

ESCUELA COLOMBIANA DE INGENIERÍA

DESARROLLO ORIENTADO POR OBJETOS

Construcción. Clases y Objetos.

2026-1 — Laboratorio 1/6

OBJETIVOS

Desarrollar competencias básicas para:

1. Apropiar un paquete revisando el diagrama de clases, la documentación y el 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.
5. Utilizar el entorno de desarrollo de BlueJ.
6. Vivenciar las prácticas XP:

Planning — el proyecto se divide en iteraciones; **Coding** — todo el código de producción se programa en parejas.

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 y separados por un guion. (Casallas-Duarte)
- Publiquen el avance al final de la sesión y la versión definitiva en la fecha indicada, en los espacios correspondientes.

1. 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 BlueJ¹. 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. Genere la documentación. De acuerdo con la documentación: (a) ¿Qué clases tiene el paquete [shapes](#)? (b) ¿Qué atributos tiene la clase [Rectangle](#)? (c) ¿Cuántos métodos ofrece la clase [Rectangle](#)? (d) ¿Qué atributos determinan el tamaño de un [Rectangle](#)? (e) ¿Cuáles métodos ofrece la clase [Rectangle](#) para cambiar su tamaño?
4. En el **código fuente** de cada clase está el detalle de la implementación. Revisen el código de la clase [Rectangle](#). 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?
5. Desde el editor, consulte la documentación. (a) Capture la pantalla. Comparando la documentación con el código: (b) ¿Qué no se ve en la documentación? (c) ¿por qué debe ser así?
6. En el código de la clase [Rectangle](#), revise el atributo [EDGES](#). (a) ¿Qué significa que sea [public](#)? (b) ¿Qué significa que sea [static](#)? (c) ¿Qué significa que sea [final](#)?
7. En el código de la clase [Rectangle](#) revisen el detalle del tipo del atributo [width](#). (a) ¿Qué se está indicando al decir que es [int](#)? (b) Si fuera [byte](#), ¿cuál sería el área más grande posible? (c) y ¿si fuera [long](#)? (d) ¿qué restricción adicional debería tener este atributo? (e) Refactoricen el código considerando (d).
8. Si creamos 100 rectángulos, (a) ¿cuántos [EDGES](#) y cuántos [width](#) tendríamos? (b) Justifique.
9. ¿Cuál dirían es el propósito del proyecto [shapes](#)?

¹Menú: [Project → Open](#).

²Menú: [Tools → Project Documentation](#).

B Manipulando objetos. Usando un objeto.

1. Creen un objeto de cada una de las clases que lo permitan³. (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 :`Rectangle`⁴. (a) Capturen la pantalla.(b) ¿Cuál es el color inicial?
4. Inspeccionen el **comportamiento** que ofrece el objeto :`Rectangle`⁵. (a) Capturen la pantalla. (b) ¿Por qué no aparecen todos los métodos que están en el código?
5. Construyan, con `shapes` sin escribir código, una propuesta de la imagen del logo de su videojuego 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. [\[En lab01.doc\]](#)

```
1 Rectangle rod;
  Rectangle bigDisk;
  Rectangle smallDisk;
 //1
 rod=new Rectangle();
 bigDisk = new Rectangle();
 smallDisk = bigDisk;
 //2
 rod.changeColor("black");
 rod.changeSize(100,10);
 rod.makeVisible();
 //3
 bigDisk.moveHorizontal(-5);
 bigDisk.moveVertical(80);
 smallDisk.moveHorizontal(-20);
 smallDisk.moveVertical(90);
 //4
 bigDisk.changeColor("red");
 bigDisk.changeSize(10,20);
 smallDisk.changeColor("blue");
 smallDisk.changeSize(10,50);
 //5
 rod.makeVisible();
 bigDisk.makeVisible();
 smallDisk.makeVisible();
 //6
```

1. Lean el código anterior. (a) ¿Cuál creen que es la figura resultante? (b) Píntenla.
2. Para cada punto señalado: (a) ¿cuántas variables existen? (b) ¿cuántos objetos existen? (no cuenten ni los objetos `String` ni el objeto `Canvas`) (c) ¿cuántos objetos se ven? (3 respuestas en cada punto)
3. Habiliten la ventana de código en línea⁶, escriban el código. (a) Capturen la pantalla.
4. Compare la figura pintada en 1. con la figura capturada en 3.: (a) ¿son iguales? (b) ¿por qué?

