OBJECTOS

PROGRAMAÇÃO ORIENTADA POR OBJECTOS

- O paradigma de programação **dominante** hoje em dia é o da **programação orientada por objetos**

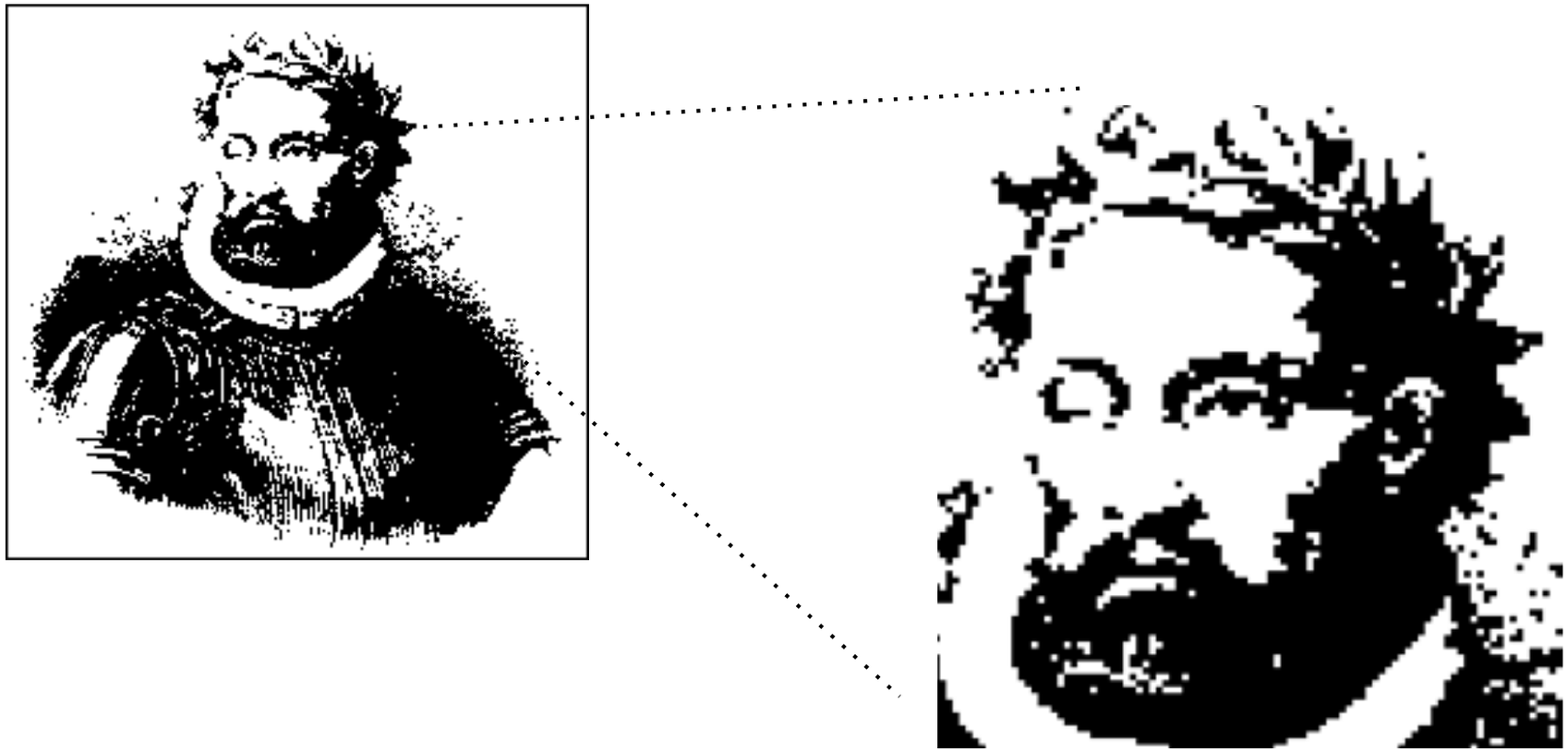
- Neste paradigma, o código manipula **objetos com identidade, estado e comportamento próprios**

  - Um objeto representa tipicamente uma **entidade** do mundo real (física ou não)

- O universo dos **objetos de determinado tipo** é designado por **classe de objetos** (descrição dos objectos de um dado tipo)

