", " + circ.posY
            + "). Possui raio " + circ.raio
            + " e area " + circ.getArea());
```

```
    }
```

```
}
```

Instanciando a classe  
Circulo e criando o  
objeto circ

Modificando os atributos  
do objeto circ

Acessando o  
atributo raio

Executando o  
método getArea()



# Orientação a Objetos em Java

## Objeto – Exemplo Prático



- A seguinte sentença realiza quatro ações:

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

- Declaração
  - *c* será um objeto da classe *Circulo*
  - Declarações não criam objetos, apenas declaram
- Instanciação
  - *new* é um operador que cria dinamicamente um novo objeto.
- Inicialização
  - Chamada ao construtor da classe *Circulo* (veremos mais adiante)
- Atribuição
  - A referência do objeto retornada pelo operador *new* é atribuída à variável *circ*

# Orientação a Objetos em Java

## Atributos



- Atributos são os dados relacionados à classe
  - São variáveis dentro do escopo de uma classe
  - Servem para armazenar informações relacionadas
- Cada atributo possui um tipo, um nome e modificadores de acesso
- Exemplos:

```
int anoPublicacao;  
double raio;  
float notaFinal = 9.8f;  
String nomeMestre = "Yoda";
```

- Nomes dos atributos
  - São nomes completos, nomes abreviados ou frases
  - A primeira palavra começa com letra minúscula
    - *as outras palavras começam com letra maiúscula sem o uso de underscores (“\_”)*

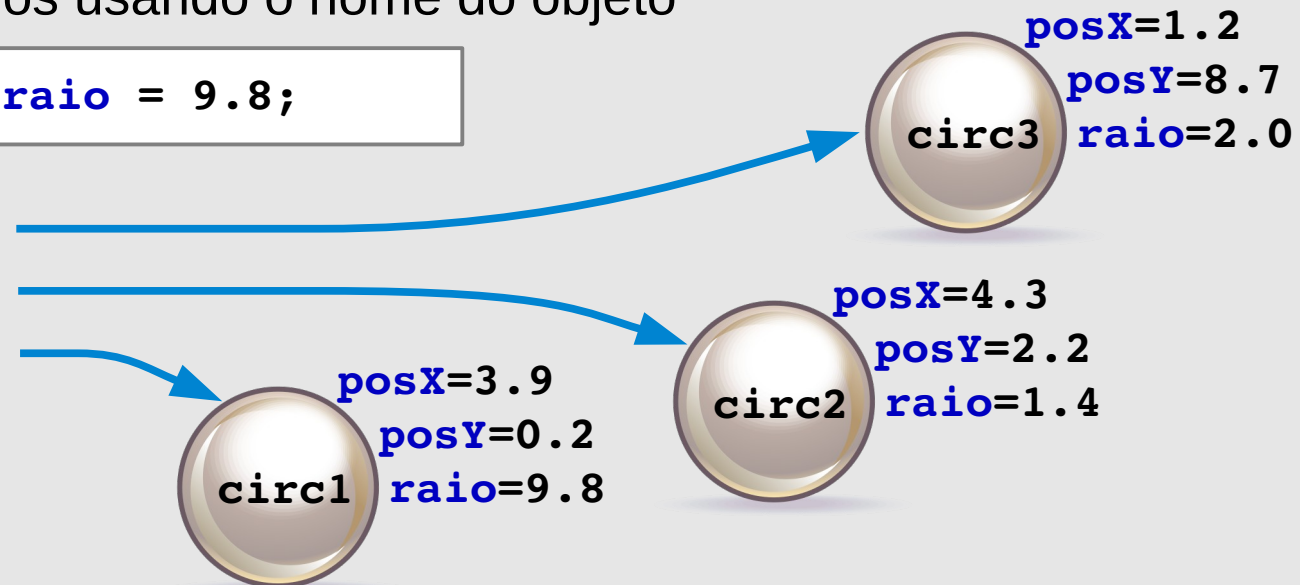
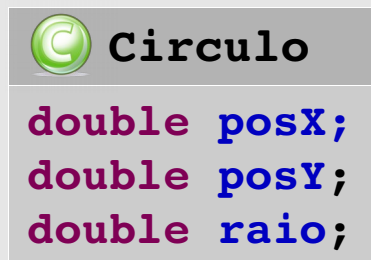
# Orientação a Objetos em Java

## Atributos de Instância e de Classe



- Atributos podem ser de Instância (padrão) ou de Classe
- Atributos de Instância
  - O atributo pertence ao objeto (instância da classe)
  - É o que usamos até o momento
  - Cada objeto instanciado terá uma região na memória para aquele atributo
  - Cada objeto poderá ter valores diferentes para aquele atributo
  - São acessados usando o nome do objeto

