**Singleton Pattern Interview Questions:**

### Another way to create singleton object...

Another way to create singleton objects::::  
  
**1>>>>>>>>>>>**  
package designPattern;  
  
class SingletonObjectAnotherDefinition{  
 private static class SingletonPattern {  
 public static SingletonObjectAnotherDefinition instance = new SingletonObjectAnotherDefinition();  
 }  
  
 public static SingletonObjectAnotherDefinition getInstance() {  
 return SingletonPattern.instance;  
 }  
}  
  
**2>>>>>>>>>>>>>>**  
class SingletonPatternAnotherImplementation {  
  
 private static SingletonPatternAnotherImplementation instance = null;  
  
 static {  
 instance = new SingletonPatternAnotherImplementation();  
 System.out.println("Static block acccessed..Class is loaded successfully,..");  
 }  
  
 public SingletonPatternAnotherImplementation() {  
 if(instance!=null) {  
 throw new IllegalStateException("Singleton double-instantiation, should never happen!");  
 }  
 }  
  
 public static SingletonPatternAnotherImplementation getSingleton() {  
 return instance;  
 }  
  
}  
  
  
/\*\*  
 \* The Class SingletonPatternAnotherWay.  
 \*/  
public class SingletonPatternAnotherWay {  
  
 /\*\*  
  \* The main method.  
  \*  
  \* @param args the arguments  
  \* @throws ClassNotFoundException  
  \* @throws IllegalAccessException  
  \* @throws InstantiationException  
  \*/  
 public static void main(String[] args) throws ClassNotFoundException, InstantiationException, IllegalAccessException {  
 System.out.println("Through SingletonObjectAnotherDefinition 1 "+ SingletonObjectAnotherDefinition.getInstance());  
 System.out.println("Through SingletonObjectAnotherDefinition 2 "+SingletonObjectAnotherDefinition.getInstance());  
 System.out.println("Through SingletonObjectAnotherDefinition 3 "+SingletonObjectAnotherDefinition.getInstance());  
 System.out.println("Through SingletonPatternAnotherImplementation 1 "+ SingletonPatternAnotherImplementation.getSingleton());  
 System.out.println("Through SingletonPatternAnotherImplementation 2 "+SingletonPatternAnotherImplementation.getSingleton());  
 System.out.println("Through SingletonPatternAnotherImplementation 3 "+SingletonPatternAnotherImplementation.getSingleton());  
 System.out.println(Class.forName("designPattern.SingletonPatternAnotherImplementation").newInstance());  
 }  
}

///////////////////////////////////////OUTPUT/////////////////////////////////////

Through SingletonObjectAnotherDefinition 1 designPattern.SingletonObjectAnotherDefinition@addbf1

Through SingletonObjectAnotherDefinition 2 designPattern.SingletonObjectAnotherDefinition@addbf1

Through SingletonObjectAnotherDefinition 3 designPattern.SingletonObjectAnotherDefinition@addbf1

Static block acccessed..Class is loaded successfully,..

Through SingletonPatternAnotherImplementation 1 designPattern.SingletonPatternAnotherImplementation@9304b1

Through SingletonPatternAnotherImplementation 2 designPattern.SingletonPatternAnotherImplementation@9304b1

Through SingletonPatternAnotherImplementation 3 designPattern.SingletonPatternAnotherImplementation@9304b1

Exception in thread "main" java.lang.IllegalStateException: Singleton double-instantiation, should never happen!

at designPattern.SingletonPatternAnotherImplementation.<init>(SingletonPatternAnotherWay.java:25)

at sun.reflect.NativeConstructorAccessorImpl.newInstance0(Native Method)

at sun.reflect.NativeConstructorAccessorImpl.newInstance(NativeConstructorAccessorImpl.java:39)

at sun.reflect.DelegatingConstructorAccessorImpl.newInstance(DelegatingConstructorAccessorImpl.java:27)

at java.lang.reflect.Constructor.newInstance(Constructor.java:513)

at java.lang.Class.newInstance0(Class.java:355)

at java.lang.Class.newInstance(Class.java:308)

at designPattern.SingletonPatternAnotherWay.main(SingletonPatternAnotherWay.java:56)

**Singleton object serializable must implement readResolve() method:**

**If the Singleton class implements the java.io.Serializable interface, when a singleton is serialized and then deserialized more than once, there will be multiple instances of Singleton created. In order to avoid this readResolve method should be implemented.**

**package** designPattern;

**import** java.io.FileInputStream;

**import** java.io.FileNotFoundException;

**import** java.io.FileOutputStream;

**import** java.io.IOException;

**import** java.io.ObjectInputStream;

**import** java.io.ObjectOutputStream;

**import** java.io.Serializable;

/\*\*

\* The Class SingletonObjectSerializing.

