SINGLETON USING WITH ENUM

- **As for now we have developed singleton object by using 2 approaches which is traditional approach and another one is Inner class approach .**

  In this section we are going to learn one best approach which is Enum based approach.

Enum :-

Group of named constants (final and static) is called enum , enum can have normal method and normal instance variables also

Enum is internally is class which is extending one predefine class called enum.

By using enum most of the problem can solve by enum which is came since java 5 on words.

By nature enum behave like singleton we no need to do anything.

Ex:-

```
1.package com.nit.dp;
2.public enum Printer {
3.  //creating instance of printer class
4.  instance;
5.  //declaring business method
6.  public static void PrintData(String data) {
7.      System.out.println("given data is="+data);
8.  }
9.}
```

“here just declaring instance which is called named constants enum creating object automatically and instance will be inisitialize with printer class object”

In above case **instance** is the object of Printer class

SINGLETON USING WITH ENUM

Test Class:-

```
package com.nit.test;
import com.nit.dp.Printer;
import com.nit.dp.ReflectionTest;

public class PrinterTest {

	public static void main(String[] args) {
		Printer p1=null,p2=null;
		p1=Printer.instance;
		System.out.println("hashcode of printer object  is="+p1.hashCode());
		p2=Printer.instance;
		System.out.println("hashcode of printer object is="+p2.hashCode());
		//calling business method
		Printer.PrintData("vikas");
		}
}
```

"here the moment class is loaded this member variable will be initialize with printer class object that is static and final so no one can change since static we can access with the class name.and here its also doing eager instantiation.

Output:-

```
hashcode of printer object is=366712642
hashcode of printer object is=366712642
given data is= vikas
```

==========****==========

SINGLETON USING WITH ENUM

Multithreading Test:-

Since enum is a internally a class which is finally and static and at once member variable initialize at the time of class loading and its object created and ready for use so its fallow egar instantiation so no issue with also multithreading.

Ex:- **printer class:-**

```
1.package com.nit.dp;
2.public enum Printer{
3.    //creating instance of printer class
4.    instance;
5.    //declaring business method
6.    public static void PrintData(String data) {
7.        System.out.println("given data is="+data);
8.    }
9.}
```

Testing Multithreading

Ex:-

```
1. package com.nit.dp;
2. public class MultiThreadTest implements Runnable{
3.
4.     @Override
5.     public void run() {
6.         Printer p=null;
7.         p=Printer.instance;
8.         System.out.println(p.hashCode());
9.         Printer.PrintData("vikas");
10.        }
11.        public static void main(String[] args) {
12.            MultiThreadTest t=new MultiThreadTest();
13.            Thread t1=new Thread(t);
14.            t1.start();
15.            Thread t2=new Thread(t);
16.            t2.start();
17.        }
18.    }
```

SINGLETON USING WITH ENUM

Output:-

```
1719197730
1719197730
given data is=vikas
given data is=vikas
```

"so with this we have solved multithreading here creating singleton object with enum no multithreading issue has created"

==========********==========

Testing enum singleton with cloning:-

Since we have already studied if we want to clone of any object so we have to implements clonable interface which is marker interface after that we have to override clone method which is available on object class.

So in enum as we know enum doesn't support of extending of class so other user can not extend another class property in enum so they can not clone of enum object.

And if another programmer create situation by implementing interface with alredy extends another interface by using this teqnic also can not clone of object because this object is final so can not create another object and also since clone method available in object class and since enum class extends alredy java.lang.enum so clone method also not available so we can not override clone method also.

Ex:-

```
1.package com.nit.dp;
2.public enum Printer implements Cloneable{
3.  //creating instance of printer class
4.  instance;
5.  //declaring business method
6.  public static void PrintData(String data) {
7.      System.out.println("given data is="+data);
8.  }
9.     //clone method can not override
10. }
```

Can not override final method enum so we can not override final method because clone method declared as final So we can not create another object of enum by using cloning

SINGLETON USING WITH ENUM

Advantages of taking enum as singleton:-

1. **Easy to develop**
2. **It perform eager instantiation of its constants , so no multithreading issues**
3. **Enum does not allow cloning in any angle**

Cloning is not possible with enum based object

Test eserialization on the enum :-

Printer class:-

```
1.package com.nit.dp;
2. import java.io.Serializable;
3.
4.public enum Printer implements Serializable{
5.//creating instance of printer class
6. instance;
7.     //declaring business method
8.     public static void PrintData(String data) {
9.         System.out.println("given data is="+data);
10.        }
11.    }
```

