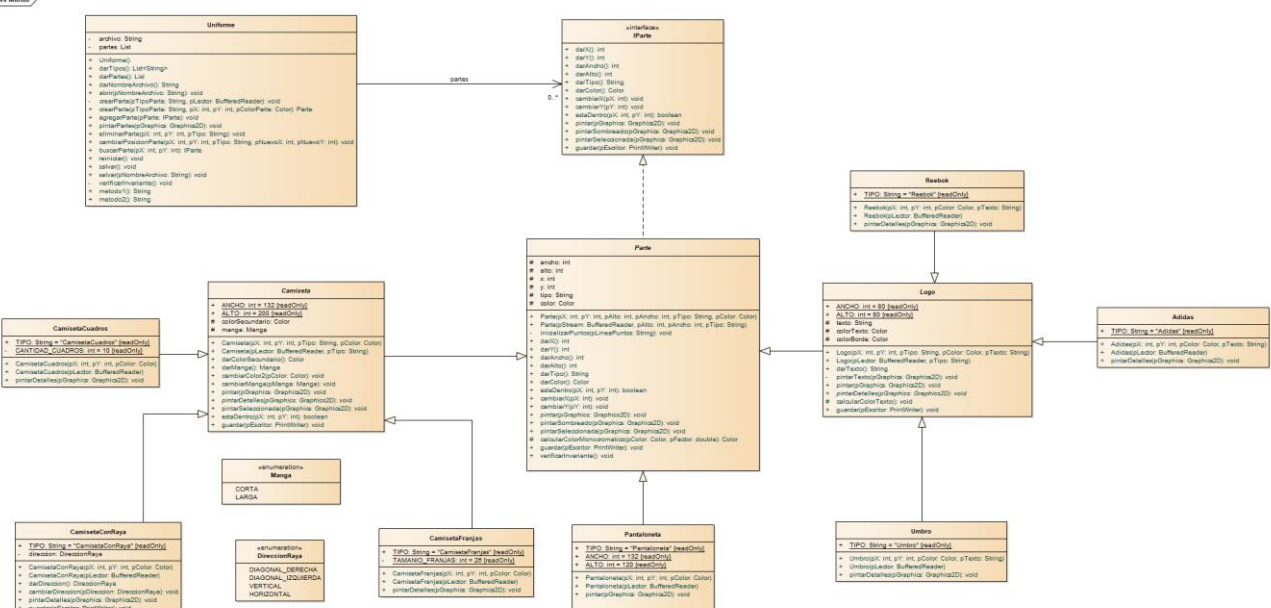


Considerando lo anterior, se propone el siguiente modelo del mundo:



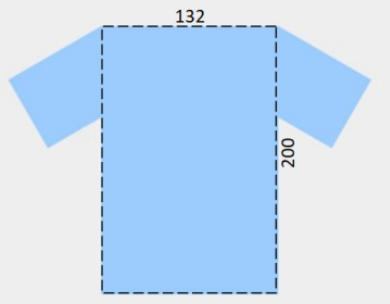
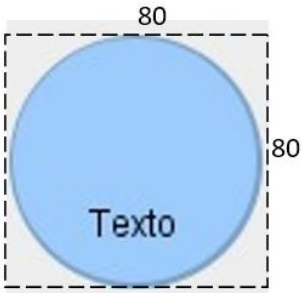
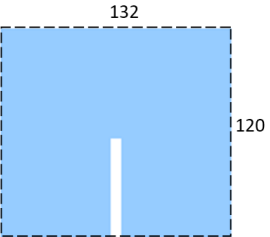
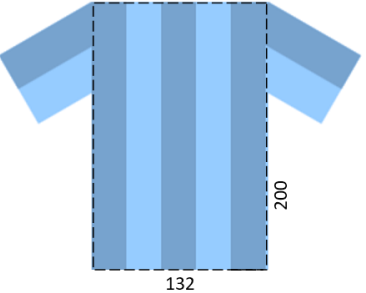
En la siguiente tabla se describen algunas de las clases incluidas para que la aplicación cumpla con las características de desacoplamiento y reutilización esperadas:

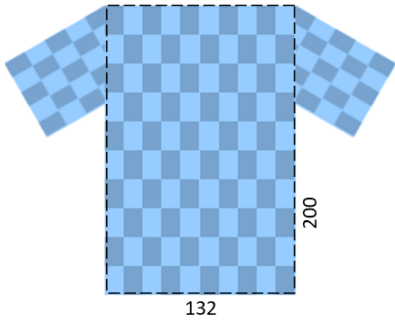
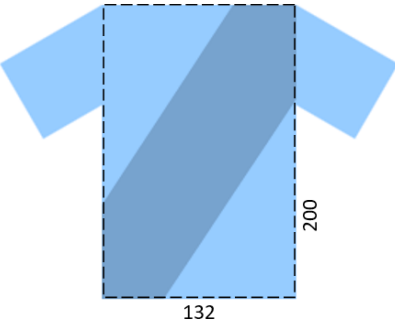

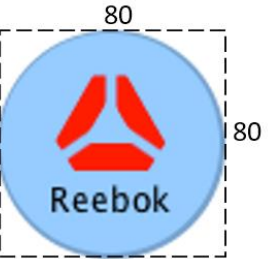
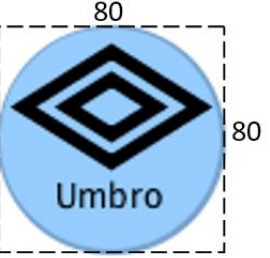
Clase	Descripción
<pre> «interface» IParte + darX(): int + darY(): int + darAncho(): int + darAlto(): int + darTipo(): String + darColor(): Color + cambiarX(pX: int): void + cambiarY(pY: int): void + estaDentro(pX: int, pY: int): boolean + pintar(pGraphics: Graphics2D): void + pintarSombreado(pGraphics: Graphics2D): void + pintarSeleccionada(pGraphics: Graphics2D): void + guardar(pEscritor: PrintWriter): void </pre>	<p>IParte (Interface)</p> <p>Interface que define las funcionalidades que debe ofrecer cualquier elemento gráfico del uniforme.</p>
<pre> Parte # ancho: int # alto: int # x: int # y: int # tipo: String # color: Color + Parte(pX: int, pY: int, pAlto: int, pAncho: int, pTipo: String, pColor: Color) + Parte(pStream: BufferedReader, pAlto: int, pAncho: int, pTipo: String) - inicializarPuntos(pLineaPuntos: String): void + darX(): int + darY(): int + darAncho(): int + darAlto(): int + darTipo(): String + darColor(): Color + estaDentro(pX: int, pY: int): boolean + cambiarX(pX: int): void + cambiarY(pY: int): void + pintar(pGraphics: Graphics2D): void + pintarSombreado(pGraphics: Graphics2D): void + pintarSeleccionada(pGraphics: Graphics2D): void # calcularColorMonocromatico(pColor: Color, pFactor: double): Color + guardar(pEscritor: PrintWriter): void + verificarInvariante(): void </pre>	<p>Parte (Abstract)</p> <p>Clase abstracta que representa un elemento gráfico. Define las propiedades (atributos y métodos) que comparten todas las partes. Sirve como base para la implementación de las clases de todos los elementos gráficos. Esta clase se compromete con el contrato funcional de la interface <i>IDibujo</i>.</p>
<pre> Camiseta + ANCHO: int = 132 {readOnly} + ALTO: int = 200 {readOnly} # colorSecundario: Color # manga: Manga + Camiseta(pX: int, pY: int, pTipo: String, pColor: Color) + Camiseta(pLector: BufferedReader, pTipo: String) + darColorSecundario(): Color + darManga(): Manga + cambiarColor2(pColor: Color): void + cambiarManga(pManga: Manga): void + pintar(pGraphics: Graphics2D): void + pintarDetalles(pGraphics: Graphics2D): void + pintarSeleccionada(pGraphics: Graphics2D): void + estaDentro(pX: int, pY: int): boolean + guardar(pEscritor: PrintWriter): void </pre>	<p>Camiseta (Abstract)</p> <p>Clase abstracta que representa una camiseta. Define los métodos que toda camiseta del uniforme ejecuta de manera similar. Esta clase hereda de la clase Parte.</p>

<pre> Logo + ANCHO: int = 80 {readOnly} + ALTO: int = 80 {readOnly} # texto: String # colorTexto: Color # colorBorde: Color + Logo(pX: int, pY: int, pTipo: String, pColor: Color, pTexto: String) + Logo(pLector: BufferedReader, pTipo: String) + darTexto(): String - pintarTexto(pGraphics: Graphics2D): void + pintar(pGraphics: Graphics2D): void + pintarDetalles(pGraphics: Graphics2D): void # calcularColorTexto(): void + guardar(pEscritor: PrintWriter): void </pre>	<p>Logo (Abstract) Clase abstracta que representa un logo. Define los métodos y constantes que todo logo debe tener. Esta clase hereda de la clase Parte.</p>
<pre> Pantaloneta + TIPO: String = "Pantaloneta" {readOnly} + ANCHO: int = 132 {readOnly} + ALTO: int = 120 {readOnly} + Pantaloneta(pX: int, pY: int, pColor: Color) + Pantaloneta(pLector: BufferedReader) + pintar(pGraphics: Graphics2D): void </pre>	<p>Pantaloneta Clase que representa una pantaloneta del uniforme.</p>

Dibujos

A continuación, se presentan los dibujos con los que cuenta el editor de uniformes, y sus dimensiones.

Nombre	Figura	Consideraciones	Clase Padre
Camiseta (clase abstracta)		Alto: 200 Ancho: 132 Dibujado en Java2D	Parte
Logo (clase abstracta)		Alto: 80 Ancho: 80 Dibujado con Java2D	Parte
Pantaloneta		Alto : 132 Ancho: 120 Dibujado en Java2D	Parte
CamisetaFranjas		Alto: 200 Ancho: 132 Dibujado en Java2D	Camiseta

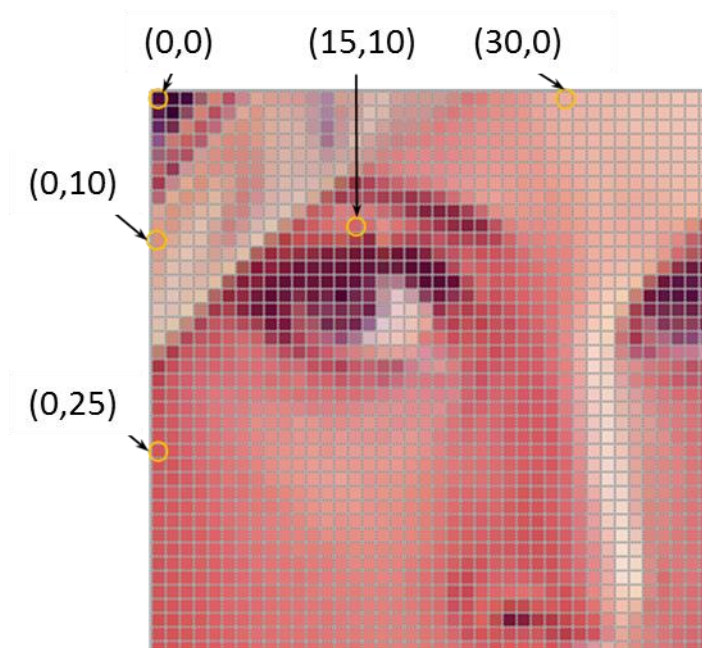
CamisetaCuadros		Alto: 200 Ancho: 132 Dibujado en Java2D	Camiseta
CamisetaConRaya		Alto: 200 Ancho: 132 Dibujado en Java2D	Camiseta
Adidas		Alto :80 Ancho: 80 Dibujado con Java2D	Logo
Reebok		Alto :80 Ancho: 80 Dibujado con Java2D	Logo
Umbro		Alto :80 Ancho: 80 Dibujado con Java2D	Logo

En adelante, se dan consejos útiles que guiarán al estudiante a dibujar correctamente cada componente del uniforme.

Sistema de coordenadas

A continuación se exponen varios detalles útiles para el desarrollo del proyecto:

- Las coordenadas de los símbolos gráficos del editor están dadas en píxeles.
- Java2D dibuja sobre una superficie de píxeles, cuya dimensión depende del panel en donde se esté dibujando.
- El origen de coordenadas se encuentra en la posición (0,0) que está situada en la esquina SUPERIOR IZQUIERDA de la superficie de dibujo (lienzo).
- La primera coordenada se refiere a la COLUMNA del píxel con el cual se está trabajando. Es creciente hacia la derecha.
- La segunda coordenada se refiere a la FILA del píxel con el cual se está trabajando. Es creciente HACIA ABAJO.
- Todo esto se ilustra en la siguiente figura:



Se debe tener esto en cuenta, pues en el código las variables de las coordenadas se llaman (x, y) , que “inducen” a pensar erróneamente en un plano cartesiano, que tiene el origen en la esquina inferior izquierda, donde la primera coordenada es creciente hacia la derecha y la segunda coordenada es creciente hacia arriba.

