APPROACH TO DEVELOP SINGLETON OBJECT

There are 3 Approach to develop singleton java object:-

a. Take normal class and add all constraints code as shown above(traditional approach)
b. Create singleton java class object inside nested inner class (static inner class) of singleton java class
c. Take enum as singleton

There are 4 types of inner class:-

1. **Normal inner class**
2. **Nested/static inner class**
3. **Local inner class**
4. **Anonymous inner class**

1. **Normal inner class:-**

   If inner class logic are required in multiple instance/non static method of outer class then go for normal inner class.

2. **Nested/static inner class:-**

   If inner class logic are required in multiple static method of outer class then go for nested/static inner class

3. **Local inner class:-**

   If inner class logic are required only in one method defition of outer class then go for local inner class

4. **Annonymous inner class:-**

   If inner class logic are required only in one method call then go for anonymous inner class.

****inner class can be private****

APPROACH TO DEVELOP SINGLETON OBJECT

By using cloning :-

deSerilization test:-

BREAKING SINGLETON BY USING REFLECTION API

APPROACH TO DEVELOP SINGLETON OBJECT

Ex:-

```
package com.nit.dp;
public class Printer {
	public Printer() {
		System.out.println("0 param constructor");
	}
	public static Printer getInstance() {
		return printerHolder.instance;
	}

	//nested/static inner class
	private static class printerHolder{
		//egar instantiation
		private static Printer instance=new Printer();

		public printerHolder() {
		System.out.println("Printer.printerHolder.printerHolder()");
		}
	}
}
```

```
package com.nit.printer;
import com.nit.dp.Printer;
public class PrinterTest {
	public static void main(String[] args) {
		Printer p1=null, p2=null;
		p1=Printer.getInstance();
		System.out.println(p1.hashCode());

		p2=Printer.getInstance();
		System.out.println(p2.hashCode());
	}
}
```

Output:-

```
0 param constructor
366712642
366712642
```

APPROACH TO DEVELOP SINGLETON OBJECT

" by using inner class we can solve problem of eager instantiation and multithreading also"

```
package com.nit.dp;
public class Multi implements Runnable {
	Printer p1=null,p2=null;

	@Override
	public void run() {
		//calling Printer class object
		p1=Printer.getInstance();
		p2=Printer.getInstance();
		System.out.println(p1.hashCode());
		System.out.println(p2.hashCode());
	}
	public static void main(String[] args) {
		Multi t=new Multi();
		Thread t1=new Thread(t);
		t1.start();

		Thread t2=new Thread(t);
		t2.start();
	}
}
```

Output:-

```
0 param constructor
455355979
455355979
455355979
455355979
```

Here same object created so no need to synchronized thread because all time same object getting new object not creating

Using inner class approach also cloning and deserialization breaking singleton object so we have to restrict with cloning and deserialization process

APPROACH TO DEVELOP SINGLETON OBJECT

By using cloning :-

Ex:-

```
package com.nit.dp;
import java.io.Serializable;
public class CloneTest implements Cloneable,Serializable {
    @Override
    public Object clone() throws CloneNotSupportedException {
        // TODO Auto-generated method stub
        return super.clone();
    }
}
```

```
package com.nit.dp;
public class Printer extends CloneTest {
    public Printer() {
        System.out.println("0 param constructor");
    }
    public static Printer getInstance() {
        return printerHolder.instance;
    }
    //nested/static inner class
    private static class printerHolder{
        //egar instantiation
        private static Printer instance=new Printer();
        public printerHolder() {
    System.out.println("Printer.printerHolder.printerHolder()");
        }
    }
       //to stop cloning
    @Override
    public Object clone() throws CloneNotSupportedException {
        throw new CloneNotSupportedException("can not clonable");
        //return this;
    }
}
```

APPROACH TO DEVELOP SINGLETON OBJECT

```
package com.nit.printer;
import com.nit.dp.Printer;
import com.nit.dp.deSerializeTest;
import com.nit.dp.Printer;

public class PrinterTest {
     public static void main(String[] args) throws CloneNotSupportedException {
          Printer p1=null, p2=null,p3=null,p4=null;
        //calling instance method of Printer class based on thraditional approach
          p1=Printer.getInstance();
          p2=Printer.getInstance();
          System.out.println(p1.hashCode());
          System.out.println(p2.hashCode());
          System.out.println();

          //cloning of p2 object
          p3=(Printer) p2.clone();
          System.out.println(p3.hashCode());
     }
}
```

