PROGRAMAÇÃO ORIENTADA A OBJETOS

PROF. JOSENALDE OLIVEIRA

josenalde.oliveira@ufrn.br
https://github.com/josenalde/poo

ANÁLISE E DESENVOLVIMENTO DE SISTEMAS - UFRN

CONSTRUTORES

- Construtor é o método invocado quando da INSTANCIAÇÃO do Objeto com o operador new. Sua assinatura básica possui visibilidade PÚBLICA (public), seguido do mesmo nome da classe. Não possui tipo de retorno (nem mesmo void). Pode ter ou não parâmetros (chamado construtor parametrizado)

- Por padrão, se não for criado o construtor sem parâmetros, fica implícito.

- Contudo, se outro construtor parametrizado for definido, o sem parâmetros não é incluído automaticamente.

```
public Robo() {

}
```

```
public Robo(String name, int batDuration, boolean turnedOn) {
        this.name = name;
        this.batDuration = batDuration;
        this.turnedOn = turnedOn;
}
```

CONSTRUTORES

- Podemos ter vários construtores, com diferentes números de parâmetros

- Em geral usamos quando um ou mais parâmetros são definidos (padrão)

```
public Robo() {

}
```

```
public Robo(String name, int batDuration, boolean turnedOn) {
        this.name = name;
        this.batDuration = batDuration;
        this.turnedOn = turnedOn;
}
```

```
public Robo(String name, int batDuration) {
        this.name = name;
        this.batDuration = batDuration;
        this.turnedOn = false;
}
```

```
Robo r1 = new Robo();
Robo r2 = new Robo("android", 100, true);
Robo r3 = new Robo("r2d2", 100);
```

DECLARAÇÃO E INSTANCIAÇÃO DE OBJETOS

- Declaramos uma "variável" de determinada Classe, a qual é inserida na STACK (pilha) da Máquina Virtual Java (JVM)
  - Exemplos: `Robo r2d2;` `Carro meuCarro;` `Elevador elevador;`
  - Se compararmos com C/C++, esta "variável" poderia ser vista como o nome do ponteiro, o qual precisará receber um endereço.
- É necessário, contudo, que seja instanciado um novo objeto desta Classe com o operador new e esta "variável" irá referenciar o endereço atribuído para este objeto. Este endereço será armazenado no HEAP da JVM.
  - `Forma 1: Robo r2d2 = new Robo(); Carro meuCarro = new Carro(); Elevador elevador = new Elevador()`
  - `Forma 2 (suas linhas): r2d2 = new Robo(); meuCarro = new Carro(); elevador = new Elevador();`

DECLARAÇÃO E INSTANCIAÇÃO DE OBJETOS

- Declaramos uma "variável" de determinada Classe, a qual é inserida na STACK (pilha) da Máquina Virtual Java (JVM)
  - Exemplos: `Robo r2d2; Carro meuCarro; Elevador elevador;`
  - Se compararmos com C/C++, esta "variável" poderia ser vista como o nome do ponteiro, o qual precisará receber um endereço.

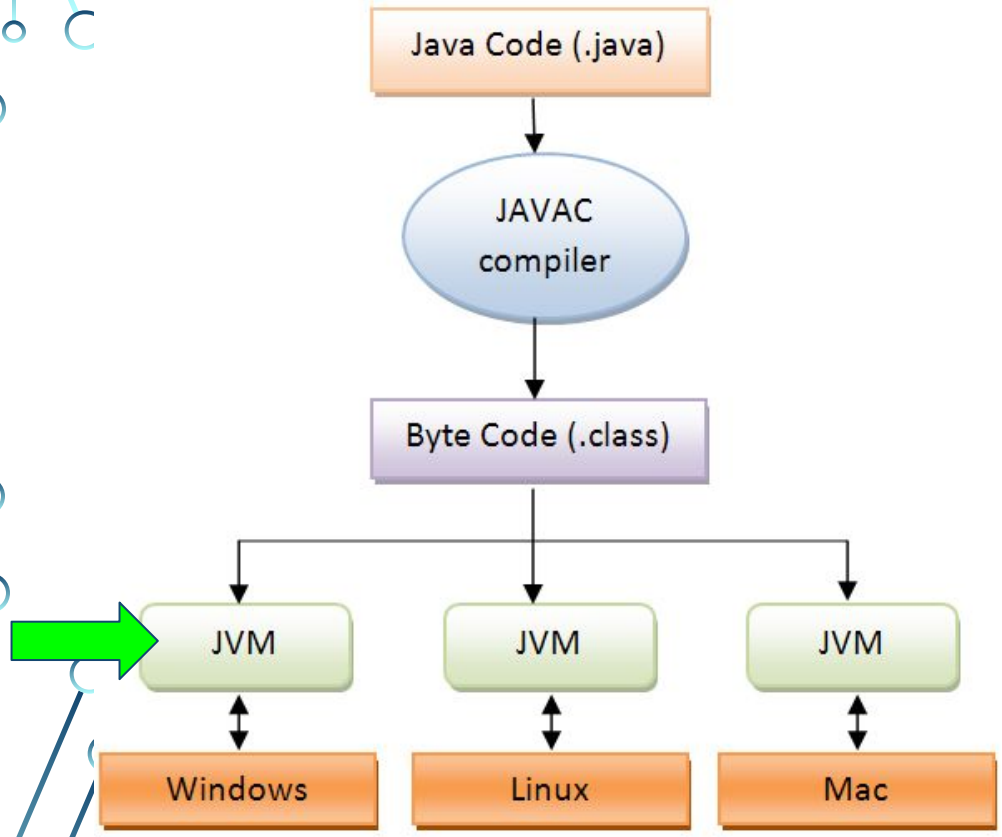
- É necessário, contudo, que seja instanciado um novo objeto desta Classe com o operador new e esta "variável" irá referenciar o endereço atribuído para este objeto. Este endereço será armazenado no HEAP da JVM.
  - `Forma 1: Robo r2d2 = new Robo(); Carro meuCarro = new Carro(); Elevador elevador = new Elevador`
  - `Forma 2: r2d2 = new Robo(); meuCarro = new Carro(); elevador = new Elevador();`

