Singleton Pattern

By “Group of Four”

Lets do some work out!

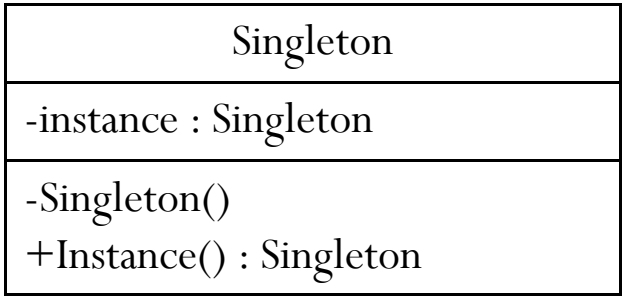
- We have a Minimart “Seven-Elephants”
- 7-Elephants uses a class to store its data on Item Stock in the Minimart.
- Cashier must know the stock class to do some transaction
- There are many cashier, so we need to organize them somehow!
- Please HELP US!! Show USYour Design!!
- Here are some Classes examples.
  - Item Class, Cashier Class, Stock Class, etc..

One of a Kind Object!!

- What will happen if there's many instance of the stock?
- Only One Instance of an object!!!
- There should be only one instance of stock right!?!?!
- Singleton Helps!!!

The Singleton Pattern

- Ensures a class has only one instance and provide a global point of access to it
- Prevent other class in creating the singleton class.



How to Singleton?

- Private your Constructor!
  - So others class can't instantiate it
- Create a static instance of the class which you want to singleton!!
  - The Unique instance
- Create a method to get the instance of the singleton class!!!
  - So other class can get the unique instance.

The Stock Class

```
public class Stock{
   List items;
   …//Some useful attributes


   public Stock(){ ….. }


   Item getItemDiscription(){ ….. }
   //more useful method
}
```

Lets Singleton the Stock Object!!!!!!

Private the Constructor

```
public class Stock {
   List items;
   …//Some useful attributes

   private Stock() { ….. }

   Item getItemDiscription() { ….. }
   //more use full method
}
```

Create an Static Instance

```
public class Stock {
   List items;
   private static Stock;
   …//Some useful attributes


   private Stock() { ….. }


   Item getItemDiscription(){ ….. }
   //more use full method
}
```

Create a get instance method

```
Public class Stock {
    List items;
     private static Stock stockInstance;
    …//Some useful attributes


    private Stock() { ….. }

    public static Stock getInstance() {
            if(stockInstance == null) {
                    stockInstance = new Stock();
            }
            return stockInstance;
    }


    Item getItemDiscription(){ ….. }
    //more use full method
}
```

- Now we have the singleton Stock class!!

We're not done yet!

- What if there's many cashier working together?
- Maybe there's problem you didn't see?
- This problem are occur frequently in multithread software.
- What happens if 2 cashier are working at the same time???

Let us be the JVM

Cashier1	Cashier2	Value of stockInstance
public static Stock getInstance() {		null
	public static Stock getInstance() {	null
if(stockInstance == null) {		null
	if(stockInstance == null) {	null
stockInstance = new Stock();		<Stock1>
return stockInstancel		<Stock1>
	stockInstance = new Stock();	<Stock2>
	return stockInstancel	<Stock2>

This is not good, there're 2 instance of stock!!

How can we improve with multithreading??

```
Public class Stock {
    List items;
     private static Stock stockInstance;
    …//Some useful attributes


    private Stock() { ….. }

    public static synchronize Stock getInstance() {
            if(stockInstance == null) {
                    stockInstance = new Stock();
            }
            return stockInstance;
    }


    Item getItemDiscription(){ ….. }
    //more use full method
}
```

Any Problems??

How can we improve with multithreading?? (Better)

```
Public class Stock {
    List items;
     private static Stock stockInstance = new Stock();
    …//Some useful attributes


    private Stock() { ….. }


    public static Stock getInstance() {
            return stockInstance;
    }


    Item getItemDiscription(){ ….. }
    //more use full method
}
```

How can we improve with multithreading?? (Even Better!!)

```
Public class Stock {
    List items;
     private volatile static Stock stockInstance;
    …//Some useful attributes


    private Stock() { ….. }


    public static Stock getInstance() {
                if (stockInstance == null) {
                    synchronized (Stock.Class) {
                                stockInstance = new Stock();
                    }
                }
                return stockInstance;
    }


    Item getItemDiscription(){ ….. }
    //more use full method
}
```

This method is called “double-checked locking”