University of Westminster
School of Computer Science and Engineering

6SENG006C Concurrent Programming Coursework (2022/23)

Module leader	Guhanathan P (Guhanathan.p@iit.ac.lk)
Unit	Coursework
Weighting:	50%
Qualifying mark	30%
Description	To develop: (a) An FSP program to model a shared printer. (b) A concurrent (multi-threaded) Java program to implement the FSP model of the shared printer.
Learning Outcomes Covered in this Assignment:	The coursework assesses learning outcomes: LO1, LO2, LO3 & LO4.
Handed Out:	Week 07
Due Date	**13:00, Thursday, 12th January, 2023**
Expected deliverables	Electronic files: (a) The FSP Process Composition Analysis & Design Form for the FSP process composition representing the complete Printing system. (PDF file format.) (b) The FSP program for the printing system, called "Printer.lts". (Plain ASCII text file.) (c) The Java source code for each individual Java class that implements the FSP model. Java code files (".java") should be plain ASCII text format. (d) Screen shots demonstrating the use of the FSP tool LTSA to animate and analyse the FSP model. Sample output from Java program. All files should be compressed into a single ZIP archive. The ZIP archive should be named using your surname & "CW", e.g. "howells_CW.zip"
Method of	Online via Blackboard

Submission:	
Type of Feedback and Due Date:	Verbal feedback in tutorials as the assessment progresses. Electronic via module's Blackboard bulletin board. Written feedback and marks 15 working days (3 weeks) after the submission deadline. **All marks will remain provisional until formally agreed by an Assessment Board.**

Assessment regulations

Refer to the “How you study” guide for postgraduate students for a clarification of how you are assessed, penalties and late submissions, what constitutes plagiarism etc.

Penalty for Late Submission

If you submit your coursework late but within 24 hours or one working day of the specified deadline, 10 marks will be deducted from the final mark, as a penalty for late submission, except for work which obtains a mark in the range 40 – 49%, in which case the mark will be capped at the pass mark (40%). If you submit your coursework more than 24 hours or more than one working day after the specified deadline you will be given a mark of zero for the work in question unless a claim of Mitigating Circumstances has been submitted and accepted as valid.

It is recognised that on occasion, illness or a personal crisis can mean that you fail to submit a piece of work on time. In such cases you must inform the FST Registry in writing on a Mitigating Circumstances (MC) form, giving the reason for your late or non-submission. You must provide relevant documentary evidence with the form. This information will be reported to the relevant Assessment Board that will decide whether the mark of zero shall stand. For more detailed information regarding University Assessment Regulations, please refer to the following website:

http://www.westminster.ac.uk/study/current-students/resources/academic-regulations

Coursework Description

1. Introduction

The coursework requires you to develop an FSP program to model a system of several students that share a printer to print documents and two technicians. One technician refills the paper when it has run out and one technician replaces the toner cartridge when it runs out. In addition, produce a Java implementation of the printing system, using the appropriate Java concurrency features: **threads**, **thread groups** and a **monitor.**

The coursework is in three parts:

1. Develop FSP processes to model the shared printer.
2. This abstract FSP program is then to be translated into a multi-threaded Java program, using appropriate Java concurrency features. The Java program is to be based on the FSP model of the printing system. You are required to develop Java classes to model the FSP processes and the complete system defined in part 1.
3. Provide screen shots of the FSP model in the LTSA tool & an example of a sample output from the Java program.

Both the FSP and Java programs should conform to the general guidelines given in: *FSP Programming Criteria, Java Programming Criteria* and the general style used for both in the Lecture notes and example programs.

NOTE: that you must use the Java monitor feature. DO NOT USE semaphores, the `synchronize` statement or some other Java synchronisation mechanism.

2. Detailed Description of Coursework Components

2.1 FSP Design & Model

You are required to develop three types of FSP processes to model the following: Printer, Student & Technician.

And a parallel composite process to model the complete system.

This is to be done by using the:

- *FSP Process Composition Analysis & Design Form* for the FSP process composition of the complete parallel system.
- The system should be developed incrementally using the LTSA tool.

NOTE: Due to the limits on the number of FSM states that the LTSA tool can draw (approximately 64), the numbers used in the processes below needs to be very small if the FSM diagrams are to be drawn, e.g. in the range 1 to 3.

