**Problem Statement 1**

**Exercise 1: Implementing the Singleton Pattern**

**Scenario:**

You need to ensure that a logging utility class in your application has only one instance throughout the application lifecycle to ensure consistent logging.

Code -

class Logger

{

private static Logger logger;

//make static too!

//private static Logger()

//constructor cannot be static I see - There is a reason for it!

//Constructor must be declared!

private Logger()

{

//logger = new Logger();

//is not it a recursive loop?

//how about below?

//logger = this;

//no need of anything

//only requirement is Constructor must be declared and must be private;

}

public static Logger getLogger()

{

if(logger == null)

{

logger = new Logger();

}

return logger;

}

}

class TestClass1

{

public static void unitTest()

{

Logger log1 = Logger.getLogger();

//System.out.println(log1);

//can even display hashcodes

//but useless, because hashcoes can turn out to be same

Logger log2 = Logger.getLogger();

//System.out.println(log2);

if(log1==log2)

System.out.println("Hurray!, Singleton working fine.");

else

System.out.println("Alas!, Singleton seems to fail.");

}

}

**Output**

Hurray!, Singleton working fine.