# PROGRAMACIÓN ORIENTADA A OBJETOS
# Construcción. Clases y objetos.
# 2025-1
# 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[1].
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 ábranlo en BlueJ[2]. 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**[3] 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) ¿Qué atributos determinan el tamaño de un `Triangle`? (d) ¿Cuáles métodos ofrece la clase `Triangle` para cambiar su tamaño?
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) ¿Quiénes pueden usar los atributos públicos?. Con respecto a los métodos: (c) ¿Cuántos métodos tiene en total? (d) ¿Quiénes usan los métodos privados? (e) Desde el editor, consulte la documentación. Capture la pantalla.
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) Actualícenlo.

---

1 http://docs.oracle.com/javase/8/docs/api/
2 Comando de menú Project-Open.
3 Menu: Tools-Project Documentation

7. En el código de la clase `Triangle` revisen el detalle del tipo del atributo `height` (a) ¿Qué se está indicándo al decir que es `int`?. (b) Si fuera `byte`, ¿cuál sería el área rectángulo más grande posible? (c) y ¿si fuera `long`? d) ¿qué restricción adicional deberían tener este atributo? (e) Refactoricen el código considerando (d).
8. ¿Cuál dirían es el propósito del proyecto "`shapes`"?

## B. Manipulando objetos. Usando un objeto.

**[En lab01.doc]**

1. Creen un objeto de cada una de las clases que lo permitan[4]. (a) ¿Cuántas clases hay? (b) ¿Cuántos objetos crearon? (c) ¿Quién se crea de forma diferente? ¿Por qué?
2. Inspeccionen los creadores de cada una de las clases. (a) ¿Cuál es la principal diferencia entre ellos? (b) ¿Qué se busca con la clase que tiene el creador diferente?
3. Inspeccionen el **estado** del objeto :`Triangle`[5], (a) ¿Cuáles son los valores de inicio de todos sus atributos? (b) Capturen las pantallas.
4. Inspeccionen el **comportamiento** que ofrece el objeto :`Triangle`[6]. (a) Capturen la pantalla. (b) ¿Por qué no aparecen todos los que están en el código?
5. Construyan, con "`shapes`" sin escribir código, una propuesta de la imagen del logo de su chatbot IA favorito. (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]**

```
Rectangle red;
Rectangle green;
Rectangle blue;
//1
red=new Rectangle();
red.changeColor("red");
//2
green=new Rectangle();
green.changeColor("green");
green.moveHorizontal(45);
red.makeVisible();
green.makeVisible();
//3
```

```
blue=red;
blue.changeColor("blue");
blue.moveVertical(35);
//4
Rectangle yellow=new Rectangle()
yellow.changeColor("yellow");
yellow.moveHorizontal(45);
yellow.moveVertical(35);
//5
blue.makeVisible();
yellow.makeVisible();
//6
```

1. Lean el código anterior. (a) ¿cuál creen que es la figura resultante? (b) Píntenla.
2. Para cada punto señalado indiquen[7]: (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?
3. Habiliten la ventana de código en línea[8], escriban el código. (a) Capturen la pantalla.
4. Compare figura pintada en 1. con la figura capturada en 3. , (a) ¿son iguales? (b) ¿por qué?

---

4 Clic derecho sobre la clase
5 Clic derecho sobre el objeto y comando inspect.
6 Clic derecho sobre el objeto.
7 Son (4*6) respuestas
8 Menú. View-Show Code Pad.

