

# Programação Orientada a Objetos

POO

## Aula 2

Universidade Federal do Rio Grande do Norte  
Unidade Acadêmica Especializada em Ciências Agrárias  
Escola Agrícola de Jundiá  
Tecnologia em Análise e Desenvolvimento de Sistemas  
**Profa. Alessandra Mendes**



# Classes e Objetos em Java

# Criação de Objetos

---

- ▶ Como declarar variável?
  - ▶ Associa variável a tipo (classe)
  - ▶ Sintaxe: *NomeClasse nomeVariável*;
  - ▶ Exemplo: *Circulo c1*;
- ▶ Como criar um objeto (ou instanciar) e fazer com que uma variável o referencie?
  - ▶ *c1 = new Circulo()*;
- ▶ Como fazer ambos em um passo?
  - ▶ *Circulo c1 = new Circulo()*;

# Criação de Objetos

```
Classe1 obj1;
obj1 = new Classe1();
```

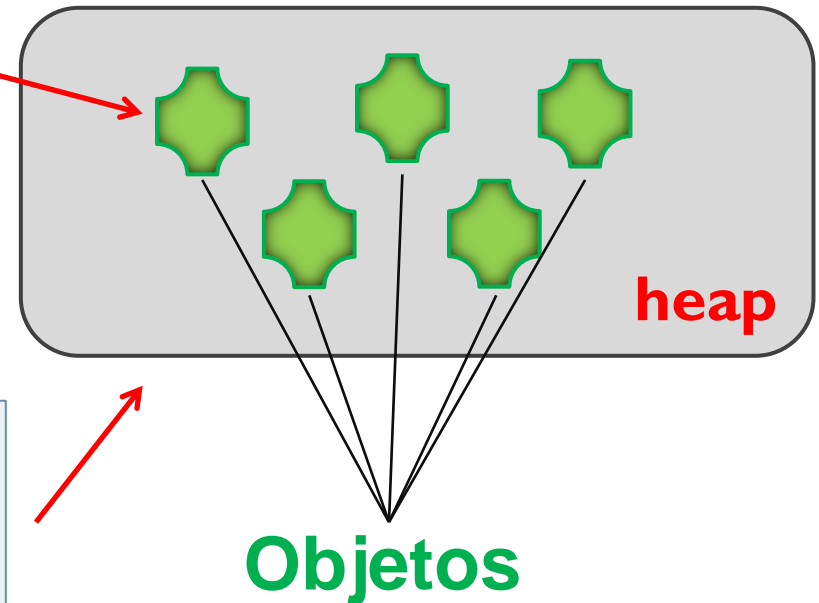
Declara **obj1** como **referência** para objeto da classe **Classe1**

**Cria** objeto e faz **obj1** referenciar objeto recém-criado

**obj1**

A variável **obj1** armazena uma **referência** para o objeto em si. Seu conteúdo é o **endereço** do objeto.

O **heap** é a organização de memória mais flexível que permite o uso de **qualquer área lógica disponível**.



# Referências

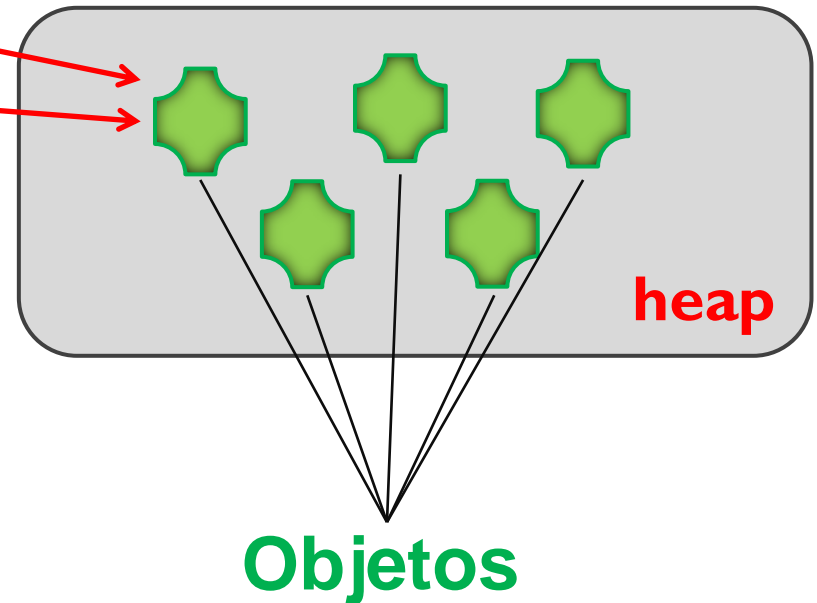
```
Classe1 obj1, obj2;  
obj1 = new Classe1();  
obj2 = obj1;
```

