

Dibujando en Java

Marco Antonio Gómez Martín

November 2, 2018

Contents

1	Introducción	1
2	Cómo se lanzan las aplicaciones	2
2.1	Posibles problemas de lanzamiento	2
3	Cómo programar un dibujo	2
4	Uso de las aplicaciones	3
5	Ejemplos de dibujos	4
5.1	Ejemplo simple	4
5.2	Cuadrado	5
5.3	Cuadrados anidados	5
6	Ejemplos de fractales	7
6.1	Curva de Koch	7
6.2	Curva de Peano	8

1 Introducción

Entre manos tienes un conjunto de aplicaciones para hacer dibujos simples utilizando líneas. Los dibujos son creados a partir de código Java que tiene las instrucciones de qué líneas pintar.

Las instrucciones que se pueden utilizar son muy limitadas y están inspiradas en la histórica *tortuga de Logo*. Dependiendo de la aplicación que se lance están disponibles más o menos instrucciones:

- **tortugaBasica**: dispone únicamente de dos instrucciones de desplazamiento básicas: una para hacer que la tortuga gire y otra para hacer que avance (pintando).
- **tortuga**: añade a la anterior la posibilidad de avanzar *sin pintar*.

2 Cómo se lanzan las aplicaciones

Las aplicaciones se lanzan con los ficheros `.bat` o `.sh` proporcionados (en caso de los `.sh` es posible que necesites añadirles los permisos de ejecución con `chmod +x *.sh`).

Todos esos *lanzadores* reciben cuatro parámetros: posición inicial de la tortuga (primero la X y luego la Y), rotación inicial (0 hacia la derecha, 90 hacia abajo) y, por último, el nombre del fichero que tiene el código fuente de pintado.

Ejemplo:

```
$ ./tortugaBasica.sh 10 20 0 cuadro.java
```

Lanza la aplicación de la tortuga básica colocando la tortuga en la posición (10, 20), "encarada" hacia la derecha y que ejecutará el fichero `cuadro.java`.

Las dimensiones se miden en píxeles (el tamaño del área de dibujo es 512x512) y la rotación en grados.

2.1 Posibles problemas de lanzamiento

Para que la aplicación funcione, el compilador de Java debe estar disponible. Para eso, es necesario que el sistema tenga instalado el JDK y accesible desde el PATH.

Por otro lado, existe un problema con las librerías externas utilizadas por la aplicación que impiden que ésta se lance si la versión de Java instalada es "un número entero" (es decir, no tiene la forma X.Y). Si ese es tu caso, tendrás que actualizar Java confiando en que al subir de versión se añada una *minor version*.

3 Cómo programar un dibujo

El fichero con el código fuente utiliza Java como lenguaje. Sin embargo, *no* se espera un fichero `.java` completo, sino únicamente uno que defina el método `paint` (público, que no devuelva nada ni tenga parámetros):

```
public void paint() {
    // ...
}
```

Dentro de ese código se puede utilizar cualquier cosa de Java (eso sí, si está dentro de un paquete, deberás poner el nombre completo de la clase, añadiendo el paquete). Además, para el manejo de la tortuga están disponibles las siguientes funciones:

- **rotate(grados)**: rota la "tortuga" los grados indicados. Un número positivo la hace girar en sentido de las agujas del reloj.
- **advanceAndPaint(1)**: avanza la tortuga en la dirección en la que está haciendo que ésta pinte una línea. El parámetro (**double**) indica la distancia (en píxeles) que avanzará.
- **getHeight()** y **getWidth()**: devuelven el número de píxeles de alto y ancho del área de dibujado.
- **setGrayLevelColor(float01)**: establece el color con el que se pintará, en escala de grises (0 equivale a negro, 1 a blanco).
- **setColor(r, g, b)**: igual que la anterior, pero definiendo cada componente de color. Los valores también están normalizados entre 0 y 1.

En la aplicación **tortuga**, además, existe una función adicional:

- **advance(1)**: avanza la tortuga en la dirección actual sin pintar en el trayecto. El parámetro (**double**) indica la distancia (en píxeles) que avanzará.