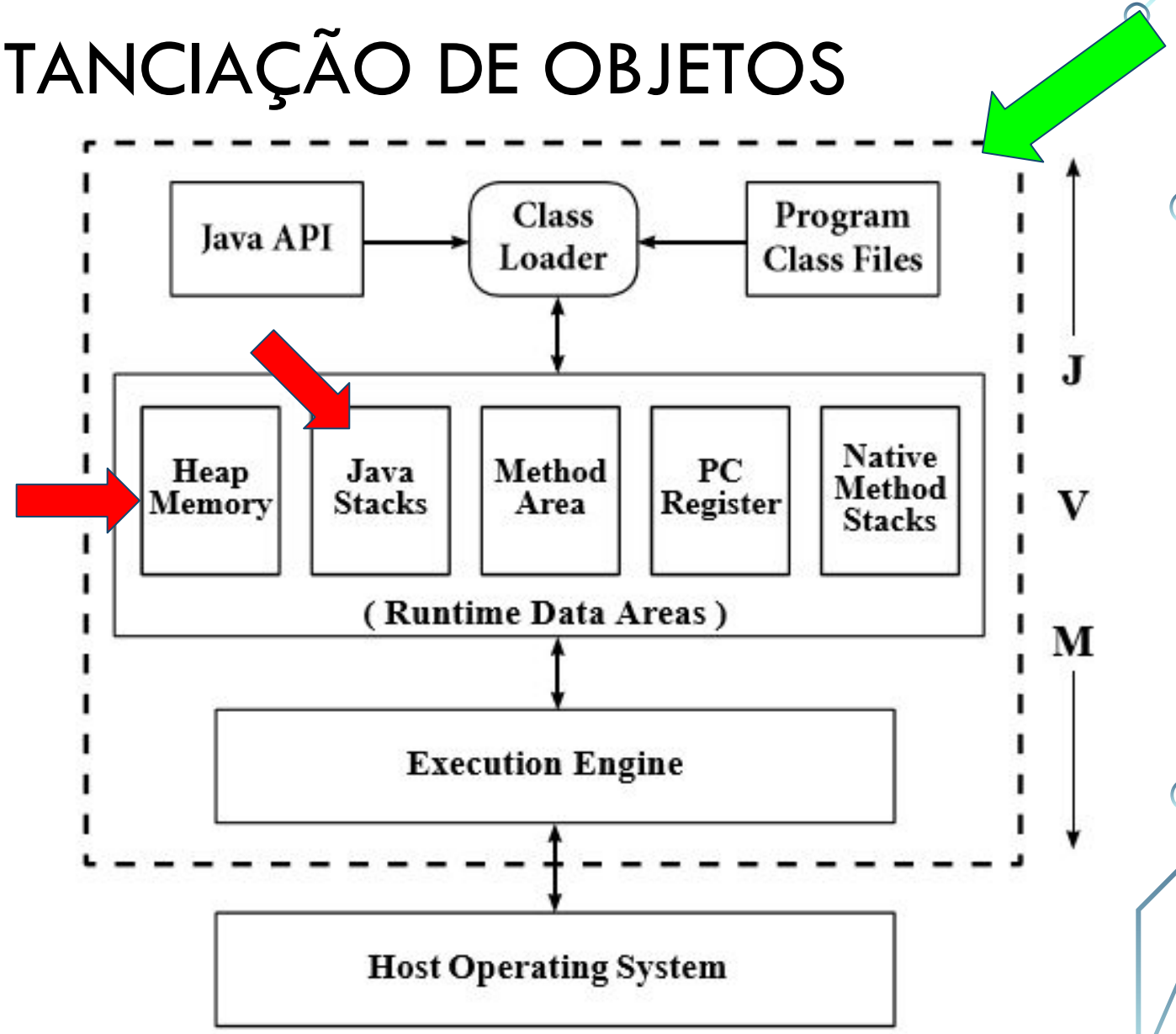
OBS: Ao tentar executar:

```
Robo r2;
System.out.println("#r2: " + Integer.toHexString(System.identityHashCode(r2)));
```