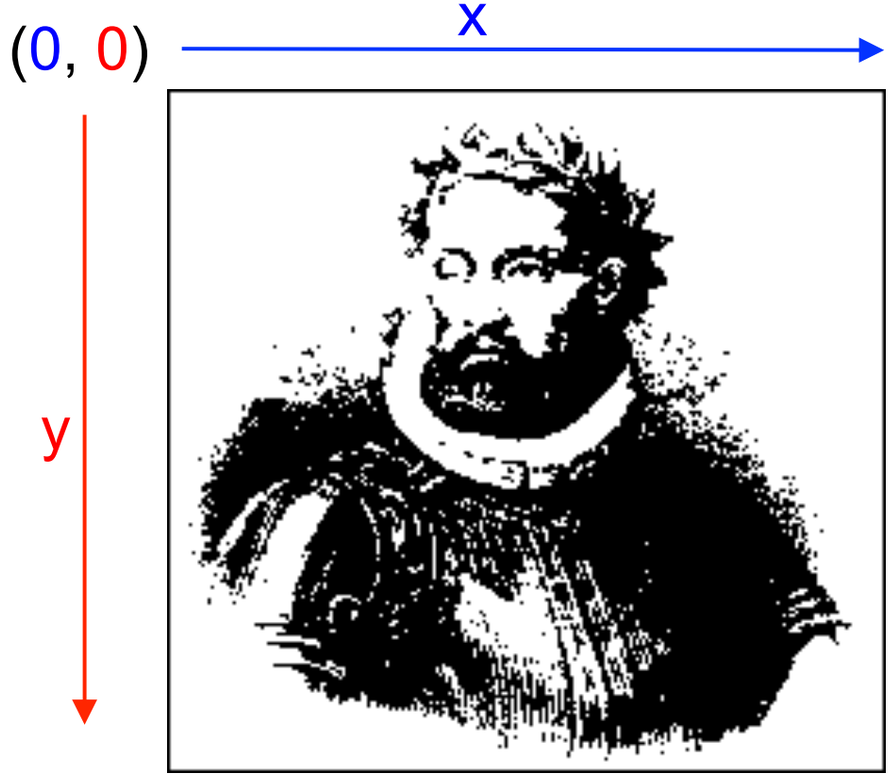
OBJECTOS: IMAGENS BINÁRIAS

- Uma imagem binária é uma imagem onde cada pixel só tem dois valores possíveis (p.e. branco ou preto)



IMAGENS: COORDENADAS DE PÍXEIS

- A coordenada (0, 0) corresponde ao píxel do canto superior esquerdo. À medida que o valor de ***x*** aumenta, deslocamo-nos para a direita na imagem e à medida que o valor de ***y*** aumenta deslocamo-nos para baixo na imagem



CLASSE DE OBJETOS: IMAGENS BINÁRIAS

- A descrição dos objetos do tipo imagem binária está na classe de objetos `BinaryImage`, cujo código está no ficheiro BinaryImage.java

```
class BinaryImage {

    …

}
```