Es importante hacer notar que el código puede incluir la definición de otros métodos, e incluso de atributos.

El nombre del fichero puede ser cualquiera; de hecho tampoco es necesario que la extensión sea **.java**. La ventaja de utilizar esa extensión es que si utilizas un editor con resaltado de sintaxis,

4 Uso de las aplicaciones

Las dos aplicaciones funcionan de forma similar. Al lanzarse, crean una ventana, cargan el fichero de código, lo compilan, y lo ejecutan.

En caso de errores de compilación, éstos se muestran por la salida estándar.

El uso de la aplicación es sencillo. Una vez lanzada, se pueden utilizar las siguientes teclas:

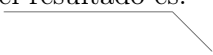
- R: Recarga el fichero de código fuente. Esto permite realizar pruebas de dibujos sin necesidad de cerrar la aplicación y volverla a lanzar.
- Q: Sale de la aplicación.
- S: Graba un fichero un `.png` con el fotograma actual. En la salida estándar aparece el nombre del fichero utilizado, que tendrá la forma `frameXXXX.png`, donde el `XXXX` es un número que se va incrementando. La aplicación intenta *no* sobrescribir ningún fichero, por lo que si en el directorio al lanzar la aplicación ya había ficheros de ejecuciones anteriores, utilizará el primer número disponible.

5 Ejemplos de dibujos

5.1 Ejemplo simple

```
public void paint() {  
    advanceAndPaint(300);  
    rotate(45);  
    advanceAndPaint(100);  
}
```

Cuando se lanza con
`./tortugaBasica.sh 10 10 0 simple.java`
el resultado es:



5.2 Cuadrado

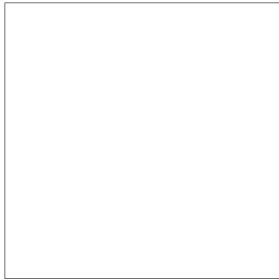
Un ejemplo de dibujo que utiliza métodos auxiliares:

```
public void paint() {
    rectangulo(getWidth() - 20, getHeight() - 20);
}

void rectangulo(double tamX, double tamY) {
    advanceAndPaint(tamX);
    rotate(90);
    advanceAndPaint(tamY);
    rotate(90);
    advanceAndPaint(tamX);
    rotate(90);
    advanceAndPaint(tamY);
    rotate(90);
}
```

Está pensado para lanzarse con los mismos parámetros que la anterior. En este caso el código reaccionaría correctamente a los cambios en el tamaño de la ventana.

El resultado es:



5.3 Cuadrados anidados

Utilizando la aplicación de la tortuga básica se pueden hacer numerosos dibujos. Sin embargo tiene una restricción importante: deben poder pintarse "con un único trazo", pues la tortuga no es capaz de avanzar sin pintar.

Para dibujos en los que se requiera levantar, hay que utilizar la aplicación *tortuga* en la que se puede utilizar *advance*. Por ejemplo:

```
// Tortuga colocada en el centro del rectángulo
```

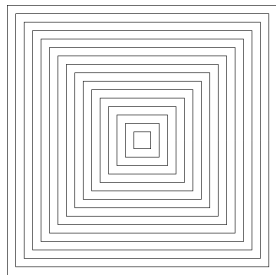
```
// que se pintará con el tamaño indicado.
// Orientación de entrada (y salida) 0
void rectangulo(double tx, double ty) {
    advance(tx/2);
    rotate(90);
    advanceAndPaint(ty/2);
    rotate(90);
    advanceAndPaint(tx);
    rotate(90);
    advanceAndPaint(ty);
    rotate(90);
    advanceAndPaint(tx);
    rotate(90);
    advanceAndPaint(ty/2);
    rotate(90);
    advance(tx/2);
    rotate(180);
}

public void paint() {

    int tamIni = 30;
    int incr = 30;

    for (int t = tamIni; t + incr < Math.min(getHeight(), getWidth()); t += incr)
        rectangulo(t, t);
}
```

Puede llamarse con
`./tortuga.sh 256 256 0 anidados.java`
 para conseguir



6 Ejemplos de fractales

(Material *copiado* de <http://www.sc.ehu.es/sbweb/fisica/cursoJava/numerico/recursivo/recursivo.htm>)

El término fractal proviene de la palabra latina "fractus" y que se acuñó a finales de la década de los 70 por el matemático Benoit Mandelbrot.