DeSerialization Test:-

```
1.  package com.nit.dp;
2.
3.  import java.io.FileInputStream;
4.  import java.io.FileOutputStream;
5.  import java.io.ObjectInputStream;
6.  import java.io.ObjectOutputStream;
7.
8.  public class DeSerializationTest {
9.
10.         public static void Serialize(Object obj) {
11.                 ObjectOutputStream oos=null;
12.                 try {
13.                         oos=new ObjectOutputStream(new FileOutputStream("vikas.txt"));
14.                         oos.writeObject(obj);
15.                         oos.flush();
16.                         oos.close();
17.                         System.out.println("DeSerializationTest.Serialize()"+" object serialized");
18.
19.                 } catch (Exception e) {
```

SINGLETON USING WITH ENUM

```
20.                     e.printStackTrace();
21.             }
22.     }
23.
24.     public static Object Deserialize() {
25.             ObjectInputStream ois=null;
26.             Object obj = null;
27.             try {
28.                     ois=new ObjectInputStream(new FileInputStream("vikas.txt"));
29.                     obj=ois.readObject();
30.                     ois.close();
31.                     System.out.println("object deserialised");
32.             } catch (Exception e) {
33.                     e.printStackTrace();
34.             }
35.             return obj;
36.     }
37.
38.     }
```

PrinterTest :-

```
1.package com.nit.test;
2.import com.nit.dp.DeSerializationTest;
3. import com.nit.dp.Printer;
4. public class PrinterTest {
5.
6.      public static void main(String[] args) {
7.           Printer p1=null,p2=null,p3=null;
8.           p1=Printer.instance;
9. System.out.println("hashcode of printer object is="+p1.hashCode());
10.                  p2=Printer.instance;
11.     System.out.println("hashcode of printer object is="+p2.hashCode());
12.                  //calling business method
13.                  Printer.PrintData("vikas");
14.                  //serializing an object
15.                  DeSerializationTest.Serialize(p1);
16.                  //derialization of object
17.                  p3=(Printer) DeSerializationTest.Deserialize();
18.     System.out.println("hashcode of printer object is="+p3.hashCode());
19.                  System.out.println();
20.                  }
21.
22.     }
```

Output:-

```
hashcode of printer object is=366712642
hashcode of printer object is=366712642
given data is=vikas
DeSerializationTest.Serialize() object serialized
object deserialised
hashcode of printer object is=366712642
```

SINGLETON USING WITH ENUM

here in enum api we have seen enum is already implementing of Serializable interface so we no need to implementing separate serializable interface that why its returning same object

enum having some restriction mean per jvm only one object will be created so no object created when Deseralization also use done not effect on enum

advantages :-

1. Enum is inherinity serializable but they allow to create only object per jvm so there is no DeSerialization problem

Testing reflection on enum:-

Since enum having by default private constructor and which will not be reflect to reflection api so we can not create object with usigng reflection api.

Ex:-

```
1.package com.nit.dp;
2. import java.io.Serializable;
3.
4. public enum Printer implements Serializable{
5.       //creating instance of printer class
6.       instance;
7.       //declaring business method
8.       public static void PrintData(String data) {
9.             System.out.println("given data is="+data);
10.            }
11.      }
```

Reflection Test:-

```
1. package com.nit.dp;
2. import java.lang.reflect.Constructor;
3. public class ReflectionTest {
4.       public static Object Reflection() {
5.             Object obj=null;
6.             Class c=null;
7.             Constructor con[];
8.             try {
9.                   c=Class.forName("com.nit.dp.Printer");
10.                        //getiing all the constructors
11.                        con=c.getDeclaredConstructors();
12.                        con[0].setAccessible(true);
13.                        obj=con[0].newInstance(con);
14.                  } catch (Exception e) {
15.                        e.printStackTrace();
16.                  }
17.                  return obj;
18.            }
19.      }
```

SINGLETON USING WITH ENUM

PrinterTest:-

```
1.package com.nit.test;
2.import com.nit.dp.Printer;
3. import com.nit.dp.ReflectionTest;
4. public class PrinterTest {
5.
6.       public static void main(String[] args) {
7.            Printer p4=null;
8.            //calling reflection test
9.            p4=(Printer) ReflectionTest.Reflection();
10.      System.out.println("hashcode of printer object is="+p4.hashCode());
11.           }
12.      }
```

Output:-

```
java.lang.IllegalArgumentException: Cannot reflectively create enum objects
at java.lang.reflect.Constructor.newInstance(Constructor.java:417)
at com.nit.dp.ReflectionTest.Reflection(ReflectionTest.java:16)
at com.nit.test.PrinterTest.main(PrinterTest.java:24)
Exception in thread "main" java.lang.NullPointerException
at com.nit.test.PrinterTest.main(PrinterTest.java:25)
```

==========**********==========

DisAdvantages of enum:-

1. **Eager instantiation of constants may waste the resource though object is not required**
2. **We should not place more then one constant in singleton Enum otherwise singleton behavior will be broken**
3. **We can not make other class extending from singleton enum…..(becoz all enum are dinal classes internally)**

Best:-