```
circ1.raio = 9.8;
```



# Orientação a Objetos em Java

## Atributos de Instância e de Classe




- Atributos de Classe (ou Atributos Estáticos)
  - O atributo pertence à classe e tem apenas uma posição na memória
  - Todos os objetos daquela classe irão enxergar o mesmo valor e
    - *se um objeto mudar o valor do atributo de classe, todos os objetos daquela classe irão enxergar o valor modificado*
  - São declarados utilizando o modificador de acesso “static”

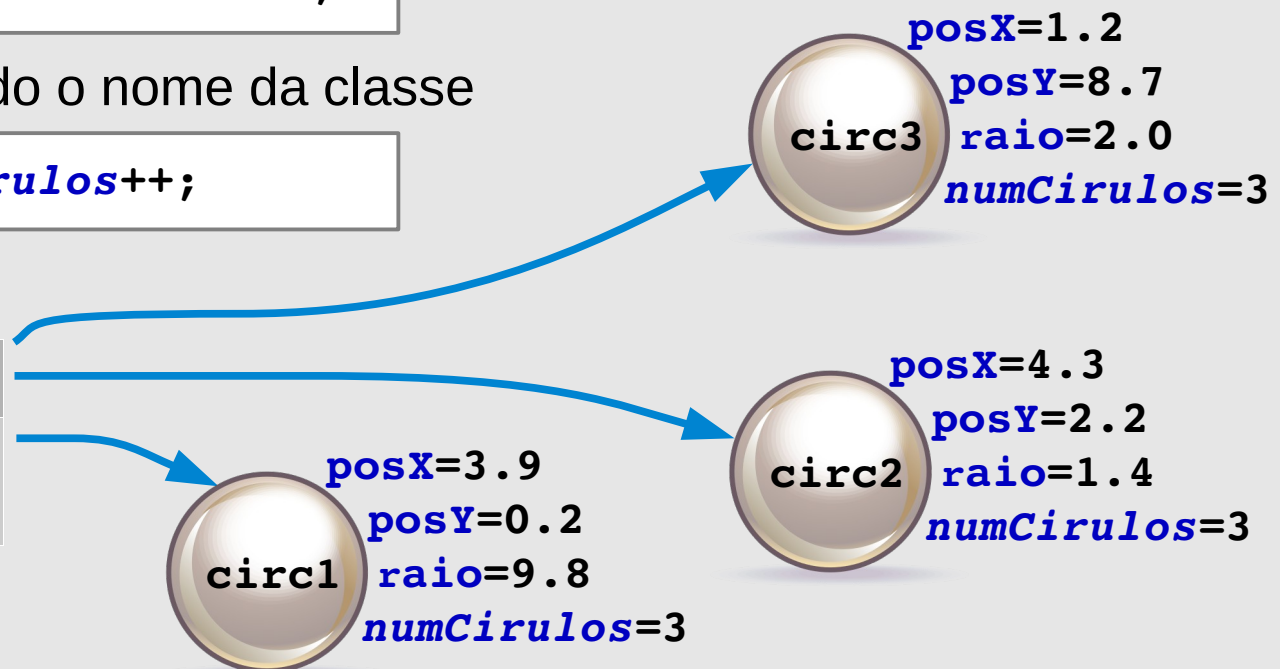
```
static int numCirculos = 3;
```

- São acessados usando o nome da classe

