Índice

1. Interfaces

Una interfaz se define del mismo modo que una clase cambiando la palabra reservada *class* por *interface*. Los métodos que define la interfaz no tienen cuerpo y terminan con un punto y coma tras la lista de parámetros. Es responsabilidad de las clases que la implementen definir el cuerpo de los métodos.
Se pueden declarar variables dentro de una interfaz y son *final* y *static*. Una interfaz se puede declarar como public o sin modificador de acceso (por lo que solo podrá ser vista por clases que estén en el mismo paquete o en el mismo directorio).

Si no se pueden crear objetos de una interfaz, ¿cuál es su utilidad? Su utilidad reside en que define los requisitos mínimos que debe cumplir cualquier objeto que creemos a partir de una clase que la implemente. Con ello nos aseguramos de que el objeto tiene como mínimo el conjunto de métodos que declara la interfaz, pero cada clase que la implemente especifica cómo se llevan a cabo las tareas que se han declarado en la interfaz.
Cuando una clase implementa una interfaz se compromete a definir cuerpo a todos los métodos impuestos por la interfaz.

Resumiendo:
Una interfaz declara, pero no implementa, las acciones mínimas que poseerá un objeto creado a partir de una clase que la implemente. Es responsabilidad de las clases que implementan la interfaz definir cómo se llevan a cabo todas y cada una de las acciones declaradas por la interfaz.

2. Los paquetes java.awt y javax.swing

El paquete java.awt proporciona una serie de clases e interfaces para realizar programas que se ejecuten en un entorno gráfico (AWT viene del inglés Abstract Window Toolkit).

A continuación se presenta un listado de algunas de las clases de este paquete que vamos a utilizar en esta práctica:

java.awt	Descripción
Component	Superclase abstracta de varios componentes del AWT. Tiene un método public Graphics getGraphics() para obtener el contexto gráfico sobre el que dibujar figuras o componentes. Declara más de 200 métodos
Container	Subclase de la anterior que puede contener otros componentes. Tiene un método que nos permite añadir un componente: Component add(Component comp)
Color	Clase que representa un color

Graphics	Encapsula el contexto gráfico que es utilizado por varios métodos para mostrar la salida en una ventana. Tiene además una serie de métodos para dibujar figuras
Point	Clase para representar las coordenadas cartesianas x e y
JPanel	Representa un área rectangular sobre la que la aplicación puede dibujar. La aplicación debe crear una clase que extienda a JPanel y sobreescribir el método void paintComponent(Graphics g) para obtener la funcionalidad que se desee. Es una subclase de Component

El paquete **javax.swing** proporciona una serie de clases e interfaces que amplían la funcionalidad del anterior. Están escritos en Java y son independientes de la plataforma. Se utiliza el término ligthweight para describir a estos elementos. La clase de este paquete que vamos a utilizar es:

javax.swing	Descripción
JFrame	Clase que representa una ventana. El método Container.getContentPane() debe utilizarse para obtener un objeto del tipo Container que se utilizará para añadir componentes a la ventana

Para realizar un programa que cree un entorno gráfico, hay que importar clases de estos dos paquetes

3. Tareas

Junto con este enunciado se proporciona un fichero ZIP que contiene 7 archivos JAVA, cada uno de los cuales contiene la definición de una clase o de una interfaz.

La primera tarea que se pide es:

- Ubicar cada fichero JAVA en el directorio que le corresponde de acuerdo con la sentencia *package* presente en cada uno de ellos
- Incluir las sentencias *import* necesarias en cada fichero JAVA.
  No utilizar la estructura: *import nombre_paquete.**
  sino: *import nombre_paquete.NombreClase*
  para observar claramente las dependencias de unas clases/interfaces con otras
- Utilizar las palabras reservadas de visibilidad adecuadas para las clases y las interfaces (*public* o nada) y los métodos (*public*, *private*, *protected* o nada) presentes en los ficheros JAVA proporcionados.
  En los ficheros proporcionados todas las clases e interfaces, y todos los métodos tienen visibilidad *de paquete*. Se trata de ser lo más restrictivo que sea posible en cuanto a visibilidad; es decir, calificar como público un método, una clase o una interfaz sólo cuando sea estrictamente necesario SIN VARIAR LA CONFIGURACIÓN EN LOS PAQUETES PROPUESTOS.

Una vez finalizada completamente esta tarea, el código se puede compilar sin errores y la aplicación se puede ejecutar invocando la clase LanzaPrac3.

Se pide una segunda tarea consistente en:

- Escribir código para una clase Linea que implemente la interfaz Figura (y no FiguraCerrada)
- Escribir código para una clase Trapecio que extienda la clase java.awt.Polygon e implemente la interfaz FiguraCerrada
- Escribir código para una clase TrapecioRelleno que extienda la clase Trapecio

Y por último se pide:

- Escribir el código en la clase LanzaPrac3 de un sencillo menú por consola que nos permita elegir qué figura queremos representar gráficamente

Comentarios a las clases e interfaces presentes en los ficheros JAVA proporcionados:

- **Figura** es una interfaz que fija los requisitos que debe tener cualquier objeto que quiera ser una figura: que se pueda obtener su longitud y que se pueda pintar
- **FiguraCerrada** es una interfaz que extiende a la anterior y define un requisito más para aquellas figuras que sean cerradas: este requisito es que se pueda calcular el área
- **Circunferencia** es una clase que implementa la interfaz FiguraCerrada, definiendo cómo se calcula su longitud, cómo se pinta y cómo se calcula su área
- **CircunferenciaRellena** es una clase que extiende Circunferencia y define su propio método para pintar
- **VentanaDeDibujo** es una clase que extiende la clase Jframe; en ella se guardan todas la figuras que se van creando para pintarlas cada vez que sea necesario (por ejemplo: cuando se mueva la ventana, cuando se maximice, cuando otra ventana que la estaba ocultando la deja visible, etc.).
  Además define una clase interna privada llamada **Superficie** (las clases internas pueden tener los mismos modificadores de acceso que un atributo) de tal forma que sólo VentanaDeDibujo pueda crear objetos del tipo Superficie. Superficie es una subclase de JPanel y hay que reescribir su método paintComponent(Graphics g) para que se vuelvan a pintar todas las figuras cuando se necesite
- **LanzaPrac3** es la clase que contiene el main(· · ·); lo único que hace es crear un objeto del tipo VentanaDeDibujo y crear las figuras que se deseen