



**SANTO  
TOMÁS**  
INSTITUTO PROFESIONAL



**SANTO<sup>®</sup>  
TOMÁS**

INSTITUTO PROFESIONAL

**PATRONES DE DISEÑO**





# Patrones de Diseño GOF

Soluciones probadas a problemas recurrentes en el diseño de software orientado a objetos.



# Objetivo de la clase

Comprender el propósito de los patrones GOF, identificar su clasificación



# ¿Qué es un patrón de diseño?

Un **patrón de diseño** es una solución **reutilizable y probada** para resolver un problema común en el diseño de software orientado a objetos.

*“Un patrón no es el código que escribes, es **la forma en que piensas al escribirlo.**”*



# ¿Quiénes son los “GOF”? (Gang of Four)

Los “**Gang of Four**” (GOF) son los autores del libro clásico sobre patrones de diseño:

**"Design Patterns: Elements of Reusable Object-Oriented Software" (1994)**

## **Autores:**

- Erich Gamma
- Richard Helm
- Ralph Johnson
- John Vlissides

*“Este libro cambió para siempre la forma de diseñar software orientado a objetos.”*



# Clasificación de los Patrones GOF

Los 23 patrones de diseño GOF se agrupan en tres categorías principales:

Tipo	¿Qué resuelven?	Ejemplos
<b>Creacionales</b>	Formas de <b>crear objetos</b> flexiblemente.	Singleton, Prototype
<b>Estructurales</b>	Formas de <b>componer clases/objetos</b> en estructuras más complejas.	Adapter, Bridge
<b>Comportamiento</b>	Formas de <b>controlar flujos de trabajo o comunicación</b> .	Observer, Strategy



# Patrón Singleton – ¿Qué problema resuelve?

El patrón **Singleton** asegura que una clase tenga **una única instancia** en todo el sistema, y proporciona un punto de acceso global a esa instancia.

## ¿Cuándo usarlo?

- Configuración global de la aplicación
- Acceso centralizado a recursos (como logs o base de datos)
- Control de acceso o autenticación

## ¿Qué pasa si no se usa?

- Se pueden crear **instancias duplicadas** innecesarias.
- Puede haber **inconsistencias** en datos compartidos.

“Un solo objeto  
para gobernarlos a  
todos.”

ONE OBJECT TO  
RULE THEM ALL







# Diagrama UML – Patrón Singleton



# Lazy vs Eager Instantiation en Singleton

## Lazy Instantiation (perezosa)



- La instancia **se crea solo cuando se llama a getInstance()** por primera vez.
-  Ahorra memoria si no se usa.
-  **No es seguro en entornos multihilo** (a menos que uses sincronización).

```
if (instancia == null) {  
    instancia = new Singleton();  
}
```



# Lazy vs Eager Instantiation en Singleton

## Eager Instantiation (ansiosa)

- La instancia **se crea al momento de cargar la clase**.
-  Seguro en entornos multihilo (por defecto).
-  Usa memoria aunque nunca se ocupe.

```
private static final Singleton instancia = new Singleton();
```



