Singleton pattern is one of the simplest design patterns in Java. This type of design pattern comes under creational pattern as this pattern provides one of the best ways to create an object.

This pattern involves a single class which is responsible to create an object while making sure that only single object gets created. This class provides a way to access its only object which can be accessed directly without need to instantiate the object of the class.

We're going to create a *Student* class. *Student* class have its constructor as private and have a static instance of itself.

*Student* class provides a static method to get its static instance to outside world. *StudentDemo*, our demo class will use *Student* class to get a *Student* object.



class Student{

private static Student student = null;

private Student() {

}

public static Student getStudent() {

if(student == null) {

student = new Student();

}

return student;

}

}

public class StudentDemo {

public static void main(String[] args) {

Student s1 = Student.getStudent();

Student s2 = Student.getStudent();

if(s1 == s2) {

System.out.println("true");

}

}

}

Does Clone method affects singleton?

By default clone method is protected , it means clone method which is inherited from Object class is protected. So other class call clone method of a class directly. If a class make its clone method as public then only it can be called.

If clone method is public , make clone method to throw cloneNotSupportedException.

class Student implements Cloneable{

private static Student student = null;

private Student() {

}

public static Student getStudent() {

if(student == null) {

student = new Student();

}

return student;

}

@Override

protected Object clone() throws CloneNotSupportedException {

// TODO Auto-generated method stub

return super.clone();

}

}

public class StudentDemo {

public static void main(String[] args) {

try {

Student s1 = Student.getStudent();

Student s2 = (Student)s1.clone();

if(s1 == s2) {

System.out.println("true");

}else {

System.out.println("false");

}

System.out.println(s1.hashCode());

System.out.println(s2.hashCode());

}catch(CloneNotSupportedException e) {

System.out.println(e.getMessage());

}

}

}

//output : gives two different objects which violates singleton pattern.

In case clone method is public , Declare clone method to throw CloneNotSupportedException and throw CloneNotSupportedException.

class Student implements Cloneable{

private static Student student = null;

private Student() {

}

public static Student getStudent() {

if(student == null) {

student = new Student();

}

return student;

}

@Override

protected Object clone() throws CloneNotSupportedException {

// TODO Auto-generated method stub

throw new CloneNotSupportedException();

}

}

public class StudentDemo {

public static void main(String[] args) {

try {

Student s1 = Student.getStudent();

Student s2 = (Student)s1.clone();

if(s1 == s2) {

System.out.println("true");

}else {

System.out.println("false");

}

System.out.println(s1.hashCode());

System.out.println(s2.hashCode());

}catch(CloneNotSupportedException e) {

System.out.println(e.getMessage());

}

}

}