**John Wangari & James Naurot**

**We went over the code and implemented the Singleton Design pattern. We also implemented the Printer class. Both of us considered how the implementation of Singleton Design pattern requires the use of a static variable and the exclusion of repeated use of the constructor.**

**COSC 436 Object-Oriented Design and Programming**

**In-class Exercise: Singleton**

**Scenario:**

A company wants to build a **PrinterManager** class to manage a number of printers available in the company. This **PrinterManager** is the only instance and entry point for anyone who wants to print a document with any printers in the company. After a print job is sent to the **PrinterManager**, the **PrinterManger** will check whether a connected printer is available. If there is an available printer, the **PrinterManger** will send the job to it. If all the printers are busy, an error message is return. Since the system is still in prototyping phase, if a printer is assigned a print job, it will stay as unavailable state, unless the **PrinterManager** reset it.

**Objective:**

Please use Singleton pattern to implement **PrinterManager** class, provide necessary functions based on the scenario described above. Create a client to demonstrate the use of it.

**Tasks: (You are encouraged to work with a partner)**

1. The **Printer** class is already implemented (code is available in Blackboard). Import it in to your package.
2. Create a new class called **PrinterManager**.
3. In **PrinterManager,** define a private static “single instance”, define an array of Printer called **printers** to store available printers.
4. Create a private constructor for **PrinterManager.**

// private constructor

private PrinterManager(int numPrinter) {

printers = new Printer[numPrinter];

// initialize them

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

printers[i] = new Printer();

}

}

1. Create a public static accessor getInstance() method in **PrinterManager**.

// important method of returning singleton instance

public static PrinterManager getInstance() {

// no existing instance

// create a new one

if (instance == null)

instance = new PrinterManager(8);

// if there is one exists

// just return it.

return instance;

}

1. Create a method called **assignJob.** It is able to assign a job to an available printer.

// assign a job to printer

public String assignJob(String printJob) {

// check available printers

for (Printer printer : printers) {

if (printer.isBusy() == false) {

printer.setBusy(true);

printer.setPrintJob(printJob);

return "job is assigned.";

}

}

// no available printers

return "all printers are busy, please try again later.";

}

1. Create a method called **showStatus**, which shows the status of every printer.

public void showStatus()

{

for (int i = 0; i < printers.length; i++) {

System.out.println("Status of Printer "+i+":");

if(printers[i].isBusy)

{

System.out.println("Busy");

}else

System.out.println("Available");

}

}

1. Create a **Client** class and use the following **main** method to test your implementation.

public static void main(String[] args) {

// get the singleton instance

PrinterManager printerManager = PrinterManager.getInstance();

// assign some job

printerManager.assignJob("print something");

printerManager.assignJob("print something again");

// show status

printerManager.showStatus();

// check if you can get another instance

PrinterManager printerManager2 = PrinterManager.getInstance();

// show status

printerManager2.showStatus();

}

1. You are welcome to add additional methods/implementation based on the scenarios.

Upload your code to the Blackboard when you are done.