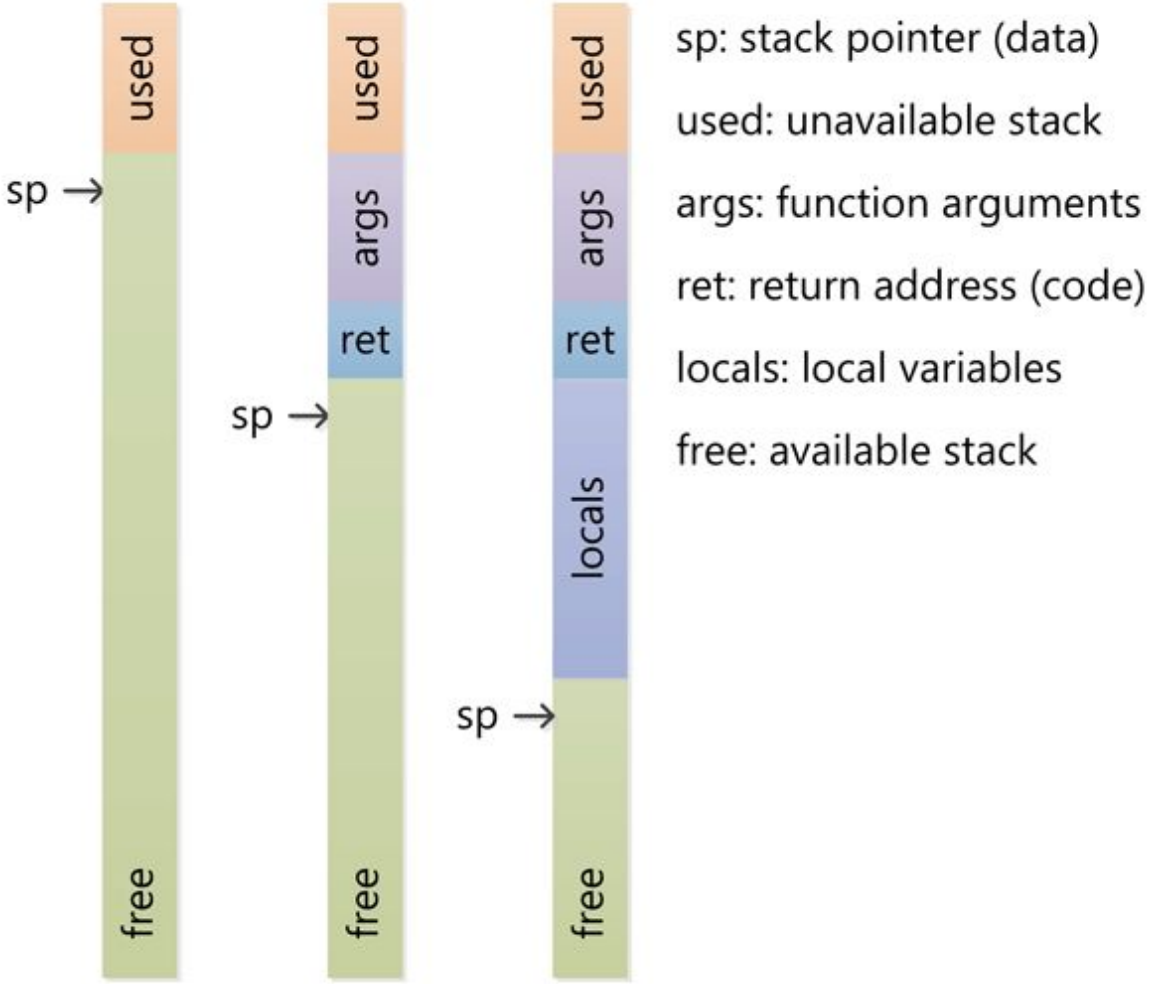
Erro! Informa que a variável r2 não foi inicializada