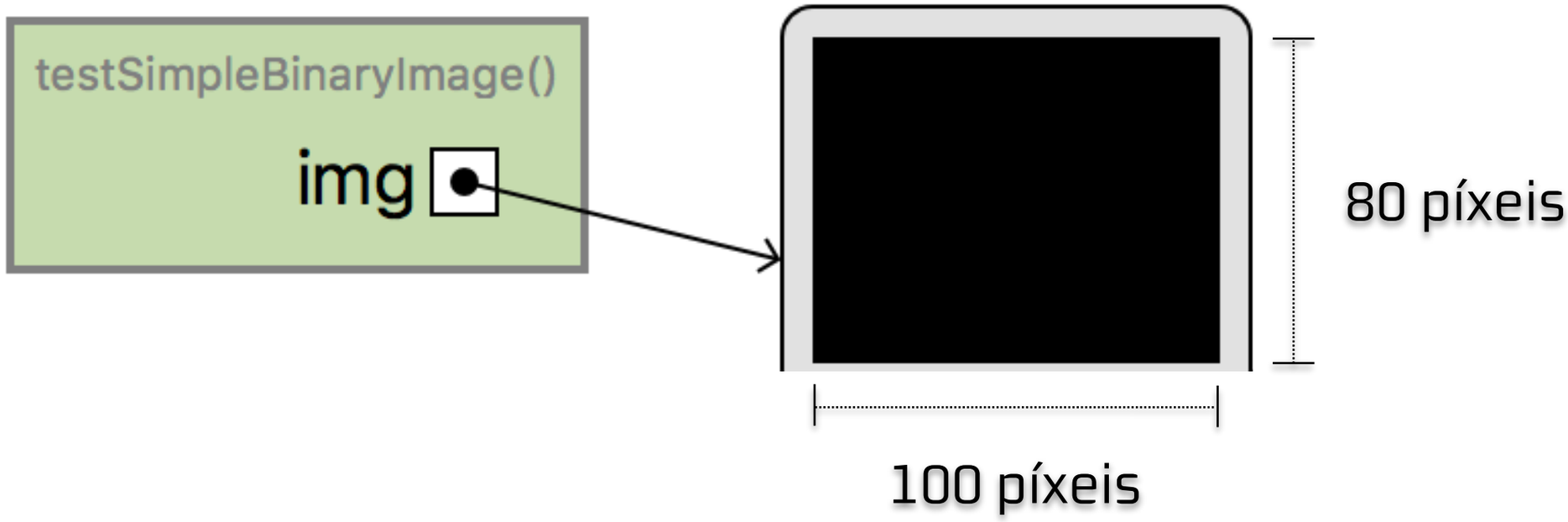
CRIAÇÃO DE OBJECTOS

- Na sua forma essencial, a criação de objetos é feita mediante um **método construtor** ou simplesmente **construtor**.

nome da classe (tipo de objeto)

Este construtor tem 2 inteiros como parâmetros

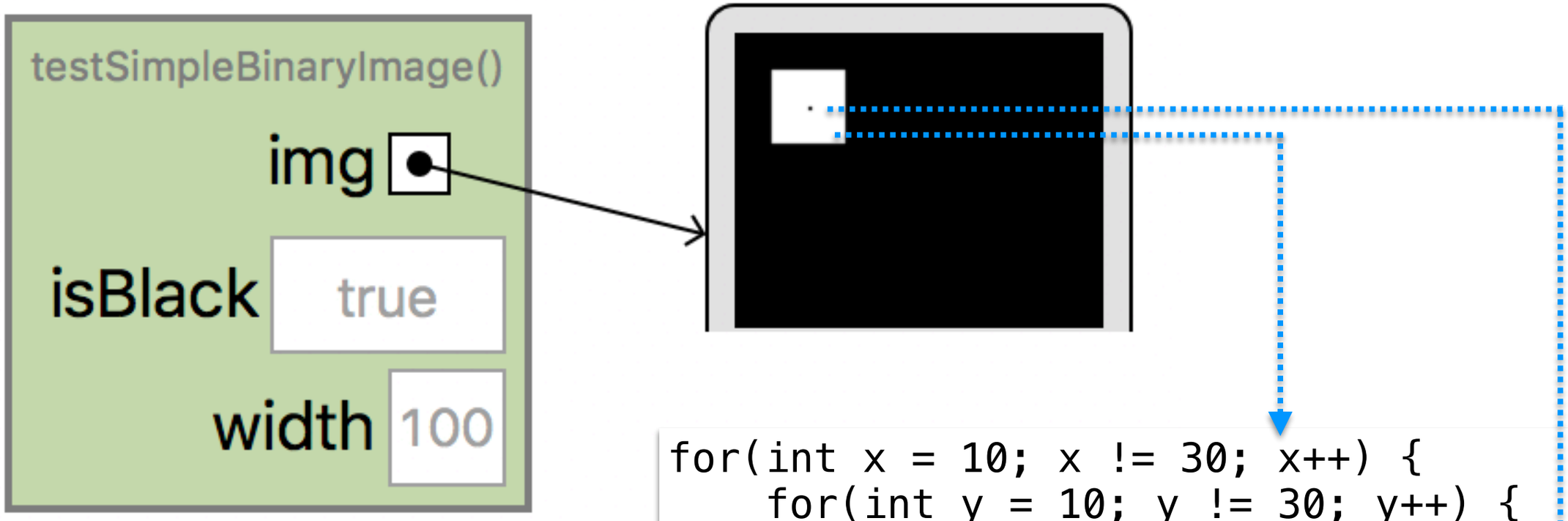
```
BinaryImage img = new BinaryImage(100, 80);
```



