

# PROGRAMACIÓN ORIENTADA A OBJETOS


## Construcción. Clases y objetos.

### 2021-2

### Laboratorio 1/6

## OBJETIVOS

Desarrollar competencias básicas para:

1. Apropiar un paquete revisando: diagrama de clases, documentación y código.
2. Crear y manipular un objeto. Extender y crear una clase.
3. Entender el comportamiento básico de memoria en la programación OO.
4. Investigar clases y métodos en el API de java<sup>1</sup>.
5. Utilizar el entorno de desarrollo de BlueJ
6. Vivenciar las prácticas XP : *Planning*  The project is divided into [iterations](#).

*Coding*  All production code is [pair programmed](#).

## ENTREGA

- ➔ Incluyan en un archivo **.zip** los archivos correspondientes al laboratorio. El nombre debe ser los dos apellidos de los miembros del equipo ordenados alfabéticamente.
- ➔ Deben publicar el avance al final de la sesión y la versión definitiva en la fecha indicada en los espacios correspondientes.

## SHAPES

### Conociendo el proyecto shapes

[En lab01.doc] [TP 20]

1. El proyecto “shapes” es una versión modificada de un recurso ofrecido por BlueJ. Para trabajar con él, bajen `shapes.zip` y ábralo en BlueJ. Capturen la pantalla.
2. El **diagrama de clases** permite visualizar las clases de un artefacto software y las relaciones entre ellas. Considerando el diagrama de clases de “shapes” (a) ¿Qué clases ofrece? (b) ¿Qué relaciones existen entre ellas?
3. La **documentación**<sup>2</sup> presenta las clases del proyecto y, en este caso, la especificación de sus componentes públicos. De acuerdo con la documentación generada: (a) ¿Qué clases tiene el paquete `shapes`? (b) ¿Qué atributos ofrece la clase `Triangle`? (c) ¿Cuántos métodos ofrece la clase `Triangle`? (d) ¿Cuáles métodos ofrece la clase `Triangle` para que la figura cambie (incluya sólo el nombre)?
4. En el **código** de cada clase está el detalle de la implementación. Revisen el código de la clase `Triangle`. Con respecto a los atributos: (a) ¿Cuántos atributos realmente tiene? (b) ¿Cuáles atributos describen la forma de la figura?. Con respecto a los métodos: (c) ¿Cuántos métodos tiene en total? (d) ¿Quiénes usan los métodos privados?
5. Comparando la **documentación** con el **código** (a) ¿Qué no se ve en la documentación? (b) ¿por qué debe ser así?
6. En el código de la clase `Triangle`, revise el atributo `VERTICES` que almacena el número de vértices de los triángulos (a) ¿Qué significa que sea `public`? (b) ¿Qué significa que sea `static`? (c) ¿De qué tipo de dato debería ser? Actualícelo. (d) ¿Qué hacemos si queremos que no se pueda modificar? Actualícelo.

---

1 <http://docs.oracle.com/javase/8/docs/api/>  
2 Menu: Tools-Project Documentation

- En el código de la clase `Triangle` revisen el detalle del tipo de los atributos `height` y `width` (a) ¿Qué se está indicando al decir que es `int`? (b) Si sabemos todos nuestros triángulos van a tener poca altura (menor a 100), ¿de que tipo deberían ser este atributo? (c) Si algunos de nuestros triángulos pueden ser muy anchos (base mayor a 220000000), ¿de que tipo deberían ser este atributo? (d) ¿qué restricción adicional tendrían estos atributos? Refactoricen el código y expliquen claramente sus respuestas.
- ¿Cuál dirían es el propósito del proyecto “shapes”?

### Manipulando objetos. Usando un objeto.

[En lab01.doc] [TP 8 ]

- Crean un objeto de cada una de las clases que lo permitan<sup>3</sup>. (a) ¿Cuántas clases hay? ¿Cuántos objetos crearon? (b) ¿Por qué?
- Inspeccionen el **estado** del objeto `:Triangle`<sup>4</sup>, ¿Cuáles son los valores de inicio de todos sus atributos? Capture la pantalla.
- Inspeccionen el **comportamiento** que ofrece el objeto `:Triangle`<sup>5</sup>. (a) Capturen la pantalla. (b) ¿Por qué no aparecen todos los que están en el código?
- Construyan, con “shapes” sin escribir código, una propuesta de la imagen del logo de su equipo favorito. (a) ¿Cuántas y cuáles clases se necesitan? (b) ¿Cuántos objetos se usan en total? (c) Capturen la pantalla. (d) Incluyan el logo original.