```
Circulo.numCirculos++;
```

 **Circulo**

```
double posX, posY, raio;  
static int numCirculos=3;
```



# Orientação a Objetos em Java

## Atributos Constantes



- Uma constante é um atributo cujo valor não muda após a primeira atribuição
  - São declarados usando o modificador de acesso “final”
  - Não precisam ser inicializados durante a declaração
    - *Mas uma vez inicializados, seu valor não muda*
  - Em geral, constantes são também atributos de classe (estáticos) e públicos (veremos futuramente). Isso facilita o acesso à constante.
  - Nomes de constantes são com letras maiúsculas e *underscores*
  - Exemplos

- *Constantes da classe Math (Java)*

```
public static final double E = 2.7182818284590452354;  
public static final double PI = 3.14159265358979323846;
```

- *Constantes da classe Integer (Java)*

```
public static final int MIN_VALUE = 0x80000000;  
public static final int MAX_VALUE = 0x7fffffff;  
public static final int SIZE = 32;  
public static final int BYTES = SIZE / Byte.SIZE;
```



- Métodos são as operações que os objetos da classe podem executar
  - São funções ou procedimentos
  - Executam um algoritmo
  - Acessam e modificam atributos
  - Retornam um valor ou `void`
  - Podem possuir variáveis locais
    - *Os nomes das variáveis locais seguem as mesmas regras dos nomes dos atributos, mas em geral são menores (acrônimos, abreviações, etc)*
- Possuem as mesmas convenções de nomes que os atributos, mas os nomes normalmente são verbos

# Orientação a Objetos em Java

## Métodos



- Cada método possui modificadores de acesso, um retorno, um nome, parâmetros e uma implementação
  - Exemplo da classe Circulo

```
double getArea() {  
    return 3.14159 * raio * raio;  
}
```

- Exemplo da classe String (Java)

```
public char charAt(int index) {  
    if ((index < 0) || (index >= value.length)) {  
        throw new StringIndexOutOfBoundsException(index);  
    }  
    return value[index];  
}
```

# Orientação a Objetos em Java

## Métodos de Classe (Estáticos)



- Assim como os atributos, também existem Métodos de Instância (usados até o momento) e os Métodos de Classe (Estáticos)
- Métodos de Classe
  - São declarados utilizando o modificador de acesso “static”
  - Não precisam de uma instância (objeto) para serem executados
    - *Podem ser executados diretamente usando o nome da classe*
  - Não podem acessar atributos de instância
    - *Só podem acessar atributos de classe (estáticos)*
    - *Porque? Imagine que não exista nenhuma instância daquela classe na memória, como o atributo será acessado? E se tiver várias instâncias? Qual seria usada?*



# Orientação a Objetos em Java

## Métodos de Classe (Estáticos)



- Exemplo de Método de Classe

- Declaração do método sqrt (raiz quadrada) da classe Math (java)

```
public static double sqrt(double a) {  
    return StrictMath.sqrt(a); // delegates to StrictMath  
}
```

- Declaração do método sqrt da classe StrictMath

```
public static native double sqrt(double a);
```

- O modificador “native” indica que o método é implementado em C/C++

- Exemplo de uso do método de classe sqrt

```
// O ponto (a,b) pertence ao círculo?  
boolean pertence(double a, double b) {  
    double dx = a - posX;  
    double dy = b - posY;  
    double dist = Math.sqrt(dx*dx - dy*dy);  
    if (dist <= raio) return true;  
    else return false;  
}
```

# Orientação a Objetos em Java

## Sobrecarga de Métodos



- Em Java, dois ou mais métodos podem ter o mesmo nome!
  - Entretanto, precisam ter parâmetros de tipos/quantidades diferentes
  - Isso é conhecido como “sobrecarga de métodos”
  - O Java diferencia um método do outro observando os parâmetros
  - Exemplo:
    - *O método `println` que você tem usado para imprimir texto, possui diversas implementações com parâmetros diferentes (classe `PrintStream`)*