Sugerencias para pintar los dibujos de Fábrica de Uniformes

El editor de uniformes cuenta con varios tipos de componentes gráficos tales como camisetas, logos o una pantaloneta para construir un uniforme. Como se explicó arriba, el panel de edición funciona como un sistema de coordenadas (x, y) que tiene su origen (0,0) en la esquina superior izquierda del panel, y aumenta sus unidades (en pixeles) hacia la derecha para 'x' y hacia abajo para 'y'. Este sistema sirve para definir e identificar la posición de las partes en el panel.

Las diferentes partes cuentan con una representación gráfica en el panel de edición. Esta representación está inscrita en un rectángulo (que no se dibuja) de dimensiones 'alto' y 'ancho'. La posición (x, y) de la parte hace referencia a la posición de la esquina superior izquierda del rectángulo en el panel de edición. La Figura 1 explica esto.

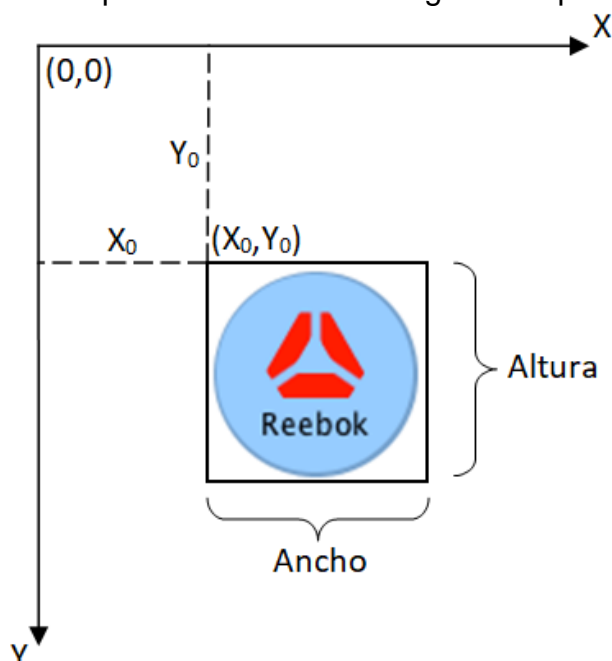


Figura 1. Ejemplo de una parte en el panel de edición.

La clase Graphics2D es utilizada para realizar el trazado de diferentes figuras básicas (rectángulos, óvalos, polígonos etc.) con o sin relleno¹, así como cargar y dibujar imágenes diseñadas que en conjunto forman cada una de los dibujos. Además de esto, permite variar el color con el que se está dibujando con el método `setColor(Color c)` y el grosor de las líneas con el método `setStroke()`².

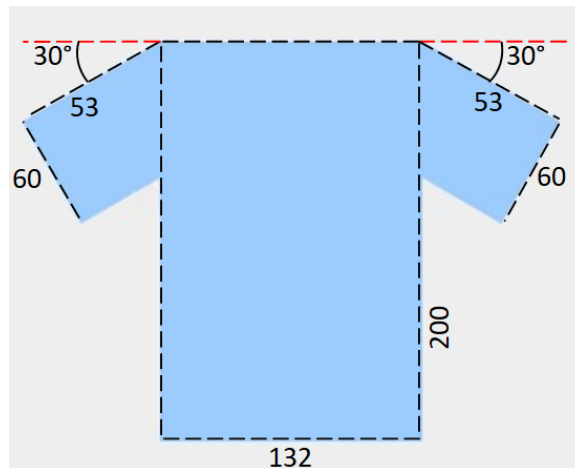
A continuación, se dan indicaciones de cómo elaborar los dibujos de los diferentes componentes que ofrece el editor de uniformes.

¹ Observe por ejemplo la diferencia entre el método `drawOval()` y `fillOval()`.

