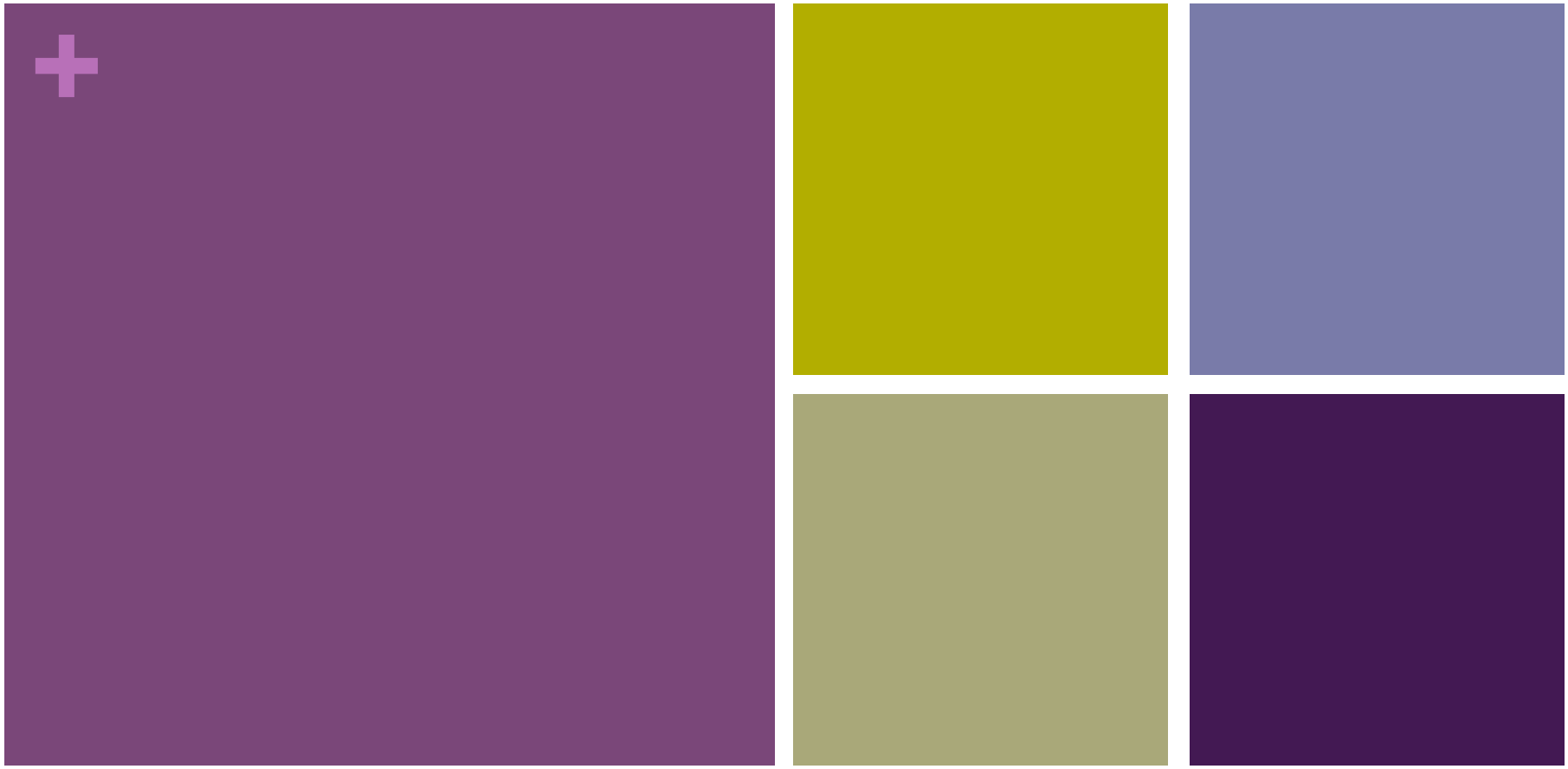


POO – Programação Orientada a Objetos

Classes e Objetos 2

Roteiro

- Criando objetos
  - Operador **new**
  - A **heap** e variáveis que referenciam objetos
- Manipulando objetos
  - Chamando métodos
- Destruição de objetos
  - O garbage collector (gc)
- Algumas classes
  - Character, StringBuilder, String, Number

Criação de objetos

- Declarar variável
  - Associa variável a tipo (classe)
  - Sintaxe

```
NomeClasse nomeVariável;
```

  - Exemplo

```
Circulo c1;
```

- Criar objeto (instanciar) e fazer variável referenciar o objeto

```
c1 = new Circulo();
```

