Requisitos:

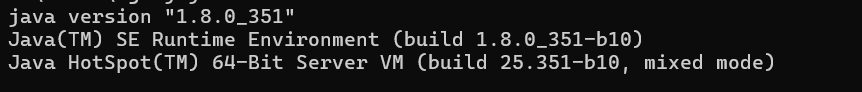
Instalar java, desde Oracle (usar vpn para abrir en Venezuela)

<https://www.oracle.com/java/technologies/downloads/?er=221886#jdk24-windows>  
también instalamos visual studio code, nuestro IDE.

Para hacer un proyecto en Java usando vsCode, debemos instalar el JAVA DEVELOPMENT KIT (paquete de desarrollo de java), para poder desarrollar en nuestro IDE.

<https://code.visualstudio.com/docs/languages/java>  
  
luego de instalarlo, abrimos la terminal de consola (bash, powershell, CMD) y ejecutamos el siguiente comando para verificar si ya se instalo:  
  
java –version

y damos ENTER.

Debería aparecer esto en consola:  
  
  
un archivo JAVA tiene la extensión .java

Creamos un proyecto de JAVA en VSCODE:

>java: create java project   
no built tools  
  
selecionamos la carpeta en la cual se creara el proyecto

Le damos un nombre al proyecto.

CONFIGURAMOS EL PATH de Windows, a veces **javac no está en tu variable de entorno PATH**

**🔥 Solución rápida:**

**1. Encuentra la ruta de instalación de tu JDK:**

* Por defecto, OpenJDK (Temurin) se instala en:

C:\Program Files\Eclipse Adoptium\jdk-17.0.15.6-hotspot\bin

* Si no está allí, busca la carpeta jdk-17 en:
  + C:\Program Files\Eclipse Adoptium\
  + C:\Program Files\Java\

**2. Agrega la ruta del JDK al PATH:**

* Abre el menú de Windows y escribe **"Editar variables de entorno"** → Selecciona **"Variables de entorno"**.
* En **"Variables del sistema"**, busca PATH y haz clic en **"Editar"**.
* Haz clic en **"Nuevo"** y pega la ruta del bin de tu JDK (ejemplo):

C:\Program Files\Eclipse Adoptium\jdk-17.0.15.6-hotspot\bin

* Haz clic en **Aceptar** en todas las ventanas.

**3. Verifica en una terminal NUEVA:**

* Abre una **nueva terminal** (PowerShell o CMD) y ejecuta:

javac -version

* Deberías ver algo como:

javac 17.0.15

# **Explicación Detallada del Código Pac-Man**

Voy a explicarte todo el código como si fueras un niño que está aprendiendo a programar, partiendo por los conceptos básicos y luego profundizando en el código.

## Conceptos Básicos de Programación Orientada a Objetos (POO)

### ¿Qué es una clase?

Imagina que una clase es como un molde para galletas. El molde define la forma que tendrán las galletas, pero no es una galleta en sí. En este código, PacMan y SoundManager son clases.

### ¿Qué es un objeto?

Un objeto es una galleta hecha con el molde. Es una "instancia" de una clase. Por ejemplo, cuando creamos pacman = new Block(...), estamos haciendo un objeto Pac-Man usando la clase Block.

### ¿Qué es un método?

Un método es como una acción que puede hacer un objeto. Por ejemplo:

* pacman.updateVelocity() es un método que actualiza la velocidad de Pac-Man.
* moveSound.playSound() es un método que reproduce un sonido.

### ¿Qué es una variable?

Una variable es como una caja donde guardamos información. Por ejemplo:

* int score = 0; es una variable que guarda la puntuación.
* Image pacmanRightImage; es una variable que guarda la imagen de Pac-Man mirando a la derecha.

