

PROGRAMACIÓN ORIENTADA A OBJETOS

Construcción. Clases y objetos.

2023-2

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 java¹.
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

A. Conociendo el proyecto shapes

[En lab01.doc]

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**² 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 tiene la clase `Triangle`? (c) ¿Cuántos métodos ofrece la clase `Triangle`? (d) ¿Cuáles métodos ofrece la clase `Triangle` para que la figura cambie su tamaño (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 determinan su forma?. 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` (a) ¿Qué significa que sea `public`? (b) ¿Qué significa que sea `static`? (c) ¿Qué significaría que fuera `final`? ¿Debe serlo? (d) ¿De qué tipo de dato debería ser (`byte`, `short`, `int`, `long`)? ¿Por qué? (e) Actualícenlo.

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 del atributo `height` (a) ¿Qué se está indicando al decir que es `int`? (b) Si `height` fuera `byte`, ¿cuál sería el triángulo más grande posible? (c) y ¿si fuera `long`? (d) ¿qué restricción adicional deberían tener este atributo? Refactoricen el código considerando (d).
- ¿Cuál dirían es el propósito del proyecto “shapes”?

B. Manipulando objetos. Usando un objeto.

[En lab01.doc]

- Crean un objeto de cada una de las clases que lo permitan³. (a) ¿Cuántas clases hay? (b) ¿Cuántos objetos crearon? (b) ¿Quién se crea de forma diferente? ¿Por qué?
- Inspeccionen el **estado** del objeto `:Triangle`⁴. (a) ¿Cuáles son los valores de inicio de todos sus atributos? (b) Capturen la pantalla.
- Inspeccionen el **comportamiento** que ofrece el objeto `:Triangle`⁵. (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 red social favorita. (a) ¿Cuántas y cuáles clases se necesitan? (b) ¿Cuántos objetos se usan en total? (c) Capturen la pantalla. (d) Incluyan el logo original.

C. Manipulando objetos. Analizando y escribiendo código.

[En lab01.doc]

```
Circle face;
Circle leftEar;
Circle rightEar;
//1
face=new Circle();
face.changeSize(50);
//2
face.moveVertical(20);
face.changeColor("black");
face.makeVisible();
//3

leftEar=new Circle();
leftEar.changeColor("black");
leftEar.moveHorizontal(-10);
//4
rightEar=leftEar;
rightEar.moveHorizontal(30);
rightEar.makeVisible();
//5
leftEar.makeVisible();
//6
```

- Lean el código anterior. (a) ¿cuál creen que es la figura resultante? (b) Píntenla.
- Habiliten la ventana de código en línea⁶, escriban el código. Para cada punto señalado indiquen: (a) ¿cuántas variables existen? (b) ¿cuántos objetos existen? (no cuenten ni los objetos `String` ni el objeto `Canvas`) (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.

D. Extendiendo una clase. Triangle.

[En lab01.doc y *.java]

1. Desarrollen en `Triangle` el método `area()`. ¡Pruébenlo! Capturen una pantalla.
2. Desarrollen en `Triangle` el método `equilateral()` (convierte el triángulo en un equilatero manteniendo el área).⁷ ¡Pruébenlo! Capturen dos pantallas.
3. Desarrollen en `Triangle` el método `blink(times:int)` (si `times>0`, cambia de color el número de veces indicado (Por ejemplo, original – nuevo – original serían dos veces. El color nuevo se escoge al azar entre los disponibles⁸). ¡Pruébenlo! Capturen dos pantallas.
4. Desarrollen en `Triangle` un nuevo creador que dada el area crea un triángulo al azar con dicha área.
5. Propongan un nuevo método para esta clase. Desarrolle y prueban el método.
6. Generen nuevamente la documentación y revise la información de estos nuevos métodos. Capturen la pantalla.

E. Extendiendo un paquete. shapes + Hexágono

[En lab01.doc y *.java]

En este punto vamos a adicionar una nueva forma al paquete shapes: un hexágono regular. Los hexágono regulares tienen todos sus lados iguales.