2.1.1 Description of Individual Processes

The requirements & design of the individual FSP processes is as follows:

(a) *Printer Process*

The maximum number of sheets of paper the printer can contain, is 3. All documents take just one sheet of paper to print. Its behaviour is as follows:

1. It is initialized with 3 sheets of paper.

2. Provided the printer has at least one sheet of paper left, it can be used to print a document.
3. To print a document:
    (i) a user must take mutually exclusive control of the printer,
    (ii) the document is then printed,
    (iii) the user releases mutually exclusive control of the printer,
    (iv) the number of sheets of paper in the printer is reduced by 1 & the printer is then ready to print another document.
4. When the printer has run out of paper, a user cannot use it to print a document.
5. When the printer has run out of paper, the technician can then refill it with the maximum amount of paper, i.e. 3 sheets.

The printer must not suffer from *"data corruption & interference"*. That is each user must have *mutually exclusive access* to the printer while it is being used and it must keep a correct count of the sheets of paper.

(b) *Student Process*

All of the documents the student wants to print, are short & take only one sheet of paper to print. A student's behaviour is as follows:

1. It is initialized with the number of documents it is to print.
2. For each document:
    (i) it takes mutually exclusive control of the printer,
    (ii) prints the document &
    (iii) then releases control of the printer.
3. When it has finished printing all its documents it terminates.

(c) *Technician Process*

Its behaviour is as follows:

1. Repeatedly, check if the printer is out of paper.
2. When the printer is out of paper, it refills the printer with the maximum number of sheets it can take, i.e. 3.

(d) *Printing System*

This combines instances of the above processes in parallel. It must ensure mutual exclusive access is maintained for use of the printer.

This combines the following four processes in parallel:

- One student process that is to print 3 documents.
- One student process that is to print 2 documents.
- One technician process that refills the printer with 3 sheets of paper.
- One printer process that can hold up to 3 sheets of paper.

2.2 Java Implementation of the FSP Model

A version of the FSP printing system model as a multi-threaded concurrent Java program, using the appropriate Java concurrency features, e.g. threads, thread groups, monitors. The Printing System's Java classes should implement the behaviour of the equivalent FSP process.

The Java implementation of the system consists of:

- a shared printer,
- four students who share it and
- two technicians who service it.

The students each use the printer to print several documents

Each technician only does one job, i.e. one only refills the paper tray & the other only replaces the toner cartridge.

You will need to develop Java classes to model: the printer, a student, a paper technician, a toner technician and the whole system.

You must use the following Java class and interfaces:

- `Document class`: this defines the *"document"* that students send to the printer.
- `Printer class`: this defines the *interface* to the shared printer for the students.
- `ServicePrinter interface`: this defines the *interface* to the shared printer for the two technicians.

You must then develop the following Java classes.

2.2.1 Description of Individual Java Components

The requirements & design of the Java classes is as following:

1. A Shared Laser Printer class

Define a class to model a shared printer, called `LaserPrinter`. **Note that this class is a "monitor", NOT a thread.**

The LaserPrinter class must:

- Implement the `ServicePrinter` interface.
  This interface defines the complete interface to the printer for both students and technicians.
- Use the `Document` class.
  Instances of this class are used to represent a *"document"* that students request the printer to be printed.
- Have private data members that hold the information associated with the printer, i.e. the printer's name/id, the current paper level, the current toner level and the number of documents printed.
- A single constructor that initializes the printer.
- A `toString()` method that returns a String representation of the current state of the printer. For example:

  [PrinterID: lp-CG.24, Paper Level: 35, Toner Level: 310, Documents Printed: 4]

LaserPrinter Behaviour

An instance of the `LaserPrinter` class, i.e. a printer, should:

- Allow students to print *"documents"* using the `printDocument( document)` method, provided it has sufficient quantities of paper and toner to be able to print the document.
  Assume that to print one page of a document you need 1 sheet of paper and 1 unit of toner.

  Example: the printer can print a 10 page document provided both the paper and toner are greater than 10; and that printing this document reduces each by 10.