Faz **obj2** referenciar o mesmo objeto que **obj1** referencia

**obj1**

**obj2**

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



# Exercício

---

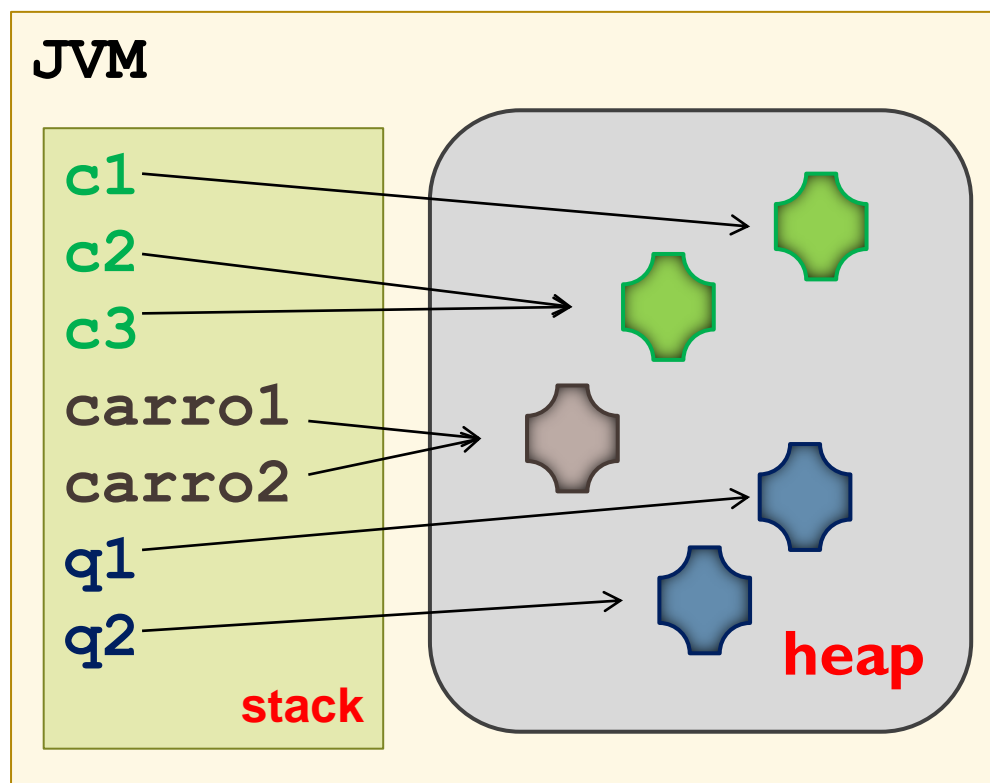
- Desenhe a heap para o código a seguir:

```
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;
```

# Exercício

- Desenhe a heap para o código a seguir:

```
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;
```



# Igualdade

---

## ▶ Entre variáveis

- ▶ Compara o valor das variáveis
- ▶ O valor de uma variável para um objeto é o **endereço** do objeto
  - ▶ O operador **==** compara se as duas variáveis **referenciam o mesmo objeto**
  - ▶ `obj1 == obj2`

## ▶ Entre objetos

- ▶ Método **equals()** verifica se dois objetos possuem o **mesmo estado** (são iguais)



