Programación orientada a objetos

Instancias únicas

{desafío} latam_

Utilizar principios básicos de diseño orientado a objetos para la implementación de una pieza de software acorde al lenguaje Java para resolver un problema de baja complejidad

- Unidad 1: Flujo, ciclos y métodos
- Unidad 2: Arreglos y archivos
- Unidad 3: Programación orientada a objetos
- Unidad 4: Pruebas unitarias y TDD

Te encuentras aquí

{desafío} latam_

¿Qué aprenderás en esta sesión?

- *Comprender el Patrón de diseño Singleton para obtener un código limpio al ejecutar.*
- *Comprender el concepto Synchronized para evitar ejecutar hilos en paralelo.*

¿Qué entendemos por instancia?

/* Instancias */

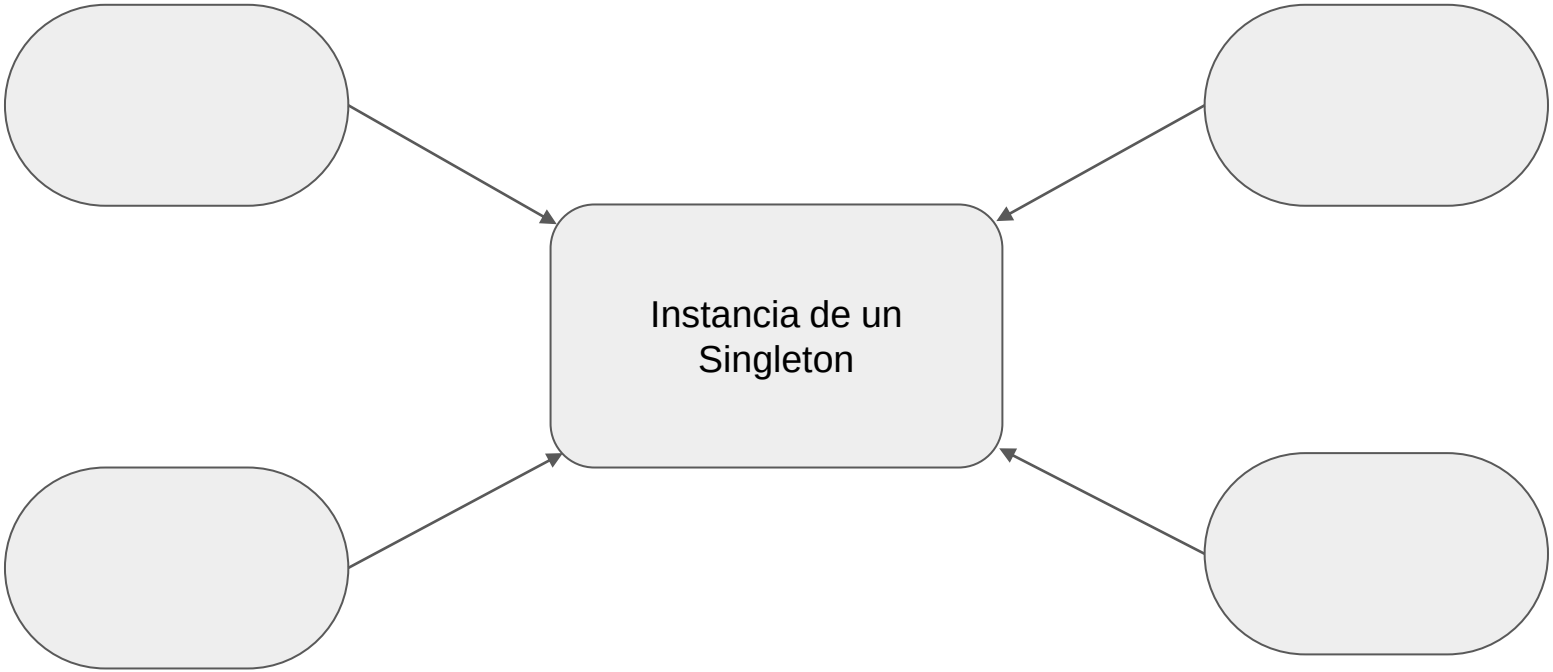
Singleton

- Patrón de diseño de software que se caracteriza porque los objetos del software se rigen por el patrón, solo se instancian una vez y esa instancia es la que se utilizará en toda la aplicación.

- Reduce la cantidad de instancias durante la ejecución de la aplicación y ayuda a tener un código más limpio porque también reduce la cantidad de variables.

- Si hay varias partes de la aplicación que comparten un mismo recurso, podrán acceder a él desde cualquier parte.

Singleton



¿Cómo se aplica el patrón Singleton?

Para aplicarlo se debe crear una clase que se instancie a sí misma en un contexto static y que tenga un constructor privado para que no se pueda acceder a él.

Por último, se debe crear un método static para que las otras clases puedan acceder a la instancia existente.

¿Cómo se aplica el patrón Singleton?