```
public void println()           { /* ... */ }
public void println(boolean x) { /* ... */ }
public void println(char x)     { /* ... */ }
public void println(char[] x)   { /* ... */ }
public void println(double x)   { /* ... */ }
public void println(float x)    { /* ... */ }
public void println(int x)      { /* ... */ }
public void println(long x)     { /* ... */ }
public void println(Object x)   { /* ... */ }
public void println(String x)   { /* ... */ }
```

Este é o que  
você vem  
utilizando,  
provavelmente

# Orientação a Objetos em Java

## Métodos Especiais



- Em Java, temos dois métodos especiais
  - Método Construtor
    - *Executado quando um novo objeto daquela classe é criado. Método que atribui valores padrões (default) para os atributos de um objeto (dentre outras coisas)*
  - Método Destrutor
    - *Executado quando o coletor de lixo vai remover o objeto da memória. Usado para liberar recursos alocados (arquivos, conexões, etc)*

# Orientação a Objetos em Java

## Métodos Construtores



- Método Construtor
  - Executado quando um novo objeto daquela classe é criado
  - Utilizado para inicializar os atributos do novo objeto
    - *dentre outras coisas*
  - Possui o mesmo nome da classe
    - *Como nomes de classes começam com letra maiúscula, este método também começará com letra maiúscula*
  - Permite sobrecarga
    - *Podemos ter diversos construtores (com o mesmo nome da classe), desde que estes possuam parâmetros diferentes*
  - Não retornam valor
    - *Estes são os únicos métodos que não retornam valor (nem mesmo o void)*

# Orientação a Objetos em Java

## Métodos Construtores – Exemplo



- Criando construtores para a classe `Circulo`

```
class Circulo {  
    double posX, posY;  
    double raio;  
  
    Circulo() {  
        posX = 0.0;  
        posY = 0.0;  
        raio = 0.0;  
    }  
  
    Circulo(double raio) {  
        posX = 0.0;  
        posY = 0.0;  
        this.raio = raio;  
    }  
  
    Circulo(double posX, double posY, double raio) {  
        this.posX = posX;  
        this.posY = posY;  
        this.raio = raio;  
    }  
  
    // Métodos getDiametro, getArea ...  
}
```

Este construtor não possui parâmetros.  
Para criar um objeto usando este construtor, usamos:  
`Circulo circ = new Circulo();`

Este construtor possui um parâmetro do tipo `double`.  
Para criar um objeto usando este construtor, usamos:  
`Circulo circ = new Circulo(1.2);`

Este `this` serve para diferenciar o atributo `raio` do parâmetro (variável local) `raio`.

Este construtor possui três parâmetros tipo `double`.  
Para criar um objeto usando este construtor, usamos:  
`Circulo circ = new Circulo(1.2, 3.5, 9.1);`

# Orientação a Objetos em Java

## Encadeamento de Construtores



- Para simplificar e reduzir código, um construtor pode chamar outro
  - Isso é conhecido como “Encadeamento de Construtores”
  - Esta é a forma recomendada de se criar construtores

```
class Circulo {  
    double posX, posY;  
    double raio;  
    Circulo() {  
        this(0.0, 0.0, 0.0);  
    }  
    Circulo(double raio) {  
        this(0.0, 0.0, raio);  
    }  
    Circulo(double posX, double posY, double raio) {  
        this.posX = posX;  
        this.posY = posY;  
        this.raio = raio;  
    }  
    // Métodos getDiametro, getArea ...  
}
```

Este construtor usa a palavra reservada `this` para chamar o construtor que possui três parâmetros do tipo `double` (último)

Mesma coisa, mas passando o `raio`

Este construtor é o principal. Inicializa todos os atributos e será usado pelos outros construtores

# Orientação a Objetos em Java

## Método Destrutor



- Método Destrutor
  - Executado quando o objeto é removido da memória
    - *Chamado pelo coletor de lixo (próximo slide)*
  - Na prática, é muito raramente utilizado
    - *Motivo: java não garante se ou quando o método será chamado*
  - É implementado através de um único método chamado `finalize`:

```
protected void finalize() { /*...*/ }
```

# Orientação a Objetos em Java

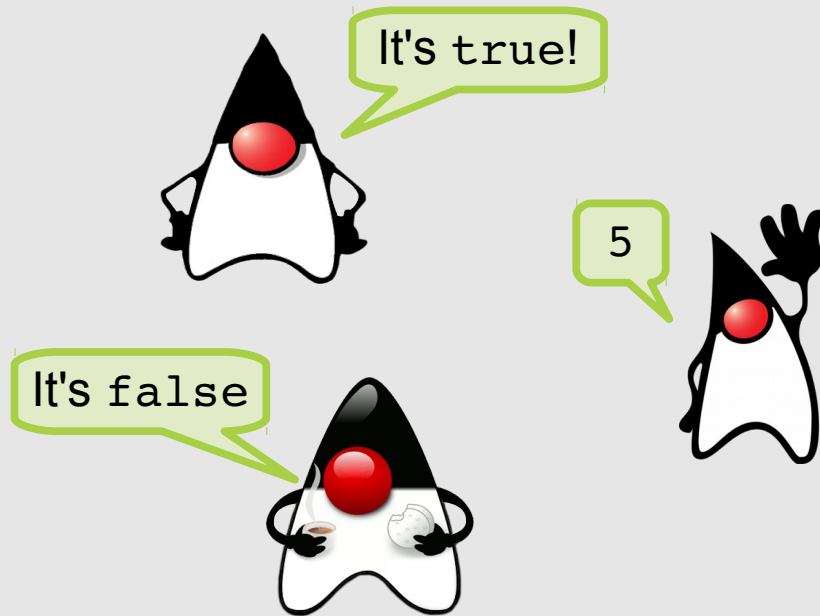
## Coletor de Lixo



- O Coletor de Lixo é uma das tecnologias chave da linguagem Java
  - Em Java, não podemos liberar memória!
  - Tudo é feito automaticamente pelo Coletor de Lixo
- Como o Coletor de Lixo sabe o que pode ou não ser liberado?
  - Por uma regra bem simples: se um objeto não possui mais referências para ele, então ele pode ser liberado
  - Apesar de isso funcionar na grande maioria das vezes, isso significa que se você não precisa mais de um objeto, mas está mantendo uma referência para ele, ele permanecerá na memória!
    - *Para evitar isso, basta setar as variáveis que apontam para o objeto para `null`*

```
circ = null;
```
    - *Isso remove a referência que `circ` tinha para o objeto e, na próxima coleta de lixo, ele irá ser removido da memória (talvez, dependendo do coletor)*





## *Tipos de Dados em Java*

# Tipos de Dados em Java

## Introdução

- Java é uma linguagem com tipagem estática
  - Toda variável e toda expressão possui um tipo que é conhecido em tempo de compilação
- Java é uma linguagem com tipagem forte
  - Tipos limitam os valores que uma variável pode armazenar
  - E limitam os valores que uma expressão pode produzir
  - Limitam também as operações suportadas pelos valores
  - E determinam o significado das operações

# Tipos de Dados em Java

## Categorias de Tipos

- Tipos em Java são divididos em duas categorias:
  - Tipos Primitivos e Tipos Referência
- Tipos Primitivos:
  - Tipos numéricos inteiros
    - *byte, short, int, long e char*
  - Tipos numéricos de ponto flutuante
    - *float e double*
  - Tipo booleano
    - *boolean*
- Tipos Referência
  - Classes
  - Interfaces
  - Vetores

# Tipos de Dados em Java

## Tipos Primitivos

- São os tipos “nativos” do hardware
  - `byte`      Inteiro, 8 bits
  - `short`      Inteiro, 16 bits
  - `int`      Inteiro, 32 bits
  - `long`      Inteiro, 64 bits
  - `float`      Ponto flutuante, 32 bits
  - `double`      Ponto flutuante, 64 bits
  - `boolean`      `true` ou `false`
  - `char`      Caractere Unicode, 16 bits, sem sinal
- Com exceção do `char` e do `boolean`, todos os tipos primitivos possuem com sinal
  - Não é possível criar números “unsigned” em Java

# Tipos de Dados em Java

## Tipos Primitivos: Operadores

- Em Java, podemos usar operadores nas variáveis

– Multiplicativos	*	/	%
– Aditivos	+	-	
– Relacionais	>	<	>= <=
– Igualdade/Diferença	==	!=	
– E lógico	&&		
– OU lógico			
– Seleção	e	?	e : e
– Atribuição	=		
– Atribuição+Operação	+=	--	*= /=
– Incremento/Decremento	++	-	
– Operadores de Bits	>>	<<	&   ^ ~

# Tipos de Dados em Java

## Tipos Primitivos: Inicialização

- **Atributos** de tipos primitivos são automaticamente inicializados para zero
  - O que acontece no código abaixo?