- Ambos em um passo

```
Circulo c1 = new Circulo();
```

Criação de objetos

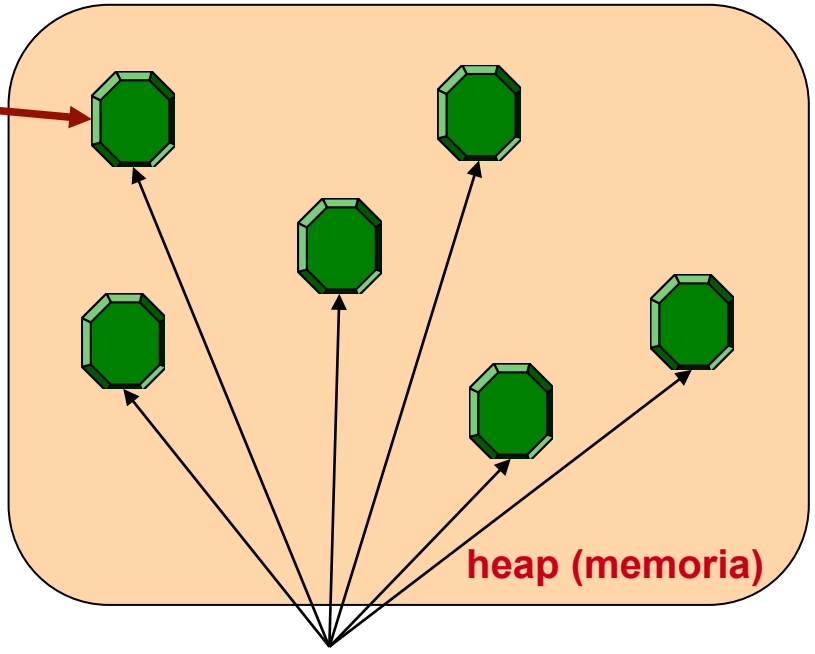

```
Circulo c1;

c1 = new Circulo();
```

Declara c*1* como referência para objeto da classe Circulo

Cria objeto e faz c*1* referenciar objeto recém-criado

c1



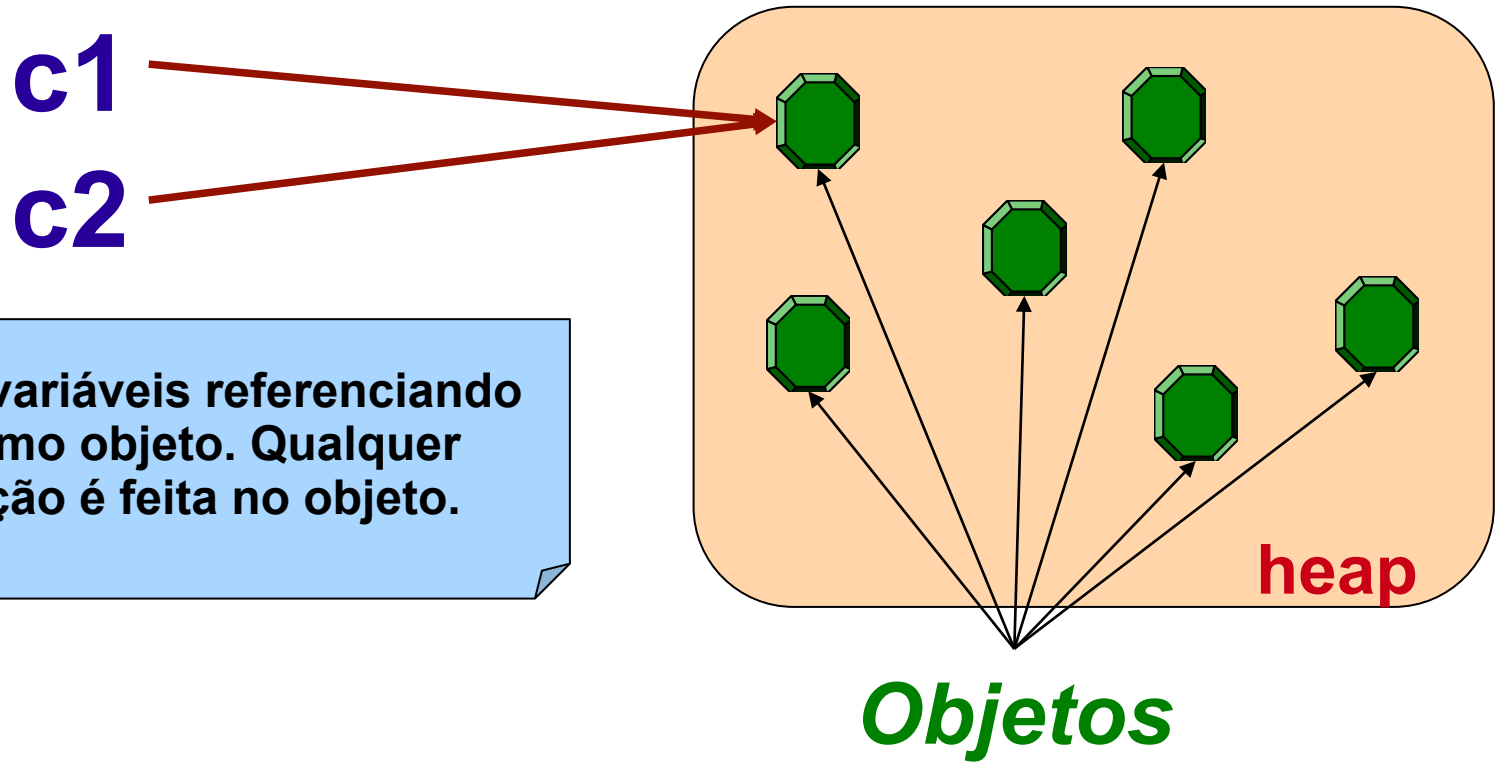
A variável c1 armazena uma referência para o objeto em sí. Seu conteúdo é o "endereço" do objeto.

