Práctica 2
Gráficos Vectoriales con SVG
(versión 20.09.14)

Programación 3
Curso 2011-2012

Departamento de Lenguajes y Sistemas Informáticos
Universidad de Alicante

1. Introducción

En esta segunda práctica comenzaremos a preparar la implementación de una pequeña aplicación que, a partir de un fichero de texto de entrada que contendrá sencillas órdenes de dibujo, generará un nuevo fichero de texto con la especificación del gráfico a generar, utilizando para ello un subconjunto del lenguaje de gráficos vectoriales *Scalable Vector Graphics Tiny 1.2*[1], al cual nos referiremos de ahora en adelante como SVG. Lo interesante de este lenguaje de especificación de gráficos es que es un estándar del World Wide Web Consortium pensado para su uso en páginas web. Algunos navegadores web[2] entienden el lenguaje SVG, con lo cual los ficheros que generemos pueden visualizarse directamente en un navegador web.

El lenguaje SVG es en realidad una especificación XML[3] donde se define un lienzo bidimensional en el cual se pueden pintar figuras de diverso tipo. Más adelante en este enunciado puedes encontrar algún ejemplo. SVG soporta diferentes clases de objetos que se pueden pintar en el lienzo.

En esta primera versión de la aplicación el únicos tipo de figura que usaremos es el círculo y solamente crearemos el modelo de objetos. La lectura desde el fichero de texto y la escritura en SVG la realizaremos en entregas posteriores.

Dado que la complejidad de la aplicación va ir creciendo conforme vayamos avanzando, con el fin de asegurar una construcción robusta y evitar errores de regresión, vamos a ir desarrollando tests unitarios para cada uno de los méto-

[1] `http://www.w3.org/TR/SVGTiny12/`

[2] Por ejemplo, Firefox.

[3] *eXtensible Markup Language*, `http://www.w3c.es/divulgacion/guiasbreves/tecnologiasXML`

dos que vayamos escribiendo. Para ello nos vamos a apoyar en la herramienta *JUnit* [4] y su integración en el entorno de desarrollo *Eclipse* [5].

1.1. Objetivo

El cometido de esta práctica no es el desarrollo de una aplicación completa que cumpla unos requerimientos, sino la correcta implementación de una librería de objetos que nos servirá para la posterior creación de una aplicación de generación de gráficos SVG. Así, la comprobación del código escrito se debe realizar mediante tests unitarios *JUnit* en lugar del uso de *mains*.

2. El diagrama de clases

Como has podido leer en la introducción, hemos dicho que un gráfico SVG se compone de un *Lienzo*, que puede contener figuras (por ahora, *Círculos*). Las figuras se pueden posicionar en el lienzo y, como habrás visto en el manual de SVG, esto se hace especificando las *coordenadas* bidimensionales de, en este caso, el centro de un círculo.

Tenemos por tanto tres clases de objetos: *Lienzo, Coordenada*, y *Círculo*. Sabemos que el sistema de coordenadas es bidimensional. Sabemos también que un Círculo tiene un centro especificado como una coordenada y un radio que indica su tamaño. También sabemos que un Lienzo es bidimensional y que se puede especificar su altura y anchura, así como un título para el lienzo, pudiendo contener éste un número indeterminado de figuras.

Esto nos lleva a un diseño de clases para nuestra pequeña aplicación como el de la Figura 1.

Se pide implementar este diagrama de clases con el comportamiento de la aplicación que se especifica a continuación.

2.1. Interfaz de las clases

Se debe implementar el constructor de las clases *Coordenada*, *Círculo*, y *Lienzo* (ver Figura 1). Los métodos cuyo nombre comienza por *get* devuelven el valor del atributo al que se refieren (por ejemplo, el *getRadio() de* Circulo devuelve el radio del círculo); los métodos *set* almacenan en el atributo correspondiente el valor que se pasa como argumento. Estos métodos *set* deben validar el valor de su argumento. Por ejemplo, no se puede asignar un valor negativo al radio de un

[4] `www.junit.org`
[5] `www.eclipse.org`

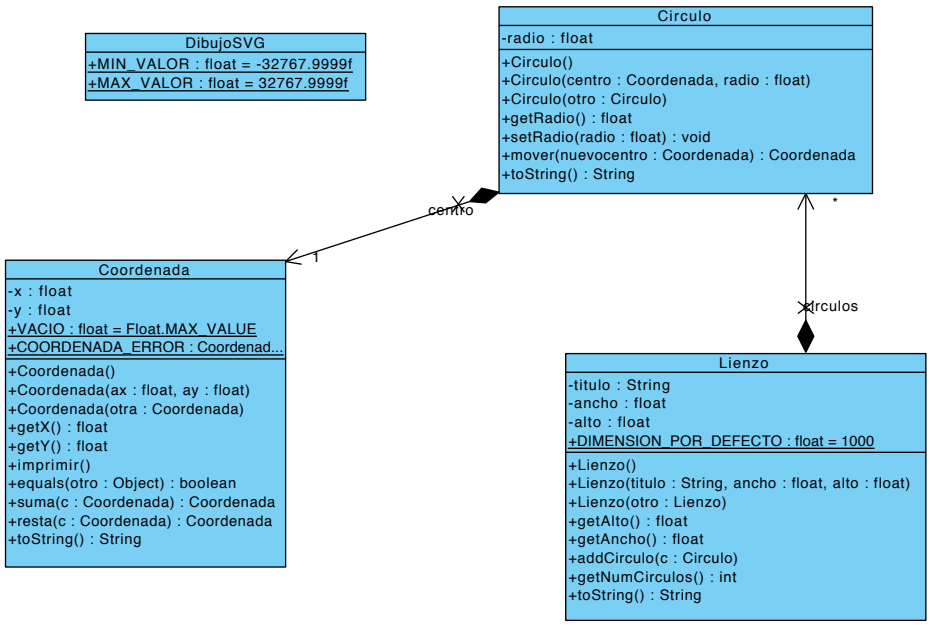

Figura 1: Diagrama de clases UML.

círculo (consultar la documentación de SVG para información sobre los rangos de valores válidos en cada caso). En caso de fallar la validación, el valor a utilizar será el mismo que se asigna en el constructor por defecto.

