Assignment 4 – Concurrency

SEG2106 – Software Construction

Part A – Monitor Development

In an office environment, there are an unspecified number of desks, and two kinds of workers: clerks and cleaners. Workers behave as follows:

Clerks may arrive at the office at any time. In order to get to work, they wait until they have exclusive access to an available desk that has been cleaned. After working for an unspecified amount of time, they leave the desk, which needs cleaning.

Cleaners work on one desk at a time. Each cleaner waits until a clerk leaves a desk, at which point the cleaner starts cleaning the desk, and then signals when cleaning is completed.

We model the office environment in Java pseudo-code using monitor classes:

• Desks corresponds to instances of a class Desk, whose details are irrelevant but has a binary state is_clean.

• The office is an instance of type Office with interface:

```
interface Office
{
  Desk arrive();              // get an available, clean desk
  void leave(Desk desk);      // leave desk ‘desk’
  Desk service();             // get a desk to clean
  void cleaned(Desk desk);    // desk ‘desk’ is now clean
}
```

• Clerks and cleaners correspond to threads sharing an instance of Office and calling its public methods continuously as follows:

```
Office office = new OpenOffice();
```

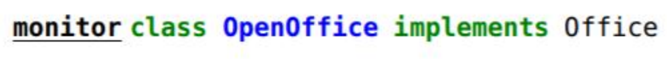
clerk$_n$

```
while (true) {
  Desk desk = office.arrive();
  // work at desk
  office.leave(desk);
}
```

cleaner$_m$

```
while (true) {
  Desk desk = office.service();
  // clean desk
  office.cleaned(desk);
}
```

The goal of this exercise is to provide a monitor class OpenOffice with interface Office, whose behavior follows the above description. You should use the Java pseudo-code for monitors that we used in class:



```
monitor class OpenOffice implements Office
```

Develop the above monitor class OpenOffice and demonstrate it using a unit test console application.

HINT:

Assume the following queues are declared in the office monitor: (https://docs.oracle.com/javase/7/docs/api/java/util/Queue.html)

Queue <Desk> cleanDesks, dirtyDesks; // sets of clean, dirty desks

Use a lock with the following conditions:

Condition clean; // condition variable: a clean desk is available

Condition dirty; // condition variable: a dirty desk is available

Part B

Q1 List the 4 necessary condition for a deadlock to happen

Q2 List the two types of synchronization