```
class Pessoa {  
    int     anoNascimento;  
    boolean solteiro;  
  
    public static void main(String args[]) {  
        Pessoa fulano = new Pessoa();  
        System.out.println("AnoNasc = " + fulano.anoNascimento  
                           + ", Solteiro? " + fulano.solteiro);  
    }  
}
```

```
$ javac Pessoa.java  
$ java Pessoa  
AnoNasc = 0, Solteiro? false
```

# Tipos de Dados em Java

## Tipos Primitivos: Inicialização

- Entretanto, **variáveis locais** de tipos primitivos **não** são inicializados
  - O que acontece no código abaixo?

```
class Pessoa {  
  
    public static void main(String args[]) {  
        int anoNasc;  
        System.out.println("AnoNasc = " + anoNasc);  
    }  
  
}
```

```
$ javac Pessoa.java  
Pessoa.java:5: variable anoNasc might not have been initialized  
    System.out.println("AnoNasc = " + anoNasc);  
                                ^  
1 error
```

# Tipos de Dados em Java

## Tipos Referência

- Também conhecidos como tipos não-primitivos
- Armazenam uma referência para um objeto
  - Uma referência é um endereço de memória aonde o objeto está armazenado
  - Por usar uma máquina virtual, o endereço armazenado em uma referência não corresponde ao endereço na memória física do computador
- Uma referência a um objeto pode ser do tipo
  - Classe
    - *Classes já existentes no Java*
    - *Classes criadas pelo usuário*
  - Interface (veremos futuramente)
  - Vetor
    - *Em java, um vetor é um objeto*



# Tipos de Dados em Java

## Tipos Referência x Tipos Primitivos

- Variáveis de Tipos Primitivos
  - São sempre utilizadas “**por valor**”
  - Quando usadas como parâmetros de métodos ou em atribuições, elas sempre serão uma **cópia** da variável original
  - Não é possível criar uma referência (ponteiro) para uma variável primitiva
    - *Por isso não existem os operadores & ou \* como na linguagem C/C++*
- Variáveis de Tipos Referência (classes)
  - São sempre utilizadas “**por referência**”
  - Quando usadas como parâmetros de métodos ou em atribuições, apenas a referência é passada. Portanto, mudando-se seus atributos dentro do método, mudará os atributos do objeto “original”
  - Para se “duplicar” um objeto, deve-se usar o método `clone`


# Tipos de Dados em Java

## Tipos Referência x Tipos Primitivos

- Exemplo

```
int a = 3;  
char b = 'c';  
Circulo c = new Circulo();
```

	Endereço	Conteúdo
a	0200	3
b	0300	'c'
c	0400	0900
	...	...
	0900	2.1,3.2,4.3



# Tipos de Dados em Java

## Objetos como Referências

- Qual a saída do código abaixo?