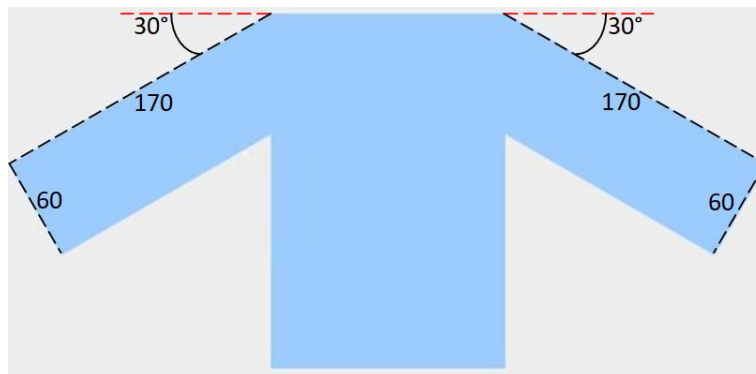
² Refiérase a la documentación de la clase `BasicStroke` y al método de `Graphics2D` `setStroke()`.

Camiseta

La clase Camiseta es una clase abstracta de la cual heredan los diferentes tipos de camisetas. Esta clase permite dibujar el torso y las mangas de la camiseta, que son iguales para todas las camisetas. Tanto el torso como las mangas están dibujados con Java2D.



Las camisetas también pueden ser de manga larga.



Tenga en cuenta que la clase Camiseta maneja 3 estilos: camiseta con cuadros, camiseta rayada y camiseta con una raya. A continuación, se explica cómo se debe dibujar cada estilo de camiseta.

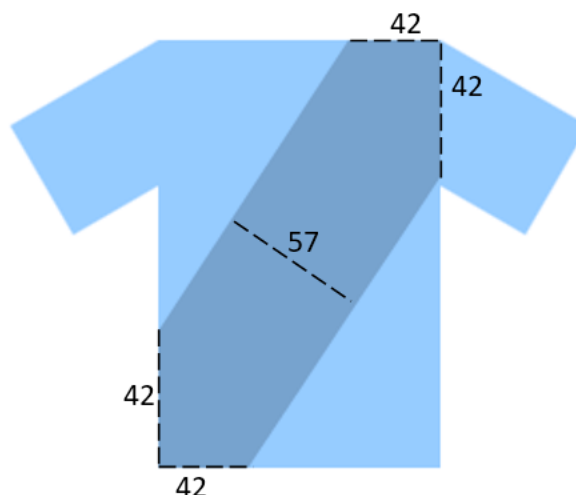
- **Camiseta con cuadros:** Esta camiseta se dibuja con el número de cuadros que la componen. En el caso de camiseta de manga larga, las cantidades se mantienen.



- **Camiseta rayada:** Esta camiseta se dibuja mediante el ancho de las rayas. Las unidades de la imagen son pixeles y estas no difieran en las camisetas de manga larga.

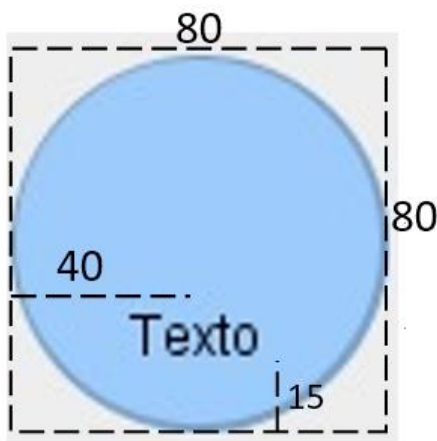


- **Camiseta con una raya:** Esta camiseta puede tener la raya en 4 direcciones, diagonal derecha, diagonal izquierda, vertical y horizontal. En todas las opciones, el ancho de la raya se mantiene.

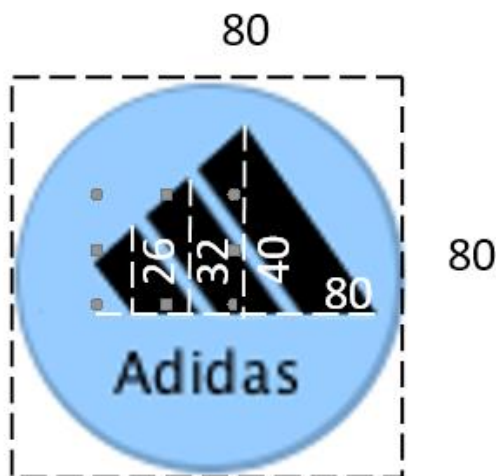


Logo

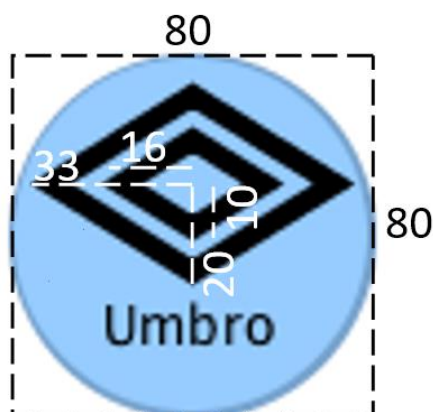
La clase abstracta Logo, define el modelo básico de cualquier tipo de logo en el uniforme. Este modelo consiste en una imagen dibujada con java2D. A continuación, se explica cómo dibujar los diferentes tipos de logos que se crean a partir de esta definición.