# Igualdade

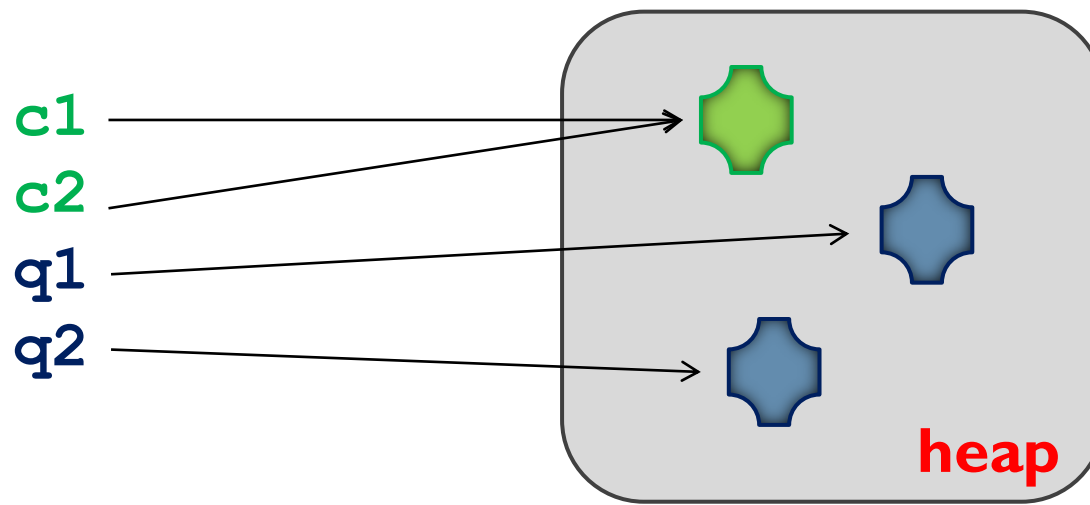
## ► Exemplo:

► **c1 == c2**

► **true** (referenciam o mesmo objeto)

► **q1 == q2**

► **false** (não referenciam o mesmo objeto)



# Métodos

---

- ▶ Usamos o operador “.” (ponto)
- ▶ Sintaxe: **objeto.metodo()**;
  - ▶ Executa método em objeto
- ▶ O objeto deve existir
  - ▶ A variável deve referenciar um objeto válido
  - ▶ Se referenciar **null** ocorre **erro**
- ▶ Exemplos:  
`obj1.nomeMetodo();`  
`obj1.nomeMetodo(arg1, arg2);`  
`(new NomeClasse()).nomeMetodo();`

# Destruindo Objetos

---

- ▶ Garbage Collector (gc) ou coletor de lixo
  - ▶ A Java Virtual Machine (JVM) realiza a **coleta de lixo** automaticamente para reivindicar a memória ocupada pelos objetos que não estão mais em uso.
  - ▶ Quando **não houver mais referências a um objeto**, o objeto é elegível para a coleta de lixo. A memória desse objeto pode ser reivindicada quando a JVM executa seu coletor de lixo.
    - ▶ Os objetos **não referenciados** são automaticamente **apagados**
  - ▶ O “quando isso é feito” depende do algoritmo de coleta de lixo usado
  - ▶ `System.gc()` – Força a coleta de lixo



# Construtores

# Construtor

---

- ▶ Determina quais ações devem ser executadas quando da **criação** do objeto.
- ▶ Em Java, o construtor é definido como um método cujo nome deve ser **o mesmo nome da classe** e **sem indicação do tipo de retorno** (nem mesmo void).
- ▶ Somente é invocado no momento da criação do objeto através do operador **new**.
- ▶ O retorno do operador new é uma **referência** para o objeto recém-criado.
- ▶ Assinatura: *public nome\_da\_classe (parâmetros)*.

# Construtores

---

- ▶ O construtor pode receber **parâmetros** como qualquer método.
- ▶ Toda classe tem **pelo menos um construtor** sempre definido.
  - ▶ Se nenhum construtor for explicitamente definido pelo programador da classe, um construtor padrão, que não recebe argumentos, é incluído para a classe pelo compilador Java.
  - ▶ No entanto, se o programador da classe criar pelo menos um método construtor, o construtor padrão não será criado automaticamente.

# Construtores

---

- ▶ São declarados conforme especificado abaixo:

```
public class Carro{  
    /* CONSTRUTOR DA CLASSE Carro */  
    public Carro(){  
        //Faça o que desejar na construção do objeto  
    }  
}
```

- ▶ A palavra reservada “new” chama o construtor:

```
public class Aplicacao {  
    public static void main(String[] args) {  
        //Chamamos o construtor sem nenhum parâmetro  
        Carro fiat = new Carro();  
    }  
}
```

# Construtores

---

```
public class Carro{
```

```
    private String cor;  
    private double preco;  
    private String modelo;
```

```
    /* CONSTRUTOR PADRÃO */
```

```
    public Carro(){  
  
    }
```

```
    /* CONSTRUTOR COM 2 PARÂMETROS */
```

```
    public Carro(String modelo, double preco){  
        //Se for escolhido o construtor sem a COR do veículo  
        // definimos a cor padrão como sendo PRETA  
        this.cor = "PRETA";  
        this.modelo = modelo;  
        this.preco = preco;  
    }
```

```
    /* CONSTRUTOR COM 3 PARÂMETROS */
```

```
    public Carro(String cor, String modelo, double preco){  
        this.cor = cor;  
        this.modelo = modelo;  
        this.preco = preco;  
    }
```

```
}
```