\*/

**public** **class** SingletonObjectSerializing {

/\*\*

\* The main method.

\* **@param** args the arguments

\*/

**public** **static** **void** main(String[] args) {

System.*out*.println("Singleton object: "+SingletonObject.*getInstance*());

**try** {

ObjectOutputStream objOut=**new** ObjectOutputStream(**new** FileOutputStream("SingletonObject.txt"));

objOut.writeObject(SingletonObject.*getInstance*());

System.*out*.println("Singleton object serialized: "+SingletonObject.*getInstance*());

} **catch** (FileNotFoundException e) {

e.printStackTrace();

} **catch** (IOException e) {

e.printStackTrace();

}

**try** {

ObjectInputStream objIn=**new** ObjectInputStream(**new** FileInputStream("SingletonObject.txt"));

SingletonObject sinObj=**null**;

**try** {

sinObj = (SingletonObject) objIn.readObject();

} **catch** (ClassNotFoundException e) {

e.printStackTrace();

}

System.*out*.println("Singleton object deserialized: "+sinObj);

} **catch** (FileNotFoundException e) {

e.printStackTrace();

} **catch** (IOException e) {

e.printStackTrace();

}

}

}

/\*\*

\* The Class SingletonObject.

\*/

**class** SingletonObject **implements** Serializable{

**private** **static** **final** **long** *serialVersionUID* = 1L;

/\*\*

\* Instantiates a new singleton object.

\*/

**private** SingletonObject(){

**super**();

System.*out*.println("Instantiated..");

}

/\*\* The instance. \*/

**private** **static** SingletonObject *instance*=**new** SingletonObject();

/\*\*

\* Gets the single instance of SingletonObject.

\*

\* **@return** single instance of SingletonObject

\*/

**public** **static** SingletonObject getInstance(){

**return** *instance*;

}

/\*\*

\* Read resolve.

\* This method is called immediately after an object of this class is deserialized.

\* This method returns the singleton instance.

\* **@return** the object

\*/

**protected** Object readResolve() {

**return** *getInstance*();

}

}

**Output>>**

**Instantiated..**

**Singleton object: designPattern.SingletonObject@addbf1**

**Singleton object serialized: designPattern.SingletonObject@addbf1**

**Singleton object deserialized: designPattern.SingletonObject@addbf1**

**package** designPattern;

**import** java.io.FileInputStream;

**import** java.io.FileNotFoundException;

**import** java.io.FileOutputStream;

**import** java.io.IOException;

**import** java.io.ObjectInputStream;

**import** java.io.ObjectOutputStream;

**import** java.io.Serializable;

/\*\*

\* The Class SingletonObjectSerializing.

\*/

**public** **class** SingletonObjectSerializing {

/\*\*

\* The main method.

\* **@param** args the arguments

\*/

**public** **static** **void** main(String[] args) {

System.*out*.println("Singleton object: "+SingletonObject.*getInstance*());

**try** {

ObjectOutputStream objOut=**new** ObjectOutputStream(**new** FileOutputStream("SingletonObject.txt"));

objOut.writeObject(SingletonObject.*getInstance*());

System.*out*.println("Singleton object serialized: "+SingletonObject.*getInstance*());

} **catch** (FileNotFoundException e) {

e.printStackTrace();

} **catch** (IOException e) {

e.printStackTrace();

}

**try** {

ObjectInputStream objIn=**new** ObjectInputStream(**new** FileInputStream("SingletonObject.txt"));

SingletonObject sinObj=**null**;

**try** {

sinObj = (SingletonObject) objIn.readObject();

} **catch** (ClassNotFoundException e) {

e.printStackTrace();

}

System.*out*.println("Singleton object deserialized: "+sinObj);

} **catch** (FileNotFoundException e) {

e.printStackTrace();

} **catch** (IOException e) {

e.printStackTrace();

}

}

}

/\*\*

\* The Class SingletonObject.