DECLARAÇÃO E INSTANCIAÇÃO DE OBJETOS





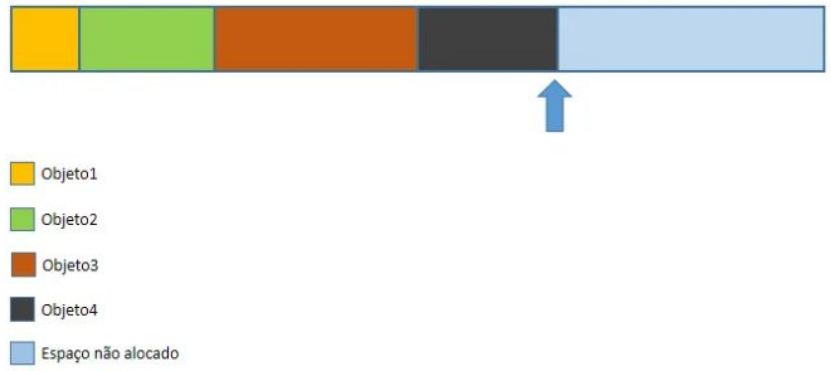
DECLARAÇÃO E INSTANCIAÇÃO DE OBJETOS

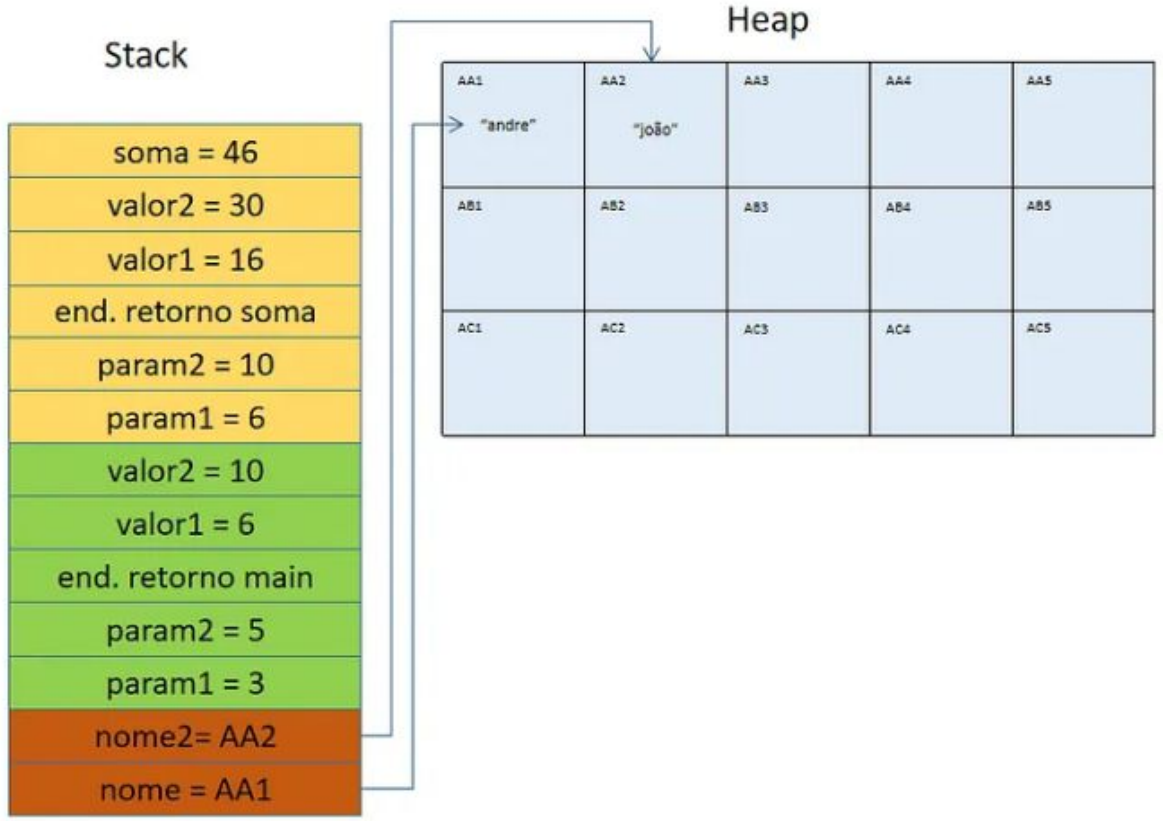


A stack é uma região de memória usada para armazenar dados temporários e informações de contexto relacionadas a cada chamada de função ou método em um programa. Cada vez que uma função ou método é chamado, um novo bloco de memória é alocado na stack para suas variáveis locais e parâmetros. Essa alocação é rápida, uma vez que basta mover um ponteiro para a stack para reservar espaço. Quando a função ou método retorna, o bloco de memória associado é desalocado e a stack é liberada para ser reutilizada. Isso implica que o tempo de vida das variáveis na stack é limitado ao escopo da função ou método em que estão definidas.

DECLARAÇÃO E INSTANCIAÇÃO DE OBJETOS

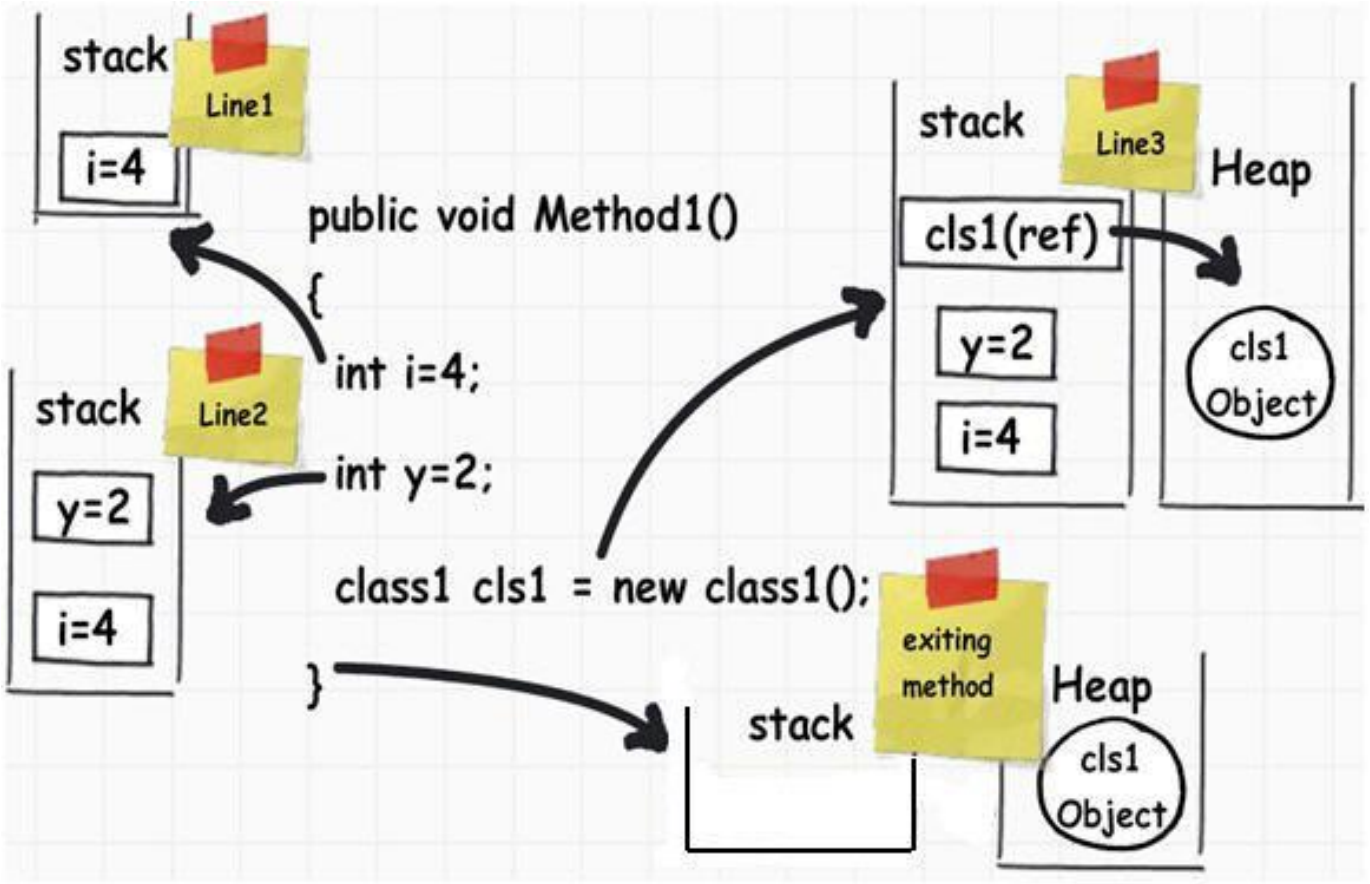
- O objeto é alocado no HEAP e apenas sua referência (endereço) é salvo pela sua variável (ponteiro) na STACK
- Os objetos no HEAP tem tamanho dinâmico