```
class A {  
    int i = 1;  
  
    public static void main(String[] args) {  
        A a1 = new A();  
        A a2 = new A();  
        System.out.println("a1.i = " + a1.i);  
        System.out.println("a2.i = " + a2.i);  
        System.out.println("a1 == a2 ? " + (a1 == a2) );  
        System.out.println("a1.i == a2.i ? " + (a1.i == a2.i));  
    }  
}
```


```
$ javac A.java  
$ java A  
a1.i = 1  
a2.i = 1  
a1 == a2 ? false  
a1.i == a2.i ? true
```

# Tipos de Dados em Java

## Objetos como Referências

- E agora? Qual a saída do código abaixo?

```
class A {  
    int i = 1;  
  
    public static void main(String[] args) {  
        A a1 = new A();  
        A a2 = a1;  
        System.out.println("a1.i = " + a1.i);  
        System.out.println("a2.i = " + a2.i);  
        System.out.println("a1 == a2 ? " + (a1 == a2) );  
        System.out.println("a1.i == a2.i ? " + (a1.i == a2.i));  
    }  
}
```



```
$ javac A.java  
$ java A  
a1.i = 1  
a2.i = 1  
a1 == a2 ? true  
a1.i == a2.i ? true
```

# Tipos de Dados em Java

## String, um Tipo Especial

- Os objetos da classe String são especiais e são tratados de forma diferente
  - Motivo: performance e facilidade (são frequentemente utilizadas)
- Strings podem ser inicializadas de duas formas:
  - Explicitamente através do operador new (como qualquer outro objeto):

```
String c = new String("Técnicas de Programação");
```

- Implicitamente através de uma "string literal":

```
String c = "Técnicas de Programação";
```

# Tipos de Dados em Java

## String, um Tipo Especial

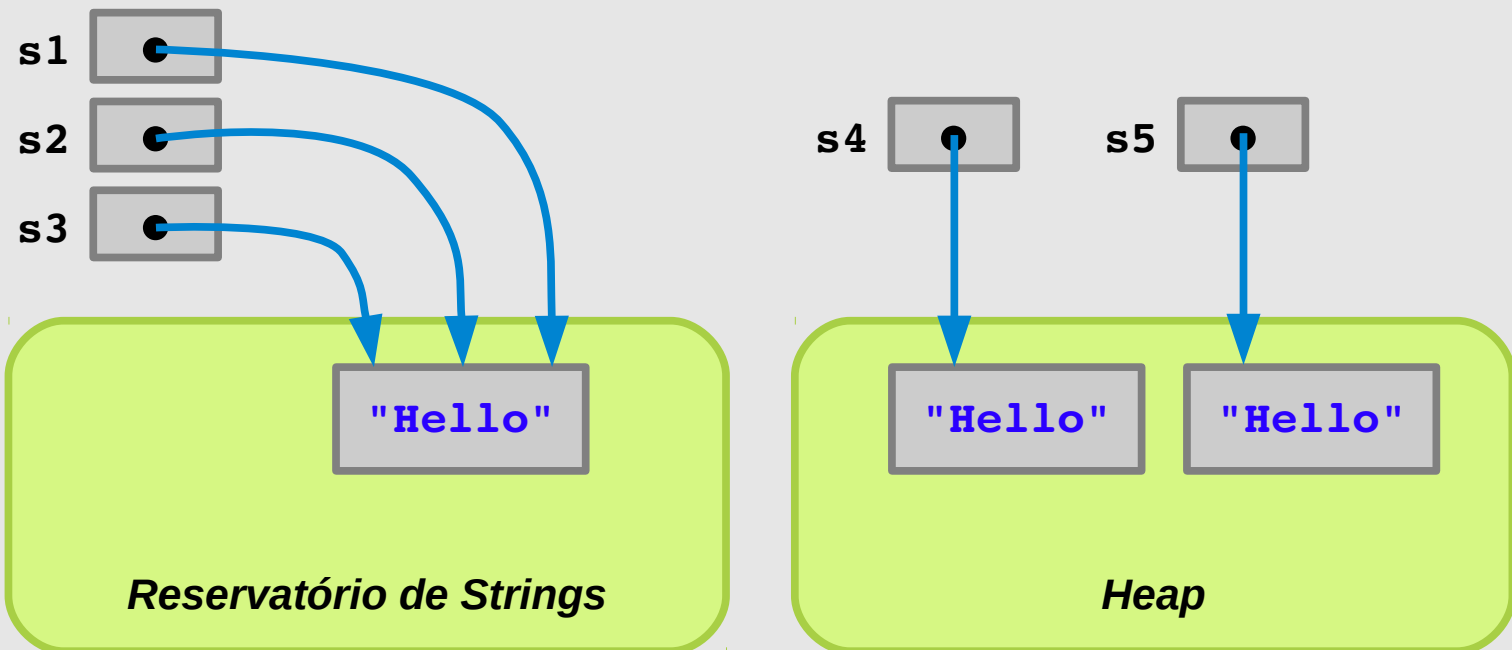
- String declaradas usando o operador `new` são armazenadas junto com os outros objetos no *heap* e se comportam como qualquer outro objeto
- Strings literais são armazenadas em um reservatório de strings:
  - Strings literais com o mesmo conteúdo, compartilham o mesmo endereço de memória
- Em ambos os casos:
  - O operador `+` é sobrecarregado para concatenar strings
  - As strings são imutáveis, ou seja, não tem como alterar seu conteúdo
    - *Para “mudar” uma string, deve-se atribuí-la a uma nova string contendo o novo valor. A string antiga, se não estiver mais em uso, será liberada pelo coletor de lixo.*

# Tipos de Dados em Java

## String, um Tipo Especial

- Exemplo