\*/

**class** SingletonObject **implements** Serializable{

**private** **static** **final** **long** *serialVersionUID* = 1L;

/\*\*

\* Instantiates a new singleton object.

\*/

**private** SingletonObject(){

**super**();

System.*out*.println("Instantiated withoutreadResolve method..");

}

/\*\* The instance. \*/

**private** **static** SingletonObject *instance*=**new** SingletonObject();

/\*\*

\* Gets the single instance of SingletonObject.

\*

\* **@return** single instance of SingletonObject

\*/

**public** **static** SingletonObject getInstance(){

**return** *instance*;

}

/\*\*

\* Read resolve.

\* This method is called immediately after an object of this class is deserialized.

\* This method returns the singleton instance.

\* **@return** the object

\*/

/\*protected Object readResolve() {

return getInstance();

}\*/

}

**Output>>**

**Instantiated without readResolve method..**

**Singleton object: designPattern.SingletonObject@19821f**

**Singleton object serialized: designPattern.SingletonObject@19821f**

***Singleton object deserialized: designPattern.SingletonObject@de6ced***

**Singleton pattern for multithreaded environment:**

**package designPattern;**

**import java.io.Serializable;**

**import java.util.Scanner;**

**/\*\***

**\* The Class SingletonPatternMultithreaded.**

**\*/**

**public class SingletonPatternMultithreaded {**

**/\*\***

**\* The main method.**

**\* @param args the arguments**

**\*/**

**public static void main(String[] args) {**

**int key=0;**

**Scanner sc=new Scanner(System.in);**

**System.out.println("Get singleton object..");**

**System.out.println("Enter a choice for singleton object-->");**

**System.out.println("Choice 1:NonStatic");**

**System.out.println("Choice 2:Static");**

**System.out.println("Choice 0:Exit");**

**key=sc.nextInt();**

**switch (key) {**

**case 1:**

**new MyThread1("non-static");**

**new MyThread2("non-static");**

**new MyThread1("non-static");**

**new MyThread2("non-static");**

**System.out.println("NON-STATIC SINGLETON OBJECT ACCESSED BY main: "+MultithreadedSingletonObject.getInstance());**

**break;**

**case 2:**

**new MyThread1 ("static");**

**new MyThread2 ("static");**

**new MyThread1("static");**

**new MyThread2("static");**

**System.out.println("STATIC SINGLETON OBJECT ACCESSED BY main: "+MultithreadedSingletonObjectBestSolution.getInstance());**