## D. Extendiendo una clase. Triangle.

**[En lab01.doc y *.java]**

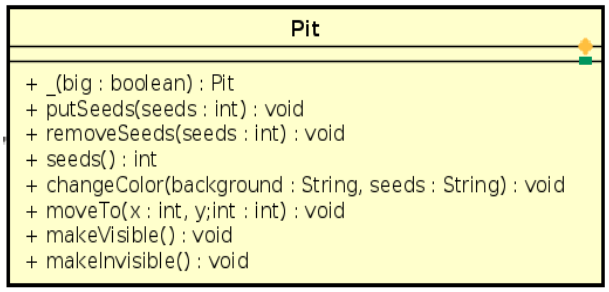
1. Desarrollen en `Triangle` el método `area()`
   (retorna el área del triangulo). ¡Pruébenlo! Capturen una pantalla.
2. Desarrollen en `Triangle` el método `equilateral()`
   (transforma el triangulo en un triangulo equilatero de área equivalente). ¡Pruébenlo! Capturen dos pantallas.
3. Desarrollen en `Triangle` el método `shrink(times:int, height: int)`
   (disminuye su tamaño **times** veces. Hasta llegar a una altura de **height**.) ¡Pruébenlo! Capturen tres pantallas.
4. Desarrollen en `Triangle` un nuevo creador que permita crear un triangulo en una posición específica.¡Pruébenlo! Capturen una pantalla.
5. Propongan un nuevo método para esta clase. Desarrollen y prueban el método.
6. Generen nuevamente la documentación y revise la información de estos nuevos métodos. Capturen la pantalla.

## E. Creando una nueva clase. Usando un paquete. shapes

**[En lab01.doc y *.java]**

En este punto vamos a crear huecos con fondos cuadrados para guardar semillas. El diseño gráfico lo definen ustedes. Estos son algunos ejemplos.

Los huecos deben ofrecer los siguiente métodos




El marco de los huecos debe ser cuadrado.
El hueco big es el doble de grande que el no big.

**Mini-ciclo: 1**
`_(big):Pit`
`putSeeds(seeds)`
`removeSeeds(seeds)`
`seeds()`
**Mini-ciclo: 2**
`makeVisible()`
`makeInvisible()`
**Mini-ciclo: 3**
`changeColors(pit,seeds)`
`moveTo(x,y)`

1. Inicie la construcción únicamente con los atributos. Justifique su selección. Adicione pantallazo con los atributos.
2. Desarrollen la clase considerando los 3 mini-ciclos. Al final de cada mini-ciclo realicen dos pruebas indicando su propósito. Capturen las pantallas relevantes.

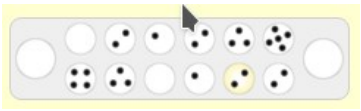
## G. De python a java

**[En lab01.doc]**

En este punto vamos a usar y evaluar dos recursos de apoyo para la transición de Python a Java. Realicen la evaluación en las encuestas preparadas con ese objetivo.

1. El video.
2. Los prompts.

## KALAH

| | |
|---|---|
|  | KALAH se juega sobre un tablero con seis casas y un almacén por jugador. Las casas y los almacenes son huecos para depositar semillas. Al inicio del juego en cada casa se colocan tres semillas. En un turno, un jugador vacía de semillas de una de sus casas, moviéndose en sentido contrario a las agujas del reloj, poniendo una semilla en cada casa y en el almacén propio, nunca en el almacén del oponente. Si la última semilla cae en el almacén del jugador, éste obtiene un turno adicional. Si la última semilla cae en una casa vacía del jugador, se apropia de las semillas de la casa que está al frente. Cuando un jugador no posee más semillas en sus casas, el juego acaba. El jugador que gana es el que queda con más semillas en su almacén. Tomado de: http://es.wikipedia.org/wiki/Kalah |

## F. Definiendo y creando una nueva clase. Kalah.

**[En lab01.doc.Kalah.java]**

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

**Requisitos funcionales**

- Crear el estado inicial
- Realizar los movimientos
- Reiniciar el juego
- Consultar el estado del juego. (Un mensaje con el número de semillas en cada almacén)
- Informar cuando alguién gana el juego. (Un mensaje de felicitación)

**Requisitos de interfaz**

- Las casas se identifican por el jugador (‘N’orth, ‘S’outh) y el número de la casa (1 a 6).
- En caso que no sea posible realizar una de las acciones, debe generar un mensaje de error.
- Para los mensajes 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 la clase `Pit` y el paquete `shapes`. Expliquen.

## BONO. Nuevos requisitos funcionales. Kalah.

**[En lab01.doc. Kalah.java]**

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

**Nuevos requisitos funcionales**

- Hacer un movimiento (la máquina decide). Explique la estrategia.
- Deshacer el último movimiento

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?
7. ¿Qué referencias usaron? ¿Cuál fue la más útil?  Incluyan citas con estándares adecuados.