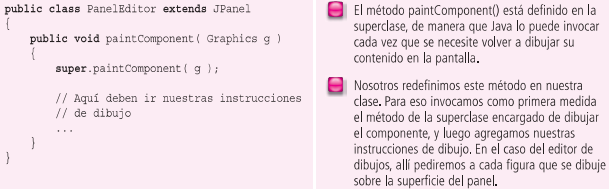
***Programación gráfica***

Todo componente gráfico tiene una superficie de dibujo que provee los métodos necesarios para hacerlo

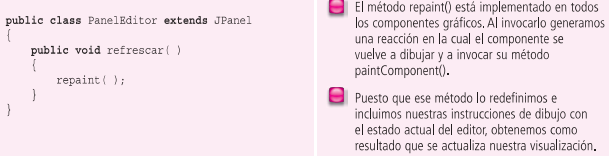
El problema es obtener esa superficie a través del componente en el que se quiere dibujar y a invocar los métodos correspondientes, en java estas superficies de dibujo se implementan por las clases ***Graphics*** y ***Graphics2D***

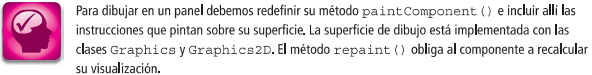
Java decide que componentes (o parte de ellos) volver a pintar, en este proceso nuestra participación es indirecta puesto que queremos que invoque nuestros métodos (de dibujo) cuando pida los componentes gráficos que se muestran en pantalla, si no se hace asi todo desaparecerá cuando el panel tenga que volver a dibujar, la solución es redefinir el método ***paintComponent()*** que tienen todos los métodos gráficos, para incluir allí nuestras instrucciones de dibujo.



Si queremos que java cambie todo lo que hemos dibujado sobre una superficie (EJ. Eliminar una figura) es nuestra obligación hacer que esto ocurra invocando los métodos necesarios.

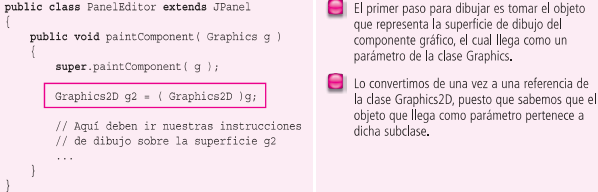
El método ***repaint()***(de los elementos gráficos) se encarga de hacer todo este trabajo por nosotros, el cual en algún momento invoca el método ***paintComponente()*** que ya hemos redefinido (si queremos incluir un método refrescar bastaría con hacer:





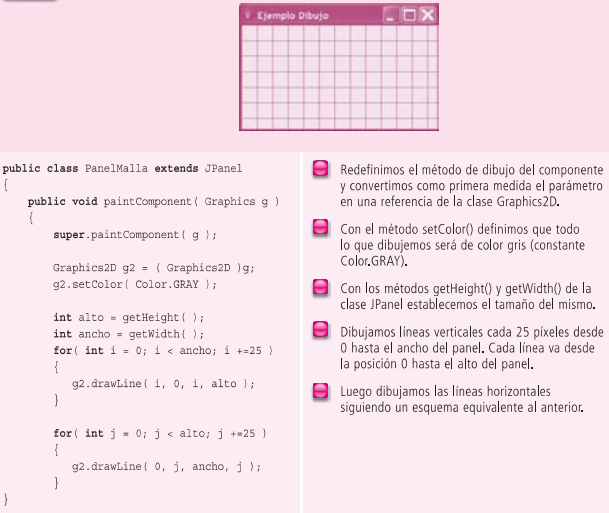
\*\*\*Todo componente gráfico de swing usa el sistema de coordenadas (x,y) como de un plano cartesiano\*\*\*

La superficie de dibujo de un componente se recibe como parámetro del método ***paintComponent()*** de manera que para iniciar el proceso de dibujo solo debemos tomar dicho parámetro y convertirlo en un elemento de la subclase ***Graphics2D***: ***(Graphics2D hereda de Graphics)***



