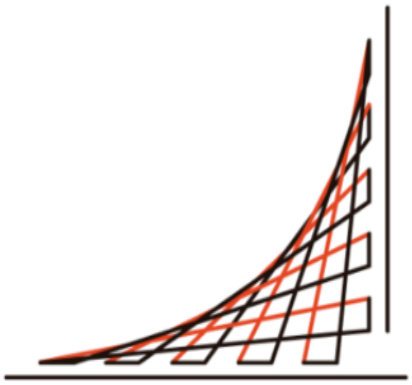


ESCUELA COLOMBIANA DE INGENIERÍA JULIO GARAVITO

UNIVERSIDAD

Nicolás Felipe Bernal Gallo

Juan Daniel Bogotá Fuetes

Desarrollo Orientada a objetos
DOPO LAB

Laboratorio #1 Construcción clases y objetos
21/08/2025

PROFESOR: María Irma Diaz Rozo

DESARROLLO ORIENTADO A OBJETOS
Construcción. Clases y objetos.
2025-2 — 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.

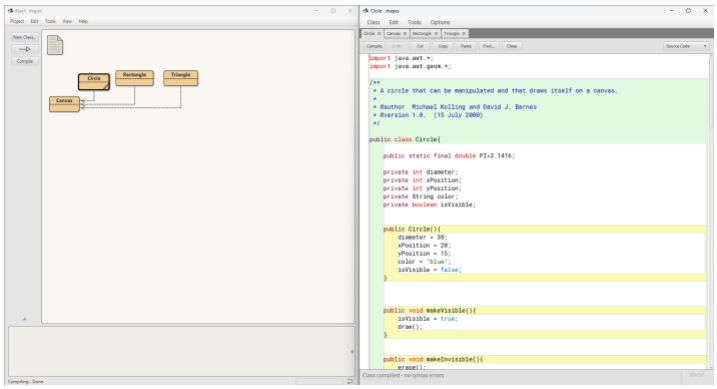
ENTREGA

- Incluyan en un archivo `.zip` los archivos correspondientes al laboratorio. El nombre debe ser los dos apellidos de los miembros del equipo ordenados alfabéticamente.
- Deben publicar 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 BlueJ[1]. 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?

   Tenemos 4 clases, Canvas, Circle, Rectangle y Triangle.

   (b) ¿Qué relaciones existen entre ellas?

   Circle, Rectangle y Triangle dependen de Canvas.

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`?

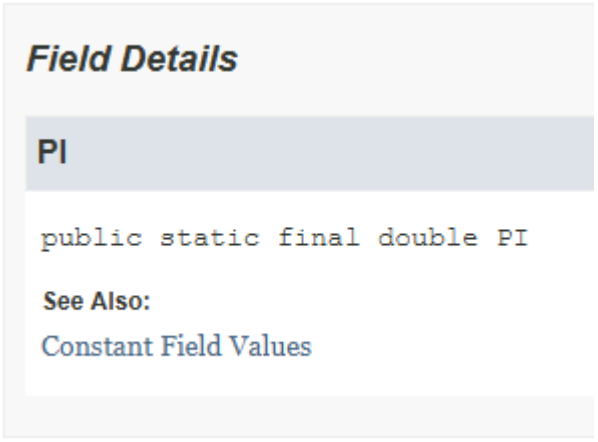
   Tenemos 4 clases, Canvas, Circle, Rectangle y Triangle.

Classes

Class	Description
Canvas	Canvas is a class to allow for simple graphical drawing on a canvas.
Circle	A circle that can be manipulated and that draws itself on a canvas.
Rectangle	A rectangle that can be manipulated and that draws itself on a canvas.
Triangle	A triangle that can be manipulated and that draws itself on a canvas.

   (b) ¿Qué atributos tiene la clase `Circle`?

   Según la documentación de la clase “Circle”, solo tiene un atributo, “PI”.



   (c) ¿Cuántos métodos ofrece la clase `Circle`?

   La clase “Circle” tiene 12 métodos.

   (d) ¿Qué atributos determinan el tamaño de un `Circle`?

   Según la documentación no se encuentra ningún atributo para determinar el tamaño de la clase “Circle”.

   (e) **¿Cuáles métodos ofrece la clase `Circle` para cambiar su tamaño?**

   Los métodos que ofrece la clase “Circle” para que cambie su tamaño es “changeSize”.

4. **En el código de cada clase está el detalle de la implementación. Revisen el código de la clase `Circle`. Con respecto a los atributos:**

   (a) ¿Cuántos atributos realmente tiene?

   Según el código, se tienen 6 atributos: PI, diameter, xPosition, yPosition, color e isVisible.

```
public class Circle{

    public static final double PI=3.1416;

    private int diameter;
    private int xPosition;
    private int yPosition;
    private String color;
    private boolean isVisible;
```

   (b) ¿Quiénes pueden usar los atributos públicos?

   Los atributos públicos pueden ser utilizados por cualquier clase dentro del mismo paquete shapes.

   Con respecto a los métodos:

   (c) ¿Cuántos métodos tiene en total?

   Tiene 14 métodos, 12 públicos y 2 privados.

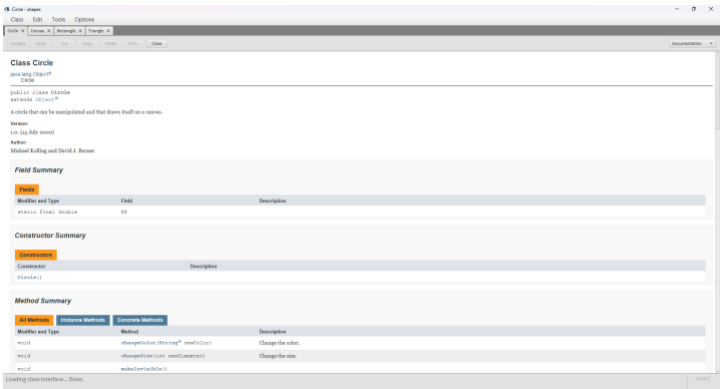
   (d) ¿Quiénes usan los métodos privados?

   Los métodos privados solo pueden ser utilizados por la misma clase donde están definidos.

   Por ejemplo: draw() y erase() en la clase “Circle”

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?

No se ven todos los métodos y atributos con el modificador de acceso prívate.

En el caso de los atributos no se ven estos:

```
private int diameter;
private int xPosition;
private int yPosition;
private String color;
private boolean isVisible;
```

En el caso de los métodos no se ven estos:

```
private void draw(){
    if(isVisible) {
        Canvas canvas = Canvas.getCanvas();
        canvas.draw(this, color,
            new Ellipse2D.Double(xPosition, yPosition,
            diameter, diameter));
        canvas.wait(10);
    }
}

private void erase(){
    if(isVisible) {
        Canvas canvas = Canvas.getCanvas();
        canvas.erase(this);
    }
}
```

(b) ¿por qué debe ser así?

Porque se deben proteger los datos y además para que así se evite que otras clases lleguen a modificar los valores. También se busca asegurar que los datos sean válidos dentro del código.

6. **En el código de la clase `Circle`, revise el atributo `PI`.**

   (a) ¿Qué significa que sea `public`?

   Esto significa que es accesible desde cualquier clase dentro del programa.

   (b)¿Qué significa que sea `static`?

   Esto significa que pertenece a la clase en lugar de a una instancia.

   (b) ¿Qué significa que sea `final`?

   Esto significaría que no se podría llegar a modificar el atributo después de inicializado. Si como es un triángulo siempre debe tener tres vértices.

7. **En el código de la clase `Circle` revisen el detalle del tipo del atributo `diameter`.**
   (a) ¿Qué se está indicando al decir que es `int`?

   Que se trata de que el atributo diameter usa un número entero, no puede usar números decimales.

   (b) Si fuera `byte`, ¿cuál sería el área más grande posible?

   Byte en java significa que puede usar valores entre -128 y 127, entonces como el área de un Circulo se calcula como:

$$r = 127$$

$$A = \pi * r^2$$

$$A = \pi * (127)^2$$

$$A \approx 3.1416 * 16129$$

$$A \approx 50653.1$$

(c) y ¿si fuera `long`?

Long en java significa que puede usar valores entre −263 y 263 en el caso de Circulo se calcula como:

$$r = 9,223,372,036,854,775,807$$

$$A = \pi * r^2$$

$$A = \pi * r^2$$

$$A = \pi * (9.22 * 10^{18})^2$$

$$A = \pi * (8.50 * 10^{37})$$

$$A \approx 2.67 * 10^{38}$$

(d) ¿qué restricción adicional debería tener este atributo?

Debería tener una restricción de números negativos, ya que el diámetro no puede ser negativo.

(e) Refactoricen el código considerando

```
/**
 * Change the size.
 * @param newDiameter the new size (in pixels). Size must be >=0.
 */
public void changeSize(int newDiameter){
    if (newDiameter >=0)
    erase();
    diameter = newDiameter;
    draw();
}
```

8. **Si creamos 100 círculos,**

   (a) ¿cuántos `PI` y cuántos `diameter` tendríamos?

   1 PI y 100 diameter.

   (b) Justifique.