Ejemplo

```
class Configurador{
        //Variable encapsulada y estática donde se almacenará la instancia
única.
        private static Configurador config;
        //Constructor privado
        private Configurador() {}
        //Método estático encapsulador para acceder a la instancia única
        public static Configurador getConfig() {
                if (config== null) {
                        config= new Configurador();
            }
                return config;
         }
}
```

Hilos / Threads

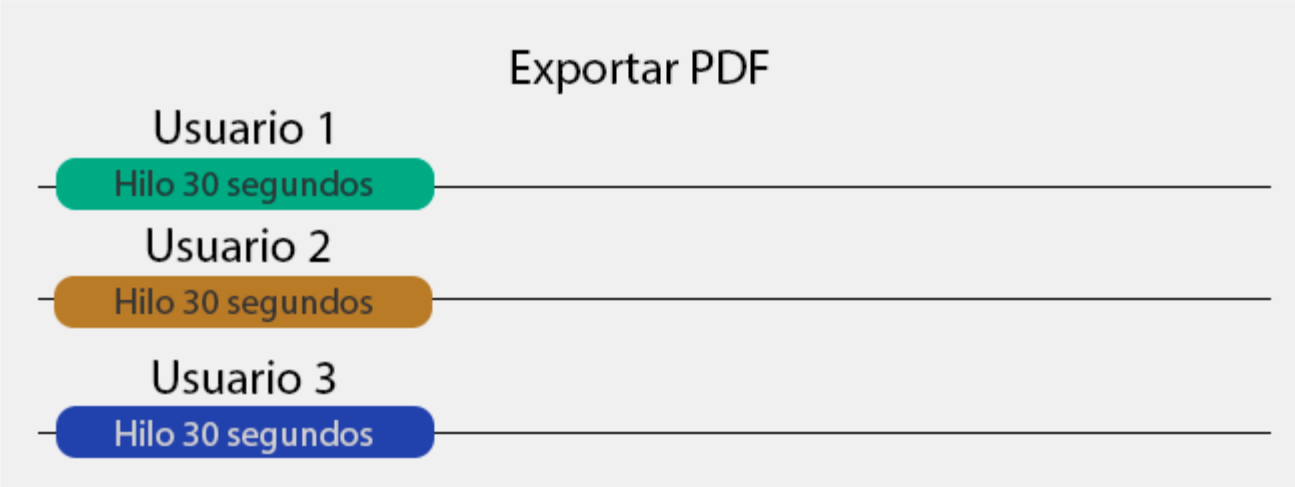
- En el software Java, cada vez que se ejecuta un método o algoritmo, se crea un hilo, hasta que termina la ejecución de este.

- Hay algunos algoritmos que tardan milésimas de segundo y otros que pueden llegar a durar meses ejecutándose, dependiendo de su complejidad.

- Por defecto, en aplicaciones online, cada usuario genera su propio hilo al ejecutar un método, esto permite que la aplicación no se bloquee con un cuello de botella, si dos usuarios quieren ejecutar el mismo método.

Hilos / Threads

Hilos / Threads



Synchronized

Cuando hablamos de Singleton, trabajaremos de una manera diferente a la que tienen por defecto los hilos.

Ejecución del método de manera sincronizada y así evitar tener hilos en paralelo al crear instancias de la misma clase. Para esto, utilizaremos la palabra reservada “synchronized".

Synchronized

```
public class Configurador{
     private static Configurador config;
     private Configurador() {}
     public static Configurador getConfig() {
          if (config == null) {
               synchronized(Configurador.class) {
                    if (config == null) {
                    config = new Configurador();
                    System.out.println("Instancia creada");
               }
          }
     }
     System.out.println("Llamada al Configurador");
     return config
}
```

Synchronized

En el código anterior, podemos ver que si no existe una instancia **config == null** , se procede a iniciar una porción de código que está sincronizada mediante una sentencia if.
Esto con la finalidad de que si hay dos o más usuarios tratando de acceder al mismo método, el primer usuario bloqueará el acceso al segundo.
Para demostrar este ejemplo, podemos llamar al método **getConfig()** 2 veces en la clase Main y corremos nuestro programa.

```
public class Main{
public static void main(String[] args){
     Configurador.getConfig();
     Configurador.getConfig();
}
```

```
--------------------------
Impresión en pantalla:
Instancia Creada
Llamada al configurador
Llamada al configurador
```

Ejercicio guiado

Instituto Educativo

Paso 1

Crear la clase InstitutoEducativo y colocar una variable del mismo tipo que la clase, pero llamada “instance”. Aquí reside el secreto de este patrón, ya que dicha variable es la que se instancia por única vez y se devuelve al cliente.

```
public class InstitutoEducativo {
     private static InstitutoEducativo instance;
}
```

Instituto Educativo

Paso 2

Privatizar el constructor para que no se pueda hacer new InstitutoEducativo() desde otro lugar que no sea dentro de la misma clase.

```
private InstitutoEducativo() {}
```

Instituto Educativo

Paso 3

Para utilizar la única instancia de la clase, los clientes deberán convocar al método getInstance(). Crear la condición if que solo será true la primera vez.

```
public static InstitutoEducativo getInstance() {
     if (instance == null) {
          instance = new InstitutoEducativo();
     }
     return instance;
}
```

Instituto Educativo

Pasos 1, 2 y 3

```
public class InstitutoEducativo {
     private static InstitutoEducativo instance;
     private InstitutoEducativo() {}
     public static InstitutoEducativo getInstance() {
          if (instance == null) {
               instance = new InstitutoEducativo();
          }
          return instance;
     }
}
```

Instituto Educativo

Paso 4

Para llamar al instituto, obtener la instancia del instituto en el Main.

```
public class Main{
     public static void main(String[] args){
          InstitutoEducativo instituto = InstitutoEducativo.getInstance();
     }
}
```

¿Qué hace la palabra reservada synchronized?

{desafío} latam_

Próxima sesión...

- *Desafío evaluado*

{desafío}
latam_

{desafío}
latam_

Academia de
talentos digitales