***Métodos básicos de dibujo(Greaphics2D):***

* ***setColor(Color.color)***:define el color con el que se van a trabajar todas las tareas de dibujo, este valor se mantiene hasta que el método sea invocado con un valor diferente
* ***drawLine(x1,y1,x2,y2):*** Dibuja una línea entre los puntos ***(x1,y1)*** y ***(x2,y2)***

******

***Dibujar cadenas de caracteres:***

* ***setFont(tipo):*** Recibe un obj de la clase Font que define el tipo de la letra, para crear un obj de la clase font se deben definir 3 caracteristicas, el nombre (Ej: “Arial”, “Tahoma” etc), el estilo (Font.BOLD, Font.PLAIN, Font.ITALIC etc) y el tamaño (12, 24, 32 ect)
* ***drawString(cadena,x,y):*** Escribe una cadena de caracteres en la superficie de dibujo comenzando en el punto **(x,y)** usa para la ejecución el tipo de letra definido en el método anterior

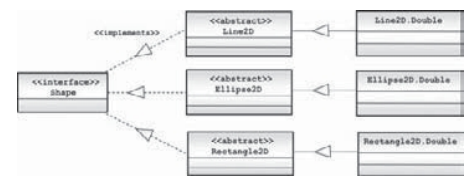
***Manejo de formas geométricas***

* ***draw(figura):*** Recibe como parámetro un objeto de una clase que implementa la interfaz **Shape** y dibuja su borde en la superficie
* ***fill(figura):*** Recibe como parámetro un objeto de una clase que implementa la interfaz **Shape** y dibuja la figura rellena en la superficie

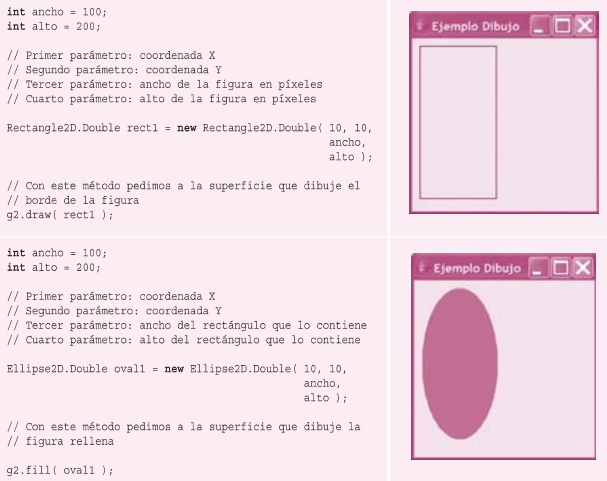
*La interfaz Shape se encuentra en el paquete java.awt y tiene dos métodos que nos interesan*

* ***contains(x,y):*** indica si el punto (x,y) se encuentra en el interior de una figura geométrica
* ***getPathIterator():*** Retorna la información geométrica necesaria para dibujar la figura. Su contenido exacto no nos interesa en este momento puesto que es la superficie la que debe ser capaz de interpretar la información para dibujar la figura

***Clases que implementan las figuras geométricas(java.awt.geom)***

****

Puesto que las clases que crean las figuras son clases abstractas, debemos crear instancias de las clases ***Line2d.Double***, ***Ellipse2D.Double*** y ***Rectangle2D.Double***



***Evolución del editor***

Extensiones del editor de dibujos para validar la facilidad de evolución del programa

Se seguirán los siguientes pasos:

1. Análisis de la modificación pedida
2. Identificar el punto del diseño que se quiere modificar y el impacto sobre el resto del programa
3. Diseñar la extensión, construyendo el diagrama de clases y asignando, entre todos los componentes, la responsabilidades que acaban de aparecer
4. Implementar y probar lo diseñado