   PI es una constante estática (static final) de la clase: existe una sola copia compartida por todos los objetos, sin importar cuántos círculos creemos.

   Diameter es un atributo de instancia: cada círculo tiene su propio diámetro, por eso con 100 círculos hay 100 diameter.

9. **¿Cuál dirían es el propósito del proyecto `shapes`?**

El propósito es dar una introducción practica a la programación orientada a objetos usando BlueJ.
Modelar figuras geométricas para practicar POO: definir clases y objetos, atributos y métodos, diferenciar miembros estáticos vs de instancia, usar constantes, validación de datos.

B Manipulando objetos. Usando un objeto.

1. **Creen un objeto de cada una de las clases que lo permitan[3].**

   (a) ¿Cuántas clases hay?

   Hay 4 clases: “Circle”, “Rectangle”, “Triangle” y “Canvas”

   (b)¿Cuántos objetos crearon?

   Se crearon 4 objetos: triangulo, rectángulo, circulo y canvas.

   (c) ¿Quién se crea de forma diferente? ¿Por qué?

   El canvas se crea de forma diferente, porque canvas es como la ventana donde se dibujan los demás objetos. Los demás objetos se crean de manera explícita.

2. **Inspeccionen los creadores de cada una de las clases.**

   (a) ¿Cuál es la principal diferencia entre ellos?

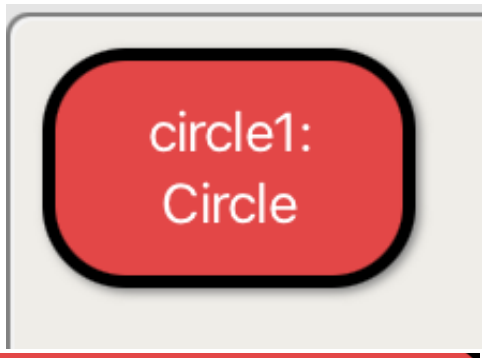
   Los objetos triangulo, círculo y rectángulo tienen las propiedades básicas de cada figura, en cambio en el canva no representa una figura, representa un lienzo donde se pueden observar las demás figuras.

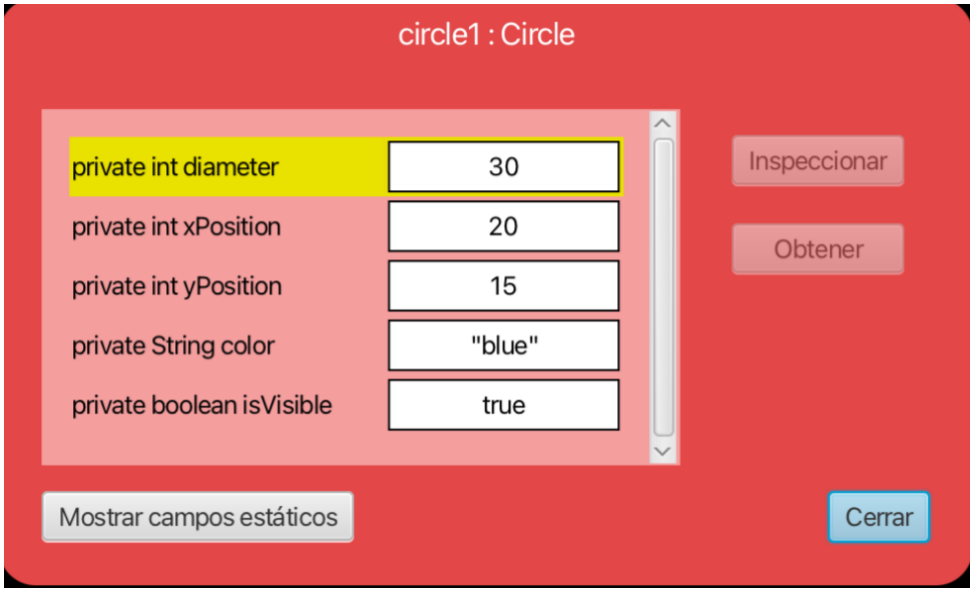
   (b) ¿Qué se busca con la clase que tiene el creador diferente?

   El canvas busca ser un gestor de dibujo donde se pueden ver los demás objetos.

3. **Inspeccionen el estado del objeto** `:Circle`[4]**.**

   (a) Capturen la pantalla.



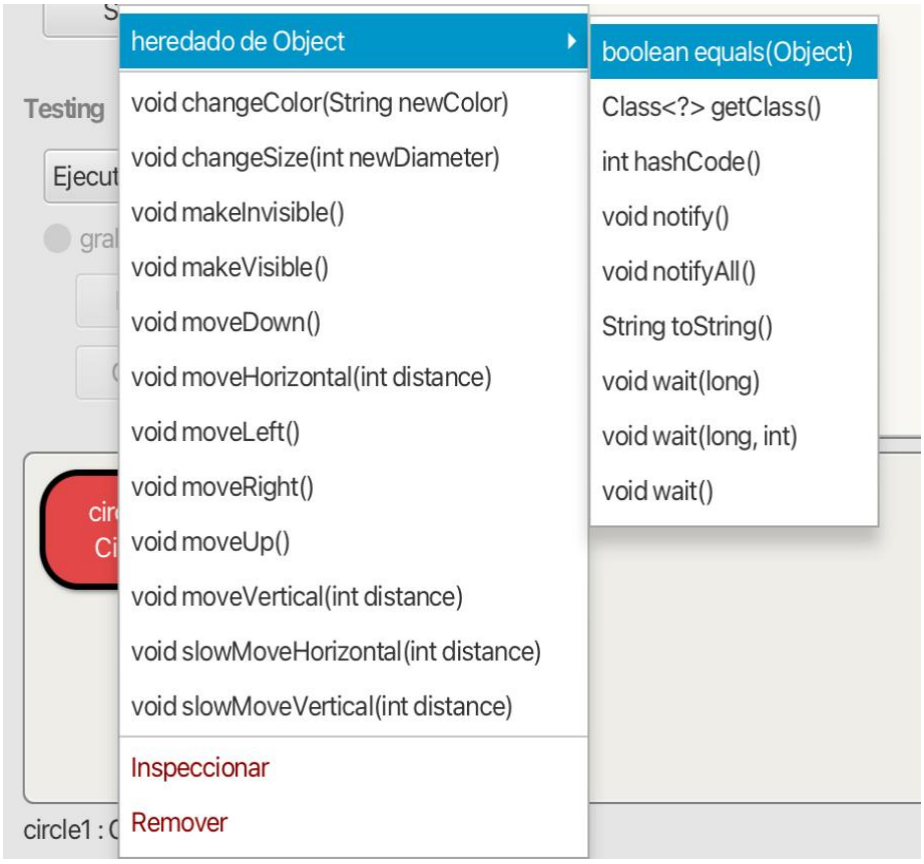


   (b) ¿Cuál es el color inicial?

El color inicial para el objeto “Circle” es el azul.

4. **Inspeccionen el comportamiento que ofrece el objeto: `Circle`[5].**

   (a) Capturen la pantalla.



   (b) ¿Por qué no aparecen todos los que están en el código?

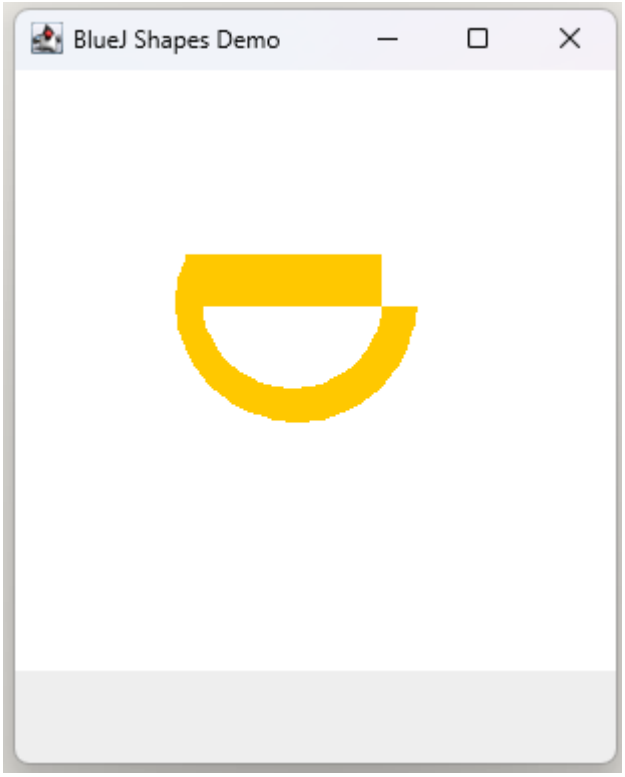
   Porque hay métodos que están con el modificador de acceso private.

5. **Construyan, con `shapes` sin escribir código, una propuesta de la imagen del logo de su domiciliario favorito.**

   (a) ¿Cuántas y cuáles clases se necesitan?
   Necesitamos dos clases, sin contar Canvas, Circle y Rectangle.

   (b) **¿Cuántos objetos se usan en total?**
   Usamos 5 objetos.

(c) Capturen la pantalla.