```
String s1 = "Hello";           // String literal
String s2 = "Hello";           // String literal
String s3 = s1;                 // Mesma Ref.
String s4 = new String("Hello"); // Objeto da classe String
String s5 = new String("Hello"); // Objeto da classe String
```



# Tipos de Dados em Java

## String, um Tipo Especial

- Exemplo

```
String s1 = "Hello";           // String literal
String s2 = "Hello";           // String literal
String s3 = s1;                 // Mesma Ref.
String s4 = new String("Hello"); // Objeto da classe String
String s5 = new String("Hello"); // Objeto da classe String
```

- Diga o resultado das expressões abaixo

```
s1 == s1;           // true, mesmo ponteiro
s1 == s2;           // true, compartilham end. do repositório
s1 == s3;           // true, mesmo ponteiro
s1.equals(s3);      // true, mesmo conteúdo
s1 == s4;           // false, ponteiros diferentes
s1.equals(s4);      // true, mesmo conteúdo
s4 == s5;           // false, diferentes ponteiros
s4.equals(s5);      // true, mesmo conteúdo
s1 == "Hello";      // true, mesmo ponteiro
s4 == "Hello";      // false, ponteiros diferentes
```



# Tipos de Dados em Java

## String, um Tipo Especial

- Strings podem ser concatenadas usando o operador +

```
String p1 = "Técnicas";  
String p2 = "Programação";  
String p3 = p1 + " de " + p2;           // Técnicas de Programação  
int ano = 2015;  
int sem = 2;  
String p4 = p3 + ano + "/" + sem; // Técnicas de Programação 2015/2
```

- O operador + gera uma nova string na memória e retorna a referência para ela
- Sempre que o Java encontra o operador + e um dos parâmetros é uma String, ele faz a concatenação

# Introdução ao Java

## Laboratório 2



- <http://tinyurl.com/slides-tp>
  - Laboratórios
  - TP – 2oLaboratorio.pdf
- Entrega por E-Mail
  - Para: [horacio.fernandes@gmail.com](mailto:horacio.fernandes@gmail.com)
  - Cópia: [moyses.lima@icomp.ufam.edu.br](mailto:moyses.lima@icomp.ufam.edu.br)
  - Assunto: TP: 2o Lab
- Data Limite:
  - Hoje, às 12hs
  - E-Mails recebidos após 12hs não serão considerados