MANIPULAÇÃO DE OBJECTOS: OPERAÇÕES

- Os objetos são manipulados através da **invocação de operações**, as quais se podem caracterizar em:
  - Funções (não modificam o objeto; inspecionam o estado do objeto)
  - Procedimentos (modificam o estado do objeto)

IMAGENS BINÁRIAS: OPERAÇÕES



Modificador

Modificador

Inspetores

```
for(int x = 10; x != 30; x++) {
    for(int y = 10; y != 30; y++) {
        img.setWhite(x, y);
    }
}
```

```
img.setBlack(20, 20);
```

```
boolean isBlack = img.isBlack(20, 20);
int width = img.getWidth();
```

IMAGENS BINÁRIAS: ESTADO

(0, 0)

```
false false false … false false
false true  true  … true  false
false true  true  … true  false
…
false true  true  … true  false
false false false … false false
```

100 x 80

(99, 79)

IMAGENS BINÁRIAS
BinaryImage

Construção (cria uma imagem binária com dimensão *width* x *height*)

- `new BinaryImage(int width, int height)`

Funções:

- `int getWidth()`
  devolve a largura da imagem

- `int getHeight()`
  devolve a altura da imagem

- `boolean isBlack(int x, int y)`
  devolve verdadeiro caso o pixel na coordenada (*x*, *y*) seja preto, falso caso contrário

Procedimentos:

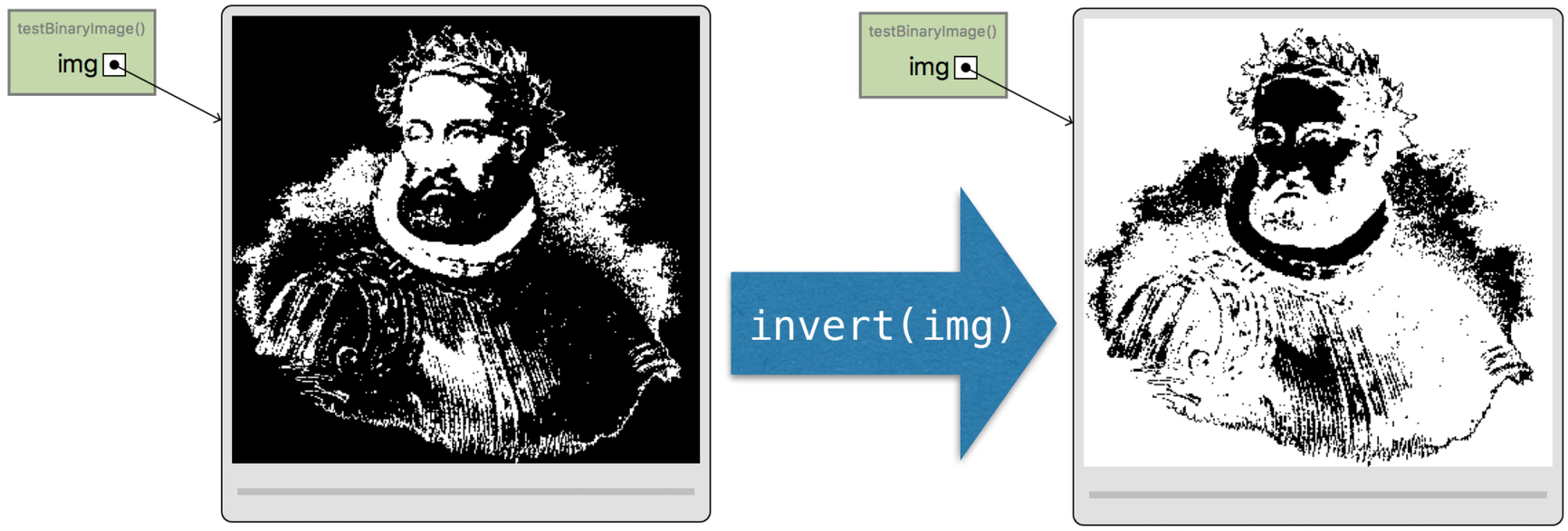
- `void setBlack(int x, int y)`
  altera o pixel na coordenada (*x*, *y*) para preto

- `void setWhite(int x, int y)`
  altera o pixel na coordenada (*x*, *y*) para branco