(d) Incluyan el logo original.

C Manipulando objetos. Analizando y escribiendo código. [En lab01.doc]

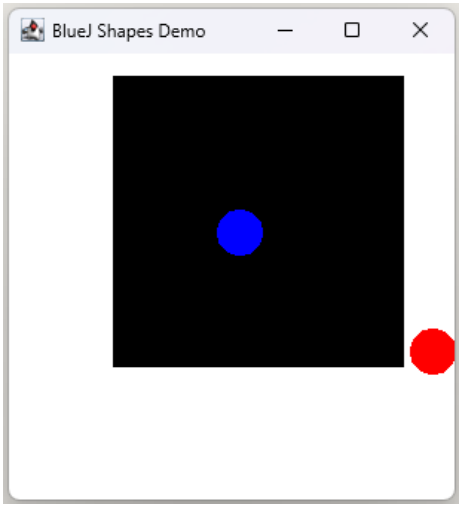
```
Rectangle face;
Circle pointOne;
Circle pointTwo;
Circle pointThree;
//1
face=new Rectangle();
pointOne = new Circle();
pointTwo = new Circle();
//2
face.changeColor("black");
face.changeSize(195,195);
face.makeVisible();
//3
pointOne.changeColor("yellow");
pointOne.moveHorizontal(70);
pointOne.moveVertical(20);
//4
pointTwo=pointOne;
pointTwo.changeColor("red");
pointTwo.moveHorizontal(180);
pointTwo.moveVertical(150);
pointTwo.makeVisible();
//5
pointThree= new Circle();
pointThree.moveHorizontal(120);
pointThree.moveVertical(90);
//6
pointOne.makeVisible();
pointThree.makeVisible();
//7
```

1. **Lean el código anterior.**

   (a) ¿Cuál creen que es la figura resultante?

   Viendo el código por encima, creemos que son dos círculos, uno rojo y otro azul, y un rectángulo negro.

   (b) Píntenla.



2. **Para cada punto señalado:**

   (a) ¿cuántas variables existen?

   Existen 4 variables:

```
Rectangle face;
Circle pointOne;
Circle pointTwo;
Circle pointThree;
```

   (b) ¿cuántos objetos existen? (no cuenten ni los objetos `String` ni el objeto `Canvas`)

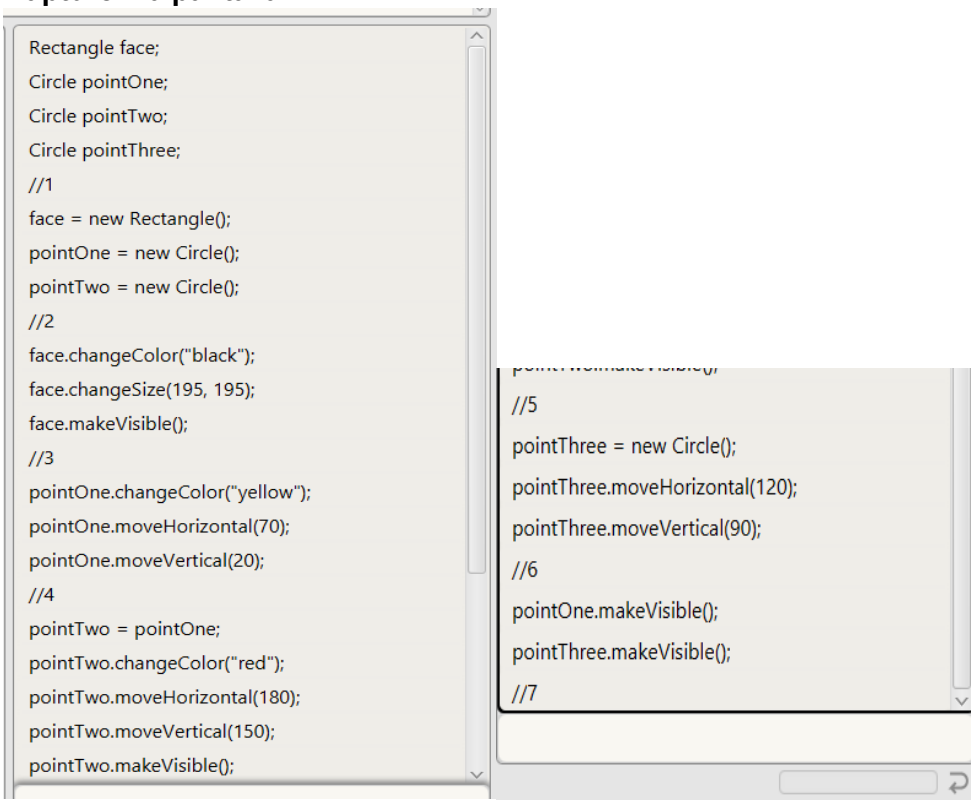
   Existen 3 objetos, dos círculos y un rectángulo.

   (c) ¿cuántos objetos se ven?

   Se ven 3 objetos, dos círculos, uno azul y uno rojo y un rectángulo.

3. Habiliten la ventana de código en línea[6], escriban el código.

   (a) Capturen la pantalla.



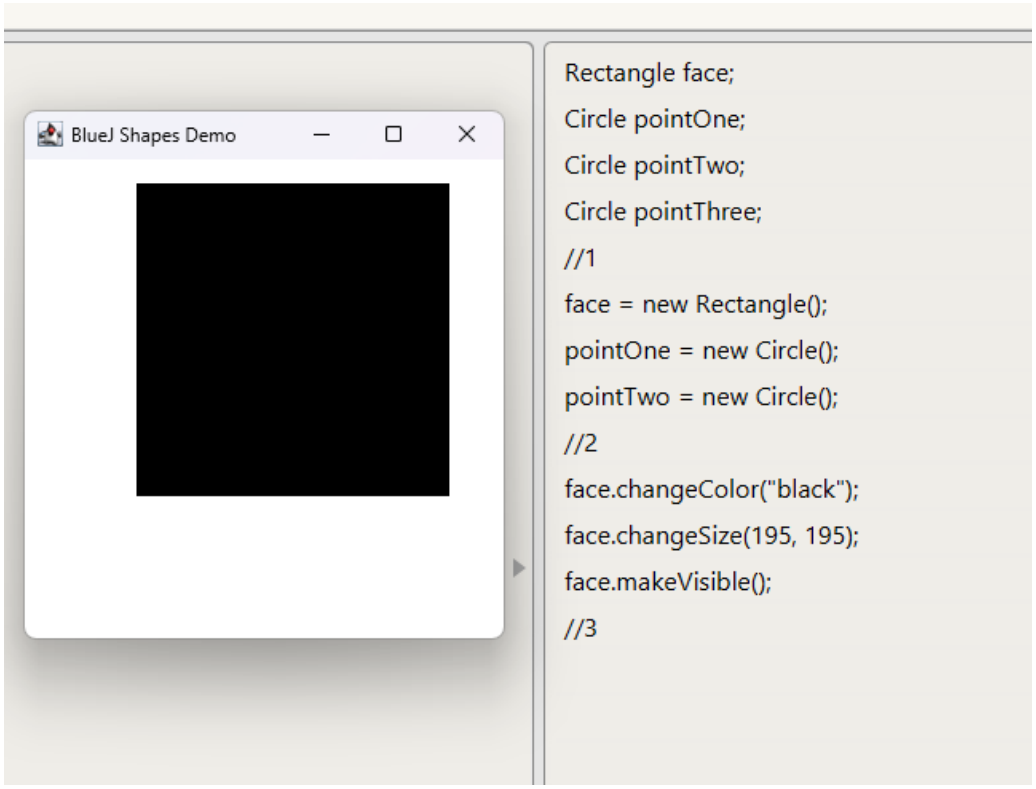
4. Compare la figura pintada en 1. con la figura capturada en 3.:

(a) ¿son iguales?

Las figuras capturadas no son iguales.

(b) ¿por qué?

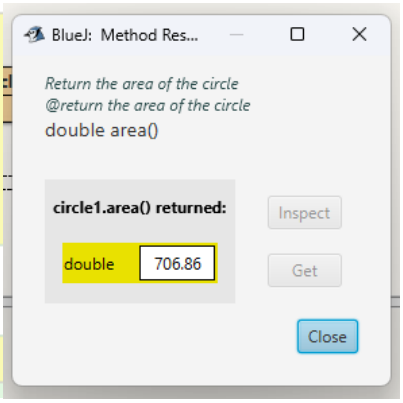
Porque en la figura 1 no se pinta nada, ni siquiera se abre el Canvas, solo se declaran variables, en cambio en la figura capturada en 3 se ve un rectángulo negro de tamaño (195,195).