  If either the paper or toner (or both) are less than 10 then the document cannot be printed and printing must wait until there is enough of both to print it.
- Allow the paper technician to refill the paper tray using the `refillPaper( )` method.
  Assume there are 50 sheets per pack of paper & the printer can hold up to 250 sheets of paper.

  Example: the printer has 150 sheets of paper, therefore, it can be refilled.
  But if it has 201 sheets of paper, then it cannot be refilled , as it would result in 251 sheets, and the technician should wait. But he or she is only prepared to wait for 5 seconds before checking if it can be refilled it again.
- Allow the toner technician to replace the toner cartridge using the `replaceTonerCartridge( )` method, provided it needs to be replaced, i.e. has a toner level of less than 10. Assume a toner cartridge can print 500 sheets of paper.

  Example: the printer has a toner level of 9, therefore, the toner cartridge can be replaced.
  But if it has a level of 10, then it cannot be replaced, as it would be a waste of toner, and the technician should wait. But he or she is only prepared to wait for 5 seconds before checking if it can be replaced it again.
- The printer status must be a correct record of its available resources and printing history & must not suffer from "data corruption & interference".
- Each user must have "mutual exclusive" access to the printer while he/she operates on it.
- Allow any user of the printer, including itself, to call the `toString( )` method, to get a String representation of its status.

2. Student class

Define a class to represent a student that can print several documents called `Student`.
Note that this class is a thread.

The `Student` class must:

- Use the `Document` class.
  Instances of this class are used to represent a *"document"* that students request the printer to be printed.
  To create five instances of this class to represent five *"documents"* that the student request the printer to print.

  For example, Joe Bloggs wants to print his 20 page CWK2 document then he would do the following:

```
Document CWK2 = new Document( "JoeBloggs", "cwk2", 20 );
printer.printDocument( CWK2 ) ;
```

- Have private data members that hold the information associated with him/her, i.e. the thread group he/she is in; his/her printer; his/her name.
- A single constructor that initializes his/her information, including placing the student in the "student" thread group.

Student's behaviour is to:

- Create and print five documents with different lengths and names.
- He/she should "sleep" for a random amount of time between each printing request.
- When he/she has finished printing, print out a message, e.g.

```
Joe Bloggs Finished Printing: 5 Documents, 95 pages
```

3. Paper Technician class

Define a class to represent a paper technician that can refill the printer's paper trays, called `PaperTechnician`. Note that this class is a thread.

The `PaperTechnician` class must:

- Have private data members that hold the information associated with him/her, i.e. the thread group he/she is in; his/her printer; his/her name.
- A single constructor that initializes his/her information, including placing the technician in the "technician" thread group.

Paper Technician's behaviour is to:

- Attempt to refill the printer's paper trays three times, using the printer's r`efillPaper()` method.
- He/she should "sleep" for a random amount of time between each attempt to refill the paper.
- When he/she has finished trying to refill the paper, print out a message, stating it has finished and how many packs of paer it refilled the printer with, e.g.

```
Paper Technician Finished, packs of paper used: 2
```

4. Toner Technician class

Define a class to represent a technician that replaces the printer's toner cartridge, called `TonerTechnician`.

This class is very similar to the paper technician class, except it attempts to replace the toner three times, using the printer's `replaceTonerCartridge`( ) method. When he/she has finished trying to replace the toner cartridge, print out a message, stating it has finished and how many toner cartridges it replaced, e.g.

```
Toner Technician Finished, cartridges replaced: 2
```

5. Printing System class

Define a class to represent the Printing System of all the clients of the printing system, called `PrintingSystem`. It is the "main" program class for the system & it combines all of the above objects & threads in parallel, see Figure 1. Note that this class is not a thread.

The `PrintingSystem` class creates and coordinates the following objects, threads and thread groups:

- One instance of the `LaserPrinter` class.
  This printer object is to be shared by the students & the two technicians.
- Creates 2 thread groups: one for the 4 students & one for the 2 technicians.
- Four instance of the `Student` class, i.e. four different students threads.
- An instance of the `PaperTechnician` class.
- An instance of the `TonerTechnician` class.
- Each of the above students and technicians is passed the appropriate information via its constructor and started, i.e 6 threads in all.
- The main program waits until all 6 threads have terminated. At which point it prints out the final status of the printer.
- In addition it must print out reports of what it is doing and when it has finished creating the threads and other objects, etc.