IMAGENS BINÁRIAS: EXEMPLO

- Inverter a cor dos píxeis (procedimento)

```
static void invert(BinaryImage img) {
    for(int x = 0; x != img.getWidth(); x++) {
        for(int y = 0; y != img.getHeight(); y++) {
            if(img.isBlack(x, y)) {
                img.setWhite(x, y);
            } else {
                img.setBlack(x, y);
            }
        }
    }
}
```



CLASSE DE OBJETOS: CORES

- Uma dos modelos de representação de cores mais utilizado é o RGB (Red, Green, Blue). Desta forma, uma cor é representada por 3 valores inteiros no intervalo [0, 255], representando as componentes de vermelho, verde, e azul

CORES: PROPRIEDADES

- Cada objeto cor é representado em termos de uma tripla de inteiros (R = [0 - 255], G = [0 - 255], B = [0 - 255])

```
Color red = new Color(255, 0, 0);
Color green = new Color(0, 255, 0);
Color blue = new Color(0, 0, 255);

Color black = new Color(0, 0, 0);
Color white = new Color(255, 255, 255);
```

CORES: OPERAÇÕES

CLASSE

`Color`

CONSTRUTOR

`Color(int r, int g, int b)`
cria uma cor com os valores RGB dados
(inteiros no intervalo [0 - 255])

OPERAÇÕES

`int getR()`
devolve o valor de vermelho [0 - 255]

`int getG()`
devolve o valor de verde [0 - 255]

`int getB()`
devolve o valor de azul [0 - 255]

`int getLuminance()`
devolve a luminância da cor [0 - 255]

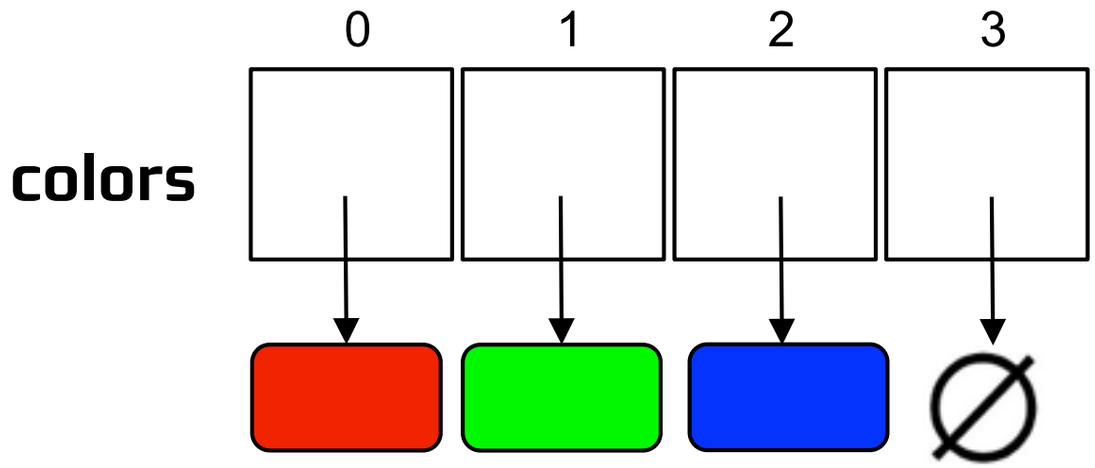
CORES: OBJETOS IMUTÁVEIS

- Os objetos `Color` só têm funções, ou seja, não têm nenhuma operação que permita modificar o seu estado
- Este tipo de objetos designam-se por **objetos imutáveis**
  - Desta forma, não faz sentido falar em procedimentos que atuam sobre objetos imutáveis
- Por outro lado, os exemplos de objetos de imagem que abordámos designam-se por **objetos mutáveis**

VETORES DE OBJETOS

- Tal como nos tipos primitivos, também é possível criar vetores (e matrizes) de objetos.

```
Color[] colors = new Color[4];
colors[0] = new Color(255, 0, 0);
colors[1] = new Color(0, 255, 0);
colors[2] = new Color(0, 0, 255);
```



A RETER

- Objectos
  - Criação
    - Construtor
  - Manipulação (operações)
    - Funções
    - Procedimentos
  - Estado
  - Objetos imutáveis
  - Exemplos
- Vetores de objetos