D Extendiendo una clase. `Circle`. [En lab01.doc][En .java]

1. **Desarrollen en `Circle` el método `area()` (retorna el área del círculo). ¡Pruébenlo! Capturen una pantalla.**

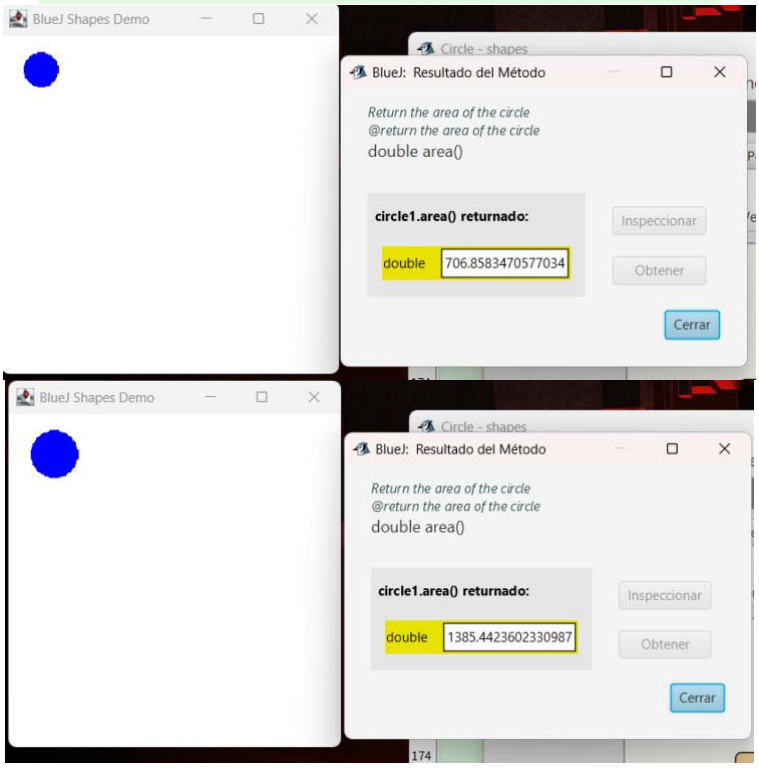
```java
/**
 * Return the area of the circle
 * @return the area of the circle
 */
public double area(){
    double radio = diameter/2.0;
    return PI * (radio * radio);
}
```



2. **Desarrollen en `Circle` el método `bigger(percentage: int)` (aumenta el área del círculo un porcentaje [0...100]). ¡Pruébenlo! Capturen dos pantallas.**

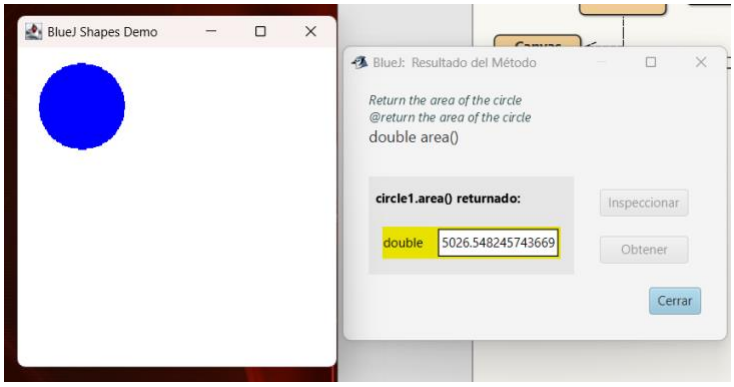
   Se aumentó 100%, no es exacto ya que el tamaño se aumenta con el diámetro y el diámetro es tipo int, al no ser tipo double se pierde exactitud.

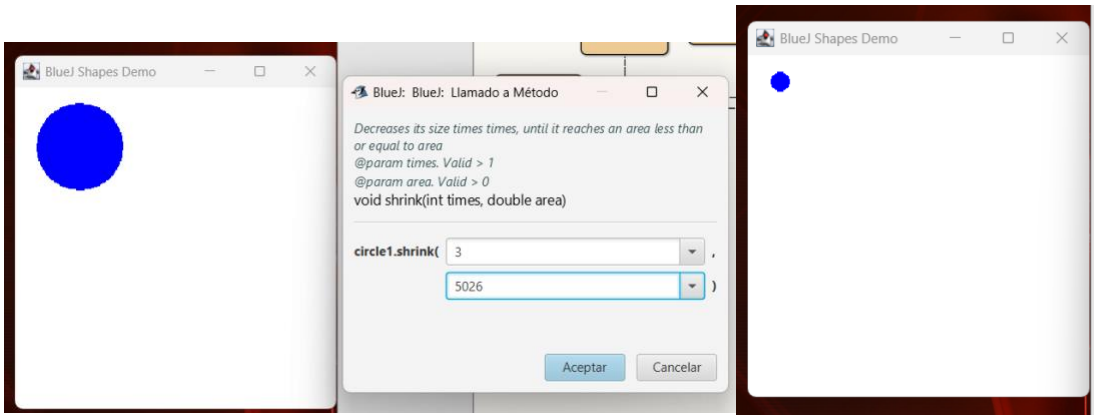
```java
/**
 * Increases the area of   the circle by a percentage [0...100]).
 * @param percentage. Valid [0...100]
 */
public void bigger(int percentage) {
    if(percentage >= 0 && percentage <= 100) {
        double areaActual = area();
        double nuevaArea = areaActual * (1 + (percentage / 100.0));
        double radio = Math.sqrt(nuevaArea / PI);
        double nuevoDiametro = 2 * radio;
        changeSize((int)Math.round(nuevoDiametro));
    }
}
```



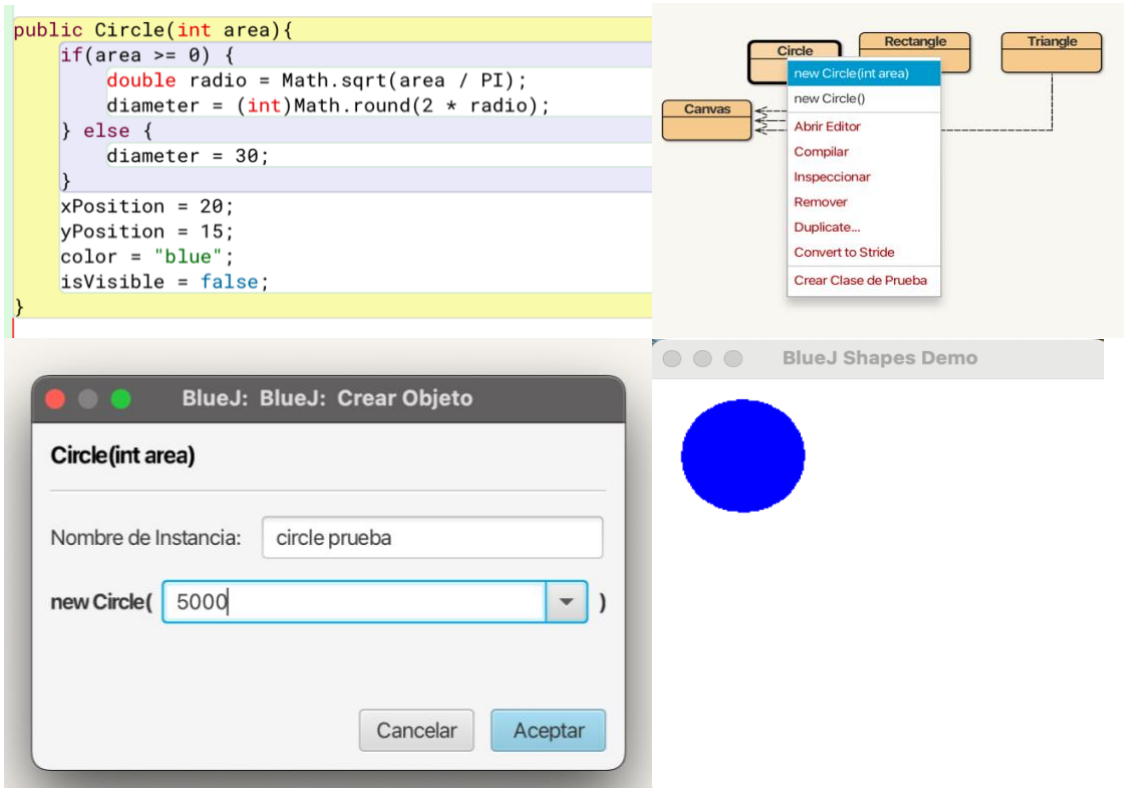
3. Desarrollen en `Circle` el método `shrink(times: int, area: int)` (disminuye su tamaño `times` veces, hasta llegar a un area menor o igual a `area`). ¡Pruébenlo! Capturen tres pantallas.

```java
/**
 * Decreases its size times times, until it reaches an area less than
 * or equal to area
 * @param times. Valid > 1
 * @param area. Valid > 0
 */
public void shrink(int times, double area) {
    if (times > 1 && area > 0) {
        for (int i = 0; i < times; i++) {
            double areaActual = area();
            double nuevoArea = areaActual / times;
            double radio = Math.sqrt(nuevoArea / PI);
            double nuevoDiametro = 2 * radio;
            changeSize((int)Math.round(nuevoDiametro));
        }
    }
}
```