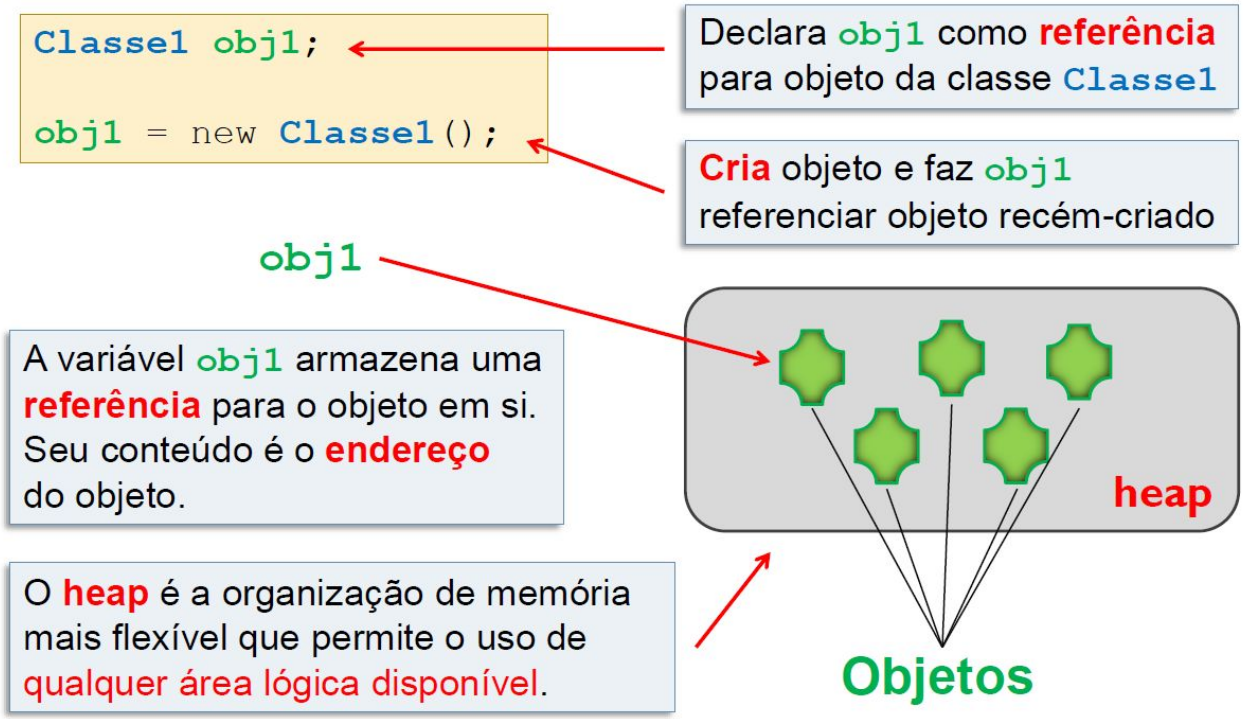
Fonte: Santarosa, A. Heap, Stack e Garbage Collector, 2021

DECLARAÇÃO E INSTANCIAÇÃO DE OBJETOS



DECLARAÇÃO E INSTANCIAÇÃO DE OBJETOS

- Agora, ao instanciar, tem-se um "endereço" atribuído:
  - `Robo r2 = new Robo();`
  - `System.out.println("#r2: " + Integer.toHexString(System.identityHashCode(r2)));`
  - Por exemplo, exibe o código hexadecimal 372f7a8d (hashcode)