► Exemplo:  
Classe Carro



# Construtores

---

- ▶ Exemplo: Classe Aplicação
  - ▶ Criando Carros com diferentes construtores.

```
public class Aplicacao {  
    public static void main(String[] args) {  
        //Construtor sem parâmetros  
        Carro prototipoDeCarro = new Carro();  
  
        //Construtor com 2 parâmetros  
        Carro civicPreto = new Carro("New Civic", 40000);  
  
        //Construtor com 3 parâmetros  
        Carro golfAmarelo = new Carro("Amarelo", "Golf", 38000);  
    }  
}
```




Vamos à prática...

# Exercício Elevador – alteração

---

- ▶ Na classe Elevador, retire o método Inicializa e insira um construtor que o substitua apropriadamente.
- ▶ Altere a classe principal, caso seja necessário.



## Métodos Acessores (*get*) e Modificadores (*set*)

# Métodos *get* e *set*

---

- ▶ Os métodos *get* e *set* são técnicas padronizadas para acessar os atributos encapsulados de uma classe;
- ▶ Determina a leitura e alteração do valor de um atributo;
- ▶ Na criação dos métodos para acesso ou modificação dos atributos privados, deve-se colocar *get* ou *set* antes do nome do atributo;

# Métodos *get* – Acessores

---

- ▶ Utilizados para **acessar** ou **ler** os valores dos atributos da classe de uma forma protegida;
- ▶ Esse método **sempre retorna um valor**, ou seja, o tipo de retorno é o mesmo tipo do atributo;
- ▶ Dentro do método existe **apenas o retorno** do atributo.
- ▶ Exemplos:

```
public String getNome () {  
    return nome;  
}  
  
public int getIdade() {  
    return idade;  
}
```

# Métodos *set* – Modificadores

---

- ▶ Utilizados para **alterar** os valores dos atributos da classe de uma forma protegida;
- ▶ Esse método não terá retorno (void), mas sempre terá parâmetro (do mesmo tipo do atributo);
- ▶ Dentro do método existe apenas a atribuição do parâmetro recebido ao atributo que será alterado.

▶ Exemplos:

```
public void setNome(String nome) {  
    this.nome = nome;  
}  
public void setIdade(int idade) {  
    this.idade = idade;  
}
```

- ▶ Observação: A utilização da cláusula **this** faz referência ao **próprio objeto** instanciado, seus atributos e métodos.

# Exemplo de Classe Contato

---

```
public class Contato {  
    private String nome;  
    private int idade;  
    public Contato(String nome, int idade) {  
        this.nome = nome;  
        this.idade = idade;  
    }  
    public String getNome(){  
        return nome;  
    }  
    public int getIdade(){  
        return idade;  
    }  
    public void setNome(String nome){  
        this.nome = nome;  
    }  
    public void setIdade(int idade){  
        this. = idade;  
    }  
}
```



# Exemplo de Classe Principal

---

```
public class Principal{

    public static void main(String[] args){

        Contato c1 = new Contato("Maria", 18);
        Contato c2 = new Contato("João", 16);

        System.out.println("Dados do primeiro contato: ");
        System.out.println("Nome: "+c1.getNome())
        System.out.println("Idade: "+c1.getIdade())

        System.out.println("Dados do segundo contato: ");
        System.out.println("Nome: "+c2.getNome())
        System.out.println("Idade: "+c2.getIdade())

    }
}
```

# Métodos *get* e *set*

---

- ▶ São métodos que acabaram tornando-se “padrões” de acesso a atributos *private*.
  - ▶ Por exemplo, se tenho um atributo chamado “nome” do tipo *String* em modo *private*, criam-se 2 métodos: um para alterar o valor deste atributo (*set*) e outro para se obter o valor deste atributo (*get*).
- ▶ As IDE’s mais utilizadas já oferecem a **geração automática** de *getters* e *setters*.
- ▶ Portanto, se uma classe puder ser acessada de fora sempre deve-se fornecer **acesso** aos valores dos atributos dos objetos instanciados a partir dela.

# Vamos à prática – Lista 1



Dúvidas?