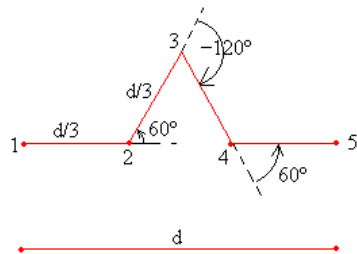
Un fractal consta de fragmentos geométricos de orientación y tamaño variable, pero de aspecto similar. Si lo ampliamos nos irá mostrando una serie repetitiva de niveles de detalle, de modo que a todas las escalas que se examine, la estructura presentada será similar.

Para representar un fractal basta con crear una rutina que tome una forma geométrica simple y la dibuje a una determinada escala. Se repite varias veces la llamada a esta rutina de forma recursiva y a diferentes escalas.

6.1 Curva de Koch

La curva de Koch fue una de las primeras curvas fractales en ser descrita (apareció en 1906). Con tres de estas curvas dispuestas en forma de triángulo equilátero se construye el famoso *copo de nieve de Koch*.

La curva de Koch comienza con un segmento de longitud l . La recursión sustituye ese segmento por cuatro segmentos, cada uno de ellos de un tercio de la longitud del anterior:



El código necesario para conseguir esta curva es:

```
public static final int nivel = 3;

public void paint() {
    double dist = getWidth() - 20;
    generaKoch(nivel, dist);
}
```

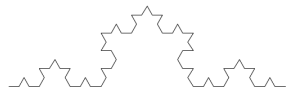
```

void generaKoch(int n, double l) {
    if (n == 0) {
        advanceAndPaint(l);
        return;
    }

    generaKoch(n-1, l/3);
    rotate(-60);
    generaKoch(n-1, l/3);
    rotate(120);
    generaKoch(n-1, l/3);
    rotate(-60);
    generaKoch(n-1, l/3);
}

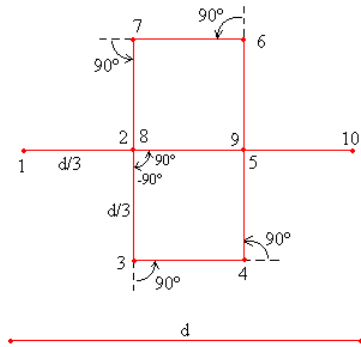
```

Que al ser ejecutado con
`./tortugaBasica.sh 10 500 0 koch.java`
 tiene como resultado lo siguiente:



6.2 Curva de Peano

La primera vez que se ve la definición de la curva de Peano sorprende por su resultado. Igual que la anterior, la recursión divide cada segmento original en tres partes de igual longitud. El segmento central es sustituido por dos cuadrados:



En nuestra aplicación el código es:

```
public static final int nivel = 3;
```

```
public void paint() {
    double dist = getWidth() - 20;
    generaPeano(nivel, dist);
}
```

```
void generaPeano(int n, double l) {
    if (n == 0) {
        advanceAndPaint(l);
        return;
    }
```

```
    double base = l / 3;
    generaPeano(n-1, base);
    rotate(90);
    generaPeano(n-1, base);
    rotate(-90);
    generaPeano(n-1, base);
    rotate(-90);
    generaPeano(n-1, base);
    rotate(-90);
    generaPeano(n-1, base);
    rotate(90);
    generaPeano(n-1, base);
    rotate(90);
```

```
generaPeano(n-1, base);  
rotate(90);  
generaPeano(n-1, base);  
rotate(-90);  
generaPeano(n-1, base);  
}
```

Que al ser ejecutado con
./tortugaBasica.sh 10 256 0 peano.java
da como resultado:

