OBJETOS COMPUESTOS

IMAGENS A CORES

CLASSE DE OBJETO: 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 yellow = new Color(255, 255, 0);
```

```
Color gray = Color.createGraytone(128);
```

CORES: OPERAÇÕES

CLASSE

`aguiaj.iscte.Color`

CONSTRUTOR

`Color(int r, int g, int b)`
cria uma cor com os valores RGB dados

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]

`boolean equals(Object color)`
devolve verdadeiro caso a cor seja igual ao objecto *color*

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

`Color toGraytone()`
devolve a cor convertida para um tom de cinzento

CORES: OBJETOS IMUTÁVEIS

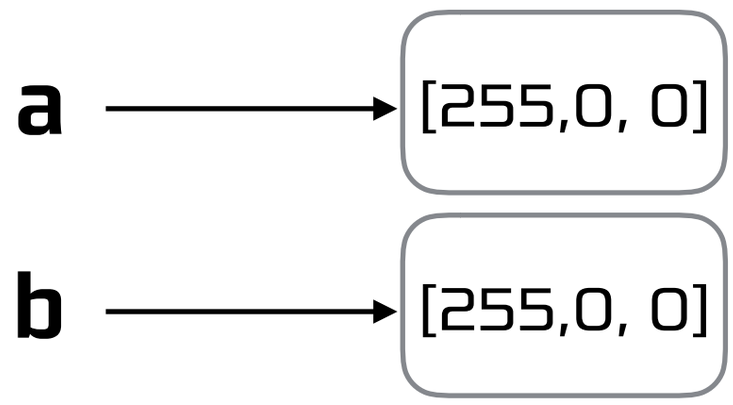
- Os objetos `Color` só têm funções, ou seja, não têm nenhuma operação que permita modificar o seu estado
- Este tipo de objetos designam-se por **objetos imutáveis**
  - Desta forma, não faz sentido falar em procedimentos que atuam sobre objetos imutáveis
- Por outro lado, os exemplos de objetos de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

COMPARAÇÃO DE OBJETOS

- Tal como no caso dos vectores/matrizes, o operador **==** compara referências, não os objectos em si

- A igualdade entre objetos deve ser verificada utilizando a operação **equals()**.

```
Color a = new Color(255, 0, 0);

Color b = new Color(255, 0, 0);


boolean sameObject = a == b;

boolean equalObjects = a.equals(b);
```

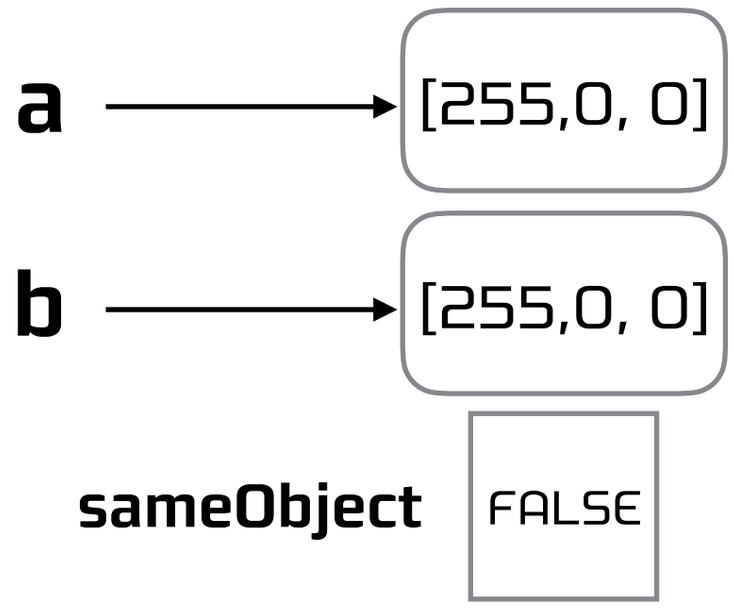


COMPARAÇÃO DE OBJETOS

- Tal como no caso dos vectores/matrizes, o operador **==** compara referências, não os objectos em si

- A igualdade entre objetos deve ser verificada utilizando a operação **equals()**.

```
Color a = new Color(255, 0, 0);

Color b = new Color(255, 0, 0);


boolean sameObject = a == b;

boolean equalObjects = a.equals(b);
```

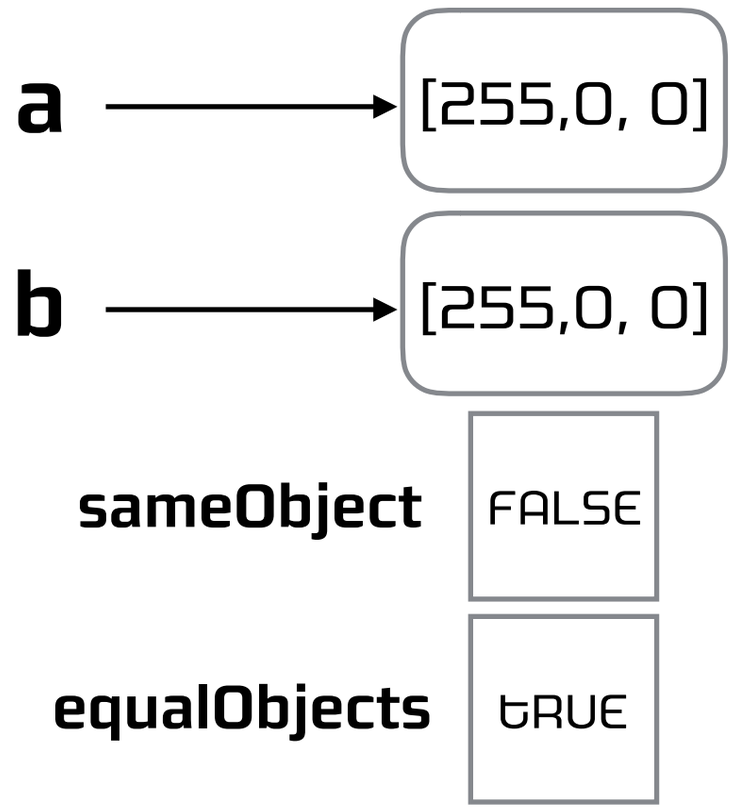


COMPARAÇÃO DE OBJETOS

- Tal como no caso dos vectores/matrizes, o operador **==** compara referências, não os objectos em si

- A igualdade entre objetos deve ser verificada utilizando a operação **equals()**.

```
Color a = new Color(255, 0, 0);

Color b = new Color(255, 0, 0);


boolean sameObject = a == b;

boolean equalObjects = a.equals(b);
```



CONSTANTES

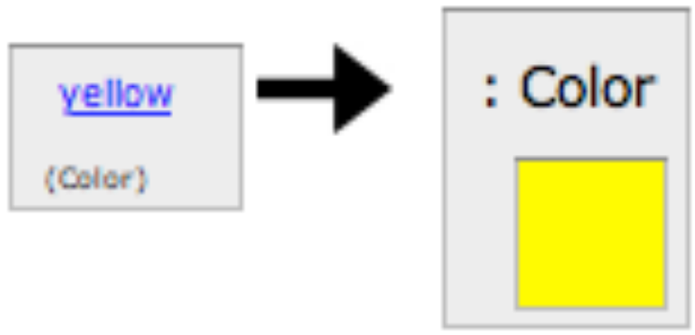
Em situações onde temos um valor constante utilizado em várias partes de um programa, torna-se útil a definição desses valores num só sítio, de modo a facilitar a sua alteração

```
class MyClass {

    static final int MAX = 100;

    static final int[][] ID3 =

        {{1, 0, 0},
         {0, 1, 0},
         {0, 0, 1}};

    static final Color RED = new Color(255, 0, 0);

    …

}
```

CORES: CONSTANTES

A classe `Color` define um conjunto de constantes para as cores básicas. As constantes são acedidas como um campo da classe.

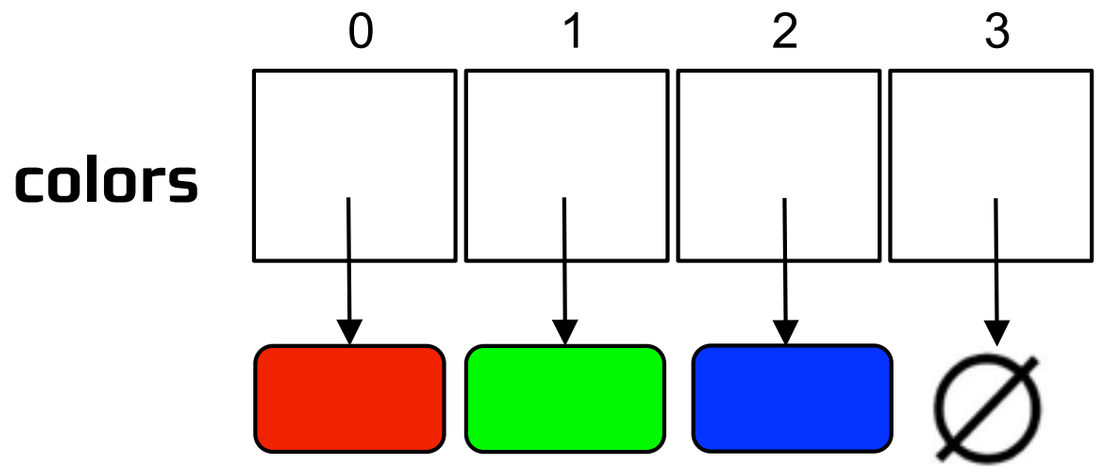
```
Color yellow = Color.YELLOW;
```



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);
```



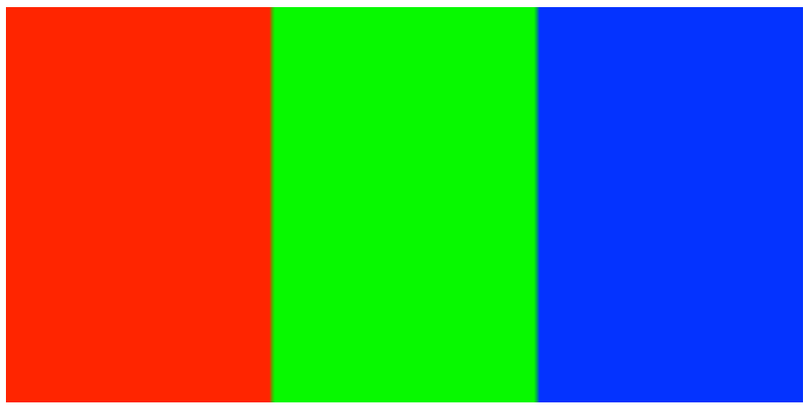
CLASSE DE OBJETOS: IMAGENS A CORES

Numa imagem a cores, cada pixel é uma cor RGB (objeto `Color`).

IMAGENS A CORES: ATRIBUTOS

objecto `Color`

[255, 0, 0]	...	[0, 255, 0]	...	[0, 0, 255]
[255, 0, 0]	...	[0, 255, 0]	...	[0, 0, 255]
...	...	...	...	...
[255, 0, 0]	...	[0, 255, 0]	...	[0, 0, 255]



IMAGENS A CORES: OPERAÇÕES

CLASSE

`aguiaj.iscte.ColorImage`

CONSTRUTOR

`ColorImage(int width, int height)`
cria uma image a cores com a dimensão *width*×*height*

OPERAÇÕES

`int getWidth()`
devolve a largura da imagem

`int getHeight()`
devolve a altura da imagem

`Color getColor(int x, int y)`
devolve a cor do pixel na coordenada (*x*, *y*)

`void setColor(int x, int y, Color color)`
altera o pixel na coordenada (*x*, *y*) para a cor *color*

EXCEÇÕES

- O lançamento de *exceções* pode ser utilizado como um mecanismo para **interromper a execução normal de um método**, caso o objeto tenha sido utilizado de forma incorreta
  - Invocação de uma operação com argumentos inválidos
  - Sequência de invocações inválida
- As exceções elas próprias **são objetos** (com atributos, operações, e construtores)

TIPOS DE EXCEÇÕES

- Muitos tipos de exceções em Java
- Tipos relacionados com a utilização incorreta de objetos
  - `IllegalArgumentException`: adequada quando um **argumento inválido** é utilizado na invocação de uma operação
  - `NullPointerException`: adequada quando é passada uma **referência `null`** não permitida como argumento
  - `IllegalStateException`: adequado quando é invocada uma operação não permitida dado o **estado atual do objeto**

LANÇAMENTO DE EXCEÇÕES:
IllegalArgumentException

```
class Point {

    final int x;
    final int y;

    Point(int x, int y) {
        if(x < 0 || y < 0)
            throw new IllegalArgumentException("Valores não negativos!”);

        this.x = x;
        this.y = y;

    }

    …

}
```

LANÇAMENTO DE EXCEÇÕES:
NullPointerException

```
class ImageUtils {

    static void invert(BinaryImage img) {
        if(img == null)
            throw new NullPointerException(“O argumento não pode ser null!”);
            …
        }

        …
}
```

LANÇAMENTO DE EXCEÇÕES:
IllegalStateException

```
class IntSet {

    …

    boolean isFull() {

        …

    }


    void add(int element) {
        if(isFull())
            throw new IllegalStateException("O conjunto está cheio!”);

        …

    }

}
```

A RETER

- A classe de objetos `Color`
  - Propriedades
  - Operações
  - Objetos imutáveis
- Comparação de objetos
- Constantes estáticas
- Vetores de objetos
- Objetos compostos, a classe `ColorImage`
  - Atributos
  - Operações
- Exceções