D Extendiendo una clase. `Rectangle`. [\[En lab01.doc\]](#) [\[En .java\]](#)

1. Desarrollen en `Rectangle` el método `perimeter()` (retorna el perímetro). ¡Pruébenlo! Capturen una pantalla.
2. Desarrollen en `Rectangle` el método `zoom(z: char)` (aumenta (+) o disminuye (-) 100 % el perímetro manteniendo la proporción). ¡Pruébenlo! Capturen dos pantallas.
3. Desarrollen en `Rectangle` el método `walk(times: int)` (va moviéndose cayendo hacia la derecha (positivo) o izquierda (negativo) `abs(times)` veces). ¡Pruébenlo! Capturen tres pantallas.
4. Desarrollen en `Rectangle` un nuevo creador que permita crear un cuadrado dado su perímetro. ¡Pruébenlo! Capturen una pantalla.
5. Propongan un nuevo método para esta clase. Desarrollen y prueben el método.
6. Generen nuevamente la documentación y revisen la información de estos nuevos métodos. Capturen la pantalla.

³Clic derecho sobre la clase

⁴Clic derecho sobre el objeto y comando *Inspect*

⁵Clic derecho sobre el objeto

⁶Menú: `View → Show Code Pad`

2. 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:

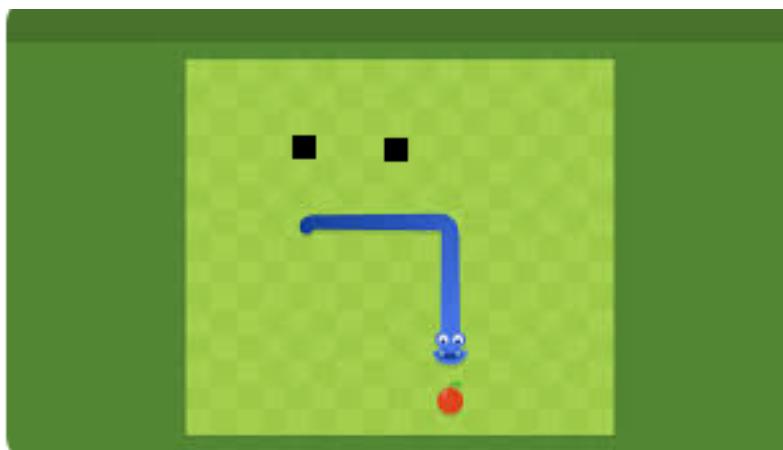
1. El video.
2. Los prompts.

3. HungrySnakeGame

El objetivo de este punto es desarrollar una mini-aplicación para una versión simplificada del juego HungrySnakeGame. Inspirado en el juego creado por Taneli Armanto en 1998 como uno de los tres juegos del teléfono celular Nokia 6110. [VINCULO](#)

El juego se desarrolla en un tablero rectangular de dimensiones fijas que una serpiente recorre buscando comida para crecer. El objetivo del juego es lograr que la serpiente obtenga la mayor longitud posible.

- El juego inicia con la serpiente de una unidad de longitud, una fruta y un número fijo de obstáculos. Todos colocados al azar.
- El jugador maniobra la cabeza de la serpiente y el cuerpo sigue la trayectoria.
Los movimientos son en las cuatro direcciones (North, South, East, West)
- Si la cabeza de la serpiente toca la fruta aumenta de longitud una unidad y otra fruta aparece en el tablero.
- Si la serpiente choca contra un obstáculo, los bordes del tablero o consigo misma, el jugador pierde.