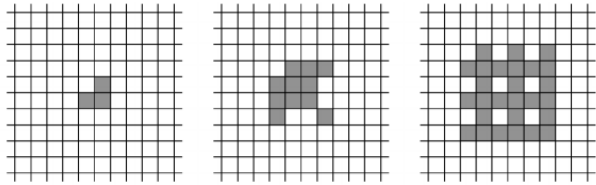
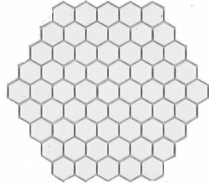
Hexagon	
<pre>- ¿? : ¿? - xPosition : int - yPosition : int - color : String - isVisible : boolean + _() : Hexagon + makeVisible() : void + makeInvisible() : void + moveRight() : void + moveLeft() : void + moveUp() : void + moveDown() : void + moveHorizontal(distance : int) : void + moveVertical(distance : int) : void + slowMoveHorizontal(distance : int) : void + slowMoveVertical(distance : int) : void + changeSize(newSize : int) : void + changeColor(newColor : String) : void - draw() : void - erase() : void</pre>	<pre>Mini-ciclo: 1 _() : Hexagon makeVisible() draw() Mini-ciclo: 2 Métodos que no cambian Mini-ciclo: 3 Los métodos que cambian</pre>

1. Para la construcción, tomen como base la clase `Triangle` inicial. (a) ¿Qué métodos podrían quedar iguales? (b) ¿Qué atributos podrían quedar iguales? (c) Justifiquen su respuesta.
2. Inicie la construcción de la clase `Hexagon` únicamente con los atributos. (a) Adicione pantallazo con los atributos.
3. Desarrollen la clase `Hexagon` considerando los 3 mini-ciclos. Al final de cada mini-ciclo realicen una prueba indicando su propósito. Capturen las pantallas relevantes.

7 Consulte la clase [Math](#) del API java

8 Consulte la clase [Math](#) del API java

REPLICATE REPLICATE

<p>Inicialmente, un conjunto de celdas se llena con una copia de la pieza que se va a replicar. En una secuencia de pasos discretos, cada celda en la red actualiza simultáneamente su estado examinando su propio estado y aquellos de sus vecinos aledaños. Si cuenta un número impar de estas celdas vecinas llenas, el siguiente estado de la celda será llena; de lo contrario, será vacío.</p> <p>La Figura G.1 muestra varios pasos en el proceso de replicación.</p> <p>[De World Final Problem G Replicate Replicate Rfplicbte] NO DEBEN RESOLVER EL PROBLEMA DE LA MARATÓN</p>	
<p>Nuestro replicate sigue las mismas reglas pero las celdas van a ser pentágonos.</p>	

F. Definiendo y creando una nueva clase. Replicate.

[En lab01.doc. SelfAssembly.java]

El objetivo de este trabajo es programar una mini-aplicación para **Replicate**.

Requisitos funcionales

- Crear un replicate indicando su dimensión.
- Llenar una celda. Deben ofrecer dos formas de crear la pieza: (a) dada su posición (b) al azar
- Replicar. Debe seguir las reglas.
- Reflejar vertical u horizontalmente el replicate

Requisitos de interfaz

- Las celdas se identifican por su fila y su columna.
- En caso que no sea posible realizar una de las acciones, debe generar un mensaje de error. Use JOptionPane.

1. Diseñen la clase, es decir, definan los métodos que debe ofrecer.
2. Planifiquen la construcción considerando algunos mini-ciclos.
3. Implementen la clase. Al final de cada mini-ciclo realicen una prueba indicando su propósito. Capturen las pantallas relevantes.
4. Indiquen las extensiones necesarias para reutilizar el paquete `shapes` y la clase `Hexagon`. Explique.

BONO. Nuevos requisitos funcionales. Replicate.

[En lab01.doc. SelfAssembly.java]

El objetivo de este trabajo es extender la mini-aplicación **Replicate**.

Nuevos requisitos funcionales

- Rotar 45% a derecha o izquierda.
 - Llenar una celda para que en la próximo replicate se llene un número dado de celdas, si es posible. Explique la estrategia.
1. Diseñen, es decir, definan los métodos que debe ofrecer.
 2. Implementen los nuevos métodos. Al final de cada método realicen una prueba indicando su propósito. Capturen las pantallas relevantes.

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?