A continuación se indica brevemente el propósito de los restantes métodos a definir para cada clase.

2.2. DibujoSVG

En esta clase de utilidad se definen dos constantes estáticas que definen los límites de los valores de tipo real (tipo *number* en SVG[6]): *DibujoSVG.MIN_VALOR* y *DibujoSVG.MAX_VALOR*.

2.3. Coordenada

En esta práctica vamos a enriquecer el comportamiento de la clase Coordenada. En concreto se añaden los elementos que especificamos a continuación.

[6] `http://www.w3.org/TR/SVGMobile12/types.html#DataTypeNumber`

COORDENADA_ERROR La constante estática `COORDENADA_ERROR` es una instancia de una Coordenada creada con los valores por defecto y nos valdrá para operaciones de comparación posteriores.

bool equals(Coordenada c) Devuelve cierto si los atributos x e y del objeto c son iguales a los del objeto receptor.

Coordenada suma(Coordenada c) Devuelve una coordenada resultado de la suma de otras dos. Si el resultado no es una coordenada válida, devuelve una coordenada cuyos valores son iguales a `Coordenada.COORDENADA_ERROR`[7].

Coordenada resta(Coordenada c) Devuelve una coordenada resultado de la resta de otras dos. Si el resultado no es una coordenada válida, devuelve `Coordenada.COORDENADA_ERROR`.

String toString(), void imprimir() Devuelve una cadena con el formato: `x,y` Dado que el `imprimir()` de la anterior práctica usa ese mismo formato podemos evitar redundancias y hacer que `imprimir()` use este nuevo método.

Coordenada(float x, float y) Este constructor cambia respecto a la práctica anterior. Ahora comprobamos que la coordenada está dentro del rango admitido por SVG, `[DibujoSVG.MIN_VALOR .. DibujoSVG.MAX_VALOR]` (ambos inclusive). En el caso de que cualquiera de las coordenadas no esté en el rango, ambas se deben establecer a `Coordenada.VACIO` creándose una coordenada cuyo contenido es igual a `Coordenada.COORDENADA_ERROR`.

2.4. Círculo

Circulo() Un círculo vacío tiene radio 0.0 y una Coordenada con valores por defecto.

Circulo(Coordenada centro, float radio) En este método ten en cuenta que Circulo se asocia con Coordenada en forma de composición, y por tanto debe crear una copia del parámetro `centro`.

[7] Esta comprobación la puedes delegar en el propio constructor de Coordenada, que comprueba el rango de los valores recibidos e inicializa a `DibujoSVG.ERR_VALOR` si son incorrectos.

Circulo(Circulo otro) El constructor de copia debe crear una copia propia de la coordenada centro.

Coordenada getCentro() Retorna la coordenada `centro`

float getRadio() Retorna el radio del círculo

void setRadio(float r) Asigna `r` al `radio` sólo si no es negativo, en ese caso asigna 0.

Coordenada mover(Coordenada porigen) Cambia el origen de la figura al nuevo origen *porigen* (recuerda copiar la coordenada para asegurar la composición). Devuelve el antiguo origen.

String toString() Devuelve una cadena con formato: `x,y,r=radio`. Por ejemplo, el código `Circulo c = new Circulo(new Coordenada(10,20),30); System.out.println(c.toString());` imprime la cadena `10,20,30`.

2.5. Lienzo

Lienzo() El constructor por defecto asigna una cadena vacía al título y `DIMENSION_POR_DEFECTO` al ancho y al alto. Recuerda inicializar el vector de cículos.

Lienzo(String titulo, float ancho, float alto) La única particularidad de este constructor es que si cualquiera de las dos dimensiones es negativa, se asigna `DIMENSION_POR_DEFECTO` a ambas.

float getAlto(), float getAncho(), int getNumCirculos() Devuelven el valor de alto, ancho, y número de círculos respectivamente.

String toString() Compone la cadena con formato: `<titulo>de acho <ancho>por <alto>con <num círculos>figuras`. Por ejemplo, el código `Lienzo z = new Lienzo("Ejemplo", 200, 400); System.out.println(l.toString());` imprime la cadena `Ejemplo de 200 por 400 con 0 figuras`.

3. Pruebas unitarias

El alumno debe realizar pruebas unitarias personales con JUnit de cada uno de los métodos. Como ejemplo inicial puedes usar las pruebas incluidas en el autocorrector de la práctica 1.

4. Evaluación

Los criterios y métodos de evaluación son los mismos que los aplicados en la práctica anterior.

5. Entrega

Toda la estructura de directorios debe estar comprimida en un fichero llamado `prog3-2-11-12.tgz` que no supere los 500 KB en la que NO se deben incluir los tests unitarios. El *main* incluido en `mains.Main1` será el mismo que el de la práctica 1.

El plazo de entrega finaliza el día **18 de Octubre de 2011** a las **23:59h**.