**break;**

**default:**

**System.exit(0);**

**break;**

**}**

**}**

**}**

**/\*\***

**\* The Class MultithreadedSingletonObject.**

**\*/**

**class MultithreadedSingletonObject implements Serializable{**

**private static final long serialVersionUID = 1L;**

**/\*\* The instance. \*/**

**private static MultithreadedSingletonObject instance;**

**/\*\***

**\* Gets the single instance of MultithreadedSingletonObject.**

**\***

**\* @return single instance of MultithreadedSingletonObject**

**\*/**

**public static MultithreadedSingletonObject getInstance(){**

**if (instance == null){**

**synchronized(MultithreadedSingletonObject.class){**

**MultithreadedSingletonObject inst = instance;**

**if (inst == null){**

**synchronized(MultithreadedSingletonObject.class) {**

**instance = new MultithreadedSingletonObject();**

**}**

**}**

**}**

**}**

**return instance;**

**}**

**/\*\***

**\* Read resolve.**

**\* This method is called immediately after an object of this class is deserialized.**

**\* This method returns the singleton instance.**

**\* @return the object**

**\*/**

**protected Object readResolve() {**

**return getInstance();**

**}**

**}**

**/\*\***

**\* The Class MultithreadedSingletonObjectBestSolution.**

**\* Most preffered singleton pattern**

**\*/**

**class MultithreadedSingletonObjectBestSolution implements Serializable{**

**private static final long serialVersionUID = 1L;**

**/\*\* The instance. \*/**

**private static MultithreadedSingletonObjectBestSolution instance=new MultithreadedSingletonObjectBestSolution();**

**/\*\***

**\* Gets the single instance of MultithreadedSingletonObject.**

**\***

**\* @return single instance of MultithreadedSingletonObject**

**\*/**

**public static MultithreadedSingletonObjectBestSolution getInstance(){**

**return instance;**

**}**

**/\*\***

**\* Read resolve.**

**\* This method is called immediately after an object of this class is deserialized.**

**\* This method returns the singleton instance.**

**\* @return the object**

**\*/**

**protected Object readResolve() {**

**return getInstance();**

**}**

**}**

**/\*\***

**\* The Class MyThread1.**

**\*/**

**class MyThread1 extends Thread{**

**/\*\* The mode. \*/**

**String mode;**

**/\*\***

**\* Instantiates a new my thread1.**

**\***

**\* @param mode the mode**

**\*/**

**MyThread1(String mode){**

**super();**

**this.mode=mode;**

**setName("MyThread1");**

**start();**

**}**

**/\* (non-Javadoc)**

**\* @see java.lang.Thread#run()**

**\*/**

**public void run(){**

**if(mode.equalsIgnoreCase("non-static")){**

**System.out.println("NON-STATIC SINGLETON OBJECT ACCESSED BY "+getName()+": "+MultithreadedSingletonObject.getInstance());**

**}else{**

**System.out.println("STATIC SINGLETON OBJECT ACCESSED BY "+getName()+": "+MultithreadedSingletonObjectBestSolution.getInstance());**

**}**

**}**

**}**

**/\*\***

**\* The Class MyThread2.**

**\*/**

**class MyThread2 extends Thread{**

**/\*\* The mode. \*/**

**String mode;**

**/\*\***

**\* Instantiates a new my thread2.**

**\***

**\* @param mode the mode**

**\*/**

**MyThread2(String mode){**

**super();**

**this.mode=mode;**

**setName("MyThread2");**

**start();**

**}**

**/\* (non-Javadoc)**

**\* @see java.lang.Thread#run()**

**\*/**

**public void run(){**

**if(mode.equalsIgnoreCase("non-static")){**

**System.out.println("NON-STATIC SINGLETON OBJECT ACCESSED BY "+getName()+": "+MultithreadedSingletonObject.getInstance());**

**}else{**

**System.out.println("STATIC SINGLETON OBJECT ACCESSED BY "+getName()+": "+MultithreadedSingletonObjectBestSolution.getInstance());**

**}**

**}**

**}**

**Output>**

Get singleton object..

Enter a choice for singleton object-->

Choice 1:NonStatic

Choice 2:Static

Choice 0:Exit

**1**

NON-STATIC SINGLETON OBJECT ACCESSED BY MyThread1: designPattern.MultithreadedSingletonObject@1270b73

NON-STATIC SINGLETON OBJECT ACCESSED BY MyThread1: designPattern.MultithreadedSingletonObject@1270b73

NON-STATIC SINGLETON OBJECT ACCESSED BY main: designPattern.MultithreadedSingletonObject@1270b73

NON-STATIC SINGLETON OBJECT ACCESSED BY MyThread2: designPattern.MultithreadedSingletonObject@1270b73

NON-STATIC SINGLETON OBJECT ACCESSED BY MyThread2: designPattern.MultithreadedSingletonObject@1270b73

Get singleton object..

Enter a choice for singleton object-->

Choice 1:NonStatic

Choice 2:Static

Choice 0:Exit

2

STATIC SINGLETON OBJECT ACCESSED BY main: designPattern.MultithreadedSingletonObjectBestSolution@173a10f

STATIC SINGLETON OBJECT ACCESSED BY MyThread2: designPattern.MultithreadedSingletonObjectBestSolution@173a10f

STATIC SINGLETON OBJECT ACCESSED BY MyThread2: designPattern.MultithreadedSingletonObjectBestSolution@173a10f

STATIC SINGLETON OBJECT ACCESSED BY MyThread1: designPattern.MultithreadedSingletonObjectBestSolution@173a10f

STATIC SINGLETON OBJECT ACCESSED BY MyThread1: designPattern.MultithreadedSingletonObjectBestSolution@173a10f