4 | Tortugas

Manuel Alcántara Juárez
Verónica Esther Arriola Ríos

Meta

Que el alumno aplique el uso de estructuras de control para resolver problemas sencillos.

Objetivos

Al finalizar la práctica el alumno será capaz de:

1. Utilizar referencias a objetos para enviarles mensajes.
2. Programar funciones utilizando objetos y estructuras de control.
3. Ver los resultados de su trabajo en una interfaz de usuario.

Antecedentes

Objetos

`Java` además de tipos primitivos nos permite definir tipos más complejos que encapsulan conjuntos de datos de varios tipos. Estos tipos complejos se definen en *clases*. Los ejemplares creados a partir de cada clase son los *objetos*. Considera que el análogo a una clase es el nombre del tipo primitivo, por ejemplo: el análogo al tipo `int` podría ser el tipo `Tortuga`. Observa la diferencia en la convención de nombrado: los tipos primitivos tienen nombres con puras minúsculas, las clases tienen nombres que comienzan con mayúsculas y siguen el fomato camello (*Camel Case*): cada palabra comienza con mayúscula y el resto se escribe en minúsculas, dando la impresión de jorobas de camello. Continuando con el ejemplo, entonces el análogo al entero `5` podría ser una *tortuga verde*.

Valores y referencias

Una diferencia muy importante entre los valores de tipos primitivos y los objetos es que, cuando se copian valores primitivos, los valores realmente se duplican.

Ejemplo 4.1. *Al crear un par de variables tipo* `int` *y asignar una a la otra, cada una tiene su propia copia del valor almacenado.*

Listado 4.1: Ejemplo.java

```
1  public class Ejemplo {
2    public static void main(String[] args) {
3      int x = 4;                  // Declara y asigna x
4      System.out.println(x);
5      int y = x;                  // Declara y asigna y al mismo valor
           ↪ que x
6      System.out.println(y);
7      x = -1;                     // Asigna un nuevo valor a x
8      System.out.println(x);
9      System.out.println(y);
10   }
11 }
```

El resultado de ejecutar el código anterior sería el siguiente:

```
4
4
-1
4
```

Simplemente y *contenía una copia del entero* 4*, de modo que al cambiar el valor de* x*,* y *conservó su propia copia.*

Ejemplo 4.2. *Ahora veamos lo que sucedería con los objetos si tuviéramos una clase llamada* `Entero`*:*

Listado 4.2: Ejemplo.java

```
1  public class Ejemplo {
2    public static void main(String[] args) {
3      Entero x = new Entero(4);  // Declara la variable y crea un
           ↪ objeto
4      System.out.println(x);
5      Entero y = x;              // Declara y asigna y al mismo valor
           ↪ que x
6      System.out.println(y);
7      x.valorNuevo(-1);          // Modifica el valor dentro del
           ↪ Entero
8      System.out.println(x);
9      System.out.println(y);
```

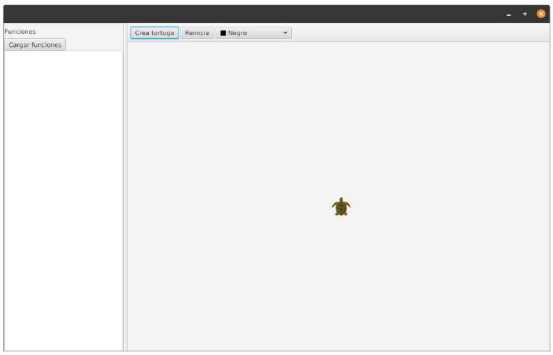

Figura 4.1 Una tortuga aparece en el centro de la pantalla tras presionar el botón `Crea tortuga`.

```
10    }
11 }
```

Ahora obtenemos:

```
4
4
-1
-1
```

Lo que sucede en este caso es que las variables con tipos clase en `Java` no contienen a los objetos directamente, si no que contienen la *dirección* del área de la memoria donde fueron almacenados. A esto le llamaremos un *referencia*. Entonces, la variable `y` almacenó una copia de la misma dirección que tenía almacenada `x`, es decir, la dirección del `Entero` con valor `4`. Cuando `x` utilizó esa dirección para pedirle al entero que modificara el número que tenía almacenado, `y` también pudo ver el cambio, porque ambas variables tienen la dirección del mismo objeto.

En esta práctica estarás trabajando con referencias a un objeto de tipo `Tortuga` y las usarás para enviarle instrucciones sobre lo que debe hacer.

Desarrollo

La interfaz de usuario

Actividad 4.1

En una terminal entra al directorio `tortugafx` y ejecuta el comando[a]:

```
mvn clean javafx:run
```

Deberás ver una ventana como la mostrada en la Figura 4.1 pero sin tortuga. Presiona el botón `Crea tortuga` y la verás aparecer.

