OBJETOS COMPOSTOS

IMAGENS A CORES

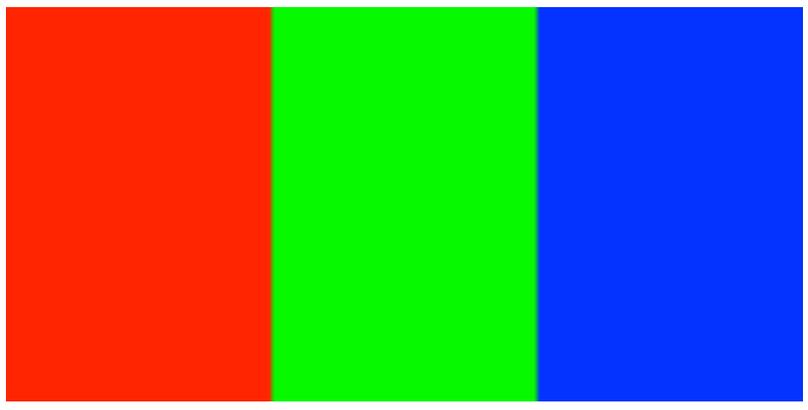
CLASSE DE OBJETOS: IMAGENS A CORES

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

IMAGENS A CORES: ATRIBUTOS

objecto `Color`

$$\begin{bmatrix} [255,0,0] & \cdots & [0,255,0] & \cdots & [0,0,255] \\ [255,0,0] & \cdots & [0,255,0] & \cdots & [0,0,255] \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ [255,0,0] & \cdots & [0,255,0] & \cdots & [0,0,255] \end{bmatrix}$$



IMAGENS A CORES: OPERAÇÕES

CLASSE

`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*)

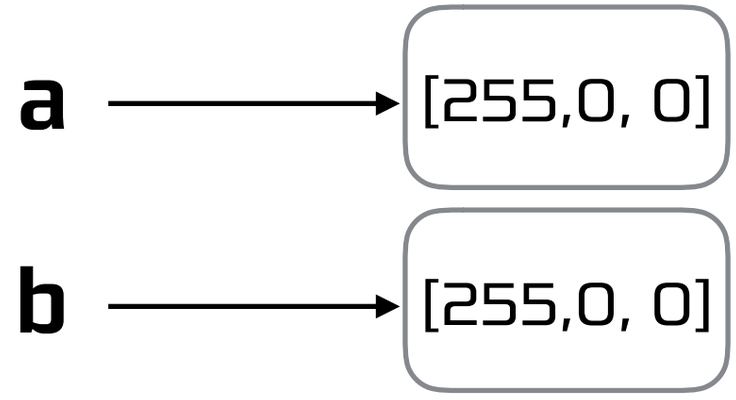
`void setColor(int x, int y, Color color)`
altera o pixel na coordenada (*x*, *y*) para a cor *color*

COMPARAÇÃO DE OBJETOS

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

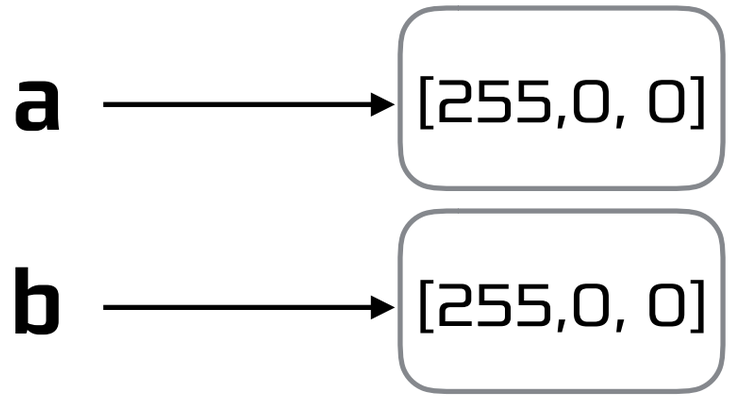
- A igualdade entre objetos deve ser verificada utilizando uma operação definida especificamente para esse efeito

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

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


boolean sameObject = a == b;

boolean equalObjects = ???
```



COMPARAÇÃO DE OBJETOS

```
class Color {
    …

    boolean isEqualTo(Color c) {
        if(c == null) {
            return false;
        }
        return rgb[0] == c.rgb[0]
               && rgb[1] == c.rgb[1]
               && rgb[2] == c.rgb[2];
    }
}
```




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

boolean sameObject = a == b;
boolean equalObjects = a.isEqualTo(b);
```

sameObject FALSE

equalObjects tRUE

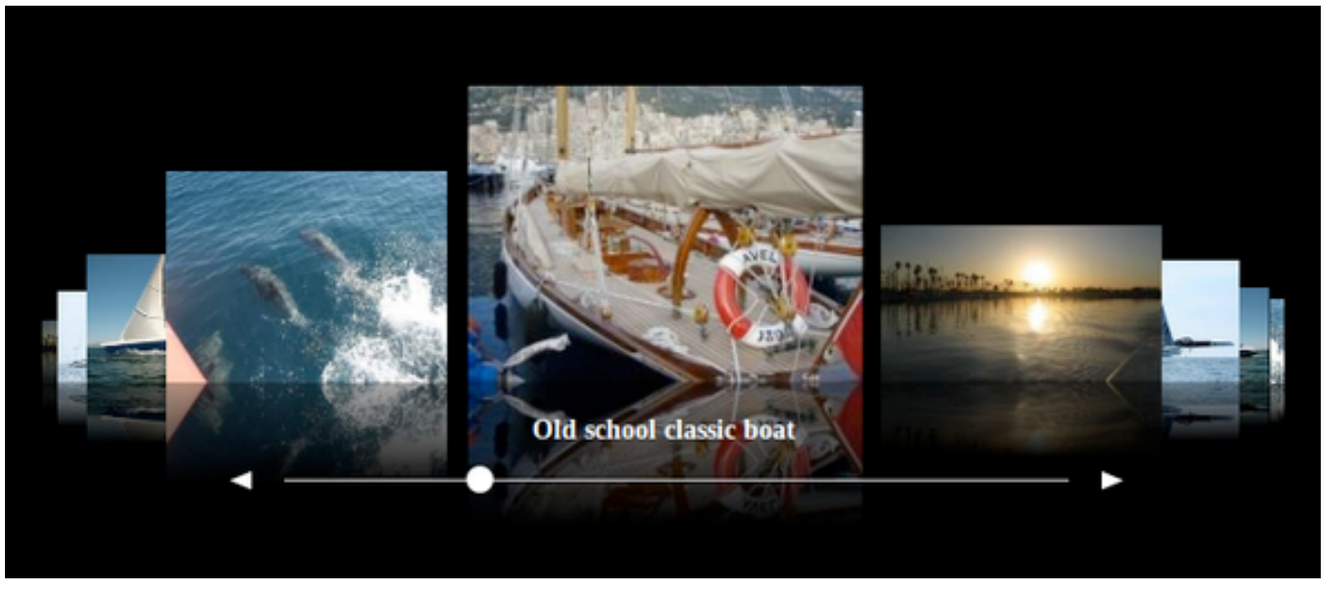
CONSTANTES

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);

    …

}
```

COLEÇÕES DE OBJETOS

- Um dos tipos mais importantes de objetos compostos usados na resolução de diversos problemas é a **coleção de objetos**

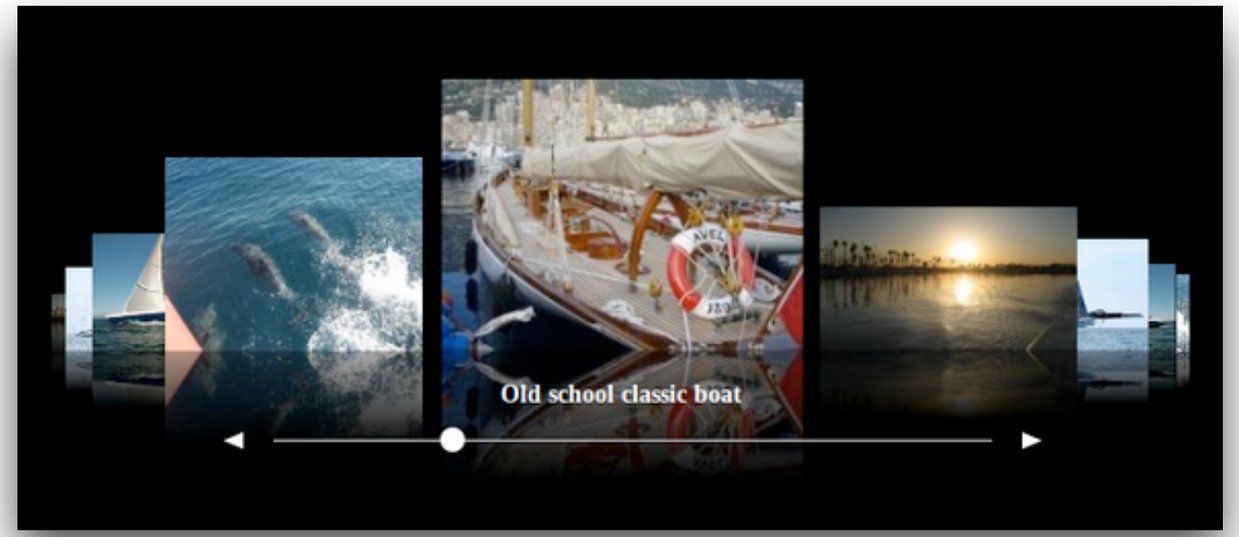


Coleção de fotografias

Coleção de contactos

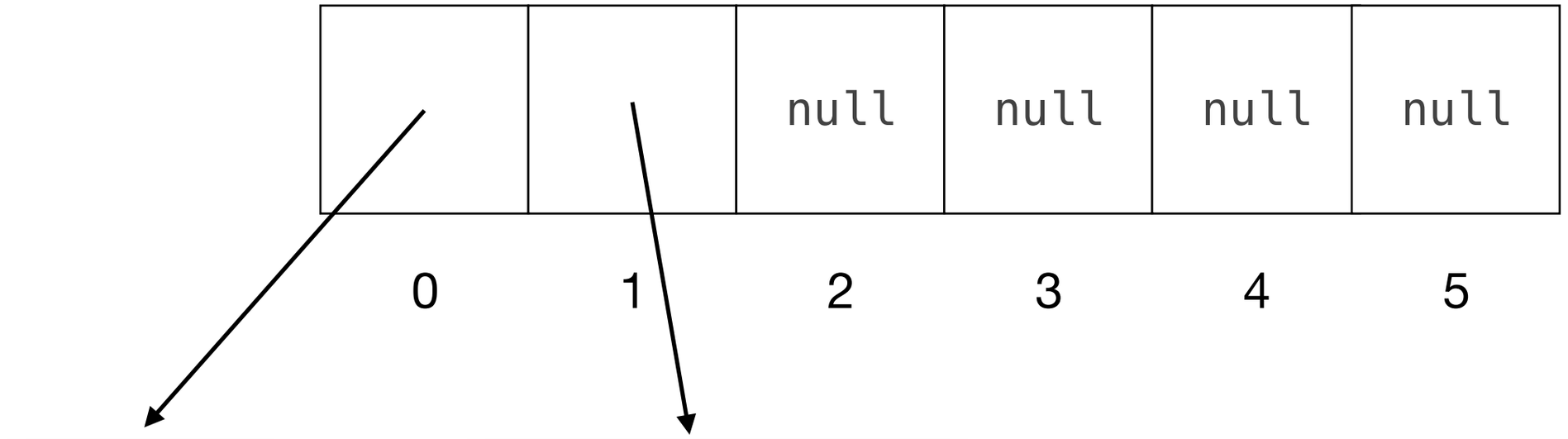
COLEÇÕES DE OBJETOS

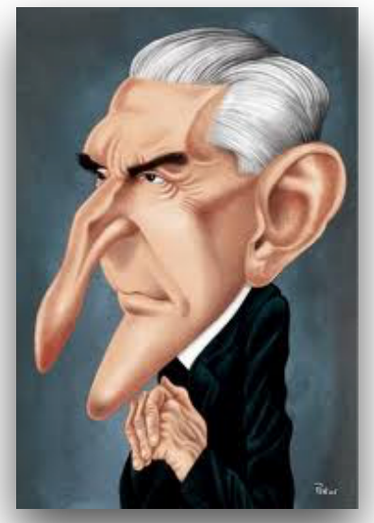
- *Organizador de fotografias*
  - As fotografias podem ser representadas usando objetos `ColorImage`
  - Manipular um organizador de fotografias consiste em
    - adicionar fotografias
    - remover fotografias
    - saber quantas fotografias há no organizador
    - saber se está vazio
    - saber se existe uma dada fotografia
    - ...



COLEÇÕES DE OBJETOS: ATRIBUTOS

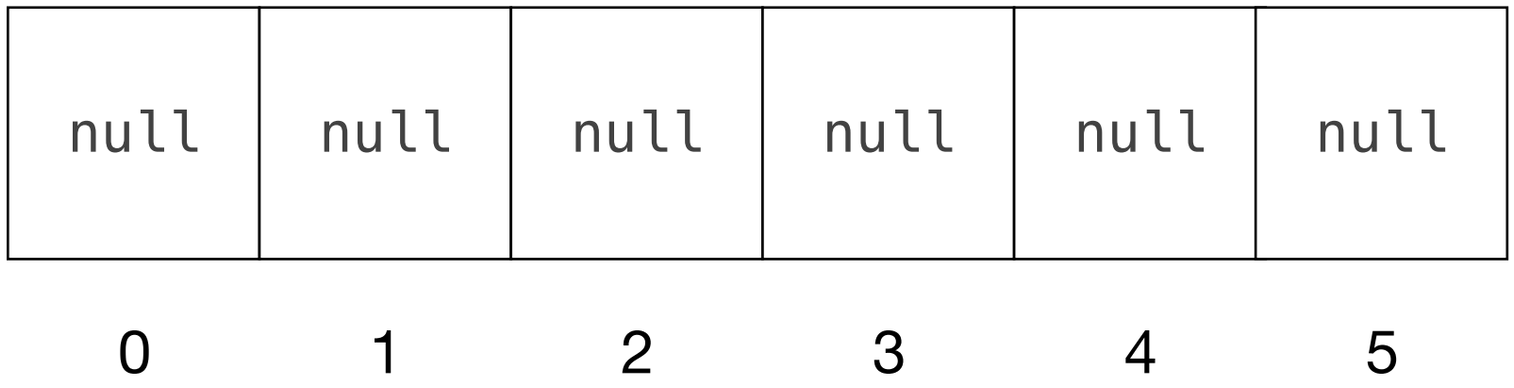
- Em geral, o principal atributo de uma coleção de objetos é um **vetor de objetos**





COLEÇÕES DE OBJETOS: ATRIBUTOS

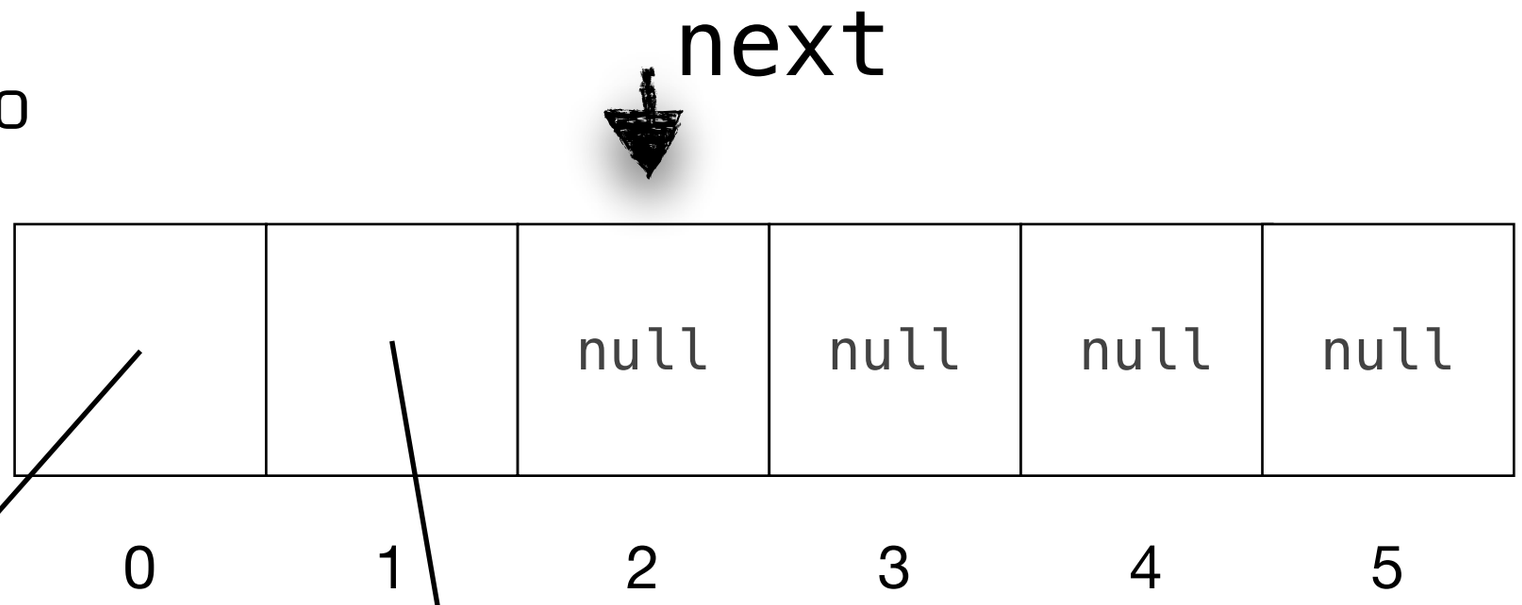
- Quando a coleção é criada, esta não contem objetos: todos os elementos do vetor são referências `null`

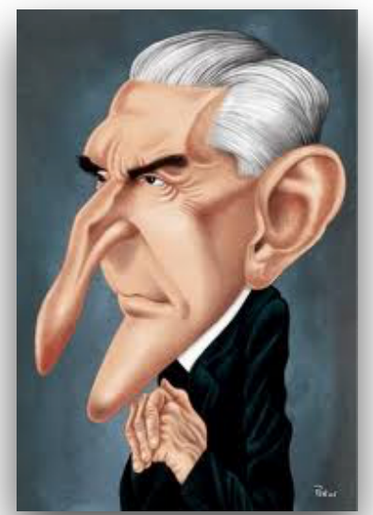


- Como adicionar objetos ao vetor de uma coleção?

COLEÇÕES DE OBJETOS: ATRIBUTOS

- Em geral, para além do vetor de objetos existe um **indicador da posição a preencher** quando um novo objeto é adicionado à coleção





Não há elementos a `null` até ao `next`!

COLEÇÕES DE OBJETOS: ATRIBUTOS

```
class PhotoOrganizer {

    static final int INITIAL_SIZE = 100;

    ColorImage[] images;
    int next;

    PhotoOrganizer() {
        images = new ColorImage[INITIAL_SIZE];
        next = 0;
    }

    …
}
```

COLEÇÕES DE OBJETOS: OPERAÇÕES

```
class PhotoOrganizer {

    …

    void add(ColorImage img) {
        images[next] = img;
        next++;
    }

    …
}
```

E se o vetor estiver totalmente preenchido?

`next == images.length`

COLEÇÕES DE OBJETOS: OPERAÇÕES

```
class PhotoOrganizer {
    …

    void add(ColorImage img) {
        if(next == images.length) {
            ColorImage[] v = new ColorImage[2*images.length];
            for(int i = 0; i != images.length; i++) {
                v[i] = images[i];
            }
            images = v;
        }
        images[next] = img;
        next++;
    }

    …
}
```

Aumentamos o espaço da coleção, criando um novo vetor que substitui o anterior

COLEÇÕES DE OBJETOS: OPERAÇÕES

```
class PhotoOrganizer {

    …

    int numberOfPhotos() {
        return next;
    }

    …
}
```

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 PhotoOrganizer {

    …

    boolean isFull() {

        …

    }


    void add(ColorImage img) {
        if(isFull())
            throw new IllegalStateException(“No space for more photos!”);

        …

    }

}
```

A RETER

- A classe de objetos compostos `ColorImage`
  - Atributos
  - Operações
- Comparação de objetos
- Constantes estáticas
- Coleções de objetos
  - Atributos
  - Operações
- Exceções