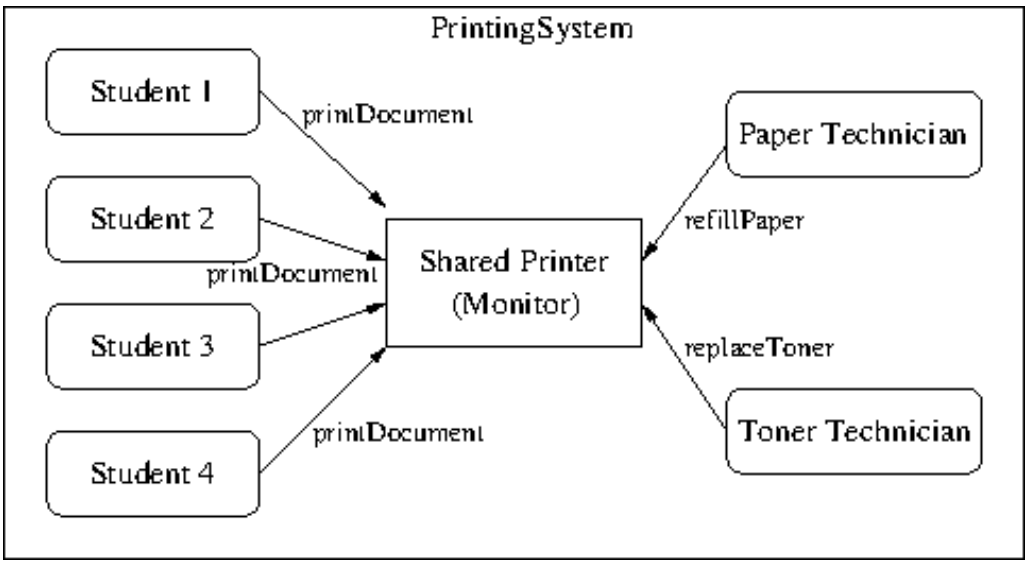

Figure 1. The shared printing system

3. Blackboard Submission

The following components are to be submitted via Blackboard. All files should be compressed **into** a single ZIP archive. The ZIP archive should be named using your surname & "CW", e.g. "Guhanathan_CW.zip".

The file formats used for the screen shot images must be either ".png", ".jpg" or ".jpeg". NOTE: If you are unable to produce screen shot image files in these format then email me with the formats you are able to produce to check that I can view them. If you submit a format that I cannot view you will get zero marks for this component of the coursework.

3.1 Components to Submit

(1) The FSP Process Composition Analysis & Design Form for the FSP process composition representing the complete Printing system. (PDF file format.)

[5 MARKS]

(2) The FSP program for the printing system, called "Printer.lts". (Plain ASCII text file.)

[25 MARKS]

(3) The Java source code for each individual Java class that implements the FSP model. Java code files (".java") should be plain ASCII text format. **Note** do not include: the given classes and interface or a full project structure from an IDE.

[50 MARKS]

(4) An example of the output produced by your Java program. (Plain Text file)

[5 MARKS]

(5) Screen shots images demonstrating the use of the FSP **LTSA tool** to animate and analyse the FSP model. (Assuming image files are in PNG format.)

- FSP program loaded into the "**Edit**" tab. (File: LTSA_Edit.png)
- "**Output**" tab showing the output after preforming:

  - the compilation and composition (File: LTSA_Output_CC.png)

  - a deadlock check (File: LTSA_Output_Deadlock.png)

- "**Draw**" tab showing the LTS for 1 student process (File: LTSA_Draw_Student.png)
- "**Animator**" pop-up window image(s) showing the states of the actions during 1 student - locking, using & unlocking the printer:

  - the printer's locking actions all available (ticked) (pre-locking) (File: Animator_before_Locking.png)

  - the printer's locking actions all unavailable (no ticks) after the student process has performed a locking action. (post-locking) (File: Animator_after_Locking.png)

  - the printer's locking actions all available (ticked) (post-unlocking) (File: Animator_After_UnLocking.png)

[15 MARKS]