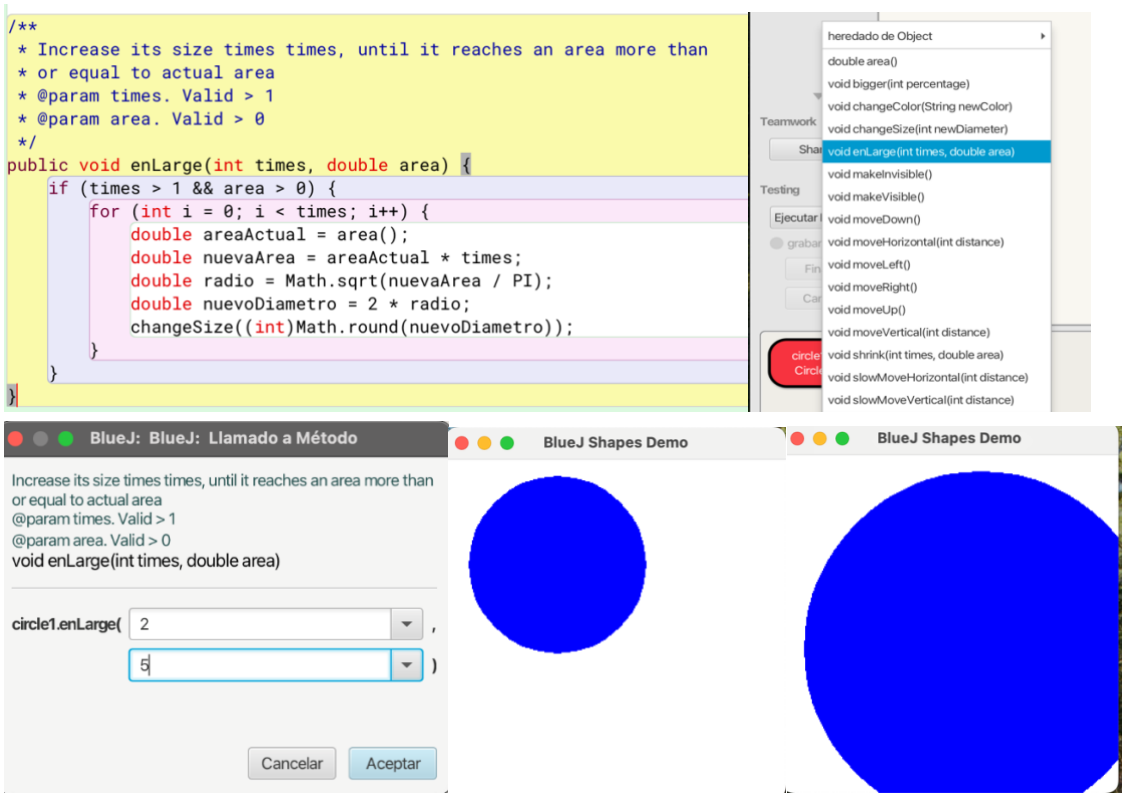




4. Desarrollen en `Circle` un nuevo creador que permita crear un círculo con un área específica. ¡Pruébenlo! Capturen una pantalla.



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.

area [Show source in BlueJ]

```
public double area()
```

Return the area of the circle

Returns:

the area of the circle

bigger [Show source in BlueJ]

```
public void bigger(int percentage)
```

Increases the area of the circle by a percentage [0...100]).

Parameters:

`percentage` - . Valid [0...100]

shrink [Show source in BlueJ]

```
public void shrink(int times,
                   double area)
```

Decreases its size times times, until it reaches an area less than or equal to area

Parameters:

`times` - . Valid > 1

`area` - . Valid > 0

enLarge [Show source in BlueJ]

```
public void enLarge(int times,
                    double area)
```

Increase its size times times, until it reaches an area more than or equal to actual area

Parameters:

`times` - . Valid > 1

`area` - . Valid > 0

Method Summary

All Methods | Instance Methods | Concrete Methods

Modifier and Type	Method	Description
double	**area**()	Return the area of the circle
void	**bigger**(int percentage)	Increases the area of the circle by a percentage [0...100]).
void	**changeColor**(**String** newColor)	Change the color.
void	**changeSize**(int newDiameter)	Change the size.
void	**enLarge**(int times, double area)	Increase its size times times, until it reaches an area more than or equal to actual area
void	**makeInvisible**()	
void	**makeVisible**()	
void	**moveDown**()	Move the circle a few pixels down.
void	**moveHorizontal**(int distance)	Move the circle horizontally.
void	**moveLeft**()	Move the circle a few pixels to the left.
void	**moveRight**()	Move the circle a few pixels to the right.
void	**moveUp**()	Move the circle a few pixels up.
void	**moveVertical**(int distance)	Move the circle vertically.
void	**shrink**(int times, double area)	Decreases its size times times, until it reaches an area less than or equal to area
void	**slowMoveHorizontal**(int distance)	Slowly move the circle horizontally.
void	**slowMoveVertical**(int distance)	Slowly move the circle vertically

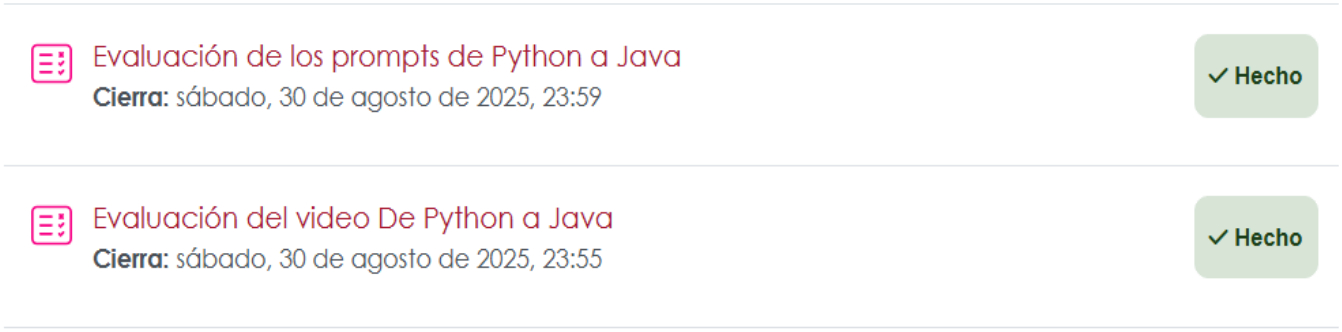
2. De Python a Java [En lab01.doc]

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.

Nicolas Bernal:

Juan Bogotá:

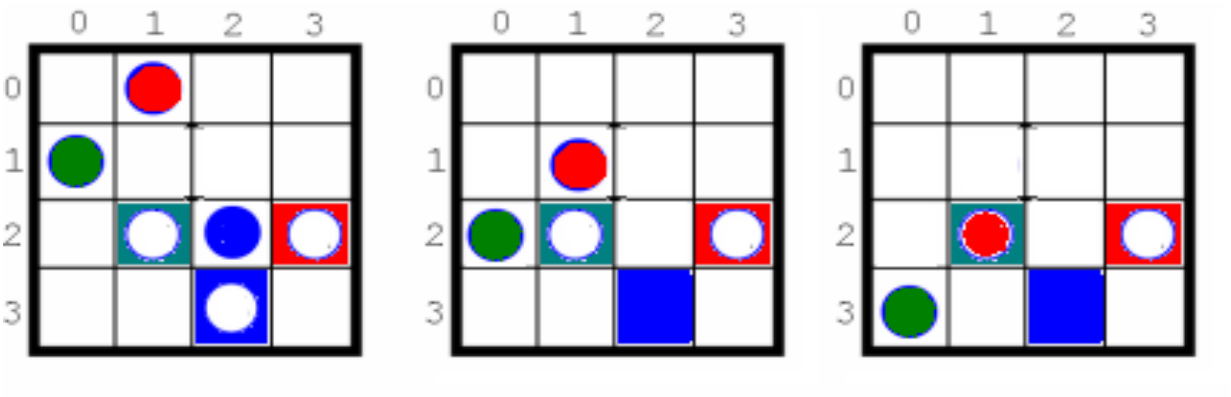


3. MINI MARBEL GAME

El objetivo de este punto es desarrollar una mini-aplicación para una versión simplificada del juego Marbel Game. Inspirado en un problema **F** de la maratón internacional 2007. VINCULO
El tablero de este juego es cuadrado (NXN) e inicia con igual número de canicas y de huecos (M), cada par de color diferente. El objetivo de este juego es hacer que todas las canicas caigan en el hueco de su mismo color. Los movimientos consisten en levantar un lado del tablero para hacer que las canicas se deslicen una posición. Las canicas caen si pasan por un agujero. Si una canica cae en un agujero que es de su color, el agujero desaparece. Si una canica cae en un agujero que no es de su color, el juego termina.

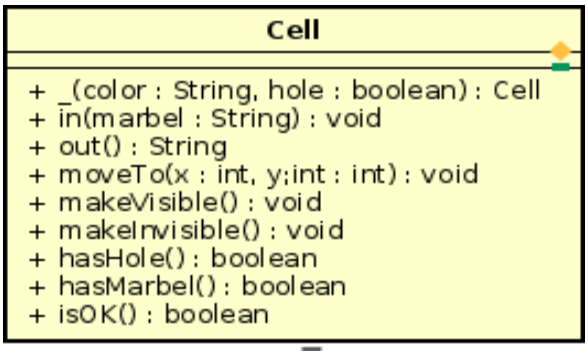
En el ejemplo se muestra un tablero de 4 x 4, con tres canicas. Los movimientos realizados fueron levantar el norte dos veces.