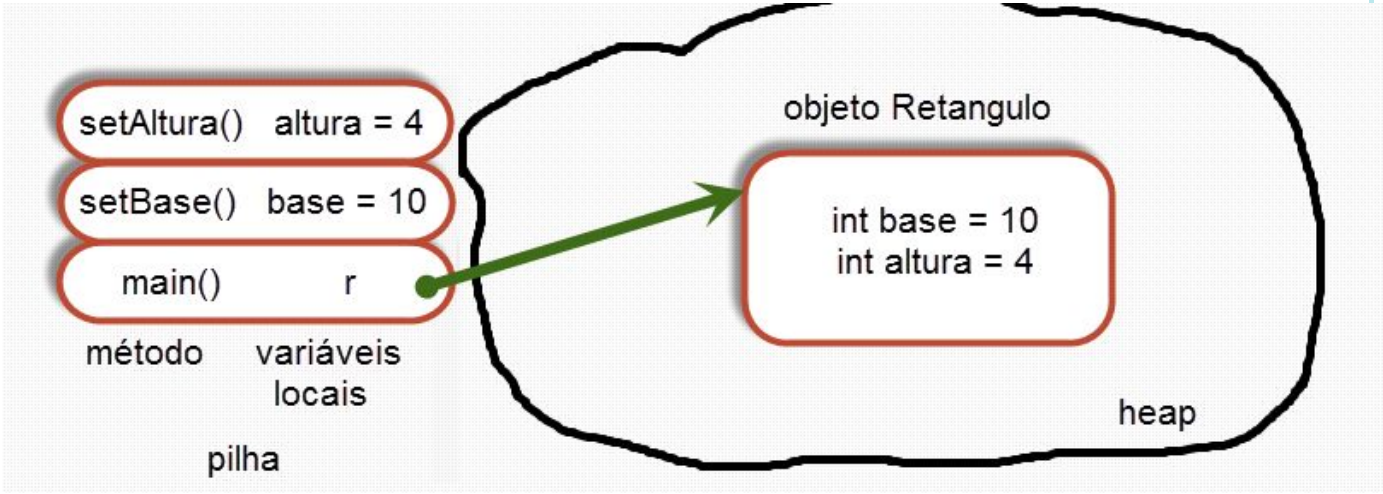


Fonte: Slides APDS Profa. Alessandra Mendes
Playlist POO (2021)

DECLARAÇÃO E INSTANCIAÇÃO DE OBJETOS

- Vejamos um outro exemplo:

```java
public class Retangulo {

private double base;
private double altura;

public double getBase() {
return base;
}

public void setBase(double base) {
this.base = base;
}

public double getAltura() {
return altura;
}

public void setAltura(double altura) {
this.altura = altura;
}

public static void main(String[] args) {
Retangulo r = new Retangulo();
r.setBase(10);
r.setAltura(4);
}

}
```



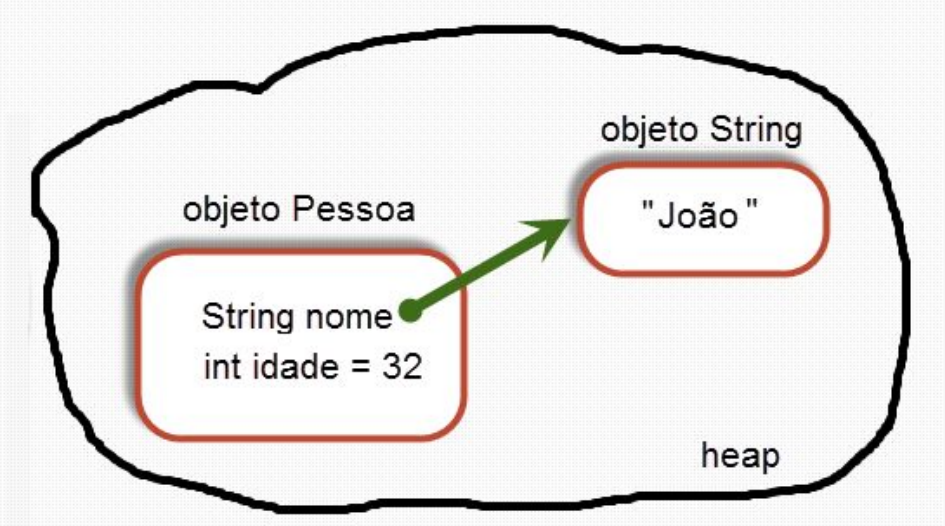
Fonte: https://www.devmedia.com.br/introducao-ao-java-garbage-collection/30326

- O método main() é colocado no topo da pilha, junto com variáveis locais e/ou referências nele, como r, que guarda o "endereço" do objeto Retangulo no HEAP
- setBase() e setAltura() são empilhados com as variáveis locais correspondentes (altura, base)
- Os atributos (variáveis de instância) e seus valores "vivem" dentro do objeto no HEAP, recebendo os valores das variáveis locais em main()

DECLARAÇÃO E INSTANCIAÇÃO DE OBJETOS

- Vejamos um outro exemplo:

```
Class Pessoa() {
        String nome; // objeto
        int idade; // primitiva
}
```



Objeto referenciado por nome no HEAP!