### ¿Qué es una librería?

Una librería es como una caja de herramientas que nos da funciones ya hechas. En este código usamos:

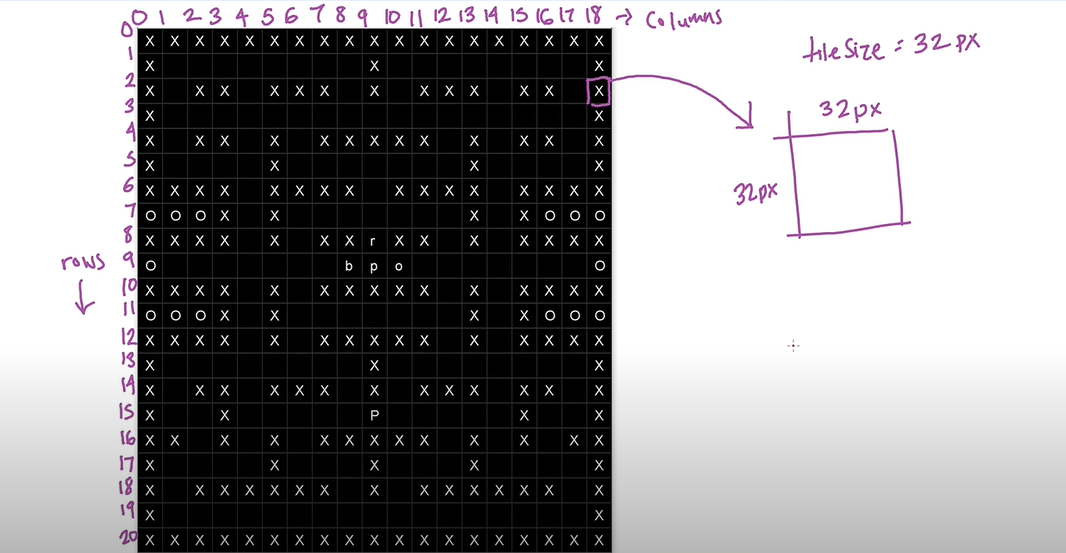
* java.awt.\* - Para gráficos y ventanas
* javax.swing.\* - Para botones y componentes de interfaz
* java.util.\* - Para listas y colecciones de datos

1. **Fase inicial:**

**Se nos crearan varias carpetas, una de ellas llamada src con el archivo App.java**

Importamos la librería import javax.swing.JFrame;

