

METODOLOGÍAS DE PROGRAMACIÓN I

Patrón creacional *Singleton*

Situación de ejemplo

Una aplicación de hotel maneja habitaciones y empleados y posee las siguientes clases:

```
class Hotel
    Empleado getNuevoEmpleado() . . .
    Habitacion getNuevaHabitacion() . . .

class Empleado
    void prepararHabitacion(Habitacion h) . . .

class Habitacion
    . . .
```

Problema

¿Cuál es el problema del siguiente código?

```
void metodo()  
    empleado = this.getUnEmpleado()  
    habitacion = this.getUnaHabitacion()  
    empleado.prepararHabitacion(habitacion)
```

```
Empleado getUnEmpleado()  
    return FabricaDeHoteles.construirHotel()  
                                   .getNuevoEmpleado()
```

```
Habitacion getUnaHabitacion()  
    return FabricaDeHoteles.construirHotel()  
                                   .getNuevaHabitacion()
```



FactoryMethod

Problema

¿Cuál es el problema del siguiente código?

Estamos creando dos instancias de la clase *Hotel* ¿Por qué?
Porque en la ejecución del método `construirHotel` hay metido un **new**

```
void metodo()  
    empleado = this.getUnEmpleado()  
    habitacion = this.getUnaHabitacion()  
    empleado.prepararHabitacion(habitacion)
```

```
Empleado getUnEmpleado()  
    return FabricaDeHoteles.construirHotel()  
                                .getNuevoEmpleado()
```

```
Habitacion getUnaHabitacion()  
    return FabricaDeHoteles.construirHotel()  
                                .getNuevaHabitacion()
```

Problema

¿Cuál es el problema del siguiente código?

```
void metodo()  
    empleado = this.getUnEmpleado()  
    habitacion = this.getUnaHabitacion()  
    empleado.prepararHabitacion(habitacion)
```

```
Emplado getUnEmpleado()  
    return FabricaDeHoteles.contruirHotel()  
                                .getNuevoEmpleado()
```

```
Habitacion getUnaHabitacion()  
    return FabricaDeHoteles.construirHotel()  
                                .getNuevaHabitacion()
```

¿Cuál es el problema en esta parte del código?

Problema

¿Cuál es el problema del siguiente código?

```
void metodo()  
    empleado = this.getUnEmpleado()  
    habitacion = this.getUnaHabitacion()  
    empleado.prepararHabitacion(habitacion)
```

```
Empleado getUnEmpleado()  
    return FabricaDeHoteles.comprarHotel()  
        .getNuevoEmpleado()
```

```
Habitacion getUnaHabitacion()  
    return FabricaDeHoteles.comprarHotel()  
        .getNuevaHabitacion()
```

El empleado y la habitación fueron creados en dos hoteles distintos.

Problema

¿Cuál es el prob

Dado que *Hotel* pareciera ser una clase tan importante que solo una instancia de ella debiera existir.

¿Existe algún mecanismo que nos garantice que solo se pueda crear una única instancia de una clase?

```
void metodo()
{
    empleado = new Empleado();
    habitacion = this.getHabitacion();
    empleado.prepararHabitacion(habitacion);
}
```

```
Empleado getUnEmpleado()  
    return FabricaDeHoteles.contruirHotel()  
                                .getNuevoEmpleado()
```

```
Habitacion getUnaHabitacion()  
    return FabricaDeHoteles.construirHotel().getNuevaHabitacion()
```

Problema

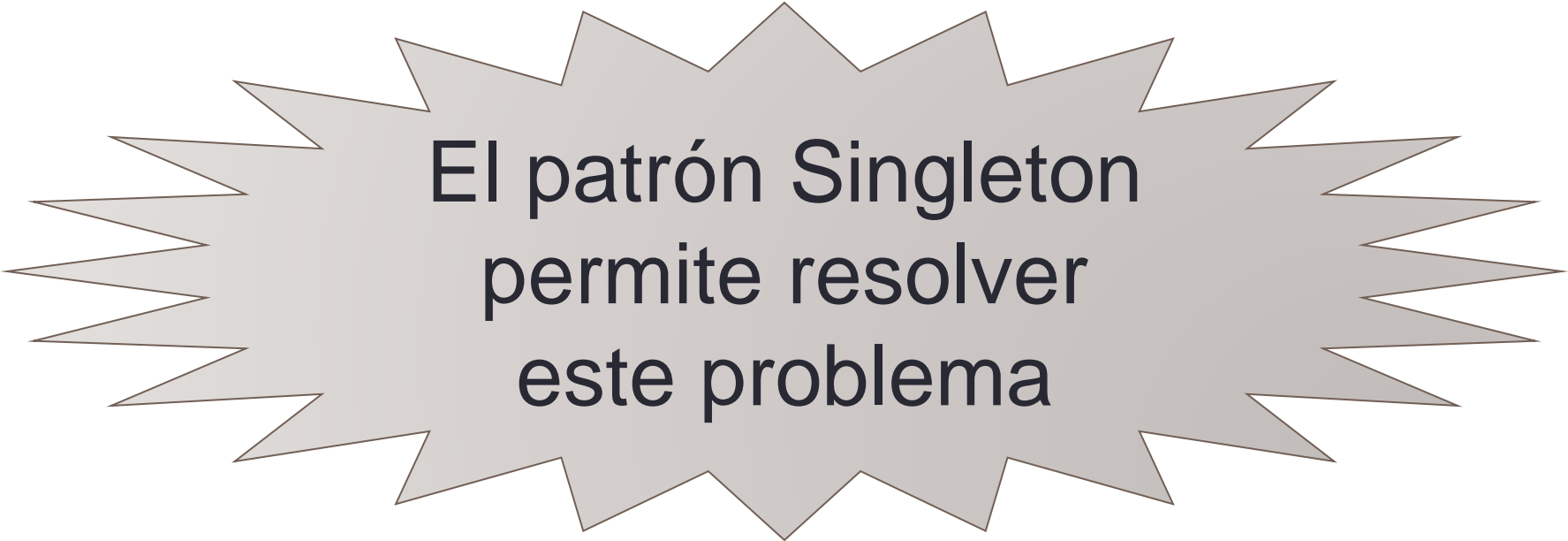
Ningún lenguaje de programación va a impedir que hagamos lo siguiente ya que sintácticamente es válido:

```
hotel1 = new Hotel()  
hotel2 = new Hotel()
```

