**Singleton**

**Definition**

One instance of a class or one value accessible globally in an application.

**Where to use & benefits**

* Ensure unique instance by defining class final to prevent cloning.
* May be extensible by the subclass by defining subclass final.
* Make a method or a variable public or/and static.
* Access to the instance by the way you provided.
* Well control the instantiation of a class.
* Define one value shared by all instances by making it static.
* Related patterns include
  + [Abstract factory](http://www.javacamp.org/designPattern/abstractfactory.html), which is often used to return unique objects.
  + [Builder](http://www.javacamp.org/designPattern/Builder.html), which is used to construct a complex object, whereas a singleton is used to create a globally accessible object.
  + [Prototype](http://www.javacamp.org/designPattern/Prototype.html), which is used to copy an object, or create an object from its prototype, whereas a singleton is used to ensure that only one prototype is guaranteed.

**Example**

One file system, one window manager, one printer spooler, one Test engine, one Input/Output socket and etc.

To design a Singleton class, you may need to make the class final like java.Math, which is not allowed to subclass, or make a variable or method public and/or static, or make all constructors private to prevent the compiler from creating a default one.

For example, to make a unique remote connection,

final class RemoteConnection {

private Connect con;

private static RemoteConnection rc = new RemoteConnection(connection);

private RemoteConnection(Connect c) {

con = c;

....

}

public static RemoteConnection getRemoteConnection() {

return rc;

}

public void setConnection(Connect c) {

this(c);

}

}

usage:

RemoteConnection rconn = RemoteConnection.getRemoteConnection;

rconn.loadData();

...

The following statement may fail because of the private constructor

RemoteConnection con = new RemoteConnection(connection); //failed

//failed because you cannot subclass it (final class)

class Connection extends RemoteConnection {}

For example, to use a static variable to control the instance;

class Connection {

public static boolean haveOne = false;

public Connection() throws Exception{

if (!haveOne) {

doSomething();

haveOne = true;

}else {

throw new Exception("You cannot have a second instance");

}

}

public static Connection getConnection() throws Exception{

return new Connection();

}

void doSomething() {}

//...

public static void main(String [] args) {

try {

Connection con = new Connection(); //ok

}catch(Exception e) {

System.out.println("first: " +e.getMessage());

}

try {

Connection con2 = Connection.getConnection(); //failed.

}catch(Exception e) {

System.out.println("second: " +e.getMessage());

}

}

}

| C:\ Command Prompt  C:\> java Connection  second: You cannot have a second instance |
| --- |

For example to use a public static variable to ensure a unique.

class Employee {

public static final int companyID = 12345;

public String address;

//...

}

class HourlyEmployee extends Employee {

public double hourlyRate;

//....

}

class SalaryEmployee extends Employee {

public double salary;

//...

}

class Test {

public static void main(String[] args) {

Employee Evens = new Employee();

HourlyEmployee Hellen = new HourlyEmployee();

SalaryEmployee Sara = new SalaryEmployee();

System.out.println(Evens.companyID == Hellen.companyID); //true

System.out.println(Evens.companyID == Sara.companyID); //true

}

}

| C:\ Command Prompt  C:\> java Test  true  true |
| --- |

The companyID is a unique and cannot be altered by all subclasses.

Note that Singletons are only guaranteed to be unique within a given class

loader. If you use the same class across multiple distinct enterprise

containers, you'll get one instance for each container.

Whether you need to use synchronized keyword to manage the method access, it depends on your project situation and thread controlling.