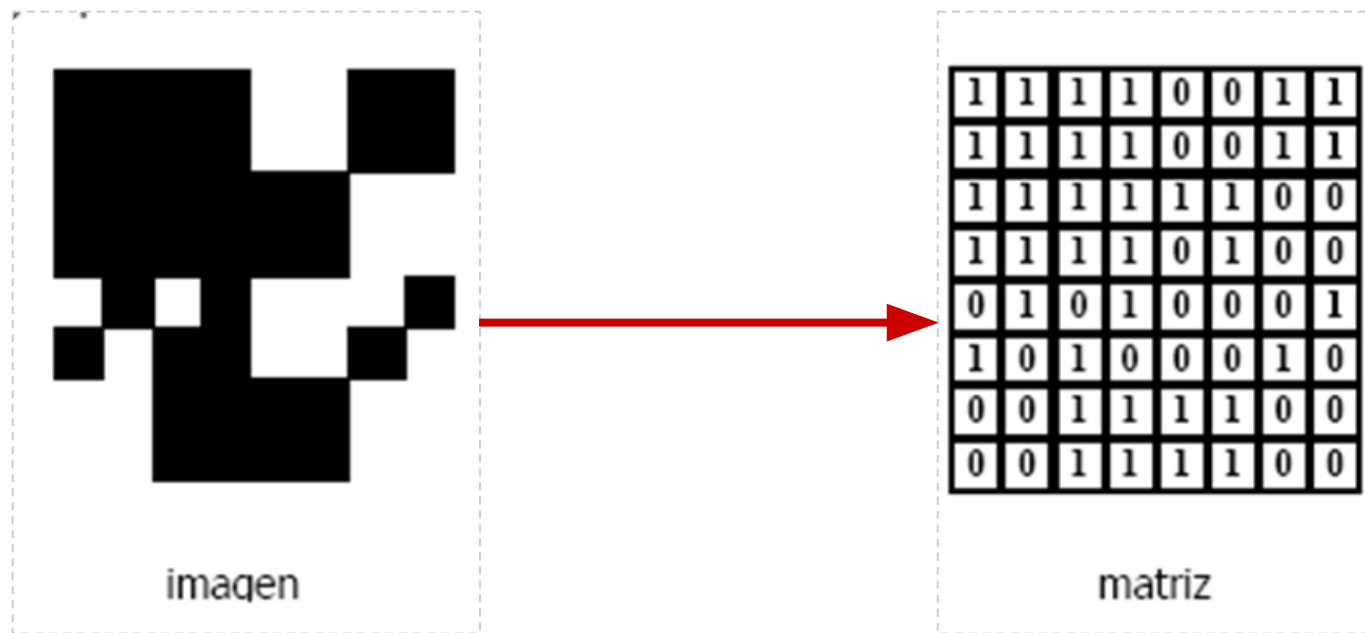


# **Algoritmos y Estructuras de Datos 2015**

Árboles Generales

# Imagen Comprimida



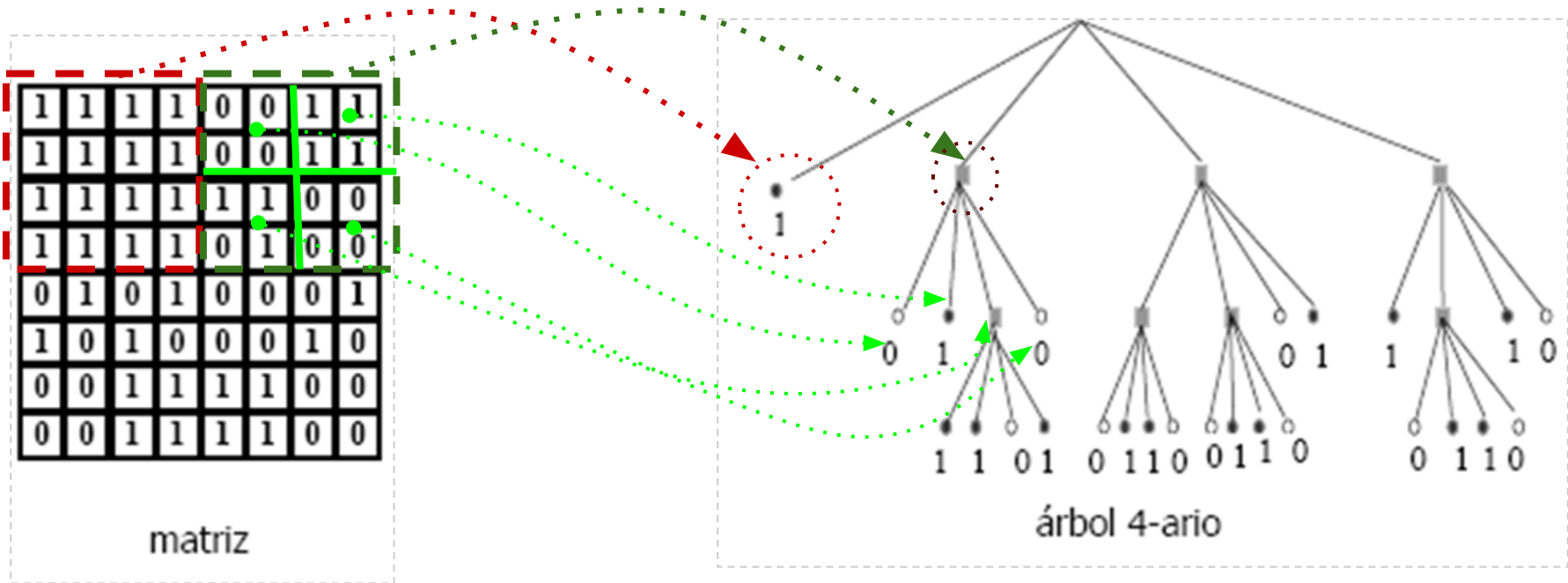
Una **Imagen** puede representarse como **una matriz** (en el ejemplo, cada casillero es blanco o es negro).

