Patrones de diseño creacionales

- También llamados patrones de construcción.
- Tienen como fundamento abstraer los mecanismos de creación de objetos.
- Vuelven independiente al sistema respecto a la forma en que se crean los objetos; es decir, a los mecanismos de instanciación de las clases concretas.

[*Laurent DEBRAUWER*].

Patrones de diseño creacionales

- Encapsulan el uso de clases concretas y favorecen así el uso de las interfaces en las relaciones entre objetos.
- Por lo tanto, aumentan las capacidades de abstracción en el diseño global del sistema.
- Por ello, simplifican y reducen el uso de constructores concretos de clases; es decir, en el código encontraremos pocas llamadas a constructores.

[*Laurent DEBRAUWER*].

Patrones de diseño creacionales

- Ayudan a hacer a un sistema independiente de cómo se crean, se componen y se representan sus objetos.
- Un patrón de creación de clases usa la herencia para cambiar la clase de la instancia a crear.
- Mientras que un patrón de creación de objetos delega la creación de la instancia en otro objeto.

[*Erich Gamma*].

Patrones de diseño creacionales

- Los patrones de creación se hacen más importantes a medida que los sistemas evolucionan para depender más de la composición de objetos que de la herencia de clases.
- Esto sucede cuando se pasa de codificar una serie de comportamientos fijos a definir un conjunto más pequeño de comportamientos fundamentales que pueden componerse con otros más complejos.

[*Erich Gamma*].

Patrones de diseño creacionales

- Ya que por ejemplo, para crear objetos con un determinado comportamiento es necesario algo más que simplemente crear una instancia de una clase.
- Estos patrones encapsulan el conocimiento sobre las clases concretas que usa el sistema; además ocultan cómo se crean y se asocian las instancias de estas clases.
- Lo que el sistema solo conoce de los objetos son sus interfaces, tal y como las definen sus clases abstractas.

[*Erich Gamma*].

Patrones de diseño creacionales

- Por lo tanto, los patrones de creación dan mucha flexibilidad sobre qué es lo que se crea, quién lo crea y cuándo.
- Permiten configurar un sistema con objetos "producto" que varían mucho en estructura y funcionalidad.
- La creación puede ser estática (esto es, especificada en tiempo de compilación) o dinámica (en tiempo de ejecución).

[*Erich Gamma*].

Patrones de diseño creacionales

- Los patrones de creación de clase comprenden: Abstract Factory y Builder; estos usa herencia para variar la clase del objeto creado.
- Los patrones de creación de objetos comprenden: Factory Method, Prototype y Singleton; estos delegan la creación en otro objeto.

[*Erich Gamma*].

Patrón de diseño singleton

- El patrón Singleton asegura que una clase sólo tiene una instancia (un ejemplar) y proporciona un punto de acceso global a ésta.
- El patrón Singleton permite construir una clase que posee una instancia como máximo.
- El mecanismo que gestiona el acceso a esta única instancia está encapsulado por completo en la clase, y es transparente a los clientes de la clase.

[*Erich Gamma*].

Patrón de diseño singleton

Ejemplos:

- En una partición de disco sólo debería haber un sistema de archivos.
- Un sistema debería tener activo un solo gestor de ventanas.
- Un filtro digital tendrá un solo convertidor de corriente alterna/directa.
- Una empresa debería tener un único sistema de contabilidad.

[*Erich Gamma*].

Patrón de diseño singleton

Aplicabilidad:

- Se requiere exactamente una instancia de una clase.
- Esta instancia debe ser accesible a los clientes desde un solo punto de acceso conocido.
- La única instancia (específicamente su clase) debería ser extensible mediante herencia.
- Los clientes deberían ser capaces de usar una instancia extendida sin modificar su código.

[*Erich Gamma*].

Patrón de diseño singleton

Participantes:

- **Define una operación (getInstance) que permite que los clientes accedan a su única instancia.**
- **getInstance es una operación o método de clase; es decir, debe ser un método estático y publico.**
- **La clase que representa al Singleton es la única responsable de crear su única instancia; por ello debe "bloquear" la creación de otras instancias.**

[*Erich Gamma*].

Patrón de diseño singleton

Consecuencias/Beneficios:

- **Acceso controlado a la única instancia. Puesto que la clase Singleton encapsula su única instancia, puede tener un control estricto sobre cómo y cuándo acceden a ella los clientes.**
- **Espacio de nombres reducido. El patrón Singleton es una mejora sobre las variables globales, ya que evita contaminar el espacio de nombres con variables globales que almacenen las instancias.**

[*Erich Gamma*].

Patrón de diseño singleton

Consecuencias/Beneficios:

- **Permite el refinamiento de operaciones (métodos) y su representación.**
- **Se puede crear una subclase de la clase Singleton, y es fácil configurar una aplicación con una instancia de esta clase extendida.**
- **Podemos configurar la aplicación con una instancia de la clase necesaria en tiempo de ejecución.**

[*Erich Gamma*].

Patrón de diseño singleton

Consecuencias/Beneficios:

- **Permite un número variable de instancias. El patrón hace que sea fácil cambiar de opinión y permitir más de una instancia de la clase Singleton.**
- **Podemos usar el mismo enfoque para controlar el número de instancias que usa la aplicación.**
- **Sólo se necesitaría cambiar la operación(método) que otorga acceso a la instancia del Singleton.**

[*Erich Gamma*].