Fonte: **https:**//www.devmedia.com.br/introducao-ao-java-garbage-collection/30326

DESALOCANDO OBJETOS: COLETOR DE LIXO

- Garbage Collector: gerenciamento automático de memória, para evitar problemas de desalocação explícita dos espaços ocupados por objeto não mais referenciados, em Java, C#, Python, Go, outras...
  - Assegura que objetos referenciados (vivos) permaneçam na memória
  - Recuperar a memória alocada a objetos não mais alcançáveis, seja por referir-se a outro objeto ou null, ou se uma variável local perde o escopo (programa retorna do método onde foi declarada)
  - Embora System.gc() force o coletor de lixo, não se faz isto! Pois o GC monitora proximidade de falta de memória e é otimizado
  - No ciclo de coleta ocorrem as etapas de suspensão da aplicação, marcação de objetos não referenciados, compactação (correção de fragmentação) e retomada

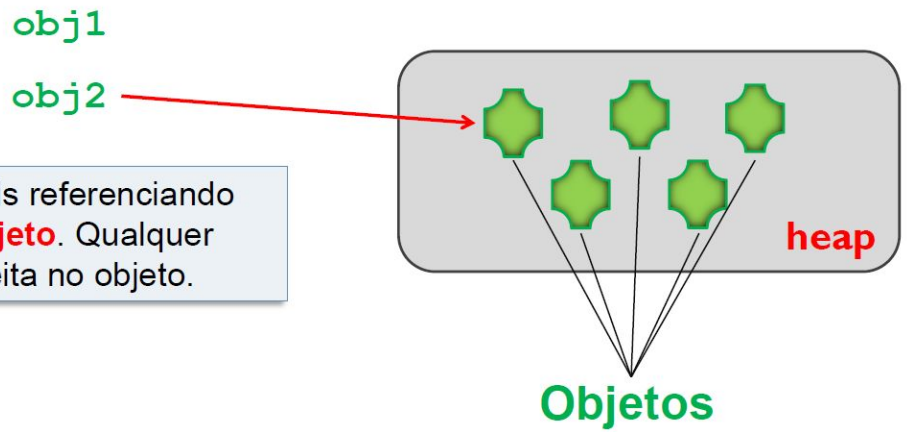
Leitura complementar sobre GC: https://www.devmedia.com.br/introducao-ao-java-garbage-collection/30326

DECLARAÇÃO E INSTANCIAÇÃO DE OBJETOS

- Fazer um objeto IGUAL a outro objeto, significa "apontar" para o mesmo local no HEAP:

```
Robo r2d2 = new Robo();
Robo r2d3;
r2d3 = r2d2;
System.out.println(r2d2 == r2d3); // O que acontece aqui?
System.out.println("#r2d2: " + Integer.toHexString(System.identityHashCode(r2d2)));
System.out.println("#r2d3: " + Integer.toHexString(System.identityHashCode(r2d3)));
```

Alterações em um refletem no outro



Duas variáveis referenciando **o mesmo objeto**. Qualquer alteração é feita no objeto.

E agora? r2d2 receberá um novo endereço, mas r2d3 referencia o endereço anterior de r2d2

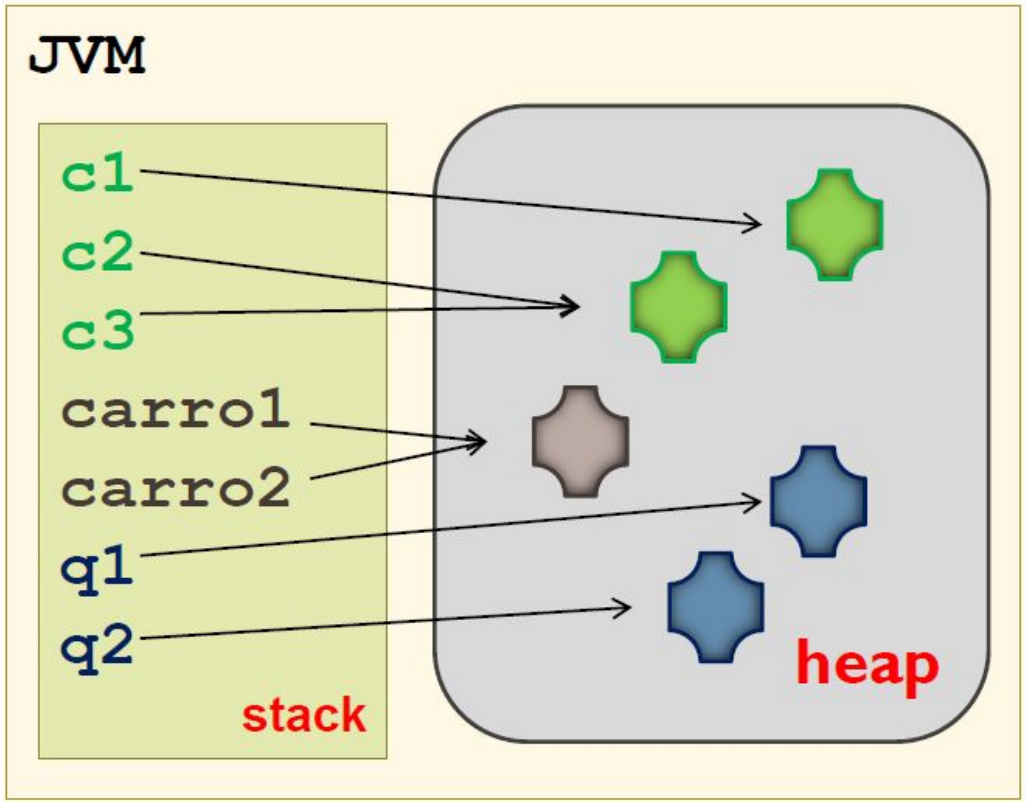
```
r2d2 = new Robo();
System.out.println(r2d2 == r2d3);
```