para crear una ventana, y le indicamos su tamaño

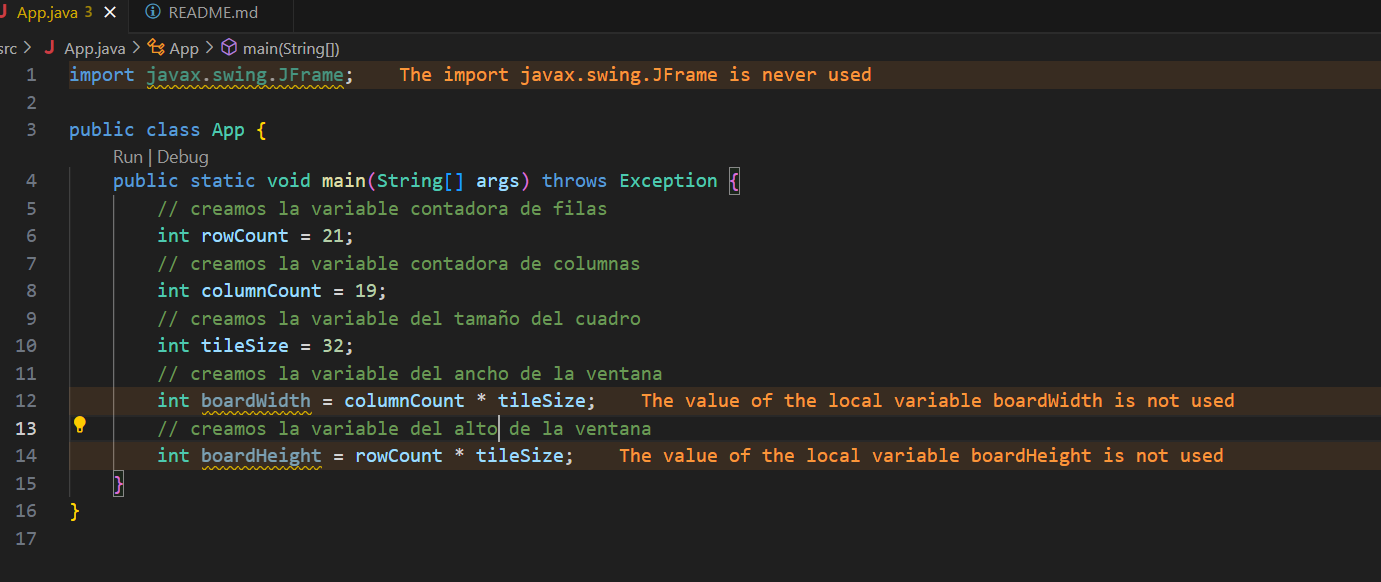


Representaremos la ventaja del juego en cuadros

Cada uno de los cuadros es de 32x32px

Entonces el ancho de la ventana seira, ancho\*19 columnas

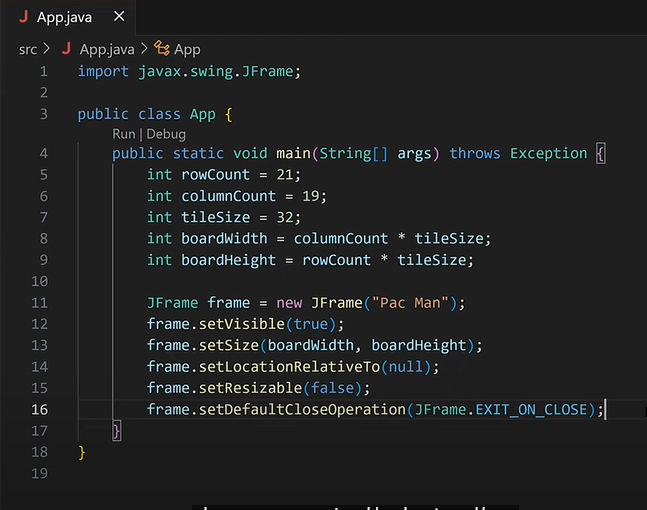
Alto seria alto\*21 filas.



Ya al tener estos datos, comenzaremos a crear nuestra ventana:

JFrame frame = new JFrame("Pac\_man");

Esto nos creara una ventaja con el nombre Pac man.



1. Hacemos el jPanel.

Creamos otro archivo de tipo class al cual llamaremos PacMan.java

import java.awt.\*;

import java.awt.event.\*;

import java.util.HashSet;

import java.util.random.\*;

import javax.swing.\*;

public class PacMan extends JPanel{

}

## Explicación del Código Pac-Man

### 1. Estructura Principal

public class PacMan extends JPanel implements ActionListener, KeyListener {

// Todo el código va aquí

}

Esto significa que:

* PacMan es una clase que representa nuestro juego
* extends JPanel - Hereda de JPanel, que es un componente para dibujar
* implements ActionListener, KeyListener - Puede escuchar eventos (teclas y tiempo)

### 2. Estados del Juego

enum GameState {

MAIN\_MENU, // Menú principal

COLOR\_SELECTION, // Selección de color

PLAYING, // Jugando

PAUSED, // Pausado

GAME\_OVER, // Fin del juego

PLAYER\_DIED // Jugador perdió una vida

}

El juego puede estar en diferentes "estados" como cuando estás en el menú, jugando o cuando pierdes.

### 3. Clase Block (Bloque)

class Block {

int x, y; // Posición

int width, height; // Tamaño

Image image; // Imagen a mostrar

// ... más propiedades

}

Esta clase representa cualquier elemento del juego: Pac-Man, fantasmas, paredes, etc. Cada uno tiene posición, tamaño y una imagen.

### 4. Variables Importantes

// Tamaño del tablero

private int rowCount = 21; // Filas

private int columnCount = 19; // Columnas

private int tileSize = 32; // Tamaño de cada celda

// Imágenes

private Image wallImage; // Paredes

private Image blueGhostImage; // Fantasma azul

// ... más imágenes

// Elementos del juego

HashSet<Block> walls; // Paredes

HashSet<Block> foods; // Comida

HashSet<Block> ghosts; // Fantasmas

Block pacman; // Jugador

**5. Mapa del Juego**

private String[] tileMap = {

"XXXXXXXXXXXXXXXXXXX",

"X X X",

// ... más líneas del mapa

};

Este es el diseño del laberinto donde:

* 'X' es una pared
* ' ' es comida
* 'P' es Pac-Man
* 'b', 'o', 'p', 'r' son fantasmas

