Aplicación del Patrón Singleton en el Sistema de Alquiler

Problema detectado

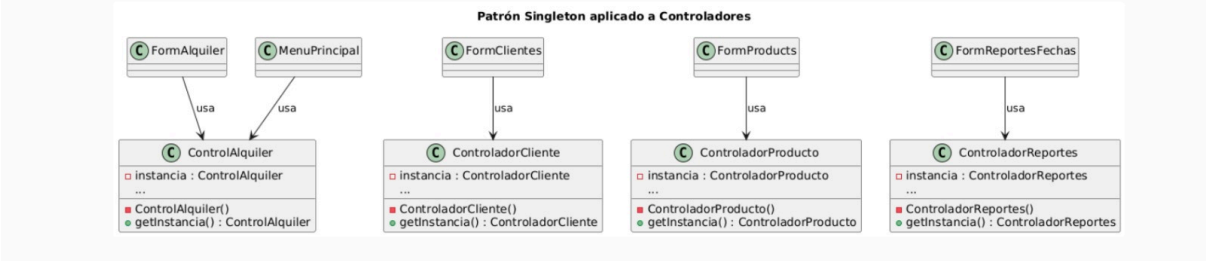
En el proyecto original, las clases controladoras como ControlAlquiler, ControladorCliente, ControladorProducto y ControladorReportes eran instanciadas múltiples veces utilizando el operador 'new'. Esto generaba varios problemas:

- Redundancia: múltiples instancias innecesarias de controladores.
- Inconsistencia: los datos o comportamientos podían variar entre instancias.
- Mayor consumo de recursos: uso innecesario de memoria y procesamiento.
- Dificultad de mantenimiento: más líneas para cambiar si hay ajustes.

Solución implementada: Patrón Singleton

Se aplicó el patrón Singleton a las clases controladoras. Este patrón garantiza que solo exista una instancia de una clase durante la ejecución del programa, permitiendo un acceso centralizado a la lógica del controlador.

Diagrama uml



Beneficios del Singleton en este sistema

- Evita duplicación de controladores.
- Mantiene un estado único y consistente.
- Facilita la reutilización del controlador.
- Mejora el rendimiento.
- Centraliza la lógica del sistema.

Cómo se implementó

Antes (código original en los controladores):

```
public class ControlAlquiler {
    // métodos...
}
```

Después (con Singleton aplicado):

```
public class ControlAlquiler {
    private static ControlAlquiler instancia;

    private ControlAlquiler() {}

    public static ControlAlquiler getInstancia() {
        if (instancia == null) {
            instancia = new ControlAlquiler();
        }
        return instancia;
    }

    // métodos...
}
```

Uso en formularios (reemplazo):

```
Antes: ControlAlquiler obj = new ControlAlquiler();
Después: ControlAlquiler obj = ControlAlquiler.getInstancia();
```