[6]Menú: View → Show Code Pad



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

En este punto vamos a crear las **casillas** del tablero. El diseño gráfico lo definen ustedes. Las casillas deben ofrecer los siguientes métodos:



- **Ciclo 1:**
  - _()
  - hasHole
- **Ciclo 2:**
  - in
  - out
  - hasMarbel
  - isOK
- **Ciclo 3:**
  - moveTo
  - makeVisible
  - makeInvisible
- **Notas:**
  - En `in` entra una canica del color indicado.
  - En `out` sale la canica que está en la casilla y se retorna su color, si es posible. Si no es posible, retorna la cadena vacía.
  - `isOK` retorna falso en el agujero está una canica de color diferente

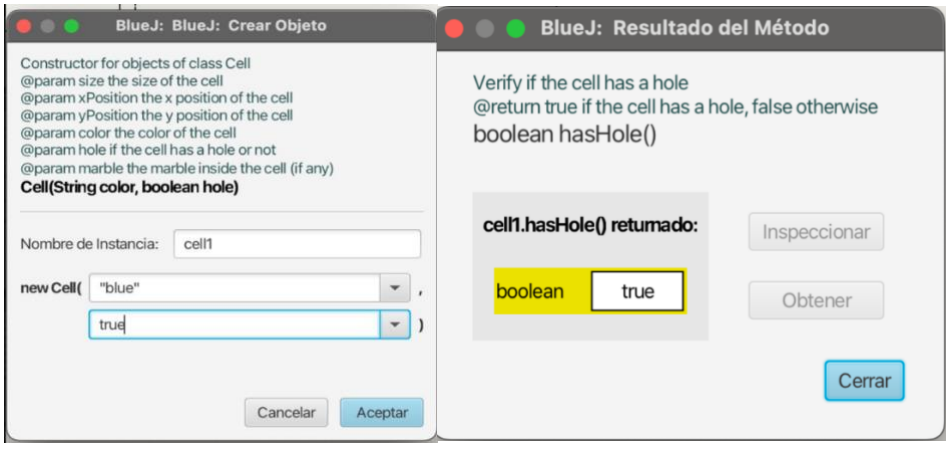
1. Inicie la construcción únicamente con los **atributos**. Justifique su selección. Agregue pantallazo con los atributos.

```java
import java.awt.geom.*;
import java.lang.Math;
import java.awt.Polygon;

/**
 * A cell of the marble game that can be manipulated and that draws itself on a canvas.
 *
 * @author  Juan Daniel Bogotá
 * @author  Nicolas Felipe Bernal
 * @version 1.0.  (27 May 2024)
 */

public class Cell{
    private String color;
    private boolean hole;
    private boolean isVisible;
    private int xPosition;
    private int yPosition;
    private int size;
    private Circle marble;
}
```

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.

   Ciclo 1:

```java
/**
 * Constructor for objects of class Cell
 * @param size the size of the cell
 * @param xPosition the x position of the cell
 * @param yPosition the y position of the cell
 * @param color the color of the cell
 * @param hole if the cell has a hole or not
 * @param marble the marble inside the cell (if any)
 */
public Cell(String color, boolean hole) {
    this.size = 20;
    this.xPosition = 0;
    this.yPosition = 0;
    this.color = color;
    this.hole = hole;
    this.isVisible = false;
    this.marble = null;
}

/**
 * Verify if the cell has a hole
 * @return true if the cell has a hole, false otherwise
 */
public boolean hasHole() {
    return hole;
}
```



Se probó el creador con los dos parámetros que se señalan en el diagrama de clase de la clase Cell, además se probó el método hasHole() con ambas posibilidades de respuesta.

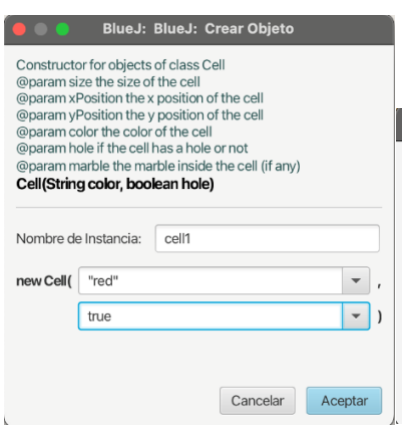
Ciclo 2:

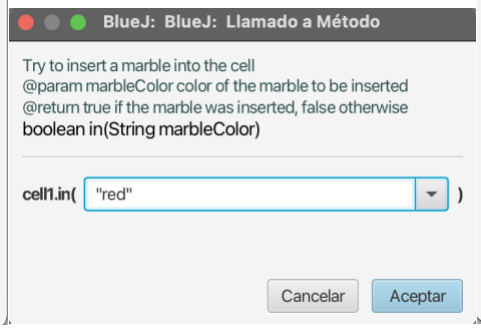
```java
/**
     * Try to insert a marble into the cell
     * @param marbleColor color of the marble to be inserted
     * @return true if the marble was inserted, false otherwise
     */
    public boolean in(String marbleColor) {
        if (marble == null) {
            int marbleSize = (int)(size * 0.8);
            marble = new Circle();
            marble.changeSize(marbleSize);
            marble.changeColor(marbleColor);
            marble.moveHorizontal(xPosition + (size - marbleSize)/2);
            marble.moveVertical(yPosition + (size - marbleSize)/2);
            if (isVisible) {
                marble.makeVisible();
            }
            return true;
        }
        return false;
    }
```

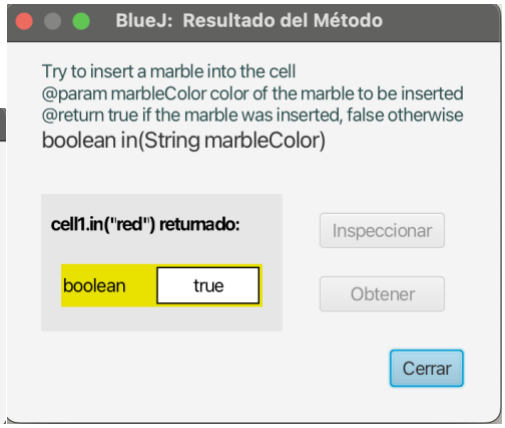
```java
/**
 * Remove the marble from the cell
 * @return color of the marble that was removed, or an empty string if there was no marble
 */
public String out() {
    if (marble != null) {
        String marbleColor = marble.getColor();
        marble.makeInvisible();
        marble = null;
        return marbleColor;
    }
    return "";
}
```

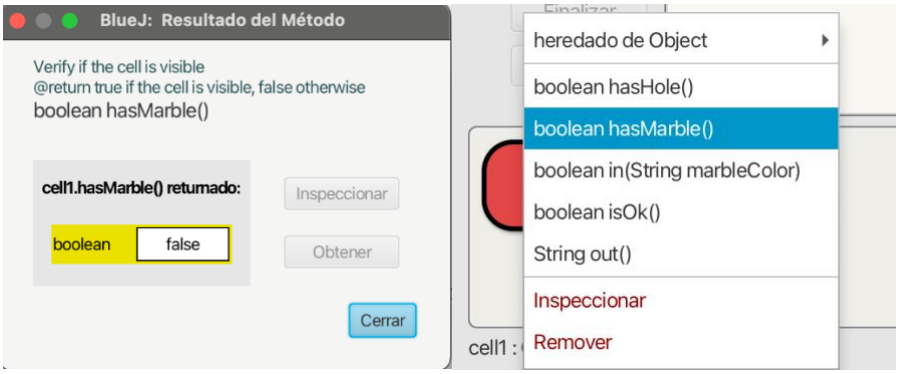
```java
/**
 * Verify if the cell has a marble
 * @return true if the cell has a marble, false otherwise
 */
public boolean hasMarble() {
    return marble != null;
}
```

```java
/**
 * Verify if the cell is in a correct state
 * @return true if the cell is in a correct state, false otherwise*/
public boolean isOk() {
    if (!hole || marble == null) {
        return true;
    }
    return color.equals(marble.getColor());
}
```