Objetos

Referência


```
Circulo c1, c2;
c1 = new Circulo();
c2 = c1;
```

faz c2 referenciar o mesmo objeto que c1 referencia



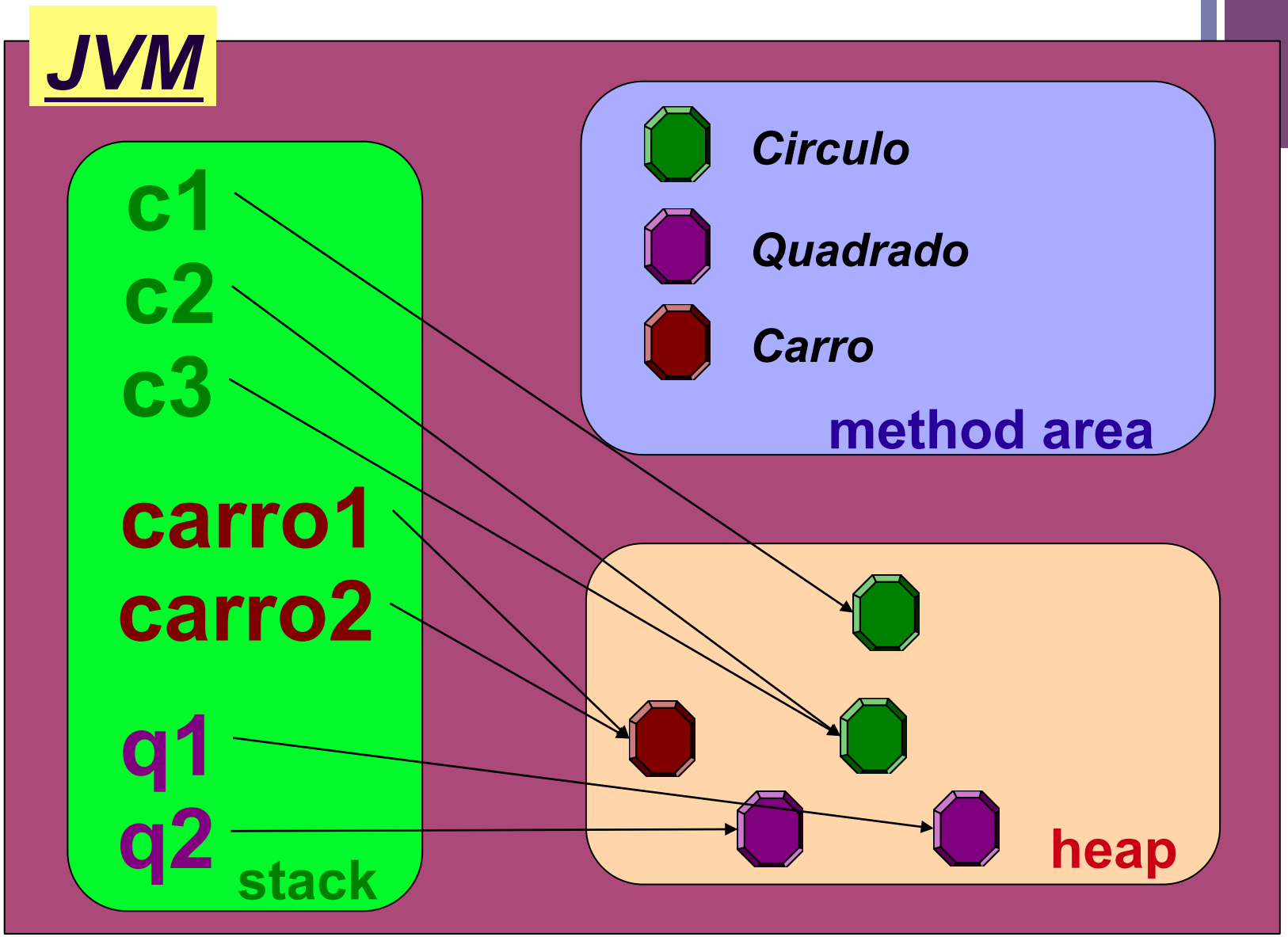
Duas variáveis referenciando o mesmo objeto. Qualquer alteração é feita no objeto.