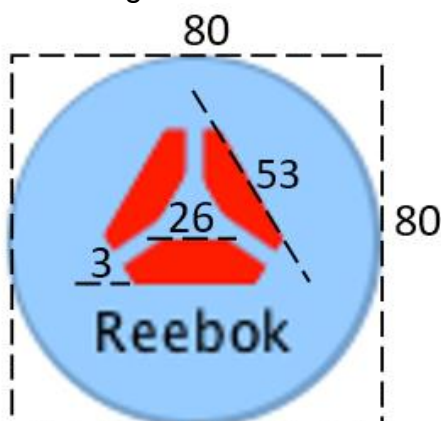
- **Logo Adidas:** Clase que modela el logo de Adidas con un campo de texto ubicado en la zona inferior.



- **Logo Umbro:** Clase que modela el logo de Umbro con un campo de texto ubicado en la zona inferior. El logo se compone por rombos con las dimensiones dadas.

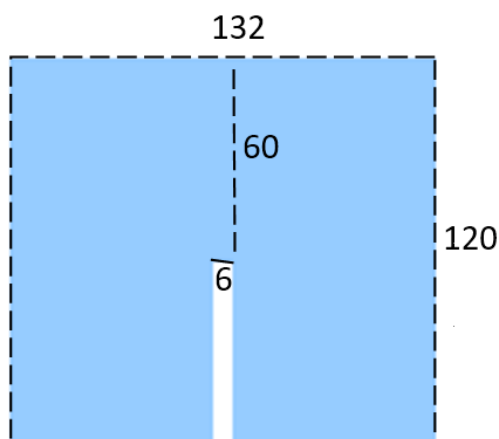


Logo Reebok: Clase que modela el logo de Reebok con un campo de texto ubicado en la zona inferior. El logo se compone por triángulos equiláteros y por líneas que unen el triángulo interior con el triángulo exterior.



Pantaloneta

La clase Pantaloneta modela una pantaloneta para el uniforme. La pantaloneta tiene las siguientes dimensiones.



Calcular el color del texto

Tanto las camisetas como los logos deben calcular el color del texto a partir de su color de fondo. Para calcular si este color debe ser negro o blanco es necesario determinar la luminosidad del fondo y encontrar cuál de los dos contrasta más. A continuación, se presenta el método de la clase “Logo” que le facilitará desarrollar esta funcionalidad:

```
/**
 * Calcula el color del texto a partir del color principal del logo.
 */
protected void calcularColorTexto( )
{
    colorTexto = Color.BLACK;
    int brillo = ( int )Math.sqrt( color.getRed( ) * color.getRed( ) * .241 +
    color.getGreen( ) * color.getGreen( ) * .691 + color.getBlue( ) * color.getBlue( )
    * .068 );
    if(brillo<130)
    {
        colorTexto = Color.WHITE;
    }
}
```

En algunos casos también será necesario calcular un color secundario a partir de uno principal. Esto se puede hacer usando un esquema monocromático en el cual el color secundario corresponderá a el color principal multiplicado por un factor dado. A continuación, se presenta el método de la clase “Parte” que le facilitará desarrollar esta funcionalidad:

```
/**
 * Calcula el color monocromático según el porcentaje dado.<br>
 * <b>Nota: </b> Para calcular un color similar se requiere multiplicar cada
componente por el factor dado.
 * @param pColor Color base. pColor != null.
 * @param pFactor Factor con el cual se calculará el color.
 * @return Color nuevo de acuerdo al factor dado.
 */
protected Color calcularColorMonocromatico(Color pColor,double pFactor)
{
    return new Color((int)(pFactor*pColor.getRed()),
                    (int)(pFactor*pColor.getGreen()),
                    (int)(pFactor*pColor.getBlue()));
}
```