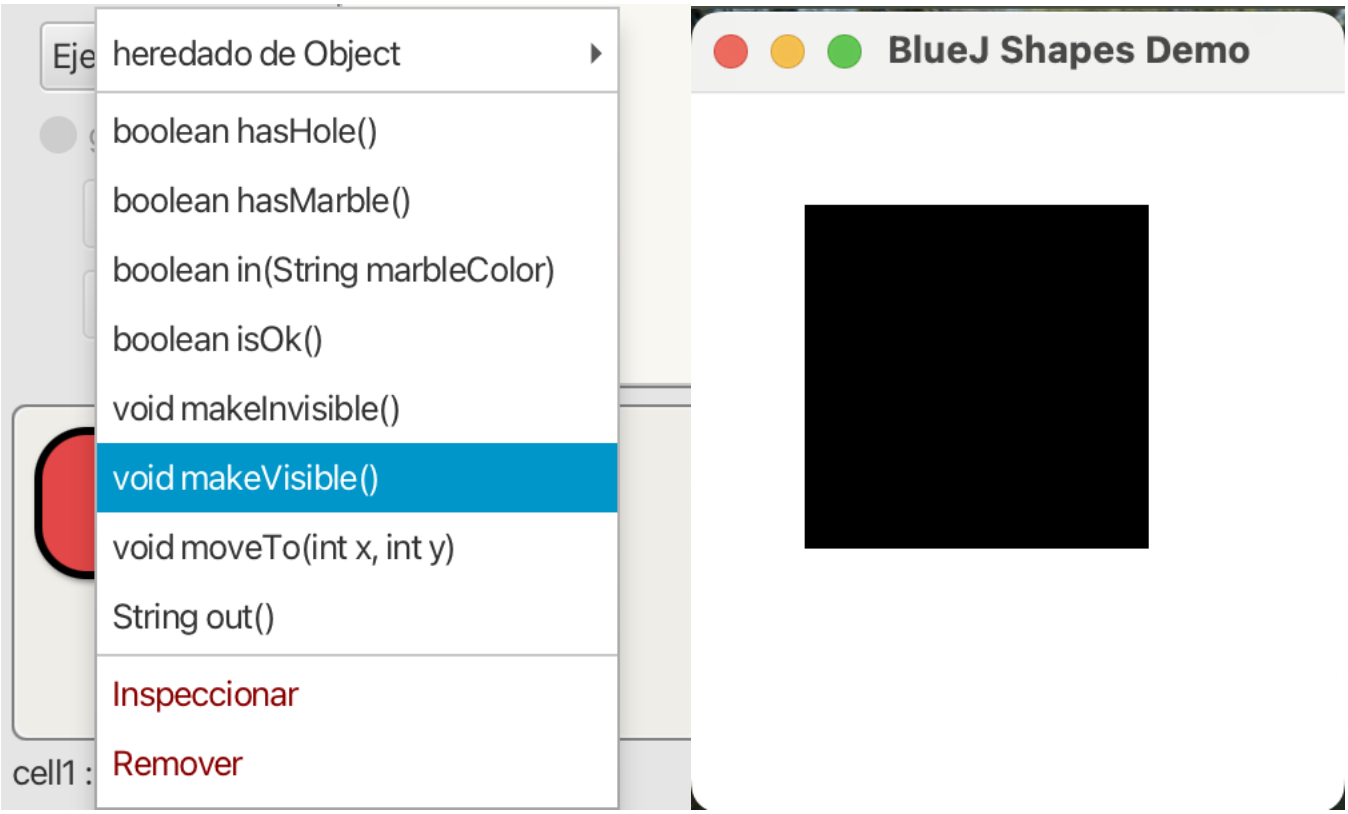


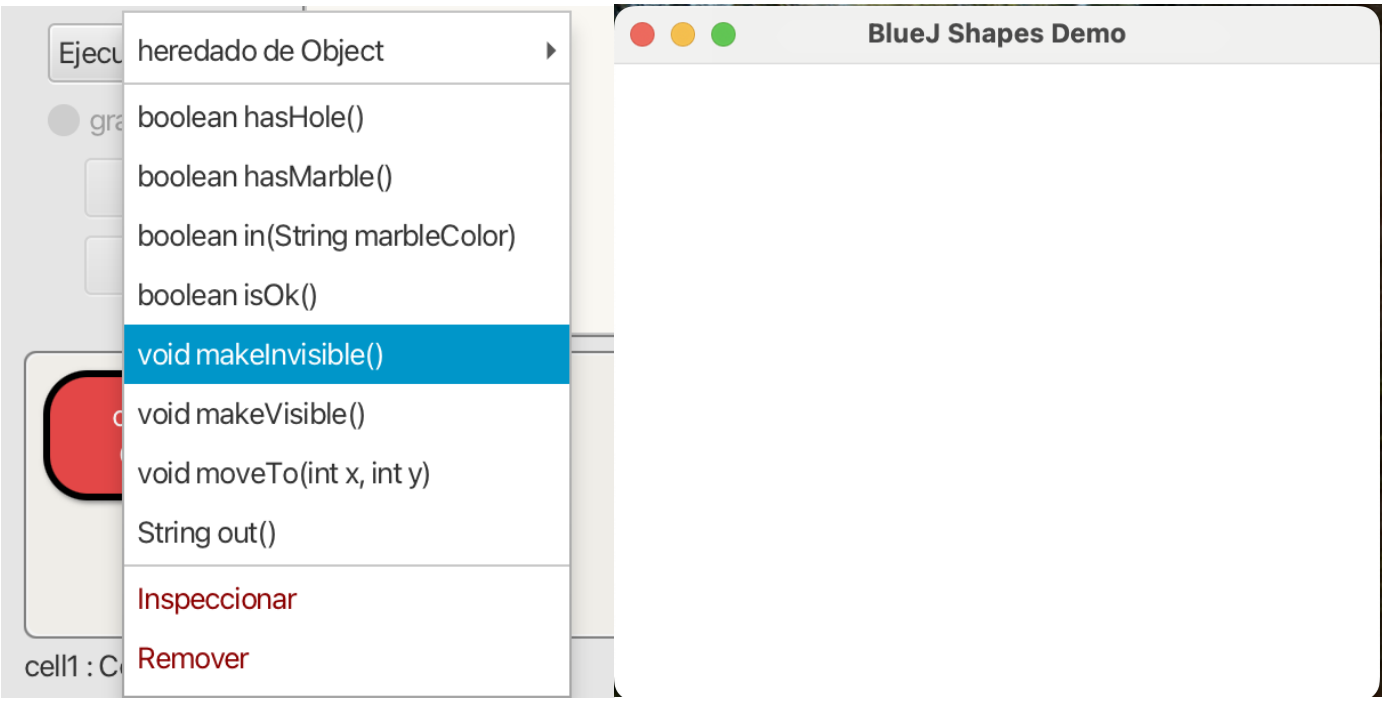




Se hicieron dos pruebas, al método in(String marbleColor) y retorna true, ya que permite insertar la canica si fuera roja y además porque no hay mas canicas en el hoyo, la otra prueba fue a el método hasMarble(), que dio false, ya que aun no existe la canica.

Ciclo 3:





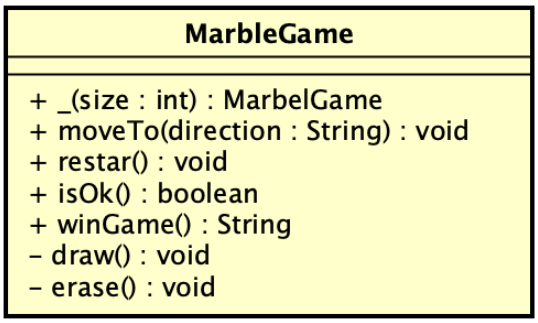
Se hicieron dos pruebas, al método makeVisible() y muestra la figura en el canvas, la otra prueba fue a el método makeInvisible(),que muestra el canvas en blanco ya que vuelve invisible la figura.

B. Definiendo y creando una nueva clase. MarbelGame. [En lab01.doc][En .java]

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

Requisitos funcionales	Requisitos de interfaz
▪ Crear el estado inicial. ▪ Realizar los movimientos. ▪ Reiniciar el juego. ▪ Consultar el estado del juego (un mensaje con el número de canicas en sus agujeros). ▪ Informar cuando alguien gana el juego (un mensaje de felicitación).	▪ Los movimientos se identifican por el lado ('North', 'South', 'East', 'West'). ▪ En caso de 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.

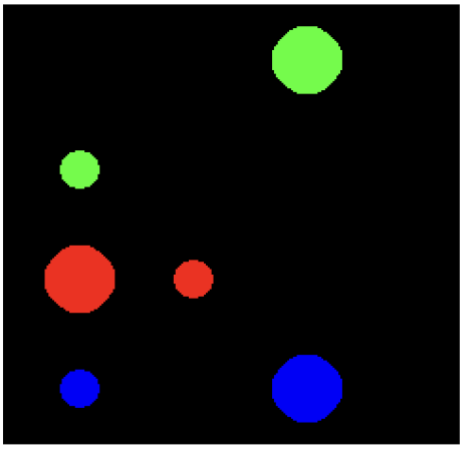


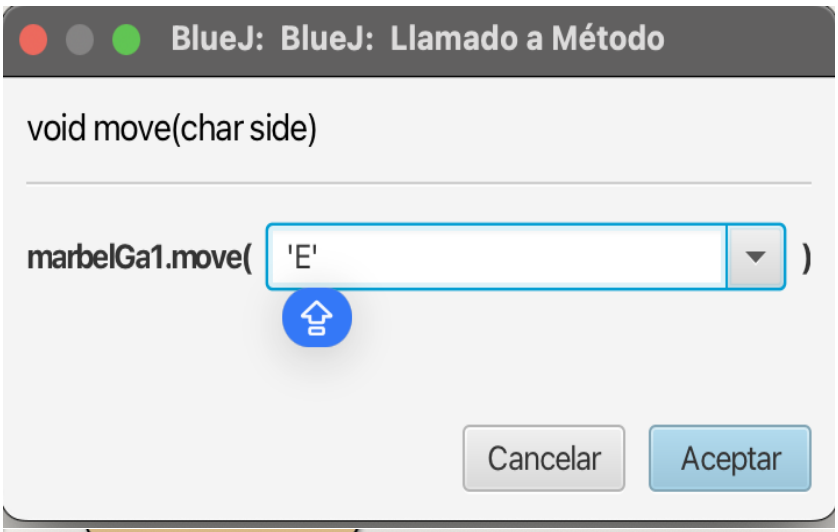
2. Planifiquen la construcción considerando algunos mini-ciclos.