+ Exercício

- Represente a **heap** para o código a seguir

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

Exercício - resposta



Igualdade

- Entre variáveis
  - Compara o valor das variáveis
  - O valor de uma variável (referência) para um objeto é o **endereço** do objeto
    - O operador == compara se as duas variaveis referenciam o mesmo objeto

```
obj1 == obj2
```

- Entre objetos
  - Metodo equals verifica se dois objetos são iguais

Igualdade

- `c1 == c2`

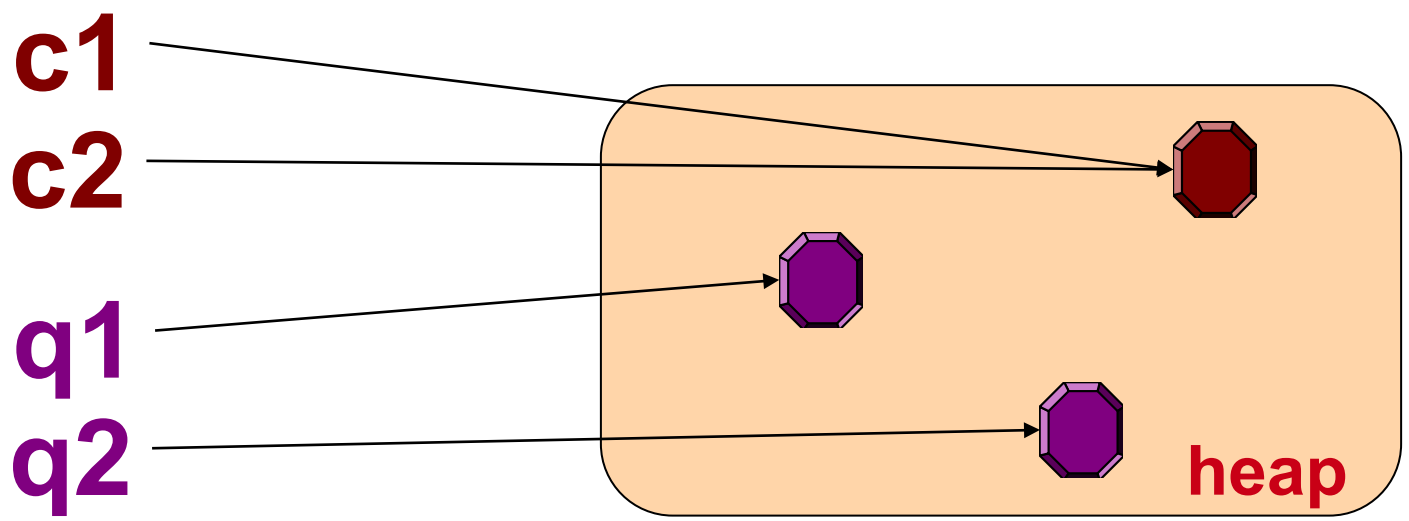
true

  - Referenciam o mesmo objeto

`q1 == q2`

false

Mesmo que os objetos sejam iguais



Métodos

- Usamos o operador "**.**" (ponto)
  - Sintaxe:
  - objeto.método();
  - Executa **método** em **objeto**
- Objeto deve existir
  - A variável deve referenciar objeto válido
  - Se referenciar **null** ocorre erro
- Exemplos:

```
obj1.nomeMetodo();
obj1.nomeMetodo(arg1, arg2);
(new NomeClasse()).nomeMetodo();
```