Patrón de diseño singleton

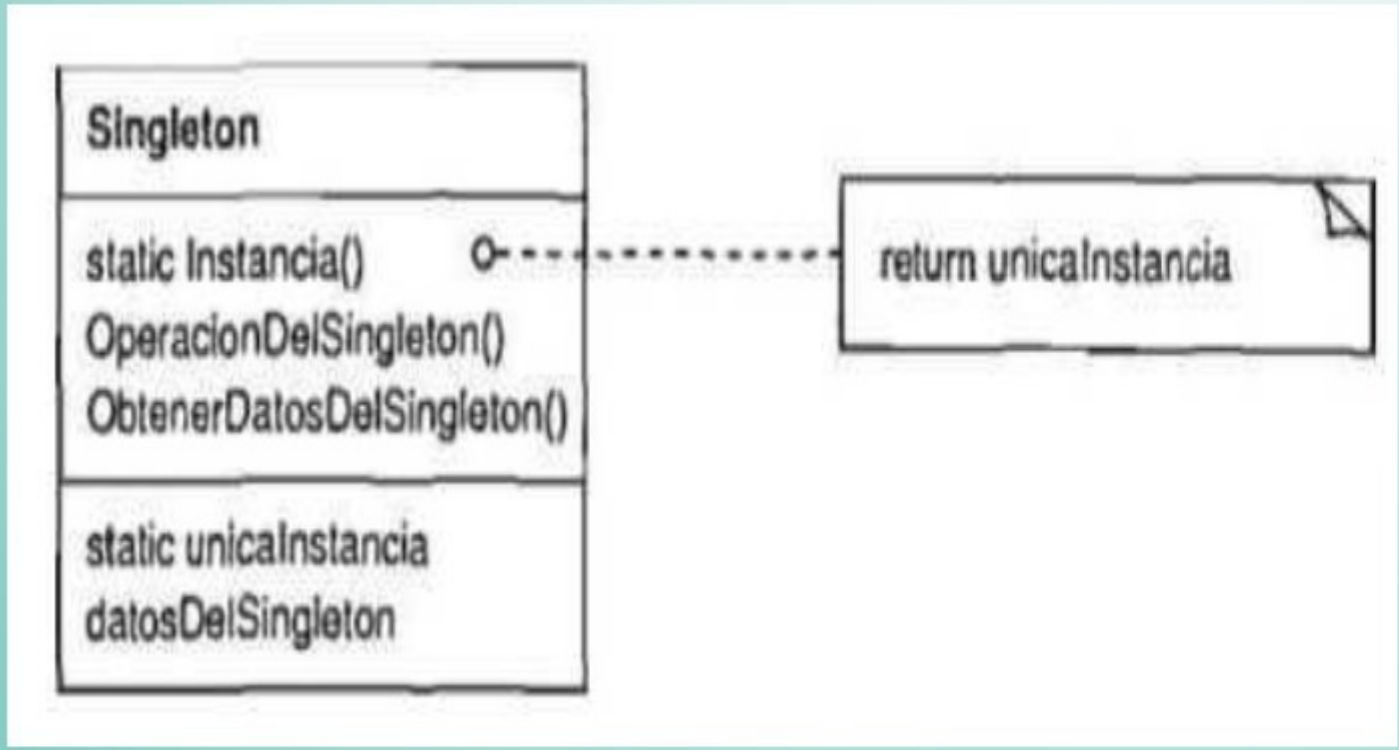
Consecuencias/Beneficios:

- **Más flexible que las operaciones de clase; es decir, otra forma de empaquetar la funcionalidad de un Singleton es usar operaciones (métodos) de clase.**
- **Sin embargo, estas técnicas en algunos lenguajes dificultan cambiar un diseño para permitir más de una instancia de una clase.**
- **Especialmente en aquellos como C++, donde las funciones miembro estáticas nunca son virtuales; por lo que las subclases no las pueden redefinir polimórficamente.**

[*Erich Gamma*].

Patrón de diseño singleton

Estructura genérica (Diseño de clase):



Singleton

- **static** *Singleton Instancia*;
Otros atributos ...

Singleton();
Otros metodos ...
+ static *Singleton* getInstancia();

[*Erich Gamma*].

Patrón de diseño singleton

Implementación:

- Garantizar una **única** instancia. El patrón **Singleton** hace que la única instancia sea una instancia **normal** de la **clase**, pero dicha clase se **escribe** de forma que sólo se pueda crear **una** instancia.
- Esto se hace usualmente al **ocultar** la operación (método) que **crea** la instancia tras una operación de **clase** (método de clase) que **garantice** que sólo se crea una **única** instancia.

[*Erich Gamma*].

Patrón de diseño singleton

Implementación:

- **Esta operación tiene acceso a la variable que contiene la instancia, y se asegura de que la variable está inicializada con dicha instancia antes de devolver su valor.**
- **Este enfoque garantiza que un Singleton se cree e inicialice antes de su primer uso.**

[*Erich Gamma*].

Patrón de diseño singleton

Implementación:

- Crear una subclase de Singleton. El principal problema no es definir la subclase sino instalar su única instancia de manera que los clientes la puedan usar.
- En esencia, la variable que hace referencia a la única instancia debe ser inicializada con una instancia de la subclase.
- La técnica más sencilla es determinar qué subtipo de Singleton queremos usar en el método getInstance de la clase Singleton.

[*Erich Gamma*].

Patrón de diseño singleton

Diseño y Código Ejemplo:


```
public class Singleton {

  //única instancia de esta clase privada y estática
  private static Singleton Instancia;

  //ocultamos el constructor de la clase (como protegido o privado)
  protected Singleton(){
  }

  //método de acceso público a la única instancia de la clase
  public static Singleton getInstancia(){
    if(Instancia==null){
      Instancia=new Singleton();
    }
    return Instancia;
  }

}
```