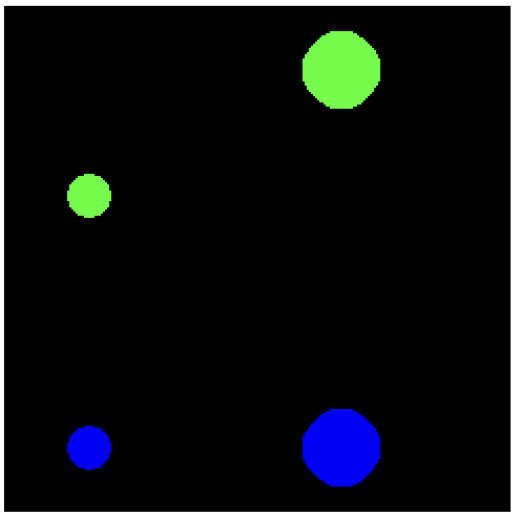
    - Ciclo 1:
        - _()
    - Ciclo 2:
        - draw
        - erase
        - moveTo
    - Ciclo 3:
        - Restar
        - isOk
    - Ciclo 4:
        - winGame

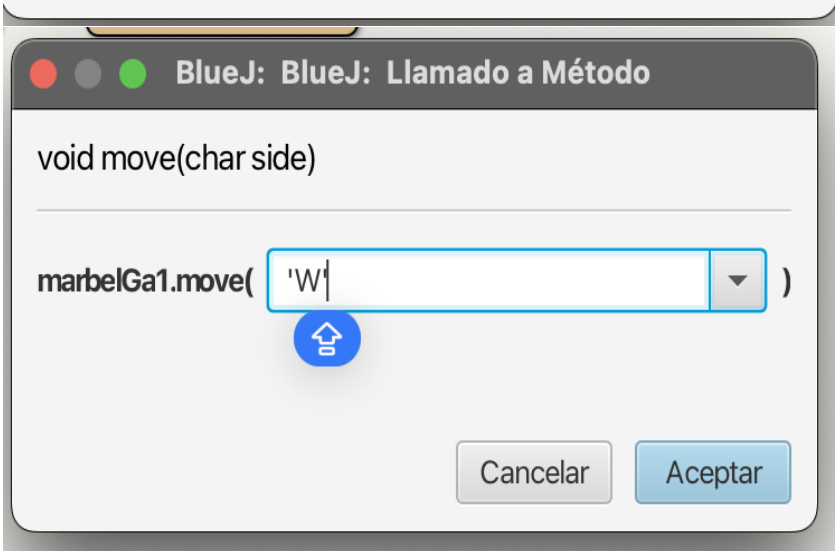
3. Implementen la clase. Al final de cada mini-ciclo realicen una prueba indicando su propósito. Capturen las pantallas relevantes.
4. Indiquen las extensiones necesarias para reutilizar la clase `Cell` y el paquete `shapes`.
5. Expliquen.

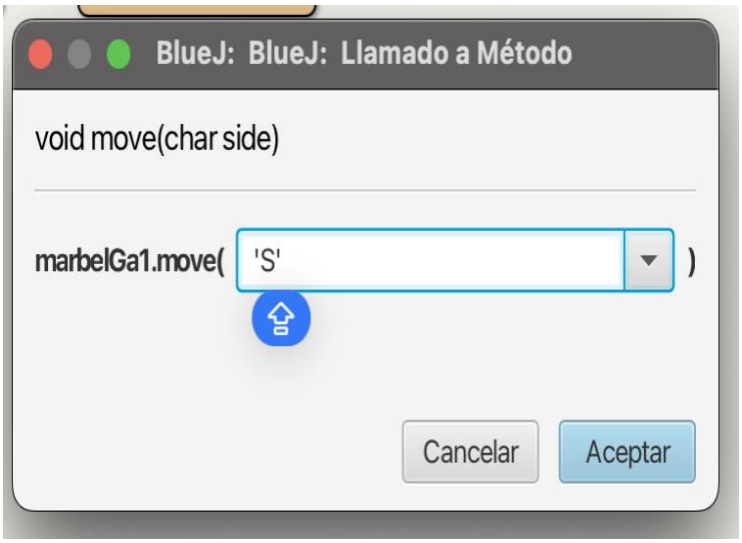

BlueJ: BlueJ: Llamado a Método
void move(char side)
marbelGa1.move(char side)
Cancelar
Aceptar

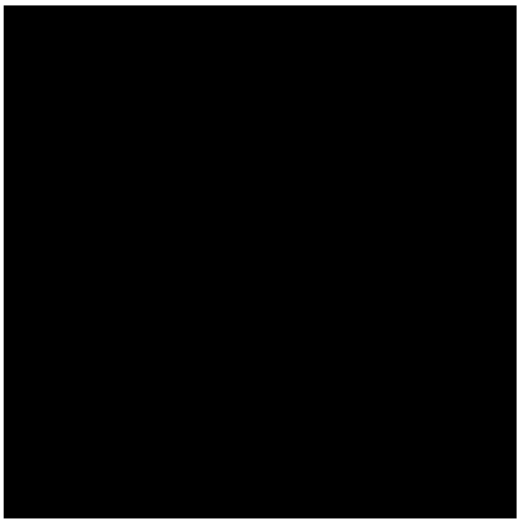



BlueJ: BlueJ: Llamado a Método
void move(char side)
marbelGa1.move('E')
Cancelar
Aceptar




BlueJ: BlueJ: Llamado a Método
void move(char side)
marbelGa1.move('W')
Cancelar
Aceptar





C BONO. Nuevos requisitos funcionales. MarbelGame.[En lab01.doc][En .java]]

Extiendan la mini-aplicación MarbelGame:

- 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)

Juan Daniel Bogotá Fuentes 28 horas, más o menos 3 horas por día (jueves, lunes,martes, miércoles, jueves, viernes, sábado).
Nicolas Felipe Bernal Gallo 28 horas (Jueves[3 horas], lunes[2 horas], martes[3 horas], miércoles[1 hora], jueves[6 horas], viernes[6 horas], sábado[7 horas])

2. ¿Cuál es el estado actual del laboratorio? ¿Por qué?

Incompleto, porque no realizamos el bono y tuvimos errores dentro del laboratorio lo que no permite que sea un laboratorio completo.

3. Considerando las prácticas XP del laboratorio, ¿cuál fue la más útil? ¿por qué?

Dos programadores trabajan juntos en el código (Pair Programming). Esta es la práctica XP que nos fue más útil a la hora de realizar este laboratorio, puesto que nosotros nos intercambiábamos la labor de escribir código y revisar el código de la otra persona. También porque intercambiamos ideas y conocimiento que fuimos adquiriendo en el auto estudio de cada persona.

4.¿Cuál consideran fue el mayor logro? ¿Por qué?

Entender todos los requerimientos que exigía el laboratorio a partir de las preguntas que fueron expuestas para su resolución. No dejando por nuestra parte lugar a la ambigüedad de nuestras respuestas.

5.¿Cuál consideran que fue el mayor problema técnico? ¿Qué hicieron para resolverlo?

El mayor problema técnico que presentamos a la hora de realizar este laboratorio fue la no interconectividad que presenta Blue J, donde no podíamos escribir en el mismo archivo al mismo tiempo, lo cual nos llevó a buscar otro entorno virtual que satisficiera esa necesidad. Luego de esto volver a Blue J y generar así el “Shapes” final.

6.¿Qué hicieron bien como equipo? ¿Qué se comprometen a hacer para mejorar los resultados?

Repartir de una buena forma el trabajo para que ambos aprendamos y además tengamos la misma carga de trabajo. Nos comprometemos a repartir mejor las horas por día para hacer el laboratorio

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

ChatGPT. (s/f). Chatgpt.com. Recuperado el 30 de Agosto de 2025, de https://chatgpt.com/

Barragán, A. (2023, octubre 12). Introducción a Java: Operadores. *Openwebinars.net*. https://openwebinars.net/blog/introduccion-a-java-operadores/

Vypirailenko, A. (2023, julio 21). *División entera Java*. CodeGym. https://codegym.cc/es/groups/posts/es.696.division-entera-java

Los 11 mejores chatbots de IA para 2025. (s/f). Getguru.com. Recuperado el 6 de febrero de 2025, de https://www.getguru.com/es/reference/best-ai-chatbots

El mejor fue Getguru.com ya que una de las cosas que más nos tomó tiempo, fue encontrar un logo de envíos adecuado para realizarlo en Bluej.