1. Provide a “traditional” implementation of the Singleton Pattern (lazy instantiation) that is thread safe (like the example we saw in class). Explain why your code guarantees thread safety.

* In order to make sure the implementation of the Singleton Pattern(lazy Instantiation) that is thread safe we need to use the keyword “synchronized” because this method will lock the object so that even there are many other thread invoked the object already locked until the process finished then we will be sure that only one instance will be created.

**public** **class** Singleton {

**private** **static** Singleton *uniqueInstance* = **null**;

**public** **synchronized** **static** Singleton getInstance() {

**if** (*uniqueInstance* == **null**) {

*uniqueInstance* = **new** Singleton();

}

**return** *uniqueInstance*;

}

**private** Singleton() {

System.***out***.println("Tranditional singleton");

}

**public** **void** doPrint() {

System.***out***.println("I am doing the printing for tranditional singleton!!!");

}

}

1. Is there a better and simpler way to implement a singleton class (that guarantees single instance and thread safety by itself)? Provide a solution with Java code. Submit a Word document for both 1 and 2

* A better and simpler way to implement a singleton class is using ***emun*** .

**public** **enum** Singleton {

***INSTANCE***;

**private** Singleton() {

System.***out***.println("I am here!!!");

}

**public** **void** doPrint() {

System.***out***.println("I am doing the printing!!!");

}

}