Práctica 11: POO en Scala (2)

Ejercicio 1

Modifica el código del ejemplo `SerpientesEscaleras` (en el tema 7) para que se pueda jugar con `n` jugadores. Modifica también los métodos que necesites del *trait* `PresentadorJuegoTablero` y/o de la clase `PresentadorTexto` para que se muestre la evolución de la nueva versión del juego. El juego termina cuando un jugador llega a la meta.

En este ejercicio no hay que hacer pruebas con `assert`. Los mensajes que se muestran por pantalla serán similares a estos:

```
Nueva tirada del jugador 1: 3
Casilla del jugador 1: 11
Nueva tirada del jugador 2: 6
Casilla del jugador 2: 20
...
Final del juego: ha ganado el jugador 5
```

Ejercicio 2

Implementa una versión genérica (con un tipo genérico `T`) de la clase `Tree` del ejercicio 3 de la práctica 10. Modifica las definiciones de los métodos para que funcionen correctamente, añadiendo algún parámetro adicional si lo consideras oportuno.

Añade la definición de un método `filtra` que en el que se pasa un predicado y se devuelve la lista de los nodos que lo cumplan.

Define pruebas para todos los métodos.

Ejercicio 3

a) Define, implementa y prueba las clases `Punto2D`, `Figura` (abstracta), `Rectangulo` y `Circulo`, de forma que las podamos probar de la siguiente manera (la función `iguales(Double, Double)` está definida en la práctica 10, y la explicación del `equals` al final del tema 7).

Todas las figuras están definidas en el plano 2D. El `Rectangulo` se crea a partir de dos puntos: el punto de su esquina inferior izquierda y el de la esquina superior derecha. El `Circulo` se crea a partir del punto de su centro y de su radio.

```
def pi = 3.1416
```

```
def sumaAreas(figuras:List[Figura]): Double = {
    if (figuras.isEmpty) 0.0
    else figuras.head.area + sumaAreas(figuras.tail)
}

val p1 = new Punto2D(3.0, 5.0)
val p2 = new Punto2D(-1.0, -1.0)
val c = new Circulo(p1, 2.0)
val r = new Rectangulo(p2, p1)

assert(p1.equals(new Punto2D(3.0, 5.0)))
assert(!p1.equals(null))
assert(iguales(r.area, 24.0))
assert(iguales(c.area, 12.5664))
val figuras = List(c,r)
assert(iguales(sumaAreas(figuras), 36.5664))
```

Añade alguna prueba adicional.

b) Añade en la clase `Figura` el siguiente método abstracto, impleméntalo en las clases hijas y pruébalo:

- `intersectaMismoTipo(otra: Figura): Boolean` : Devuelve `true` si la figura del parámetro es del mismo tipo e intersecta el objeto que ejecuta el método.

c) Define las funciones y pruébalas:

- `cuentaTipos(figuras: List[Figura]): (Int, Int)` : Devuelve una tupla con el número de círculos y rectángulos que hay en la lista.
- `intersectanMismoTipo(figuras: List[Figura], figura: Figura): List[Figura]` : Devuelve las figuras de la lista que son del mismo tipo e intersectan con una figura dada.

d) Define en la clase `Rectangulo` el siguiente método y añade algunas pruebas. La explicación de `Option` está al final del tema 7.

- `interseccion(otro: Rectangulo): Option[Rectangulo]` : Calcula la intersección del objeto que ejecuta el método con otro rectángulo y devuelve `Some(Rectangulo)` con el resultado de la intersección o `None` si no existe intersección.

Ejercicio 4

Supongamos las siguientes definiciones de clases y traits:

```
class Clase1 {
    def g(x:Int, y:Int) = x+y
}
```

```
trait Trait1 {
    def h(x:Int, y:Int) = x*y
}

class Clase2 extends Clase1 with Trait1 {
    override def g(x:Int, y:Int) = x+y+100
    def f(x:Int) = x+30
}

class Clase3 {
    def g(x:Int, y:Int) = x+y+200
}

trait Trait2 extends Clase1 {
    abstract override def g(x:Int, y:Int) = super.g(x,y+10)
}
```

a) Indica cuáles de las siguientes definiciones son incorrectas y explica por qué (explícalo con un comentario en el mismo fichero `practica-11.scala` en el que entregues la práctica):

```
val a = new Clase2
val b = new Clase3 with Trait2
val c = new Clase2 with Trait2
val d: Clase1 = new Clase2
```

b) En las siguientes expresiones tacha las que correspondan a las expresiones incorrectas anteriores. Con las expresiones no tachadas, índica qué valor devuelven o si dan un error y explica por qué devuelven ese valor o por qué resultan en un error.

```
a.h(2,4)
a.g(2,4)
b.g(2,4)
b.h(2,4)
c.g(2,4)
c.h(2,4)
d.f(2)
```

Ejercicio 5

Supongamos la siguiente definición de las clases `Ping` y `Pong`:

```
import scala.actors.Actor
import scala.actors.Actor._

class Ping(count: Int, pong: Actor) extends Actor {
    def act() {
        for(i <- 1 to count) {
```

```
            println("Ping envía Ping")
            pong ! "Ping"
            }
        pong ! "Stop"
        }
}

class Pong extends Actor {
    def act() {
        receive {
            case "Ping" =>
                println("Recibido mensaje")
                case "Stop" =>
                println("Pong se para")
                exit()
            }
    }
}
```

Sin ejecutarlo en el intérprete di qué va a aparecer por pantalla al ejecutar las siguientes instrucciones y explica por qué (escríbelo entre comentarios en la práctica)

```
val pong = new Pong
val ping = new Ping(10,pong)
ping.start
pong.start
```
