A singleton class means only one object can be created from the class.

public class SingletonA {

// a private static reference to the only object.

private static final SingletonA instance = new SingletonA();

// a private constructor, make sure outsider cannot use new keyword.

private SingletonA() {

if (instance != null)

throw new RuntimeException("Cannot create");

}

// the only public method to return the only reference.// it is static so we can use Class name to call it. SingletonA.getInstance()

public static SingletonA getInstance() {

return instance;

}

}

// combine all of the above, private constructor, public get instance, static reference.// object is create eagerly. No thread safety issue in this one.

Also we can create a double-lock checking style, which lazily instantiats the object.

public class SingletonB {

private static volatile SingletonD instance= null; // volatile - prevent instruction reordering

private SingletonB(){} // private constructor

public static SingletonB getInstance(){// check once. none blocking return.

if(instance == null){

// if null, race for the lock

synchronized(SingletonB.class){

if(instance == null){ // check twice in case the previous thread who got the lock already created the instance.

instance = new SingletonB();

}

}

}

return instance;

}}

Testing Code:

public class SingletonTest {

public static void main(String[] args) {

Runnable r = ()->{

System.out.println(SingletonD.getInstance());

}; // lambda

ExecutorService es = Executors.newFixedThreadPool(40); // thread pool

for(int i = 0 ; i < 50; i++){

es.submit(r);

}

es.shutdown();

}}// if the class is thread safe, all the objects should be the same name.// if you see two or more result got printed out. that means multiple object got create which is against the defination o