+ Destruindo objetos

- Garbage Collector (gc)
  - Objetos não referenciados são automaticamente apagados
    - Quando isso é feito depende do algoritmo de coleta de lixo usado
  - **System.gc()**
    - Força a coleta de lixo

+ JAVA

- Algumas classes JAVA
  - Caractere
    - Character
  - Strings
    - StringBuilder, String
  - Números
    - Byte, Short, Integer, Long, Float, Double, BigInteger, BigDecimal

Character

- Classe que representa um único caracter
  - método charValue() retorna o caractere desse objeto

```
Character c1 = null, c2 = null;
System.out.println("O valor de c1 é "+c1);
c1 = new Character('a');
System.out.println("O valor de c1 é "+c1.charValue());
c2 = c1;
if (c1 == c2)
   System.out.println("c1 e c2 são o mesmo (1)");
if (c1.equals(c2))
   System.out.println("c1 e c2 são iguais (1)");
c2 = new Character('a');
if (c1 == c2)
   System.out.println("c1 e c2 são o mesmo (2)");
if (c1.equals(c2))
   System.out.println("c1 e c2 são iguais (2)");
```

+ Strings

- Três classes para Strings
  - String
    - Objetos são **imutáveis**
    - Possuem tratamento especial
  - StringBuilder
    - Objetos **mutáveis**
    - Métodos de manipulação
  - StringBuffer
    - Mesmo que StringBuilder
    - Usada na programação concorrente

Classe String

- Operador **new**
  - String s = new String("CEFET");
  - String nome = new String ("João");
- São indexados a partir do zero
  - Implementadas como array de char
  - "CEFET" usa os índices 0, 1, 2, 3, 4
  - Não terminam com **'\0'**
- Pode ser concatenada com o operador **+**
  - Tratamento especial para String
  - String s3 = s1 + s2;
  - s3 = s3 + "bla";

+ String

- Alguns métodos
  - int length()
    - retorna o tamanho da String
  - char charAt(int i)
    - retorna o caractere no índice i
  - int indexOf(char c)
    - Retorna o indice do caractere c
  - char[] toCharArray();
    - Retorna a String em forma de array
  - String toLowerCase();
    - Retorna nova String toda minúscula
  - String toUpperCase();
    - Retorna nova String toda maiuscula
  - String trim();
    - Retorna nova String sem os espaços no inicio e fim
  - int compareTo(String s);
    - Compara duas Strings

Literais String

- Texto entre aspas são objetos da classe String
  - String s = "Isto é uma String"
  - mais eficiente que usar **new**
- Usada normalmente onde pede-se String
  - int tamanho = "Qual o tamanho?".length();
  - String s1 = "um nome qualquer".toUpperCase();
  - String s = "CEFET" + "-RN";
    - São criada **3** Strings: "CEFET", "-RN" e "CEFET-RN"

Exemplo


```
String cefet = "Centro Federal de Educação Tecnológica";
System.out.println(cefet);
int tamanho = cefet.length();
System.out.println("O tamanho é "+tamanho);
char c = cefet.charAt(7);
System.out.println("O caractere no índice 7 é "+c);
int x = cefet.indexOf('ç');
System.out.println("O ç está no índice "+x);
String cefetMin = cefet.toLowerCase();
System.out.println(cefetMin);
String cefetMai = cefet.toUpperCase();
System.out.println(cefetMai);
cefet = cefet + " do RN";
System.out.println(cefet);
```

NÃO muda a String. Cria uma nova e referencia a nova

StringBuilder

- Classe de texto mutável
  - Necessário criar objeto com **new**
  - Mais eficiente para tratar textos mutáveis
    - Evita processamento com criação de objetos
- Classe StringBuffer
  - Mesmo que StringBuilder, porém usado quando há acesso concorrente

StringBuilder

- Alguns métodos
  - int length();
    - retorna o tamanho do StringBuilder
  - char charAt(int i);
    - retorna o caractere no índice i
  - StringBuilder append(x);
    - Adiciona **x** no final da String
      - **x** pode ser: boolean, char, char[], CharSequence, double, float, int, long, Object, String, StringBuffer.
  - int capacity();
    - Retorna o a capacidade do StringBuilder (espaço reservado) - diferente de length()
    - StringsBuilder são implementadas com arrays de char. capacity retorna o tamanho do array, enquanto length retorna o espaço usado no array.