Para jugar con ella un poco presiona el botón `Cargar funciones`. Esto hará que aparezcan una serie de botones con movimientos básicos. Velos presionando y observa lo que puede hacer nuestra tortuga.

Después de haberla movido observa que la puedes dejar quieta, crear otra y mover ahora a la nueva tortuga.

[a]Para este paso asumimos que tienes *Maven* instalado, si no es así usa el comando de instalación correspondiente a tu distribución de linux. En Ubuntu este es `sudo apt install maven`.

Proyecto Tortuga

Trabajar con interfaces gráficas de usuario es sumamente laborioso y te requerirá todo el curso llegar a crear una interfaz de usuario como esta desde cero, puede que incluso requieras aún algo más de práctica en cursos futuros. Por ello si intentas mirar los archivos que se te entregan es posible que experimentes un poco de vértigo. No te preocupes, todo se irá aclarando.

Por el momento, el archivo que debes abrir es `ProyectoTortuga.java`, éste es bastante amigable, se encuentra en el directorio:

`tortugafx/src/main/java/ciencias/icc/tortugafx`

y el Listado 4.3 lo muestra en resumen.

Listado 4.3: ProyectoTortuga.java

```
1  package ciencias.icc.tortugafx;
2
3  import java.util.logging.Level;
4  import java.util.logging.Logger;
5  import ciencias.icc.tortugafx.figuras.FunciónFigura;
6
7  import javafx.event.ActionEvent;
8  import javafx.scene.paint.Color;
9
10 /**
11  * Controlador con funciones específicas para que la tortuga las
      ↪ ejecute.
12  * @author blackzafiro
13  */
14 public class ProyectoTortuga extends ControladorTortuga {
15
16   protected final static Logger LOGGER = Logger.getLogger("ciencias.
```

```
      ↪ icc.tortugafx.ProyectoTortuga");
17  static { LOGGER.setLevel(Level.FINE); }
18
19  @FunciónFigura(nombre = "Avanza")
20  public void avanza(ActionEvent t) {
21    LOGGER.log(Level.FINE, "Avanza␣");
22    Tortuga tortuga = dameTortugaActiva();
23    tortuga.avanza(50);
24  }
25
26  @FunciónFigura(nombre = "Rota␣a␣la␣derecha")
27  public void rotaDerecha(ActionEvent t) {
28    LOGGER.log(Level.FINE, "Rota␣a␣la␣derecha␣");
29    Tortuga tortuga = dameTortugaActiva();
30    tortuga.rota(-10);
31  }
32
33  //
34  // Inserta a partir de aquí otras funciones
35  //
36 }
```

Esto es lo que significa cada sección:

- Primero se declara el paquete al que pertenece este archivo, es algo así como los apellidos de nuestra aplicación.
- Todas las líneas `import` indican que estamos invitando a varios tipos clase programados por otras personas y que tienen otros apellidos.
- La interfaz de usuario crea un objeto de tipo `ProyectoTortuga` y le envía mensajes cada vez que el usuario presiona los botones, aquí se programarán los que responden a los botones que aparecen en el panel de la izquierda.
- Hay otros elementos en la interfaz que responden al usuario, como el menú para cambiar el color de la pluma con que dibuja la tortuga. `ControladorTortuga` se encarga de los asuntos complicados.
- El `LOGGER` es una herramienta muy útil para imprimir mensajes en pantalla cuando necesitamos ver qué está haciendo nuestro programa, puedes ver ejemplos de su uso en las líneas 21 y 28. Pero si el programa ya está funcionando no queremos seguir viendo esos mensajes. Es muy sencillo apagarlos cambiando el nivel de detalle con que se reportan. Para ello, en la línea 17, se puede cambiar `Level.FINE` por `Level.OFF`.
- Finalmente se encuentran las funciones que serán llamadas cuando los botones son presionados: `avanza` y `rotaDerecha`. El archivo original tienen aún más. Tu trabajo será agregar otras, así que lee estas con cuidado.

@FunciónFigura

Los métodos con la forma:

```
<acceso>         ::= public|private|protected|∅
<modificador>    ::= final|∅
<identificador>  ::= (<letra>|_) (<letra>|<dígito>|_)*

<parámetro>      ::= <tipo><identificador>
<parámetros>     ::= <parámetro>(,<parámetros>)*|∅

<función>        ::=
<acceso> <modificador> <tipo> <identificador>(<parámetros>) {
  <implementación>
}
```

es decir, que no incluyen `static`, son métodos típicos para objetos.

Estos de aquí tienen una pieza extra. En la parte de arriba está escrito:

```
@FunciónFigura(nombre="un texto").
```

Esto no es estándar de `Java`. Se llaman anotaciones y la anotación `@FunciónFigura` es otro elemento definido dentro de este programa. Sirve para indicarle a la interfaz que queremos que agregue más botones en el menú izquierdo con nuevas habilidades para la tortuga. Lo usarás para agregar más botones con cosas qué hacer.