So here also we can use clone method to solve cloning problem

APPROACH TO DEVELOP SINGLETON OBJECT

deSerilization test:-

```
package com.nit.dp;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;

public class deSerializeTest {
     //serialization of object
     public static void serialize(Object obj) {
               ObjectOutputStream oos=null;
          try {
               oos=new ObjectOutputStream(new
FileOutputStream("vikas.txt"));
               oos.writeObject(obj);
               oos.flush();
               oos.close();
               System.out.println("object Serialized");
               System.out.println();
          } catch (Exception e) {
               e.printStackTrace();
          }
     }
     public static Object Deserialize() {
          ObjectInputStream ois=null;
          Object obj=null;
          try {
               ois=new ObjectInputStream(new
FileInputStream("vikas.txt"));
               obj=ois.readObject();
               ois.close();
               System.out.println("Object DeSerilized");
               System.out.println();
          } catch (Exception e) {
               e.printStackTrace();
          }
          return obj;
     }

}
```

APPROACH TO DEVELOP SINGLETON OBJECT

```
package com.nit.dp;
public class Printer extends CloneTest {
    public Printer() {
        System.out.println("0 param constructor");
    }
    public static Printer getInstance() {
        return printerHolder.instance;
    }
    //nested/static inner class
    private static class printerHolder{
        //egar instantiation
        private static Printer instance=new Printer();
        public printerHolder() {

    System.out.println("Printer.printerHolder.printerHolder()");
        }
    }
       //to stop  deserialization
        public Object readResolve() {
        System.out.println("Printer.readResolve()");
        throw new IllegalArgumentException("deserialization s
not allow");
    }
}
```

APPROACH TO DEVELOP SINGLETON OBJECT

```
package com.nit.printer;
import com.nit.dp.Printer;
import com.nit.dp.deSerializeTest;
import com.nit.dp.Printer;
public class PrinterTest {
    public static void main(String[] args) throws
CloneNotSupportedException {
        Printer p1=null, p2=null,p3=null,p4=null;
//calling instance method of Printer class based on thraditional approach
        p1=Printer.getInstance();
        p2=Printer.getInstance();
        System.out.println(p1.hashCode());
        System.out.println(p2.hashCode());
        System.out.println();

        //serialize of object
        deSerializeTest.serialize(p1);

        //Deserialize of Object
        p4=(Printer) deSerializeTest.Deserialize();
        System.out.println(p4.hashCode());
    }
}
```

APPROACH TO DEVELOP SINGLETON OBJECT

BREAKING SINGLETON BY USING REFLECTION API

Ex:-

```
package com.nit.dp;
import java.lang.reflect.Constructor;
public class ReflectionTest {
    private static Printer instance;
    public static Printer reflection() {
        Class c=null;
        Constructor con[]=null;
        try {
            //loading class
            c=Class.forName("com.nit.dp.Printer");
            //getting all the constructor
            con=c.getDeclaredConstructors();
            //intialise this constructor to index
            con[1].setAccessible(true);
            //creating object of this construcro
            instance=(Printer) con[1].newInstance();

        } catch (Exception e) {
            e.printStackTrace();
        }
        System.out.println("calling reflection api");
        return instance;
    }
}
```

APPROACH TO DEVELOP SINGLETON OBJECT

```
package com.nit.dp;
public class Printer extends CloneTest {
    private static boolean flag=false;

    /*public Printer() {
        System.out.println("0 param constructor");
    }*/

    //for reflection api we have to restrict constructor to create object
    private Printer(){
        if(flag==true) {
            throw new IllegalArgumentException("object
already created");
        }
        flag=true;
        System.out.println("0 param constructor");
    }
    public static Printer getInstance() {
        return printerHolder.instance;
    }

    //nested/static inner class
    private static class printerHolder{
        //egar instantiation
        private static Printer instance=new Printer();
        public printerHolder() {

    System.out.println("Printer.printerHolder.printerHolder()");
        }
    }
}
```

APPROACH TO DEVELOP SINGLETON OBJECT

```
package com.nit.printer;

import com.nit.dp.Printer;
import com.nit.dp.ReflectionTest;

public class PrinterTest {

    public static void main(String[] args) throws
CloneNotSupportedException {
        Printer p1=null, p2=null,p3=null,p4=null,p5=null;
        //calling instance method of Printer class based on
thraditional approach
        p1=Printer.getInstance();
        p2=Printer.getInstance();
        System.out.println(p1.hashCode());
        System.out.println(p2.hashCode());
        System.out.println();

        //creating object using reflection api
        p5=ReflectionTest.reflection();
        System.out.println(p5.hashCode());
    }
}
```