Creational Patterns

1. 1.1. Singleton Pattern

Bir sınıfa ait çalışma zamanı örneğinin (instance) tek olmasını sağlamak, garanti etmek adına kullanılan bir yazılım tasarım kalıbıdır.

1.1. 1.1.1. Örnek

Singleton sınıfından ancak iznimiz dahilinde nesne oluşturulabilmeli. Bunu sağlamak adına ilgili sınıfın kurucu (contructor) metodunu private ayarlarız.

```
public class Singleton {
    private static Singleton INSTANCE = null;

    private Singleton() {}
}
```

Nesnenin erişimi ve üretimi getInstance metodu ile yapılır.

Eğer INSTANCE nesnesi üretilmemişse, nesne üretilip döndürülür. Halihazırda üretilmiş ise ilgili nesne döndürülür. Bu şekilde Singleton tasarım kalıbı ile nesne erişimi sağlanır.

```
public static Singleton getInstance() {
        if (INSTANCE == null)
            INSTANCE = new Singleton();

        return INSTANCE;
    }
```

SingletonDemo.java.

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

        Singleton singletonObj = Singleton.getInstance();
        System.out.println("1st Instance ID: " +
 System.identityHashCode(singletonObj));
        singletonObj.getMessage();
```

```
        Singleton singletonObj2 = Singleton.getInstance();
        System.out.println("2nd Instance ID: " +
 System.identityHashCode(singletonObj2));
        singletonObj.getMessage();
    }
}
```

```
1st Instance ID: 22307196
Singleton Design Pattern!
2nd Instance ID: 22307196
Singleton Design Pattern!
```

1.2. 1.1.2. Thread-Safe Singleton Design Pattern

Singleton Pattern ile ilgili sınıfın kurucu metodunu private yaparak nesne oluşturma kontrolünü bir metot içerisinde kontrol ettik. Fakat multi-thread işlemlerde nesne oluşturulan metoda aynı anda birden fazla thread erişmeye çalışabilir, bir race condition oluşabilir.

Bunu engellemek adına, getInstance metodu aynı anda en fazla bir thread tarafından erişilebilir olacak şekilde tasarlanmalıdır.

Java'da **synchronized** ile metot ya da sınıfa aynı anda çoklu erişim kısıtlamaktadır.

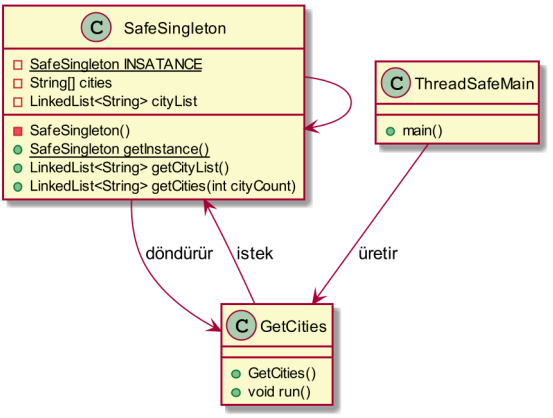

Figure 1. Singleton Pattern UML Diyagramı

SafeSingleton.java.

```
// Threadler bu metoda ayni anda erismeye calisabilir.
public synchronized static SafeSingleton getInstance() {
        if (INSTANCE == null)
                INSTANCE = new SafeSingleton();

        return INSTANCE;
    }
```

SafeSingleton sınıfı içerisinde iki yeni değişken oluşturulur. Şehirler kullanıcılara atanacaktır. Bununla birlikte thread-safe yapı test edilecektir.

SafeSingleton.java.

```
    String[] cities = {"Beauclair", "Novigrad", "Oxenfurt",
                        "Vizima", "Spikeroog", "Velen"};

    private LinkedList<String> cityList = new LinkedList<>
 (Arrays.asList(cities));
```

ThreadSafeMain.java.

```
public class ThreadSafeMain {

    public static void main(String[] args) {
        Runnable getCities = new GetCities();
        Runnable getAnotherCities = new GetCities();

        new Thread(getCities).start();
        new Thread(getAnotherCities).start();
    }
}

class GetCities implements Runnable {

    public GetCities() {}

    @Override
    public void run() {
        SafeSingleton singletonInstance = SafeSingleton.getInstance();
        System.out.println("Instance ID: " +
 System.identityHashCode(singletonInstance));

        System.out.println(singletonInstance.getCityList());
```

```
        LinkedList<String> playerCityList =
 singletonInstance.getCities(3);
        System.out.println("Player: " + playerCityList);
        System.out.println("Player visited cities.");
        System.out.println("");
    }
}
```

Threadler oluşturulur, çıktılar kontrol edilir.

```
new Thread(getCities).start();
new Thread(getAnotherCities).start();
```

Görüldüğü üzere, Instance ID'ler eşittir, threadler senkronize çalışmaktadır:

```
Instance ID: 2452427
Instance ID: 2452427
[Beauclair, Novigrad, Oxenfurt, Vizima, Spikeroog, Velen]
[Beauclair, Novigrad, Oxenfurt, Vizima, Spikeroog, Velen]
Player: [Beauclair, Novigrad, Oxenfurt, Vizima]
Player visited cities.

Player: [Beauclair, Novigrad, Oxenfurt, Vizima]
Player visited cities.
```

SafeSingleton sınıfında, **getInstance** metodunda **synchronized** ön eki kaldırıldığında ise singleton tasarım kalıbı yok sayılır, her iki thread de kendi yeni nesnesini üretir:

```
Instance ID: 218531
Instance ID: 33096305
[Beauclair, Novigrad, Oxenfurt, Vizima, Spikeroog, Velen]
[Beauclair, Novigrad, Oxenfurt, Vizima, Spikeroog, Velen]
Player: [Beauclair, Novigrad, Oxenfurt, Vizima]
Player visited cities.

Player: [Beauclair, Novigrad, Oxenfurt, Vizima]
Player visited cities.
```