Puedes agregar más segmentos de código con el mismo formato, sólo cambia el nombre de la función y dale un nombre adecuado. Es muy importante que este código quede *dentro de la clase*. Es decir, entre la llave `{` que abre al final de `ControladorTortuga` y la que cierra `}`, al final del archivo y fuera del código de cualquier otro método.

dameTortugaActiva()

Otro elmento que es importante explicar es lo que hace la llamada a la función `dameTortugaActiva()`. Esta devuelve la dirección del objeto tortuga que está en la pantalla. Si no había una tortuga la crea y devuelve su dirección. Por ello, si haces click sobre los botones para dibujar figuras sin haber creado primero una tortuga, aparece una tortuga nueva junto con la figura que pediste. Necesitarás usar esta función para acceder a la tortuga y darle instrucciones.

Cuando se declara una variable de tipo tortuga en el interior de uno de estos métodos, como en las líneas 22 y 29, obtenemos una variable tipo referencia lista para almacenar la dirección de una tortuga. Al asignarle el valor devuelto por `dameTortugaActiva()` la dirección de la tortuga queda almacenada en ella.

Puedes utilizar el código siguiente dentro de cualquier método que crees:

```
Tortuga t;                // Declaración de una variable
t = dameTortugaActiva() // Se le asigna valor
```

A partir de ese momento, podemos utilizar la variable `t` para enviarle mensajes a la tortuga. Tu variable puede tener el nombre que quieras, pero es buena práctica que el nombre sea representativo del objeto al que apunta.

```
t.avanza(-20);
t.rota(-45);
t.plumaAbajo(false);
t.asignaColor(Color.RED);
```

Los mensajes que le puedes enviar son:

- `void plumaAbajo(boolean estaAbajo)`
- `void avanza(double distancia)`
- `void rota(double grados)`
- `void asignaColor(Color color)`

Puedes revisar la documentación en el archivo `Tortuga.java`, sin embargo puede resultarte más sencillo si generas la versión `html`. Para ello invoca:

```
mvn javadoc:javadoc
```

Encontrarás los archivos en el directorio:

`target/site/apidocs`.

Si quieres ver qué colores puedes pasar revisa la documentación en `Color`.

Ejercicios

1. Crea una función en la que dibujes cualquier cosa en la pantalla utilizando todas las funciones que tiene la tortuga.
2. Crea una función que dibuje un **cuadrado** en la posición donde se encuentre la tortuga. Es obligatorio que esta función utilice un ciclo para lograrlo. No se vale que dibujes a mano los cuatro lados.
3. Crea una función que dibuje un círculo. La forma tradicional de resolver esto es avanzar una distancia pequeña y rotar un grado tantas veces como sea necesario hasta completar los 360°.

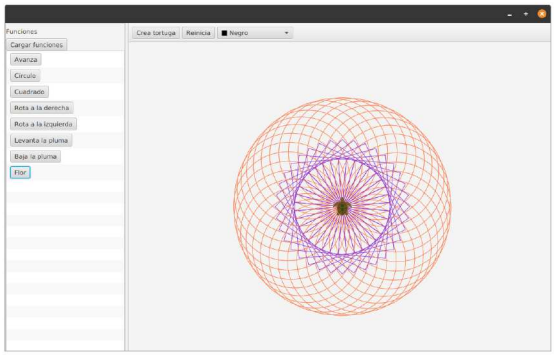

Figura 4.2 Flor hecha con círculos y cuadrados.

a. b.

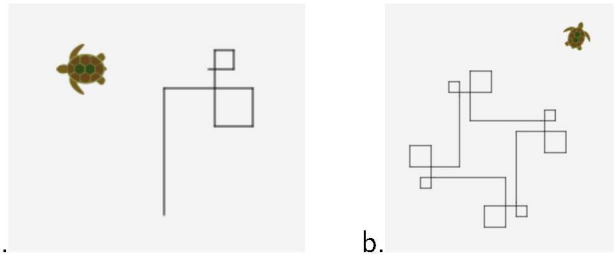

Figura 4.3 Patrones. Se levantó la pluma y se desplazó a la tortuga de su posición final para mostrar mejor la figura.

4. Observa que, una vez programada una función, la puedes llamar desde otra, sólo pasa como parámetro el mismo `ActionEvent t` que recibiste. Utiliza las funciones anteriores para obtener algo como lo mostrado en la Figura 4.2. No tiene que ser idéntica, pero debe tener las mismas características: se debe cerrar el círculo y elaborarla debe requerir al menos dos figuras base.

5. Dibuja el patrón de la Figura 4.3 a. y luego repítelo para obtener b.

Entregables

El único archivo que debes modificar es `ProyectoTortuga.java`, al ejecutar tu programa podremos ver los botones con las funciones que agregaste en cada uno de los puntos de la sección de ejercicios.