StringBuilder

- Mais métodos
  - delete (int i , int f)
    - Apaga caracteres entre os índices e e f
  - StringBuilder insert (int i, x)
    - Insere x no indice i (similar a append)
  - StringBuilder replace (inicio, fim, String);
    - Substitui o texto entre inicio e fim por String
  - substring(inicio, fim)
    - Retorna String contida entre os índices inicio de fim
  - String toString();
    - Retorna a String correspondente a StringBuilder

Exemplo


```
StringBuilder cefet;
cefet=new StringBuilder("Centro Federal");
System.out.println(cefet.toString());
int tamanho = cefet.length();
System.out.println("O tamanho é " + tamanho);
char c = cefet.charAt(9);
System.out.println("O caractere no índice 9 é " + c);
cefet.append(" de educação ");
System.out.println(cefet);
System.out.println("O tamanho é " + cefet.length());
System.out.println("Capacidade " + cefet.capacity());
cefet.append(" tecnologica do RN");
System.out.println("Capacidade " + cefet.capacity());
```

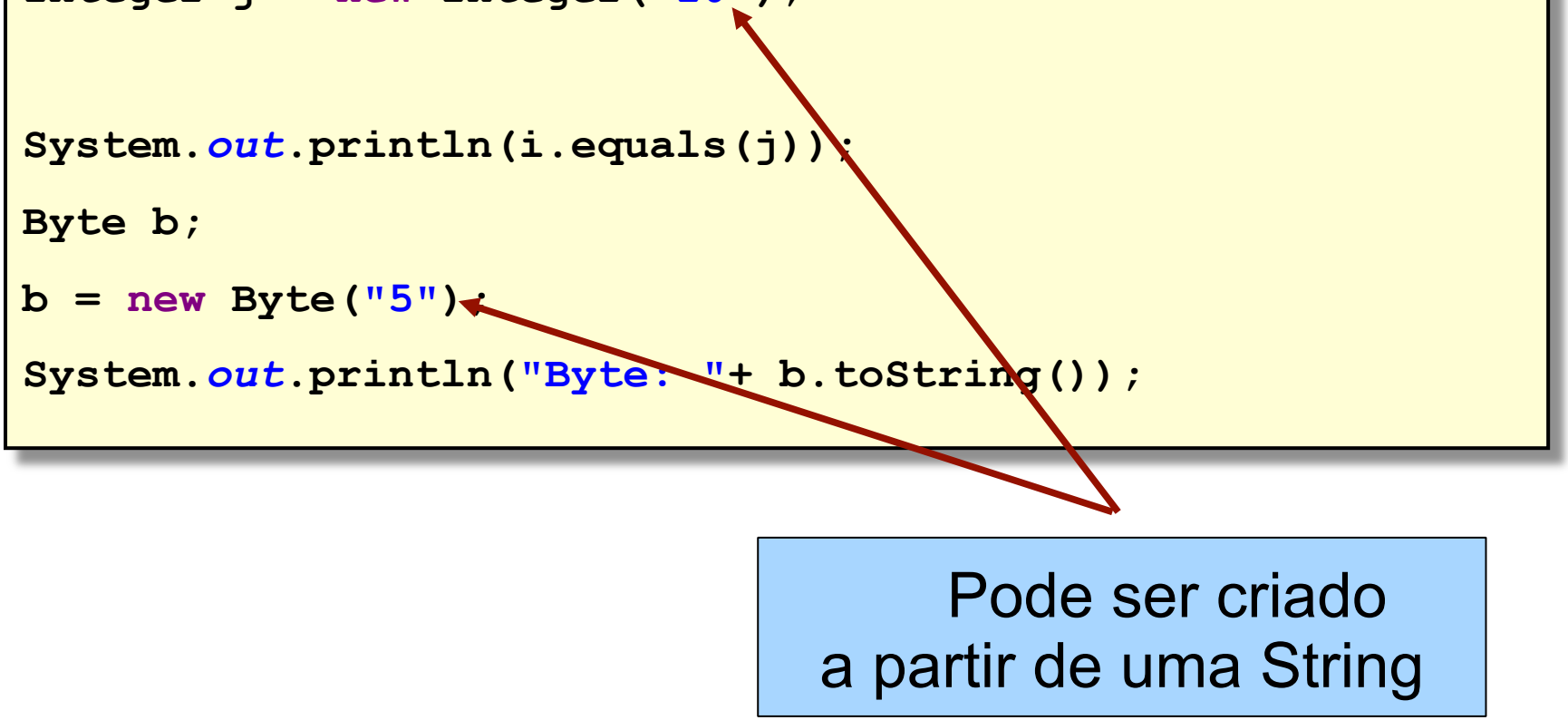
Números

- Classes que substituem os tipos primitivos
  - Byte, Short, Integer, Long, Float, Double, Boolean
  - Metodos comuns a todos
    - byteValue(), shortValue(), intValue(), longValue(),
      - retorna o número no tipo especificado
  - São imutáveis

Exemplo


```
Integer i;
i = new Integer(10);
Integer j = new Integer("10");


System.out.println(i.equals(j));
Byte b;
b = new Byte("5");
System.out.println("Byte: "+ b.toString());
```



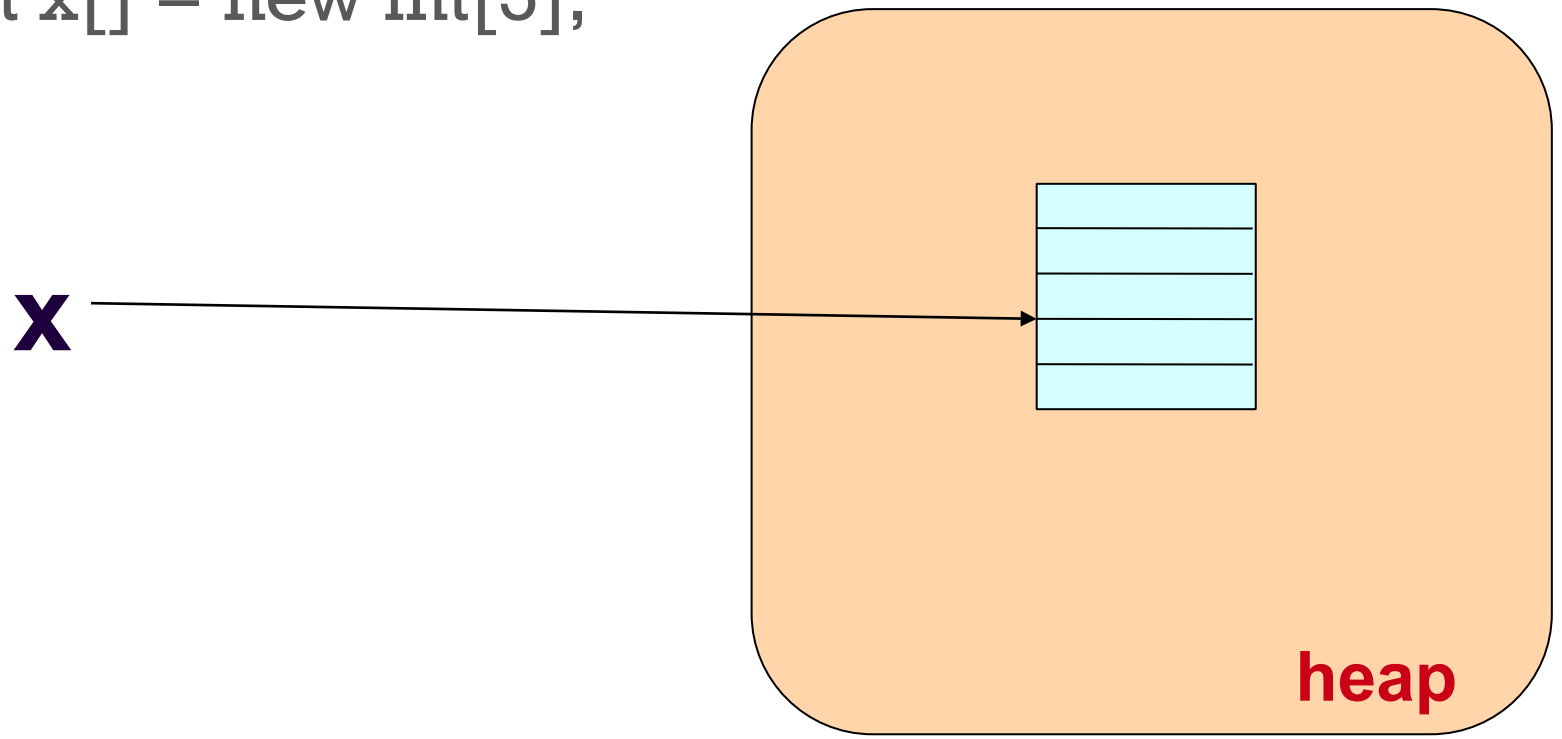
Pode ser criado
a partir de uma String

+ Arrays

- Arrays são objetos
  - Precisa ser instanciado
- Declaração e instanciação
  - tipo id[] = new tipo[10];
- Exemplo
  - int x[] = new int[10];
- Podem ser de tipos primitivos e de objetos
  - Arrays de objetos **não** instanciam os objetos
    - Cada índice armazena uma **referência**