### 6. Métodos Principales

public PacMan() {

// Configura la ventana

// Carga imágenes y sonidos

// Crea botones

// Inicia el juego

}

#### Movimiento de Pac-Man

private void movePacman() {

// Actualiza posición según teclas presionadas

// Verifica colisiones con paredes

// Maneja el teletransporte en bordes

}

#### Movimiento de Fantasmas

private void moveGhosts() {

// Cada fantasma se mueve según su estrategia

// Algunos persiguen, otros van aleatorio

// Verifican colisiones con Pac-Man

}

#### Dibujado del Juego

@Override

protected void paintComponent(Graphics g) {

// Dibuja todo según el estado del juego

// Menú, juego, pausa, game over

}

### 7. Manejo de Sonidos

La clase SoundManager se encarga de reproducir y parar sonidos:

class SoundManager {

public void playSound(String path, boolean loop) {

// Reproduce un sonido

// Puede loopear (repetirse)

}

public void stop() {

// Detiene el sonido

}

}

### 8. Main (Punto de Entrada)

public static void main(String[] args) {

// Crea la ventana principal

// Añade el juego

// Muestra todo

}

## **¿Cómo Funciona Todo Junto?**

1. **Inicio**: El programa comienza en main(), crea una ventana y el juego.
2. **Menú**: Muestra botones para empezar, seleccionar color, etc.
3. **Juego**:
   * Tú controlas a Pac-Man con las teclas
   * Los fantasmas se mueven solos
   * Comes puntos y ganas puntos
   * Si te tocan los fantasmas, pierdes vidas
4. **Fin**: Cuando pierdes todas las vidas, termina el juego.

## ¿Por qué se Hizo Así?

* **Clases**: Para organizar el código (Pac-Man, fantasmas, sonidos son objetos separados)
* **Métodos**: Cada acción importante tiene su propio método (mover, dibujar, etc.)
* **Variables**: Guardan el estado del juego (puntos, vidas, posición)
* **POO**: Para modelar el mundo del juego como objetos que interactúan

# Guía Completa para Identificar Elementos de POO en Código Java

Voy a crear una guía detallada para que puedas identificar y entender cada componente de la Programación Orientada a Objetos (POO) en este código y en general.

## 1. Clases

\*\*¿Qué es?\*\* Una clase es una plantilla o molde que define las características y comportamientos de un tipo de objeto.

\*\*Cómo identificarlas:\*\*

- Busca la palabra clave `class`

- Empiezan con mayúscula por convención (PascalCase)

\*\*Ejemplos en el código:\*\*

```java

public class PacMan extends JPanel implements ActionListener, KeyListener {

// Contenido de la clase

}

class Block {

// Contenido de la clase

}

class SoundManager {

// Contenido de la clase

}

```

\*\*Qué hace cada clase en este código:\*\*

- `PacMan`: La clase principal del juego, maneja toda la lógica y visualización

- `Block`: Representa cualquier elemento del tablero (Pac-Man, fantasmas, paredes)

- `SoundManager`: Maneja la reproducción de sonidos

## 2. Constructores

\*\*¿Qué es?\*\* Un método especial que se ejecuta al crear un objeto (instancia de una clase). Inicializa el objeto.

\*\*Cómo identificarlos:\*\*

- Tiene el mismo nombre que la clase

- No tiene tipo de retorno (ni siquiera void)

\*\*Ejemplos en el código:\*\*

```java

// Constructor de PacMan

public PacMan() {

// Configuración inicial del juego

setPreferredSize(new Dimension(boardWidth, boardHeight));

// ... más configuraciones

}

// Constructor de Block

Block(Image image, int x, int y, int width, int height) {

this.image = image;

this.x = x;

// ... inicializa propiedades

}

```

\*\*Qué hace cada constructor:\*\*

- `PacMan()`: Configura el tamaño del panel, carga imágenes, crea botones, inicializa temporizadores

- `Block()`: Crea un nuevo elemento del juego con su imagen, posición y tamaño

## 3. Métodos

\*\*¿Qué es?\*\* Funciones asociadas a una clase que definen su comportamiento.

