SINGLETON WITH REFLECTION API

By breaking encalpulation by getting access to private constructor there is provihision to create the object

Problem :-

```
package com.nit.printer;
import java.lang.reflect.Constructor;
import java.lang.reflect.InvocationTargetException;
import javax.management.InstanceAlreadyExistsException;

public class Printer {
	//egar instantiation
	private static Printer instance;

	//private  constructor
	private Printer() {
		System.out.println("0 param constructor");
	}
	//static factory method that is return object of class
	public static Printer getInstance()  {
		synchronized (Printer.class) {
			if(instance==null) {
				//double null check
				if(instance==null) {
					instance=new Printer();
				}//if
			}//if
		}
		return instance;
	}
	//breaking singlton using reflection API
	public static Printer reflection() {
		Class c=null;
		Constructor con[];
		try {
			//load the class
			c=Class.forName("com.nit.printer.Printer");
			//get all the constructor
			con=c.getDeclaredConstructors();
			con[0].setAccessible(true);
			//creting object of this construor using new instance metod
			instance=(Printer) con[0].newInstance();
		} catch (Exception e) {
			e.printStackTrace();
		}
		return instance;
	}
	//displaying hashcode of object
	public static void display(Object obj) {
		System.out.println("hashcode is="+obj.hashCode());
	}
}
```

Test:-

```
package com.nit.test;

import java.lang.reflect.Constructor;

import com.nit.printer.Printer;

public class ReflectionTest {
	public static void main(String[] args){
		Printer p1=null,p2=null,p3=null;
		Class c=null;
		Constructor con[];

		p1=Printer.getInstance();
		System.out.println("***calling p1 object***");
		Printer.display(p1);
		System.out.println();

		System.out.println(" calling p2 object");
		p2=Printer.getInstance();
		Printer.display(p2);
		System.out.println();
		System.out.println("p1==p2?"+(p1==p2));
		System.out.println();

		p3=Printer.reflection();
		Printer.display(p3);
		System.out.println("p1==p2?="+(p1==p3));
	}
}
```

Output:-

```
0 param constructor
***calling p1 object***
hashcode is=366712642

calling p2 object
hashcode is=366712642

p1==p2?true

0 param constructor
hashcode is=1829164700
p1==p2?=false
```

SINGLETON WITH REFLECTION API

Solution :-

there is no solution to do by override any class or method we have to write strict logic for constructor to not create another object for our singleton class

Ex:-

Printer class:-

```
package com.nit.printer;
import java.lang.reflect.Constructor;
import java.lang.reflect.InvocationTargetException;
public class Printer {

    //egar instantiation
    private static Printer instance;
    private static boolean flag=false;

        //private  constructor
        private Printer() throws Exception {
            if(flag==true) {
            throw new InstantiationException("object already created");
            }
            flag=true;
            System.out.println("0 param constructor");
        }

        //static factory method that is return object of class
        public static Printer getInstance() throws Exception {
            synchronized (Printer.class) {
                if(instance==null) {
                    //double null check
                    if(instance==null) {
                        instance=new Printer();
                    }//if
                }//if
            }
            return instance;
        }
        //breaking singlton using reflection API
        public static Printer reflection() {
            Class c=null;
            Constructor con[];
            try {
                //load the class
```

```
                        c=Class.forName("com.nit.printer.Printer");
                        //get all the constructor
                        con=c.getDeclaredConstructors();
                        con[0].setAccessible(true);
                        //creting object of this construor using new instance metod
                        instance=(Printer) con[0].newInstance();
                } catch (Exception e) {
                        e.printStackTrace();
                }
                return instance;
        }
        //displaying hashcode of object
        public static void display(Object obj) {
                System.out.println("hashcode is="+obj.hashCode());
        }
}
```

Reflection test:-

```
package com.nit.test;
import java.lang.reflect.Constructor;
import com.nit.printer.Printer;
public class ReflectionTest {
        public static void main(String[] args) throws Exception {
                Printer p1=null,p2=null,p3=null;
                Class c=null;
                Constructor con[];

                p1=Printer.getInstance();
                System.out.println("***calling p1 object***");
                Printer.display(p1);
                System.out.println();

                System.out.println(" calling p2 object");
                p2=Printer.getInstance();
                Printer.display(p2);
                System.out.println();
                System.out.println("p1==p2?"+(p1==p2));
                System.out.println();

                p3=Printer.reflection();
                Printer.display(p3);
                System.out.println("p1==p2?="+(p1==p3));
        }
}
```

Output:-

```
0 param constructor
***calling p1 object***
hashcode is=366712642

 calling p2 object
hashcode is=366712642

p1==p2?true

java.lang.reflect.InvocationTargetException
	at sun.reflect.NativeConstructorAccessorImpl.newInstance0(Native Method)
	at sun.reflect.NativeConstructorAccessorImpl.newInstance(NativeConstructorAccessorImpl.java:62)
	at sun.reflect.DelegatingConstructorAccessorImpl.newInstance(DelegatingConstructorAccessorImpl.java:45)
	at java.lang.reflect.Constructor.newInstance(Constructor.java:423)
	at com.nit.printer.Printer.reflection(Printer.java:46)
	at com.nit.test.ReflectionTest.main(ReflectionTest.java:25)
Caused by: java.lang.InstantiationException: object already created
	at com.nit.printer.Printer.<init>(Printer.java:17)
	... 6 more
hashcode is=366712642
p1==p2?=true
```

we cannot use reflection api singleton class breakage problem for ever because same reflection api can be used to access private flag variable and instance variable and can change their data

This is not a perfect solution because when people can access private constructor by using reflection api then they can also can access private members also they can break singleton so this is also not perfect solution this is only one possible solution .