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.

public class SingletonLazy {

private static class Singleton{

private static SingletonLazy *uniqueInstance* = new SingletonLazy();

}

public static SingletonLazy getInstance() {

return Singleton.*uniqueInstance*;

}

private SingletonLazy() {}

// Accessors and mutators here!

}

When the singleton class is loaded, inner class is not loaded and hence doesn’t create object when loading the class. Inner class is cre­ated only when getInstance() method is called.

2) 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.

public enum SingletonEnum {

INSTANCE;

// Accessors and mutators here!

}