Arrays

- Arrays de tipos primitivos
  - Instanciado o array os espaços de cada índice são usados para armazenar o valor
  - int x[] = new int[5];

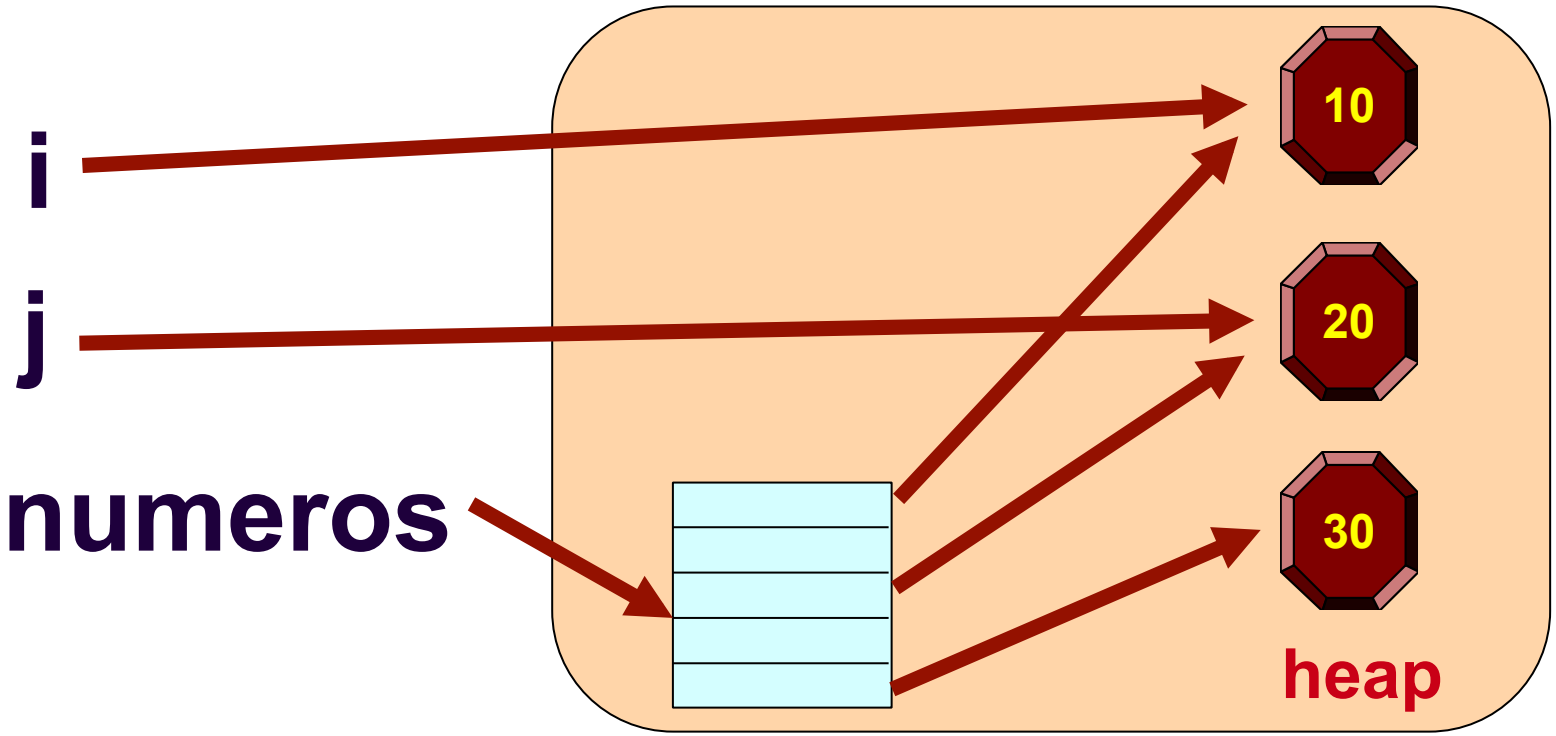


+ Arrays

- Arrays de objetos
  - Cada índice armazena uma **referência** a um objeto
  - É necessário instanciar cada objeto individualmente

Arrays


```
Integer i = new Integer(10);
Integer j = new Integer(20);
Integer numeros[] = new Integer[5];
numeros[0] = i;
numeros[2] = j;
numeros[4] = new Integer(30);
```



+ Dúvidas