### Manipulando objetos. Analizando y escribiendo código.

[En lab01.doc] [TP 30]

```

Rectangle yellow;
Rectangle blue;
Rectangle red;
//1
yellow= new Rectangle();
blue= new Rectangle();
red= yellow;
yellow.makeVisible();
//2
yellow.changeSize(30,80);
yellow.changeColor("yellow");
//3

blue = new Rectangle();
blue.changeSize(20,80);
blue.changeColor("blue");
blue.moveVertical(30);
//4
red.changeColor("red");
red.changeSize(20,80);
red.moveVertical(50);
red.makeVisible();
//5
blue.makeVisible();
//6

```

- Lean el código anterior. (a) ¿cuál es la figura resultante? (b) Píntenla.
- Habiliten la ventana de código en línea<sup>6</sup>, escriban el código. Para cada punto señalado indiquen: (a) ¿cuántas variables existen? (b) ¿cuántos objetos existen? (c) ¿qué color tiene cada uno de ellos? (d) ¿cuántos objetos se ven? Al final, (e) Expliquen sus respuestas. (f) Capturen la pantalla.
- Compare figura pintada en 1. con la figura capturada en 2. , (a) ¿son iguales? (b) ¿por qué?

3 Clic derecho sobre la clase

4 Clic derecho sobre el objeto

5 Hacer clic derecho sobre el objeto.

6 Menú. View-Show Code Pad.


## Extendiendo una clase. Triangle.

[En lab01.doc y \*.java]

1. Desarrollen en `Triangle` el método `duplicate()` (duplica su área) . ¡Pruébenlo! Capturen dos pantallas.
2. Desarrollen en `Triangle` el método `perimeter()` . ¡Pruébenlo! Capturen una pantalla.
3. Desarrollen en `Triangle` el método `walk(int steps)` (camina sobre sus lados dando el número de pasos definido) . ¡Pruébenlo! Capturen dos pantallas.
4. Propongan un nuevo método para esta clase.
5. Generen nuevamente la documentación y revise la información de estos nuevos métodos. Capturen la pantalla.

## Codificando una nueva clase. Digit.

[En lab01.doc. Digit.java]

	<table><tr><th>Digit</th></tr><tr><td>+ _(digit : byte) : Digit + get() : byte + next() : void + change(digit : byte) : void + change() : void + moveTo(x : int, y : int) : void + changeColor(color : String) : void + makeVisible() : void + makeInvisible() : void</td></tr></table>	Digit	+ _(digit : byte) : Digit + get() : byte + next() : void + change(digit : byte) : void + change() : void + moveTo(x : int, y : int) : void + changeColor(color : String) : void + makeVisible() : void + makeInvisible() : void	<p><b>Iterativo - Incremental</b></p> <p><b>Miniciclos</b></p> <ol style="list-style-type: none"><li>1. _(digit) get ()</li><li>2. next () change (digit) change ()</li><li>3. makeVisible () makeInvisible ()</li><li>4. moveTo (x,y) changeColor ()</li></ol>
Digit				
+ _(digit : byte) : Digit + get() : byte + next() : void + change(digit : byte) : void + change() : void + moveTo(x : int, y : int) : void + changeColor(color : String) : void + makeVisible() : void + makeInvisible() : void				

1. Revisen el diseño y clasifiquen los métodos en: constructores, analizadores y modificadores.
2. Desarrollen la clase `Digit` considerando los 4 mini-ciclos. Al final de cada mini-ciclo realicen una prueba indicando su propósito. Capturen las pantallas relevantes.

## Diseñando y codificando una nueva clase. DigitsGame

[En lab01.doc. DigitsGame.java]

El objetivo de este trabajo es programar una mini-aplicación para una **DigitsGame**<sup>7</sup>.