El operador ***new*** sirve para crear instancias y resulta muy útil... pero en algunos casos se puede convertir en un problema.

Motivación

¿Podemos decirle al lenguaje que nos impida el uso indiscriminado del operador *new*?



El patrón Singleton
permite resolver
este problema

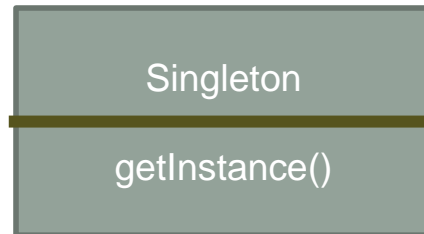
Singleton

Propósito: garantiza que una clase tenga una única instancia y proporciona un único punto de acceso a ella.

Aplicabilidad: usarlo cuando

- Deba haber exactamente una única instancia de una clase y esta deba ser accedida por los clientes desde un punto de acceso conocido.
- La única instancia debería ser extensible mediante herencia, y los clientes deberían ser capaces de usar una instancia extendida sin modificar su código.

Singleton - Estructura



Singleton - Estructura



Singleton - Implementación

```
class Hotel
    private static Hotel unicoHotel = null

    . . .

    private constructor()
        . . .

    public static Hotel getInstance()
        if(unicoHotel == null)
            unicoHotel = new Hotel()
        return unicoHotel
```

Singleton - Implementación

```
class Hotel
    private static Hotel unicoHotel = null

    . . .

    private constructor()
        . . .

    public static Hotel getInstance()
        if(unicoHotel == null)
            unicoHotel = new Hotel()
        return unicoHotel
```

Si ponemos el modificador privado al único constructor de la clase entonces las instancias de esta clase solo pueden ser construidas dentro de ella misma.

Singleton - Implementación

```
class Hotel
    private static Hotel unicoHotel
    . . .

    private constructor()
        . . .
```

Si el único constructor es privado, y por ende no puede ser utilizado desde fuera, ofreceremos un método público y estático que devuelva una instancia de esta clase

```
public static Hotel getInstance()
    if(unicoHotel == null)
        unicoHotel = new Hotel()
    return unicoHotel
```

Singleton - Implementación

```
class Hotel
    private static Hotel unicoHotel = null
```

Notar que la primera vez que se invoca a este método, se crea una instancia la cual es devuelta.

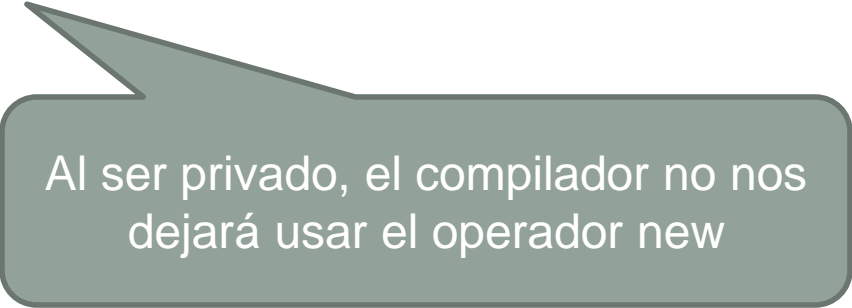
Las siguientes invocaciones devolverán la misma instancia creada previamente, ya que queda almacenada en la variable de clase unicoHotel

```
public static Hotel getInstance()
    if(unicoHotel == null)
        unicoHotel = new Hotel()
    return unicoHotel
```


Singleton - Implementación

Ahora nuestro código "problemático":

```
hotel1 = new Hotel()  
hotel2 = new Hotel()
```



Al ser privado, el compilador no nos dejará usar el operador new

Singleton - Implementación

Ahora nuestro código "problemático":

```
hotel1 = new  
hotel2 = new
```

Por ende, el único mecanismo que nos queda para construir instancias es el método estático `getInstance()`, quien nos garantiza devolver **SIEMPRE** la misma instancia

lo debemos reemplazar por:

```
hotel1 = Hotel.getInstance()  
hotel2 = Hotel.getInstance()
```

Singleton – Ventajas

- Acceso controlado a una única instancia.
- Espacio de nombres reducido, evita contaminar el código con variables globales.
- Permite el refinamiento de operaciones. Se puede crear una subclase de la clase Singleton.
- Permite un número variable de instancias. Es fácil controlar el número de instancias que se crean de una clase.