# Algoritmo

*Si toda la matriz tiene **un mismo color**, se debe definir **un nodo** con ese color.*

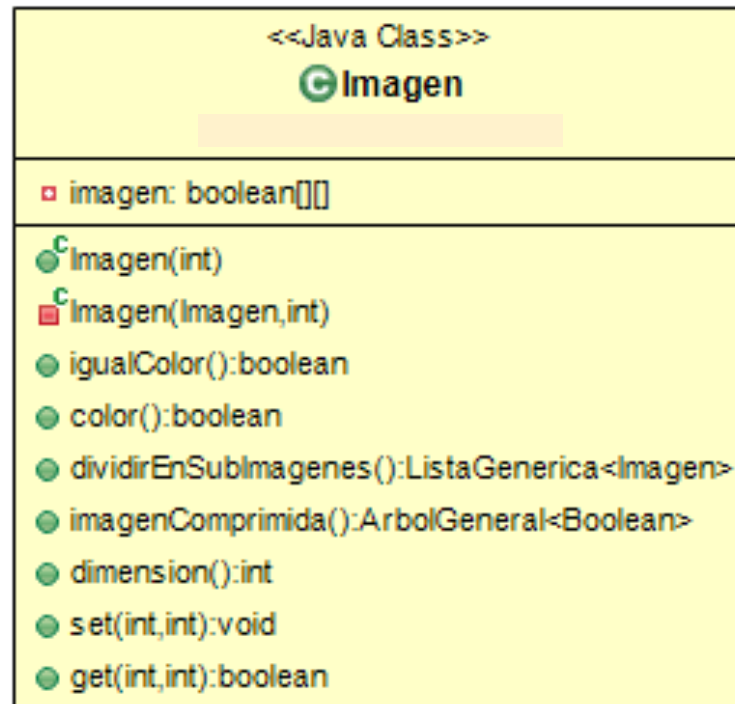
*En caso contrario, se **divide la matriz en cuatro partes**, se define **un nodo con 4 hijos**, y cada hijo es la conversión de cada una de las partes de la matriz*

# Árbol 4-ario




Un **árbol 4-ario** es una representación de la **imagen comprimida**.

# Clase Imagen



# Métodos de la clase Imagen

```
public ArbolGeneral<Boolean> imagenComprimida() {  
    ArbolGeneral<Boolean> agTmp = null;  
    if (this.igualColor()) {  
        agTmp = new ArbolGeneral<Boolean>(color());  
        return agTmp;  
    } else {  
        ListaGenerica<Imagen> imagenes = this.dividirEnSubImagenes();  
        Imagen imgProcesar = null;  
        imagenes.comenzar();  
        agTmp = new ArbolGeneral<Boolean>(null);  
        while (!imagenes.fin()) {  
            imgProcesar = imagenes.proximo();  
            agTmp.agregarHijo(imgProcesar.imagenComprimida());  
        }  
    }  
    return agTmp;  
}
```



Cada imagen generada a partir de un sector de la imagen general, devolverá un Árbol General como su representación

# Métodos de la clase Imagen

```
public int dimension() {  
    return this.imagen.length;  
}
```

Como la matriz es cuadrada  
da lo mismo pedir el número  
de filas que el número de  
columnas

```
public void set(int fila, int columna) {  
    this.imagen[fila][columna] = true;  
}
```

```
public boolean get(int fila, int columna) {  
    return this.imagen[fila][columna];  
}
```

# Métodos de la clase Imagen

```
public boolean igualColor() {
    boolean color = this.imagen[0][0];
    for (int i = 0; i < this.imagen.length; i++) {
        for (int j = 0; j < this.imagen.length; j++) {
            if (this.imagen[i][j] != color)
                return false;
        }
    }
    return true;
}

// solo devuelve el color de la primera posición
// como muestra
public boolean color() {
    return this.imagen[0][0];
}
```



**Constructor** que crea una Imagen **copiando un sector** de una imagen original


1	2
3	4

```
private Imagen(Imagen original, int sector) {
    this(original.dimension() / 2);
    int medio = original.dimension() / 2;
    int fila = 0;
    int columna = 0;
    int inicioX = 0;
    int inicioY = 0;
    switch (sector) {
        case 1:
            inicioX = 0;
            inicioY = 0;
            break;
        case 2:
            inicioX = medio;
            inicioY = 0;
            break;
        case 3:
            inicioX = 0;
            inicioY = medio;
            break;
        case 4:
            inicioX = medio;
            inicioY = medio;
            break;
        default:
            break;
    }

    for (int i = inicioX; i < inicioX + medio; i++) {
        columna = 0;
        for (int j = inicioY; j < inicioY + medio; j++) {
            if (original.get(i, j))
                this.set(fila, columna);
            columna++;
        }
        fila++;
    }
}
```

# Métodos de la clase Imagen

```
public ListaGenerica<Imagen> dividirEnSubImagenes() {  
    ListaGenerica<Imagen> imagenes = new ListaEnlazadaGenerica<Imagen>();  
    Imagen imagen = null;  
    for (int i = 1; i <= 4; i++) {  
        imagen = new Imagen(this, i);  
        imagenes.agregarFinal(imagen);  
    }  
    return imagenes;  
}
```



Invoca al constructor que devuelve una imagen por sector!