

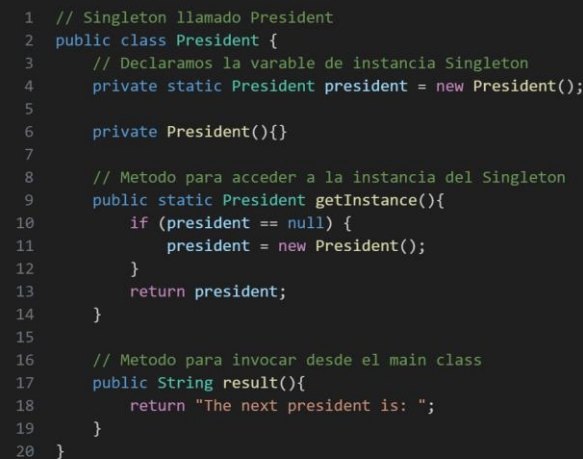
Documentación Singleton Presidente

Patrones de Diseño en la programación

Los patrones de diseño son soluciones generales y reutilizables para problemas comunes en la programación. Proporcionan un enfoque probado y estructurado para resolver problemas específicos al diseñar nuestras aplicaciones o sistemas, promoviendo buenas prácticas de diseño y facilitando la comunicación entre devs. Cada patrón describe un problema que ocurre una y otra vez en nuestro entorno y luego describe el núcleo de la solución a ese problema, de tal manera que puede usar la solución un millón de veces en el contexto de nuestra aplicación.

¿Qué es el Singleton?

Es un patrón de diseño de software que se utiliza para garantizar que una clase tenga una única instancia y proporcionar un punto de acceso global a esa instancia. Tomando como ejemplo, tenemos este código en Java, que es un Singleton aplicado en un contexto de la vida cotidiana.



```
1 // Singleton llamado Presidente
2 public class Presidente {
3     // Declaramos la variable de instancia Singleton
4     private static Presidente presidente = new Presidente();
5
6     private Presidente(){}
7
8     // Metodo para acceder a la instancia del Singleton
9     public static Presidente getInstance(){
10         if (presidente == null) {
11             presidente = new Presidente();
12         }
13         return presidente;
14     }
15
16     // Metodo para invocar desde el main class
17     public String result(){
18         return "The next president is: ";
19     }
20 }
```

Las características de este son las siguientes que podemos analizar

1. **Única instancia:** El patrón Singleton asegura que solo existe una única instancia de algo en todo el programa. En este caso, esa "instancia" sería el presidente elegido. No puede haber múltiples personas siendo consideradas presidentes simultáneamente.

2. **Acceso global:** Proporciona un punto de acceso global a esa única instancia. Esto significa que todos los ciudadanos y entidades involucradas en el proceso electoral pueden referirse a la misma persona como presidente, independientemente de su ubicación geográfica o rol en el país.
3. **Control de creación:** Controla cómo y cuándo se crea esa instancia única. En el contexto de las elecciones presidenciales, esto implica seguir un proceso electoral bien definido y asegurar que solo una persona sea elegida como presidente al final del proceso.

Clase Partidos Politicos (Party)

Durante las elecciones, los ciudadanos votan por sus candidatos, a través de los partidos que representan. Al final del conteo de votos, se asegura que solo una persona gane y se convierta en presidente. Anunciando el candidato ganador y al partido que representa



```
1 // Clase llamada Party
2 public class Party {
3     // Variables de referencias declaradas
4     private StringBuilder party;
5     private String name;
6
7     public Party(String name){
8         this.name = name;
9         this.party = new StringBuilder();
10    }
11    // Getters & Setters de las variables
12    public StringBuilder getParty(){
13        return party;
14    }
15    public void setParty(StringBuilder name){
16        this.party = name;
17    }
18    public String getName(){
19        return name;
20    }
21    public void setName(String name){
22        this.name = name;
23    }
24 }
```

Clase Main

Ahora tenemos toda la lógica de todo nuestro proyecto, podemos conocer quien será nuestro próximo presidente, en este caso, para temas prácticos vamos a generar de manera aleatoria nuestro próximo presidente con su respectivo partido político que representa. Haciendo el uso del Singleton y de la clase de los partidos políticos (Party).

Tal y como veremos a continuación en el siguiente código que se muestra con todas las llamadas, instancias y métodos que tenemos descritas.

```

1 // Paqueterías importadas para su funcionamiento
2 import java.util.ArrayList;
3 import java.util.List;
4 import java.util.Random;
5
6 // Clase main de la lógica
7 public class App {
8     public static void main(String[] args) throws Exception {
9         // Declaración de un ArrayList
10        List<Party> parties = new ArrayList<>();
11
12        // Creación e inicialización de instancias de la clase Party
13        Party party0 = new Party("Santiago");
14        Party party1 = new Party("Ernesto");
15        Party party2 = new Party("Humberto");
16
17        // Se agregan los valores hacia el StringBuilder de la clase Party
18        party0.getParty().append("PRD");
19        party1.getParty().append("PAN");
20        party2.getParty().append("PRI");
21
22        // Se agregan para almacenar las instancias a un ArrayList
23        parties.add(party0);
24        parties.add(party1);
25        parties.add(party2);
26
27        // Obtener la única instancia de President (Singleton)
28        President president = President.getInstance();
29        // Seleccionamos aleatoriamente nuestro próximo presidente
30        Party win = winner(parties);
31        // Imprimimos nuestro ganador de la contienda
32        System.out.println(president.result() + win.getName() + " from: " + win.getParty());
33    }
34
35    /** Metodo creado para seleccionar aleatoriamente un candidato a través de la lista
36     * @param parties => Lista que contiene las instancias de Party
37     * @return Una instancia de Party ganadora aleatoriamente
38     */
39    private static Party winner(List<Party> parties) {
40        // Validamos si esta vacía la lista
41        if (parties.isEmpty()) {
42            return null;
43        }
44        // Creamos e inicializamos la instancia random para generar números aleatoriamente
45        Random random = new Random();
46        // Inicializamos la variable index apuntando al objeto de referencia del tamaño de la lista
47        int index = random.nextInt(parties.size());
48        // Devolvemos el elemento de la lista de la posición seleccionada aleatoriamente
49        return parties.get(index);
50    }
51 }

```

Finalmente, esto se ve en sus aplicaciones en la vida real, en este caso pusimos la situación de un país donde se está organizando una elección para elegir un presidente. En este escenario, el patrón Singleton se refleja en que solo hay un único presidente elegido y que todos los ciudadanos y partes involucradas en el proceso electoral pueden referirse a esa única persona como presidente desde cualquier lugar del país.

Referencias Bibliográficas

Staff Refactoring.Guru. (2014). *¿Qué es un patrón de diseño?* Refactoring Guru
<https://refactoring.guru/es/design-patterns>

Staff IONOS, (19 de febrero de 2021). *Patrón singleton: una clase propia*. IONOS.
<https://www.ionos.mx/digitalguide/paginas-web/desarrollo-web/patron-singleton/>