\*\*Cómo identificarlos:\*\*

- Tienen un tipo de retorno (void si no retorna nada)

- Siguen la convención camelCase

- Pueden tener parámetros entre paréntesis

\*\*Ejemplos en el código:\*\*

```java

// Método para mover Pac-Man

private void movePacman() {

// Lógica de movimiento

}

// Método para verificar colisiones

public boolean collision(Block a, Block b) {

return a.x < b.x + b.width &&

a.x + a.width > b.x &&

// ... más condiciones

}

// Método para dibujar

@Override

protected void paintComponent(Graphics g) {

// Lógica de dibujo

}

```

\*\*Tipos de métodos:\*\*

- \*\*De instancia\*\*: Operan sobre un objeto específico (`pacman.reset()`)

- \*\*Estáticos\*\*: Pertenecen a la clase, no a instancias (no hay en este código)

- \*\*Sobrescritos (override)\*\*: Redefinen métodos de la clase padre (como `paintComponent`)

## 4. Variables

\*\*¿Qué es?\*\* Almacenan datos del objeto o de la clase.

\*\*Cómo identificarlas:\*\*

- Declaradas con un tipo (int, String, etc.)

- Pueden ser de instancia o locales a métodos

**\*\*Tipos de variables:\*\***

- \*\*De instancia\*\*: Pertenece a cada objeto

```java

class Block {

int x; // Variable de instancia

int y;

}

```

- \*\*De clase (static)\*\*: Compartida por todos los objetos (no hay en este código)

- \*\*Locales\*\*: Dentro de métodos

```java

public void move() {

int oldX = pacman.x; // Variable local

// ...

}

```

**## 5. Palabras con @ (Anotaciones)**

\*\*¿Qué son?\*\* Proporcionan metadatos sobre el código que sigue.

\*\*Anotaciones comunes en el código:\*\*

```java

@Override // Indica que el método sobrescribe uno de la clase padre

protected void paintComponent(Graphics g) {

// ...

}

```

Otras anotaciones comunes (aunque no aparecen aquí):

- `@Deprecated`: Indica que el elemento está obsoleto

- `@SuppressWarnings`: Suprime advertencias del compilador

**## 6. Herencia**

\*\*¿Qué es?\*\* Cuando una clase recibe características de otra.

\*\*Cómo identificarla:\*\*

- Palabra clave `extends`

```java

public class PacMan extends JPanel {

// Hereda de JPanel

}

```

**## 7. Interfaces**

\*\*¿Qué es?\*\* Un contrato que la clase debe implementar.

\*\*Cómo identificarla:\*\*

- Palabra clave `implements`

```java

public class PacMan implements ActionListener, KeyListener {

// Debe implementar métodos de estas interfaces

}

```

## 8. Enumeraciones (enum)

\*\*¿Qué es?\*\* Tipo especial que define un conjunto de constantes.

\*\*Ejemplo en el código:\*\*

```java

enum GameState {

MAIN\_MENU, // Constantes

COLOR\_SELECTION,

PLAYING,

// ...

}

```

## Guía Rápida de Identificación

| Elemento | Características de Identificación | Ejemplo |

|---------------|-----------------------------------------------|---------|

| Clase | `class NombreClase` | `class PacMan` |

| Constructor | `NombreClase(...)` sin tipo retorno | `PacMan()` |

| Método | `tipoRetorno nombreMetodo(...)` | `void move()` |

| Variable | `tipo nombreVariable` | `int score` |

| Instancia | `new Constructor()` | `new Block()` |

| Herencia | `extends` | `extends JPanel` |

| Interface | `implements` | `implements ActionListener` |

| Anotación | `@nombre` antes de elemento | `@Override` |

## ¿Qué Hace Cada Cosa? - Resumen

1. \*\*Clase\*\*: Define qué es un tipo de objeto (sus datos y comportamientos)

- Ejemplo: `Block` define qué es un elemento del juego

2. \*\*Constructor\*\*: Inicializa el objeto cuando se crea

- Ejemplo: `PacMan()` configura el juego al inicio

3. \*\*Método\*\*: Acciones que puede realizar el objeto

- Ejemplo: `movePacman()` actualiza su posición

4. \*\*Variable\*\*: Datos que almacena el objeto

- Ejemplo: `score` guarda la puntuación actual

5. \*\*Herencia\*\*: Permite reutilizar código de otra clase

- Ejemplo: `PacMan` hereda de `JPanel` para ser un componente visual

6. \*\*Interfaces\*\*: Define comportamientos que la clase debe implementar

- Ejemplo: `KeyListener` obliga a implementar métodos para teclado

7. \*\*Anotaciones\*\*: Da información adicional al compilador

- Ejemplo: `@Override` indica que el método reemplaza uno heredado