# Lazy vs Eager Instantiation en Singleton

## ¿Cuál conviene?

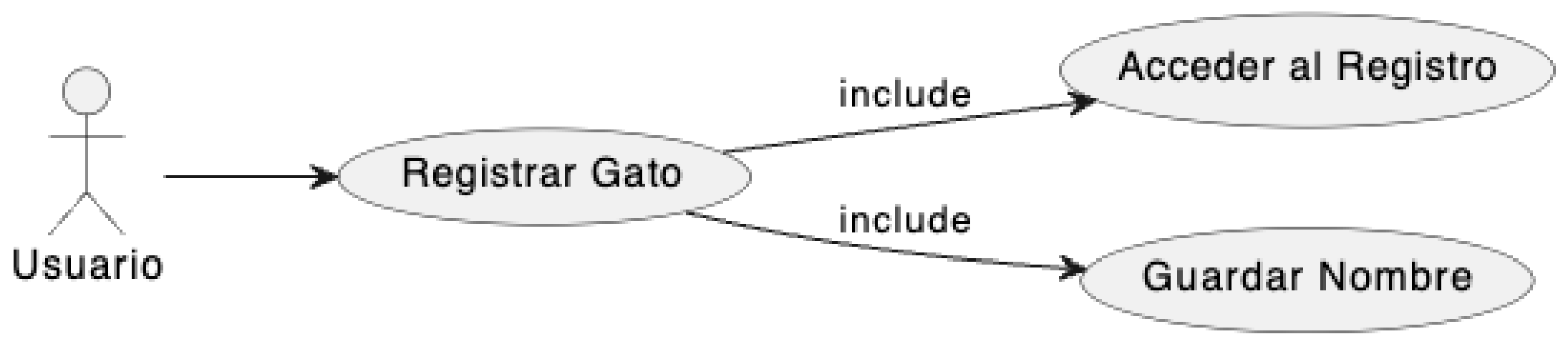
- **Lazy:** si la creación del objeto es costosa y podría no usarse.
- **Eager:** si el objeto es liviano o se necesita sí o sí.

# Código Java – Patrón Singleton (Lazy)

```
public class Singleton {  
    // 1. Instancia estática (aún no creada)  
    private static Singleton instancia;  
  
    // 2. Constructor privado  
    private Singleton() {  
        System.out.println("Instancia creada");  
    }  
  
    // 3. Método de acceso  
    public static Singleton getInstance() {  
        if (instancia == null) {  
            instancia = new Singleton();  
        }  
        return instancia;  
    }  
  
    // 4. Método de ejemplo  
    public void saludar() {  
        System.out.println("Hola desde el Singleton");  
    }  
  
    // 5. Uso  
    public static void main(String[] args) {  
        Singleton s1 = Singleton.getInstance();  
        Singleton s2 = Singleton.getInstance();  
  
        s1.saludar();  
  
        if (s1 == s2) {  
            System.out.println("Ambas instancias son iguales");  
        }  
    }  
}
```

# De UML a Código – Singleton aplicado al Registro de Gatos





---

Caso de uso

# Diagrama de Clases

Clase Singleton que controla el registro de todos los gatos. Solo puede haber una instancia.





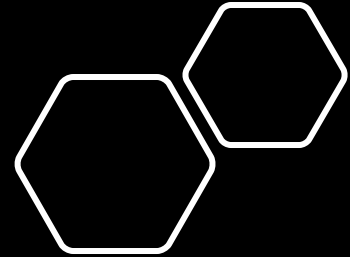
```
// Clase que representa un registro único para gatos
public class RegistroGatos {

    // 1. Instancia estática privada (única en todo el sistema)
    private static RegistroGatos instancia;

    // 2. Constructor privado para evitar creación externa
    private RegistroGatos() {
        System.out.println("Registro de gatos iniciado");
    }

    // 3. Método público para acceder a la única instancia
    public static RegistroGatos getInstancia() {
        if (instancia == null) {
            // Si no existe, la crea
            instancia = new RegistroGatos();
        }
        return instancia;
    }

    // 4. Método que registra un gato (simulado con un print)
    public void registrar(String nombre) {
        System.out.println("🐱 Gato registrado: " + nombre);
    }
}
```



