Coursework Specification

CW1_Specification_CSI_5_ADP_22/23

Read this coursework specification carefully, it tells you how you are going to be assessed, how to submit your coursework on-time and how (and when) you'll receive your marks and feedback.

Module Code	CSI-5-ADP
Module Title	ADVANCED PROGRAMMING
Lecturer	MIKE CHILD
% of Module Mark	60%
Distributed	07 FEB 2023
Submission Method	Submit online via this Module's Moodle site
Submission Deadline	16:00 Friday 31 March 2023
Release of Feedback & Marks	Feedback and provisional marks will be available in the Gradebook on Moodle within 15 working days of submission.

Coursework Aim:

To demonstrate understanding of the module subject matter and the ability to apply it, including critical analysis and judgement. To achieve this, you are provided with an example application that must be analysed and assessed, and then enhanced to solve its existing weaknesses.

Coursework Details:

Type:	Report on development task
Word Count:	No formal word count is required but for general guidance no more than 2500 words would be expected. The word count will always be ambiguous because the requirement for embedded code will skew it. There are no specific penalties for exceeding or failing to reach the word count guidance.
Presentation:	▪ Any quoted material must be referenced, and a bibliography provided, however, formal referencing is not an assessed criteria – however the identification of technical resources consulted in order to solve specific described problems is an assessed criteria (see main task specification) ▪ Work must be submitted as a Word document (.doc/docx) or a PDF accompanied by separate ZIP archive files for your source code as specified below. ▪ Course work must be submitted using Arial font size 11 (or larger if you need to), with a minimum of 1.5 line spacing ▪ Your student number must appear at the front of the coursework. Your name must **not** be on your coursework.
Referencing:	Harvard Referencing should be used, see your Library Subject Guide for guides and tips on referencing.
Regulations:	Make sure you understand the University Regulations on expected academic practice and academic misconduct. Note in particular: ▪ Your work must be your own. Markers will be attentive to both the plausibility of the sources provided as well as the consistency and approach to writing of the work. Simply, if you do the research and reading, and then write it up on your own, giving the reference to sources, you will approach the work in the appropriate way and will cause not give markers reason to question the authenticity of the work. ▪ All quotations must be credited and properly referenced. Paraphrasing is still regarded as plagiarism if you fail to acknowledge the source

	for the ideas being expressed. **TURNITIN:** When you upload your work to the Moodle site it will be checked by anti-plagiarism software.

Learning Outcomes

This coursework will fully or partially assess the following learning outcomes for this module.

On completion of the module you will be able to:

- Demonstrate an understanding of software development using a variety of software engineering techniques.
- Critically evaluate the quality of a software artefact.
- Use sources of information to improve your knowledge and understanding.
- Develop multi-threaded applications.
- Effectively plan the development of a practical project from design to implementation.

Assessment Criteria and Weighting

LSBU marking criteria have been developed to help tutors give you clear and helpful feedback on your work. They will be applied to your work to help you understand what you have accomplished, how any mark given was arrived at, and how you can improve your work in future.

Full details of the criteria, interpretation, weightings and rubric to be applied are included in the main task specification below.

How to get help

We will discuss this Coursework Specification in class. However, if you have related questions, please contact Mike Child, childm@lsbu.ac.uk as soon as possible.

Resources

Course materials and Oracle Tutorials linked to from the exercises are expected to be the main resources required for this assignment.

Quality assurance of coursework specifications

Coursework specifications within CSI division go through internal (for new modules with 100% coursework also through external) moderation. This is to ensure high quality, consistency and appropriateness of the coursework as well as to share best practice within the CSI division.

Details of the moderators for this coursework specification are below:

Moderated (internal)	Aarbaz Alam, Philip Burrell
Moderated (CSI lead)	George Ubakanma
Signed off by (HoD/DHoD)	

----------For Internal use by CSI lead only----------

Changes required to CW?	Yes, No *
Examples of good practice	

*** if changes are required, moderator to complete the below:**

List of changes required	[These needs to be met before signoff can be achieved]
ML Response	[ML response, date]
Moderator Response	[ML response, date]

Advanced Programming Coursework Assignment

Introduction

You are provided with a partially working application that is intended to graphically demonstrate the solving of the Travelling Salesperson Problem using a simple brute-force algorithm. The Travelling Salesperson Problem is simply to work out the shortest route to follow to visit some number of cities and return to the starting city. While this is a famous and interesting problem in computer science that rapidly becomes very difficult as the number of cities increases (see https://en.wikipedia.org/wiki/Travelling_salesman_problem) this assignment is not really concerned with the problem itself, only the threading issues involved in displaying an animation of it.

The ***intended*** behavior of the program is as follows:

On launching, the program displays a GUI which allows the user to choose a number of cities and which has a "Go" button to start the demonstration. When "Go" is pressed 'cities' are randomly located in the square area of the display and the program is repeatedly draws routes between the cities and displays the corresponding length. It highlights the shortest route found so far in white and keeps trying different routes until every possible route has been measured. Finally the best route is shown in green together with its distance. After this process has completed, the "Replay longest to shortest" button can be used to rapidly display all the routes from the longest to the shortest. Alternatively, the "Go" button can be pressed again to repeat the demonstration with a new set of random cities. During either the finding of the shortest route or the replay of the routes, the "Cancel" button can be pressed to abort the process. If the window is closed, the application should terminate cleanly.

Unfortunately, the developer has not succeeded in achieving these goals.

Your tasks are as follows:

1. Run the application and observe its behaviour. Note that the application may take a significant length of time to complete (at least with 9 cities selected) and you must be patient to see what happens. Consider especially the behaviour of the various parts of the GUI (including the window closing/resizing behaviour) and the difference between the intended functionality and what the program actually does.

2. