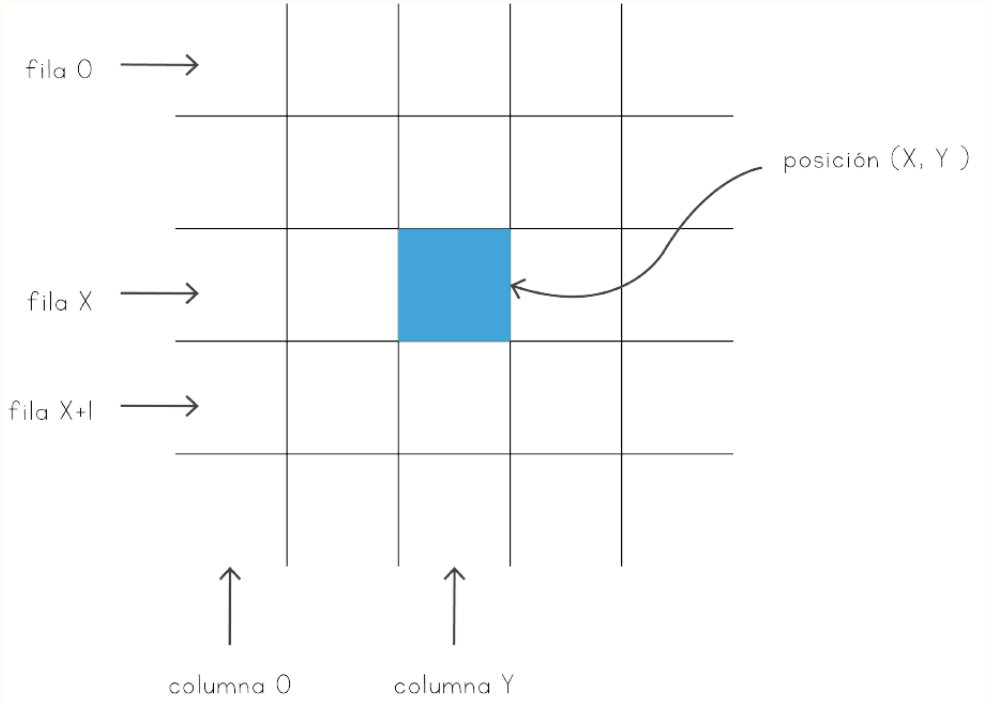
6 NÍVEL

Manejo de estructuras de 2 dimensiones y persistencia

{Contenedoras de 2 dimensiones - Matrices}

PLANO CARTESIANO - MATRICES

Algunos problemas del mundo serían más sencillos de modelar si tuviéramos una estructura que permitiera la manipulación de los elementos utilizando la posición de la fila y de la columna como en un plano cartesiano, así podríamos hablar de:



Un objeto se encuentra en la posición (fila, columna)

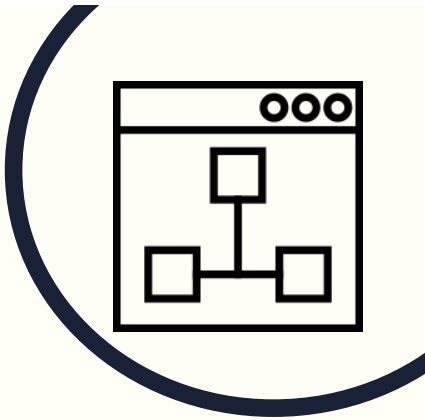
CASO – VISOR DE IMÁGENES

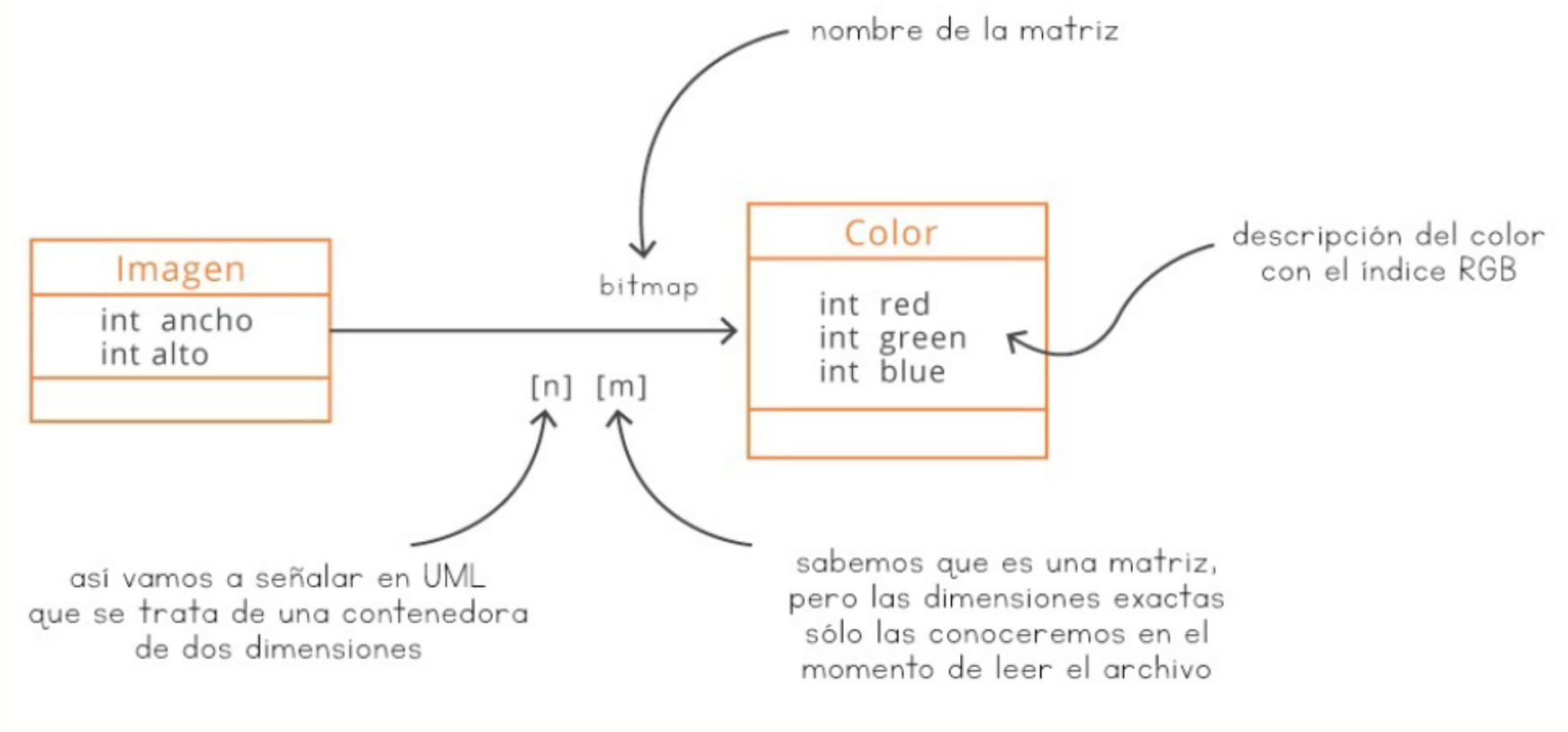
Aplicación que permite visualizar imágenes en formato BMP (guarda la información del color de cada píxel que la conforma). El color de un píxel está representado en el sistema RGB (*Red-Green-Blue*). Un color se forma de la combinación de sus tres componentes, cada uno representado por un número que indica la proporción del color en el color final.

La aplicación mostrará una imagen de máximo 400 x 300 píxeles. Si al cargarla se exceden estos máximos se debe tomar solo dicha porción. Siendo posible cargar imágenes de dimensiones menores.

Adicionalmente, el programa ofrece seis (6) métodos de transformación.

MODELO DEL MUNDO





CONTENEDORAS 2D - MATRICES

Estructura contenedora de dos dimensiones de tamaño fijo, cuyos elementos son referenciados utilizando 2 índices (fila, columna). Se utiliza cuando el mundo del problema se adapta a una representación bidimensional

```
public class Imagen
{ …

        // Atributos
        private int ancho;
        private int alto;
        private Color [ ] [ ] bitmap;
}
```

La clase Color es una clase de Java, que se encuentra en java.awt y permite trabajar colores en formato RGB

INICIALIZACIÓN & ACCESO

```
public class Imagen {

public Imagen ( ){

        ancho = 400;
        alto = 300;
        bitmap = new Color [alto][ancho];
}
```

Length permite consultar el número de filas o columnas que una matriz tiene:

bitmap.lenght -> Filas

bitmap[0].lenght -> Columnas

```
public void imagenAzul ( ) {
  int i, j;
  for ( i = 0; i < bitmap.lenght ; i++ ) {
      for ( j = 0; j < ancho; j++ )
          bitmap [ i ][ j ] = new Color (0,0,255);
  }
}
```

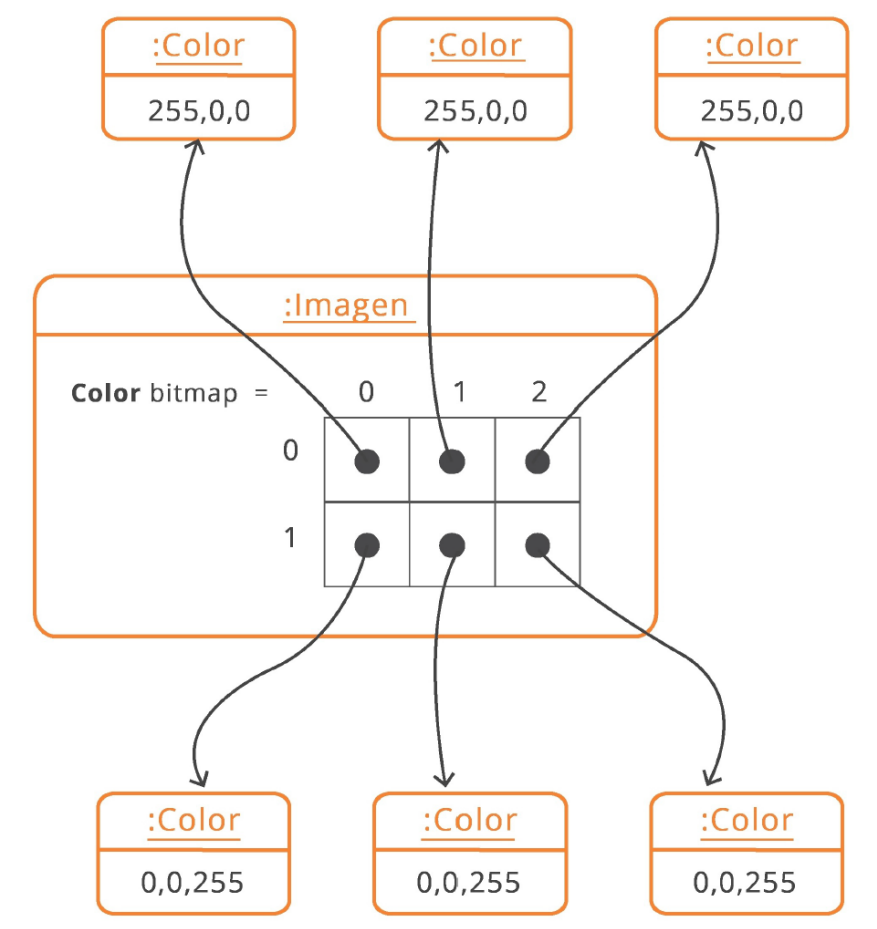
Al igual que cualquier atributo de una clase, en el método constructor es necesario inicializar la matriz antes de poder utilizarla. Para hacerlo es necesario definir las dimensiones, es decir, la cantidad de elementos que va a contener (reservar espacio en memoria

Para acceder a una posición de la matriz necesitamos dos índices (Recuerde que un índice es un valor entero que va desde el 0 (cero) hasta el número de elementos - 1)

<matriz>[<indice_filas>] [<indice_columnas>]

Si tratamos de acceder un elemento de una matrices con un índice no válido obtendremos el error **ArrayIndexOutOfBoundsException**

COMPARANDO ELEMENTOS



La expresión:
bitmap[0][0] == bitmap[0][1]
Es falsa. Ambas referencias llevan a objetos que representan el color rojo, pero son objetos distintos.

La expresión:
bitmap[0][0].equals(bitmap[0][1])
Es verdadera. Ambas referencias llevan a objetos que representan el color rojo y el método no tiene en cuenta que sean instancias distintas.

EJERCICIO — CLASE IMAGEN

/** Calcular el número de píxeles cuyo color es el dado como parámetro.
*@param pColor objeto por el que se quiere preguntar, != null
*@return número de píxeles en la matriz cuyo color es igual al dado
*/
Nota: Se debe utilizar el método equals de la Clase Color

```
public int cuantosPixelesColor ( Color pColor )
{




}
```

PATRÓN DE RECORRIDO TOTAL

- ✓ Problemas que involucran recorrer todos los elementos de la matriz
- ✓ Como el ejercicio anterior

Patrón

```
int i, j;
for ( i = 0; i < NUM_FILAS; i++ ) {
    for ( j = 0; j < NUM_COLUMNAS; j++ ) {
        < cuerpo >
}
}
```

Las condiciones para continuar es que los índices sean menores al número de filas y columnas respectivamente

El avance es sumarle uno al índice

Los indices debe empezar en cero

PATRÓN DE RECORRIDO TOTAL

- ✓ Primera Variante: Todos los elementos de la matriz serán modificados siguiendo una regla
- ✓ Ejemplo: Oscurecer una imagen

```
public void oscurecerImagen ( ) {
 int i,j;
 for ( i = 0; i < alto; i ++ ) {
      for ( j = 0; j < ancho; j ++ )
           bitmap[ i ][ j ] = bitmap[ i ][ j ].darker();
  }
}
```

Se reemplaza el *cuerpo* del patrón por las instrucciones que hacen las modificaciones pedidas

PATRÓN DE RECORRIDO TOTAL

- ✓ Segunda Variante: Calcular algo sobre el conjunto de elementos de la matriz
- ✓ Ejemplo: Contar cuántos puntos tienen su componente rojo igual a cero

```
public int rojoCero ( )
{
    int cuantosRojosCero = 0;
    int i, j;
    for ( i = 0; i < alto; i++ ) {
        for ( j = 0; j < ancho; j++ ) {
            if  (bitmap[ i ][ j ].getRed( ) == 0 )
                cuantosRojosCero ++;
        }
    }
return cuantosRojosCero;
}
```

Acumular información: inicialización del acumulado, cálculo y retorno

EJERCICIO – CLASE IMAGEN

Método que indica cual es la tendencia de color de la imagen. Un pixel tiene tendencia roja si su índice es mayor que los otros dos, lo mismo para los otros colores. Debe retornar 0 sino tiene tendencia, 1 si es hacia el rojo, 2 si es hacia verde y 3 si es hacia el azul

```
public int calcularTendencia ( )
{




}
```

PATRÓN DE RECORRIDO PARCIAL

- ✓ Problemas que involucran recorrer los elementos de una secuencia hasta que encontramos algo que nos hace terminar, encontrar la solución

Patrón

```
boolean termino = false;
int i, j;
for ( i = 0; i < NUM_FILAS && ! termino; i ++ ) {
    for ( j = 0; j < NUM_FILAS && ! termino; j ++ )
        < cuerpo >
        if ( <ya se cumplio el objetivo> )
            termino = true;
}
```

Variable que controla la salida del ciclo

Si su valor es verdadero no se debe volver a entrar

Si ya se resolvió el problema se le asigna el valor verdadedo

En ocasiones se deben utilizar 2 variables distintas para controlar la salida de cada uno de los ciclos, el patrón se aplica de forma anidada

PATRÓN DE RECORRIDO PARCIAL

- ✓ Ejemplo: Permite saber si al menos hay un punto negro

```
public boolean hayPuntoNegro ( )
{
      int i, j;
      boolean termino = false;

      for ( i = 0; i < NUM_FILAS && !termino; i++ ) {
            for ( j = 0; j < NUM_COLUMNAS && !termino; j++) {
                  if ( bitmap[ i ][ j ].equals(Color.BLACK))
                  termino = true;
            }
      }
     return termino;
}
```

PATRÓN DE RECORRIDO PARCIAL

- ✓ Ejemplo: Permite saber si hay más de 50 filas con un píxel negro

```
public boolean muchasFilasConPixelNegro ( )
{
    boolean termino1 = false;
    int i, j, numFilas = 0;
    for ( i = 0; i < NUM_FILAS && !termino1; i++ ) {
        boolean termino2 = false;
        for ( j = 0; j < NUM_COLUMNAS && !termino2; j++) {
            if (bitmap[ i ][ j ].equals(Color.BLACK))
                numFilas++;
                termino2 = true;
        }
        if ( numFilas > 50)
            termino1 = true;
    }
    return termino1;
}
```

Objetivo del ciclo2 encontrar un punto negro en la fila

Objetivo del ciclo1 encontrar mas de 50 filas que cumplan la regla

EJERCICIO — CLASE IMAGEN

Contar el número de píxeles que son completamente verdes en la fila numFila

```
public int contarVerdes (  int numFila)
{









}
```

EJERCICIO – CLASE IMAGEN

Método que permite binarizar una imagen. Es decir, ponerla en blanco y negro cumpliendo que: si la suma de sus 3 componentes es menor que 100 lo reemplaza por el color blanco (255,255,255), en caso contrario por el color negro (0,0,0)

```
public void binarizar ( )
{




}
```

EJERCICIO – CLASE IMAGEN

Método que permite filtrar una imagen. Es decir, mejorar su calidad calculando de nuevo el valor de cada pixel, como el promedio de los 8 vecinos (no se deben incluir los bordes, porque no tienen los vecinos necesarios)

```
public Color [ ][ ] imagenFiltrada( )
{




}
```

EJERCICIO — CLASE IMAGEN

Método que rota la imagen 90 grados a la derecha

```
public void rotarDerechar90( )
{









}
```