A Creando una nueva clase. Usando un paquete shapes [\[En lab01.doc\]](#) [\[En .java\]](#)

En este punto vamos a crear la clase `Snake` del juego. El diseño gráfico lo definen ustedes.

<table border="1"><tr><td>Snake</td></tr><tr><td>+ <code>_row : int, column : int</code> : Snake</td></tr><tr><td>+ <code>move(direction : char) : void</code></td></tr><tr><td>+ <code>grow(direction : char) : void</code></td></tr><tr><td>+ <code>head() : int[]</code></td></tr><tr><td>+ <code>tail() : int[]</code></td></tr><tr><td>+ <code>length() : int</code></td></tr><tr><td>+ <code>makeVisible() : void</code></td></tr><tr><td>+ <code>makeInvisible() : void</code></td></tr><tr><td>+ <code>isOK() : boolean</code></td></tr></table>	Snake	+ <code>_row : int, column : int</code> : Snake	+ <code>move(direction : char) : void</code>	+ <code>grow(direction : char) : void</code>	+ <code>head() : int[]</code>	+ <code>tail() : int[]</code>	+ <code>length() : int</code>	+ <code>makeVisible() : void</code>	+ <code>makeInvisible() : void</code>	+ <code>isOK() : boolean</code>	<ul style="list-style-type: none">■ Ciclo 1:<ul style="list-style-type: none">- <code>_()</code>- <code>head</code>- <code>tail</code>■ Ciclo 2:<ul style="list-style-type: none">- <code>move</code>- <code>grow</code>- <code>isOk</code>■ Ciclo 3:<ul style="list-style-type: none">- <code>length</code>- <code>makeVisible</code>- <code>makeInvisible</code>■ Notas:<ul style="list-style-type: none">- <code>head</code> y <code>tail</code> retornan la posición [fila, columna]- En <code>move</code>, la serpiente sólo se mueve- En <code>grow</code>, la serpiente se mueve y crece- <code>isOk</code> retorna falso si no se pudo hacer el último movimiento
Snake											
+ <code>_row : int, column : int</code> : Snake											
+ <code>move(direction : char) : void</code>											
+ <code>grow(direction : char) : void</code>											
+ <code>head() : int[]</code>											
+ <code>tail() : int[]</code>											
+ <code>length() : int</code>											
+ <code>makeVisible() : void</code>											
+ <code>makeInvisible() : void</code>											
+ <code>isOK() : boolean</code>											

1. Inicie la construcción únicamente con los **atributos**. Justifique su selección. Agregue pantallazo con los atributos.
2. Desarrollen la clase considerando los 3 mini-ciclos. No olviden la documentación. Al final de cada mini-ciclo realicen dos pruebas indicando su propósito. Capturen las pantallas relevantes.

B Definiendo y creando una nueva clase. HungrySnakeGame. [\[En lab01.doc\]](#) [\[En .java\]](#)

En este punto vamos a desarrollar la *mini-aplicación* para HungrySnakeGame.

Requisitos funcionales	Requisitos de interfaz
<ul style="list-style-type: none">■ Crear el estado inicial.■ Realizar los movimientos.■ Reiniciar el juego.■ Consultar el estado del juego (un mensaje con la longitud de la serpiente).■ Informar cuando se pierde el juego (un mensaje de consolación).	<ul style="list-style-type: none">■ Los diferentes movimientos se realizan cada uno con un comando especial (north, south, east, west).■ En caso que no sea posible realizar una de las acciones, debe generar un mensaje de error.■ Para los mensajes use <code>JOptionPane</code>.

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 `Snake` y el paquete `shapes`. Expliquen.

C BONO. Nuevos requisitos funcionales. HungrySnakeGame. [\[En lab01.doc\]](#) [\[En .java\]](#)

Extiendan la mini-aplicación HungrySnakeGame con los siguientes requisitos:

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