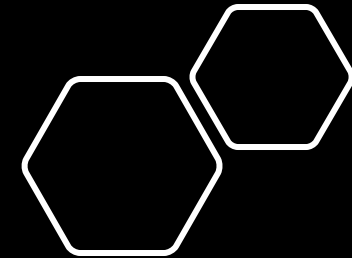
```
// 5. Método main para demostrar cómo funciona el patrón Singleton
public static void main(String[] args) {
```

```
    // Se obtiene el registro y se registra un gato
    RegistroGatos registro1 = RegistroGatos.getInstance();
    String nombre1;
    System.out.println("Ingrese el nombre del gato1:");
    Scanner scanner = new Scanner(System.in);
    nombre1 = scanner.nextLine();
    registro1.registrar(nombre1);
```

```
    // Se vuelve a pedir el registro (no se crea uno nuevo)
    RegistroGatos registro2 = RegistroGatos.getInstance();
    String nombre2;
    System.out.println("Ingrese el nombre del gato2:");
    nombre2 = scanner.nextLine();
    registro2.registrar(nombre2);
    scanner.close();
```

```
    // Verificación: ambos apuntan al mismo objeto
    if (registro1 == registro2) {
        System.out.println("✓ Ambos usan el mismo registro");
    }
```

```
}
```





# Resumen de la clase

Aprendimos qué son los **patrones de diseño**



Conocimos a los "**Gang of Four**" (GOF)



Clasificamos los patrones: **Creacionales** y **Estructurales**



Profundizamos en el patrón **Singleton**:

- Se asegura que exista **una única instancia** de una clase
- Útil para registros, configuraciones, logs
- Vimos cómo se modela en **UML**
- Lo implementamos y probamos en **Java**
- Usamos un ejemplo **realista y divertido** con **gatos**



# Actividad Evaluativa







## Instrucciones

1. Trabaja en **pareja**
2. Elige uno de estos patrones:
  1. Prototype (creacional)
  2. Adapter (estructural)
  3. Bridge (estructural)



# Actividad Evaluativa




Graba un video de **máximo 2 minutos y 30 segundos** explicando:

-  ¿Qué problema resuelve?
-  **Diagrama UML** explicado (puede ser a mano, digital o en pantalla)
-  **Código Java** funcional y explicado
-  **Caso de uso realista**
-  **Ventajas y desventajas**
-  **Recursos visuales** de apoyo (ej.: presentación, esquemas, fragmentos de código bien mostrados)



# Actividad Evaluativa

El video debe tener:

-  **Audio claro** (voz entendible, sin ruido molesto)
-  **Buena calidad visual** (pantalla clara, sin pixelaciones)
-  Formato recomendado: .mp4, .mov, .avi

**Fecha de entrega:** *martes 30 de abril, 23:59*

- **Medio de entrega:** correo

Nº	Criterio	Descripción	Puntaje Máximo
1	Explicación del patrón	Describe qué problema resuelve el patrón, su categoría (creacional / estructural), y cuándo usarlo.	2.0 pts
2	Diagrama UML explicado	Muestra un diagrama UML bien construido y lo explica claramente. Puede ser digital o manuscrito si es legible.	2.0 pts
3	Código Java funcional	Presenta código válido, bien estructurado y explica cómo se relaciona con el patrón.	2.0 pts
4	Caso de uso realista	Aplica el patrón en un ejemplo concreto, creíble y contextualizado (ej.: sistema escolar, ecommerce, etc.).	1.0 pt
5	Claridad y síntesis	Mensaje claro, uso correcto del tiempo (máx. 2:30), sin desorden ni divagaciones.	1.0 pt
6	Calidad de video y audio	Voz clara, sin ruido de fondo. Imagen estable y nítida (se entiende el texto y lo que se muestra en pantalla).	1.0 pt
7	Uso de recursos visuales de apoyo	Utiliza elementos visuales útiles: subtítulos, esquemas, fragmentos de código, apoyo gráfico o presentación.	1.0 pt



“Quien enseña, aprende dos veces.” – Joseph Joubert

“Explicar algo bien es el mejor test de si lo entendiste.”

Dominar un patrón no es copiarlo: es poder enseñarlo con tus propias palabras.”





SANTO  
TOMÁS<sup>®</sup>  
INSTITUTO PROFESIONAL



SANTO<sup>®</sup>  
TOMÁS

INSTITUTO PROFESIONAL

Fin

