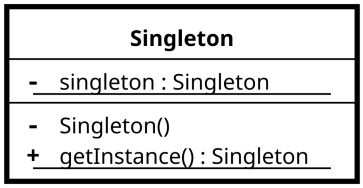
Singleton

Singleton es un patrón de diseño creacional que permite que una clase tenga una única instancia, proporcionando un punto de acceso global a dicha instancia.

Estructura



- Campo estático privado a la clase para almacenar la instancia Singleton.
- Constructor de clase como privado. El método estático de la clase seguirá siendo capaz de invocar al constructor, pero no a los otros objetos.
- Método de creación estático público para obtener la instancia Singleton.
- Implementar una inicialización dentro del método estático. Se debe crear un nuevo objeto en su primera llamada y colocarlo dentro del campo estático. El método deberá devolver siempre esa instancia en todas las llamadas siguientes.

Ejemplo

El siguiente ejemplo se desarrolla en una cafetería en donde únicamente se tiene contratado a un mesero. La clase Mesero es la que corresponde al singleton, pues una vez que el mesero se presenta a trabajar (instancia del mesero) no es posible realizar solicitudes a un mesero diferente (instancia de otros objetos).

```
public class Mesero {
    private static Mesero mesero; //Campo para almacenar la instancia Singelton

    private Mesero() { //Constructor
        System.out.println("Ernesto se puso su uniforme y está listo para trabajar);
    }

    public static Mesero getMesero(){ //Método de creación para obtener la instancia
        if (mesero == null)
            mesero = new Mesero();
        return mesero;
    }
}
```

En el método main es donde se realiza la instancia del mesero, posterior a eso se utiliza nuevamente el método getMesero() cada que alguien realiza una orden, sin embargo, no se crea otro mesero pues únicamente se tiene uno. En la parte final del código se realiza la comprobación de que solo se ha creado un objeto.

```
public class Main {
    public static void main(String[] args) {
```

```
        System.out.println("Nuestro café ha abierto, Ernesto es nuestro único mesero y llegó a s
u jornada de hoy");
        Mesero mesero = Mesero.getMesero();//Creando una instancia de nuestro mesero

        System.out.println("Un cliente solicita un mesero para decirle su orden");
        Mesero meseroOrden1 = Mesero.getMesero();

        System.out.println("Otro cliente solicita un mesero para decirle su orden");
        Mesero meseroOrden2 = Mesero.getMesero();


        System.out.println("¿Es Ernesto (mesero) quien atiende la orden 1 (meseroOrden1)?: " +
(mesero == meseroOrden1));
        System.out.println("¿Es Ernesto (mesero) quien atiende la orden 2 (meseroOrden2)?: " +
(mesero == meseroOrden2));

        System.out.println(mesero);
        System.out.println(meseroOrden1);
        System.out.println(meseroOrden2);

    }
}
```

El resultado de ejecutar el código es el siguiente:

```
Nuestro café ha abierto, Ernesto es nuestro único mesero y llegó a su jornada de hoy
Ernesto se puso su uniforme y está listo para trabajar (constructor)
Un cliente solicita un mesero para decirle su orden
Otro cliente solicita un mesero para decirle su orden
¿Es Ernesto (mesero) quien atiende la orden 1 (meseroOrden1)?: true
¿Es Ernesto (mesero) quien atiende la orden 2 (meseroOrden2)?: true
Mesero@75b84c92
Mesero@75b84c92
Mesero@75b84c92
```

Referencias

- “Singleton.” [Online]. Available: https://refactoring.guru/es/design-patterns/singleton