

Singleton Pattern

Singleton
- Singleton : Singleton
- Singleton(): + getInstance() : Singleton

Advantage of Singleton design pattern

- Saves memory because object is not created at each request.
- Only single instance is reused again and again.
- Role of a Global Variable
- Mostly used in multi-threaded and database application

```
import java.util.*;
public class MySingleton {
    //the static singleton object
    private static MySingleton theObject;
    private MySingleton() {
    }
    public static MySingleton createMySingleton() {
        if (theObject == null)
            theObject = new MySingleton();
        return theObject;
    }
}
```

```
import java.util.*;
public class MySingleton {
    //the static singleton object
    private static MySingleton MySingleton;
    private MySingleton() {
    }
    public static MySingleton createMySingleton() {
        if (MySingleton == null)
            MySingleton = new MySingleton();
        return MySingleton;
    }
}
```

```
public class Main {  
    public void createSingleton() {  
        MySingleton ms1 =  
            MySingleton.createMySingleton();  
        MySingleton ms2 =  
            MySingleton.createMySingleton();  
        System.out.println( ms1 == ms2 );  
    }  
    public static void main(String[] args) {  
        new Main().createSingleton();  
    }  
}
```

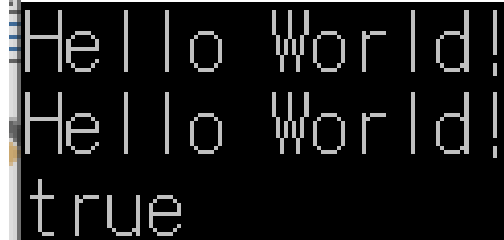
true

The 2nd Example

```
public class SingleObject {  
    private static SingleObject instance = new SingleObject(  
        private SingleObject(){}  
        public static SingleObject getInstance(){  
            return instance;  
        }  
        public void showMessage(){  
            System.out.println("Hello World!");  
        }  
    }  
}
```

```
public class SingletonPatternDemo {  
    public static void main(String[] args) {  
  
        //illegal construct  
        //Compile Time Error: The constructor SingleObject() is not visible  
        //SingleObject object = new SingleObject();  
  
        //Get the only object available  
        SingleObject object = SingleObject.getInstance();  
  
        //show the message  
        object.showMessage();  
    }  
}
```

```
public class SingletonPatternDemo2 {  
    public static void main(String[] args) {  
  
        SingleObject object1 = SingleObject.getInstance();  
        SingleObject object2 = SingleObject.getInstance();  
        //show the message  
        object1.showMessage();  
        object2.showMessage();  
        System.out.println( object1 == object2 );  
  
    }  
}
```



```
Hello World!  
Hello World!  
true
```


Yuki Book Example

```
public class TicketMaker {  
    private int ticket = 1000;  
    private static TicketMaker singleton = new TicketMaker();  
    private TicketMaker() {  
    }  
    public static TicketMaker getInstance() {  
        return singleton;  
    }  
    public synchronized int getNextTicketNumber() {  
        return ticket++;  
    }  
}
```

```
public class TicketMain {  
    public static void main(String[] args) {  
        System.out.println("Start.");  
        for (int i = 0; i < 10; i++) {  
            System.out.println(i + ":" +  
                TicketMaker.getInstance().getNextTicketNumber());  
        }  
        System.out.println("End.");  
    }  
}
```

```
Start.  
0:1000  
1:1001  
2:1002  
3:1003  
4:1004  
5:1005  
6:1006  
7:1007  
8:1008  
9:1009  
End.
```

```
public class Triple {  
    private static Triple[] triple = new Triple[]{  
        new Triple(0), new Triple(1), new Triple(2),  
    };  
    private int id;  
    private Triple(int id) {  
        System.out.println("The instance " + id + " is created");  
        this.id = id;  
    }  
    public static Triple getInstance(int id) {  
        return triple[id];  
    }  
    public String toString() {  
        return "[Triple id=" + id + "]";  
    }  
}
```

```
public class Main {  
    public static void main(String[] args) {  
        System.out.println("Start.");  
        for (int i = 0; i < 9; i++) {  
            Triple triple = Triple.getInstance(i % 3);  
            System.out.println(i + ":" + triple);  
        }  
        System.out.println("End.");  
    }  
}
```

```
Start.  
The instance 0 is created.  
The instance 1 is created.  
The instance 2 is created.  
0:[Triple id=0]  
1:[Triple id=1]  
2:[Triple id=2]  
3:[Triple id=0]  
4:[Triple id=1]  
5:[Triple id=2]  
6:[Triple id=0]  
7:[Triple id=1]  
8:[Triple id=2]  
End.
```