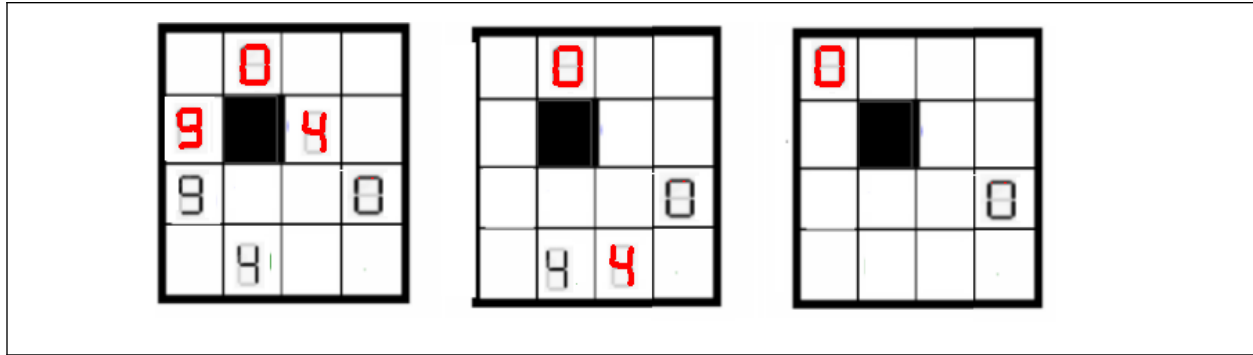
El tablero de este juego es cuadrado (NXN), con igual número de dígitos y de huecos con forma de dígitos (M); y un número de barreras (B). El objetivo de este juego es hacer que las dígitos caigan en el agujero con su mismo correspondiente a su valor.

Los movimientos consisten en levantar un lado del tablero para hacer que los dígitos se deslicen. Los dígitos caen si pasan por un hueco que lo contenga (por ejemplo un 1 puede caer en un hueco 8). Los dígitos pueden pasar sobre huecos que no los contiene y sobre huecos llenos; y no pueden pasar por las barreras.

En el ejemplo se muestra un tablero de 4 x 4, con tres dígitos (de color rojo), tres huecos (de color negro) y una barrera. Los movimientos realizados fueron levantar el norte y luego este.

---

<sup>7</sup> Inspirado en un problema de la maratón internacional 2007.



### Requisitos funcionales

- Permitir iniciar el juego indicando la configuración (N,M,B). Los elementos se seleccionan y ubican al azar.
- Permitir levantar uno de los lados del tablero (N, S, W, O)
- Permitir cambiar el color a los dígitos.

### Requisitos de interfaz

- Las barreras y los agujeros de los dígitos deben ser negros.
- Todos los dígitos son del mismo color: no pueden ser ni negros ni grises.
- Cuando un dígito cae en su propio hueco la celda queda en blanco.
- Cuando un dígito cae en un agujero que no le corresponde la celda queda de color gris.
- Debe “sonar” y “parpadear” cuando se logre el objetivo: todos los dígitos en sus huecos.
- Debe construirse usando las figuras del paquete `shapes`
- Se debe presentar un mensaje amable al usuario si hay algún problema. Consulte y use el método `showMessageDialog` de la clase `JOptionPane`.

1. Diseñen la clase `DigitsGame`, es decir, definan los métodos que debe ofrecer.
2. Planifiquen la construcción definiendo algunos miniciclos. Recuerden que al final de cada miniciclo deben poder probar.
3. Implementen la clase. Al final de cada miniciclo realicen una prueba de aceptación. Capturen las pantallas relevantes.
4. Indiquen las extensiones necesarias para reutilizar el paquete `shapes` y la clase `Digit`. Explique.

## Extendiendo una nueva clase. `DigitsGame`

[En `lab01.doc`. `DigitsGame.java`]

El objetivo es extender la clase para tener un mejor juego.

### Nuevos requisitos

1. Hace el mejor siguiente movimiento
2. Calcula el número mínimo de movimientos que faltan para ganar

## RETROSPECTIVA

1. ¿Cuál fue el tiempo total invertido en el laboratorio por cada uno de ustedes? (Horas/Hombre)
2. ¿Cuál es el estado actual del laboratorio? ¿Por qué?
3. Considerando las prácticas XP del laboratorio. ¿cuál fue la más útil? ¿por qué?
4. ¿Cuál consideran fue el mayor logro? ¿Por qué?
5. ¿Cuál consideran que fue el mayor problema técnico? ¿Qué hicieron para resolverlo?
6. ¿Qué hicieron bien como equipo? ¿Qué se comprometen a hacer para mejorar los resultados?