Fonte: Slides APDS Profa. Alessandra Mendes

DECLARAÇÃO E INSTANCIAÇÃO DE OBJETOS

- Vamos analisar o estado final do HEAP, após a sequência de comandos:

```
Circulo c1,c2,c3;
Carro carro1, carro2;
c1 = new Circulo();
Quad q1 = new Quad();
c2 = c1;
carro1 = new Carro();
Quad q2 = q1;
q1 = new Quad();
c3 = c1;
c1 = new Circulo();
carro2 = carro1;
```



Fonte: Slides APDS Profa. Alessandra Mendes

COMPARAÇÃO ENTRE OBJETOS

- O uso do operador de comparação (igualdade, ==) verifica se os valores (endereços) das variáveis são iguais, ou seja, se referenciam o mesmo objeto
- Já o método equals() verifica se dois objetos possuem o mesmo ESTADO, ou seja, se todos os seus atributos possuem os mesmos valores.

```
Robo r1 = new Robo();
r1.setName("android");
Robo r2 = new Robo();
r2.setName("android");
r1.setBatDuration(10);
r2.setBatDuration(10);
r1.setTurnedOn(true);
r2.setTurnedOn(true);
System.out.println(r2.equals(r1)); // mesmo ESTADO
```

Para que o método **equals** funcione corretamente, a documentação da API Java recomenda sobrescrever os métodos **hashCode** e **equals** na definição da classe. Iremos ver o que significa sobrescrever mais tarde, mas por ora no VSCODE pode-se criar automaticamente com o botão direito do mouse (source action – generate hashCode e equals)

COMPARAÇÃO ENTRE OBJETOS

Turma, alguns esclarecimentos sobre os métodos **hashCode() e equals()** no Java, visto ser um tópico de dúvida comum nos fóruns de discussão (quora, stackoverflow, medium etc.):

1) por padrão, o método equals() pertence à classe Object, a classe de mais alta hierarquia em Java. Iremos entender melhor este conceito de hierarquia ao estudarmos herança, mas já é possível entender que todas as demais classes Java estão abaixo da classe Object, são portanto suas filhas. A implementação portanto de equals() compara os "endereços", os hashCodes() na JVM para cada objeto. Então, se fizermos algo como:

Caneta canetaA = new Caneta();

Caneta canetaB;

CanetaB = canetaA;

E usarmos o teste (**canetaA == canetaB**) ou o teste (**canetaA.equals(canetaB)**) ambos os resultados serão TRUE, pois o "endereço" de ambas as canetas é o mesmo.

Se fizermos:

Caneta canetaA = new Caneta();

Caneta canetaB = new Caneta();

E usarmos o teste (canetaA == canetaB) ou o teste (canetaA.equals(canetaB)) ambos os resultados serão FALSE, pois os "endereços" das canetas são diferentes.

Agora suponha que fazemos o mesmo código, com um construtor com parâmetros, e ambas as canetas com os mesmos parâmetros:

Caneta canetaA = new Caneta("compactor","azul",0.7,10,true);

Caneta canetaB = new Caneta("compactor","azul",0.7,10,true);

Agora, o teste (canetaA == canetaB) retornará FALSE (pois os endereços são diferentes) e (canetaA.equals(canetaB)) também retornará FALSE. Mas deveria ser TRUE, já que as canetas tem o mesmo estado. Concordam?

A explicação é que o método equals() que está sendo usado é o da Classe Object, que compara endereços e não estados. Para que funcione, precisamos REESCREVER o método equals() dentro da classe Caneta, para que compare os atributos de uma caneta com outra caneta. Este processo de REESCREVER um método que já existe em outra classe (com a mesma assinatura) se chama SOBRESCRITA, algo que iremos estudar em breve, quando estudarmos HERANÇA. Felizmente não precisa reescrever neste caso, pois a IDE já permite GERAR automaticamente o equals().

Mas então por que também aparece o hashCode() SOBREESCRITO? Posso comentá-lo e não usá-lo para este exemplo? Sim, pode comentar. A documentação Java recomenda usá-los sempre em conjunto, pois várias estruturas de dados (HashMap, HashSet, ArrayList etc.) e métodos de ordenação e comparação usam o código hash dos objetos, portanto, calculam para cada objeto um código que os permita agrupar por similaridade etc. e tornar a ação (busca, ordenação etc.) mais rápida. Em conclusão, não há problema em deixar o método hashCode() já implementado nas suas Classes, pois se for usado em algum método já está lá, não dará erro semântico.

Prof. Josenalde Oliveira

Tem uma explicação adicional com exemplo, bem didática aqui: https://blog.algaworks.com/entendendo-o-equals-e-hashcode/