SOLUTION FOR MULTITHREADING ENV

"breaking singleton with multithreading problem is solved by using synchronized"

Solution :-

1. **Use synchronization (best)**
2. **Eager instantiation**

1. Ex:- using Eager Instantiation

Printer class:-

```
package com.nit.dp;
public class printer {
	private static printer instence=new printer();		//egar instantiation only if object creation at the time of declaration
	private printer() {
		System.out.println("0 param constructor");
	}
	//statif factory method
		public static printer getInstance() {
			//static factory method
			if(instence==null) {
			instence =new printer();	//lazy instantiation good practice
			}
			return instence;
		}
	public void printDate(String data) {
		System.out.println(data);
	}
}
```

ticketBooking.java

```
package com.nit.dp;
public class ticketBooking implements Runnable  {
	printer p=null;
	@Override
	 public synchronized void run() {
		p=printer.getInstance();
		System.out.println("hascode of printer class
				is="+p.hashCode()+"
				and thread
				is="+Thread.currentThread().getName());
	}
}
```

ticketBookingTest.java

```
package com.nit.test;
import com.nit.dp.ticketBooking;
public class ticketBookingtest {
      public static void main(String[] args) {
            ticketBooking t=new ticketBooking();
            Thread t1=new Thread(t);
            Thread t2=new Thread(t);
            Thread t3=new Thread(t);
            t1.start();
            t2.start();
            t3.start();
      }
}
```

Output:-

```
0 param constructor
hascode of printer class is=1618742454 and thread is=Thread-0
hascode of printer class is=1618742454 and thread is=Thread-2
hascode of printer class is=1618742454 and thread is=Thread-1
```

"Eager instantiation is bad practice because we don't call object it create object so we should not depend on this approach"

SOLUTION FOR MULTITHREADING ENV

2. Using synchronized

Synchronized is meant for lock on the method using synchronized one thread act on method only one time until one that execution is complete once one thread execution is completed then other thread act on that method

Ex:-

Printer class:-

```
package com.nit.dp;

public class printer {
     //private static printer instence=new printer();         //egar
instantiation only if object creation at the time of declaration
     private static printer instence;
     private printer() {
          System.out.println("0 param constructor");
     }
     /*public static printer getinstence() {
          return instence;
     }*/
     //statif factory method
     public synchronized static printer getInstance() {
          //static factory method
          if(instence==null) {
               instence =new printer();    //lazy instantiation good
practice
          }
          return instence;
     }
     public void printDate(String data) {
          System.out.println(data);
     }
}
```

SOLUTION FOR MULTITHREADING ENV

ticketBooking.java

```
package com.nit.dp;

public class ticketBooking implements Runnable  {
	printer p=null;
	@Override
	 public void run() {
		p=printer.getInstance();
		System.out.println("hascode of printer class is="+p.hashCode()+" and
thread is="+Thread.currentThread().getName());
	}
}
```

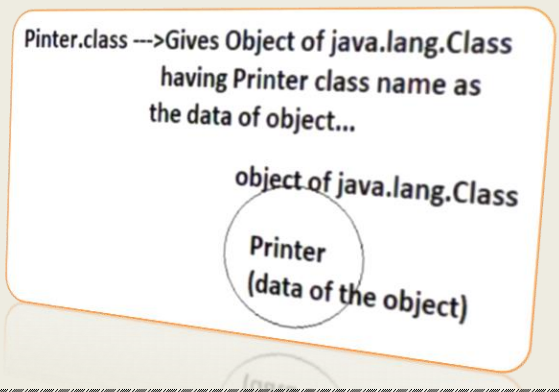
tiketBookingTest.java

```
package com.nit.test;
import com.nit.dp.ticketBooking;
public class ticketBookingtest {
	public static void main(String[] args) {
		ticketBooking t=new ticketBooking();
		Thread t1=new Thread(t);
		Thread t2=new Thread(t);
		Thread t3=new Thread(t);
		t1.start();
		t2.start();
		t3.start();
	}
}
```

Output:-

```
	0 param constructor
	hascode of printer class is=1618742454 and thread is=Thread-0
	hascode of printer class is=1618742454 and thread is=Thread-2
	hascode of printer class is=1618742454 and thread is=Thread-1
```

"instead of making whole method synchronized its recommended to work with synchronized block inside the method…



SOLUTION FOR MULTITHREADING ENV

in side synchronized block we can write this but since method is static so we can not write this then we should be write ClassName ……..it gives (object…….{java.lang.Ojbect}) of class having class name so indirectly it gives class level locking "

ex:-

```
package com.nit.dp;

public class printer {
//egar instantiation only if object creation at the time of declaration
	//private static printer instence=new printer();
		private static printer instence;
	private printer() {
		System.out.println("0 param constructor");
	}

	/*public static printer getinstence() {
		return instence;
	}*/

	//statif factory method
	/*public synchronized static printer getInstance() {
		//static factory method
		if(instence==null) {
			instence =new printer();    //lazy instantiation good practice
		}
		return instence;
	}*/

	public static printer getInstance() {
		synchronized (printer.class) {
			//static factory method
			if(instence==null) {
				instence =new printer();    //lazy instantiation good practice
			}
		}

		return instence;
	}
	public void printDate(String data) {
		System.out.println(data);
	}
}
```

***** by changing synchronized block it will also create one object only*****

SOLUTION FOR MULTITHREADING ENV

Class level locking:-

One thread at a time to access the class to create the object.

Object level locking:-

One thread at a time to act on a object

Q:- in the synchronization what Is good ?

Ans:- synchronized block is good

==========

"still using thread block having some problem ...it may consume performance to improve performance we have to do double null checking so performance will increase and time also will save"

Double null checking :-

Ex:-

```
package com.nit.dp;
public class printer {
//egar instantiation only if object creation at the time of declaration
	//private static printer instence=new printer();
		private static printer instence;
	private printer() {
		System.out.println("0 param constructor");
	}

	/*public static printer getinstence() {
		return instence;
	}*/

	//statif factory method
	/*public synchronized static printer getInstance() {
		//static factory method
		if(instence==null) {
			instence =new printer();    //lazy instantiation good practice
		}
		return instence;
	}*/
```

SOLUTION FOR MULTITHREADING ENV

```
public static printer getInstance() {
      if(instence==null) {                          //1st null checking
            synchronized (printer.class) {
                  //static factory method
                  if(instence==null) {              //2nd null checking
                        instence =new printer();    //lazy instantiation good practice
                  }
            }
      }
            return instence;
}
      public void printDate(String data) {
            System.out.println(data);
      }
}
```

“in above program we are doing double null check solution because once more then one thread comes to the method first one thread comes inside and verify wather object is created or not condition will true it comes inside synchronized block and thread gone lock again it will check object is created or not condition true it create object and return object to the method then thread 2 comes and check object created or not …condition false then directly control go outside the method with exacting object so no need to check inside the synchronized block so it will improve performance so we can verify before synchronized block only from outside block its returning object”

“Synchronized block is good with double null check ”

SOLUTION FOR MULTITHREADING ENV

SOME IMPORTANT POINTS

1. We should not decide whether object is created or not based on constructor executed or not because some exception cases happen some situation.
2. When we create subclass object of the abstract class along with sub class constructor then abstract class construct also executed but it doesn't mean object is created for abstract class.
3. Similarly when we create of object through cloning and deserialization process object will be created but constructor will not execute.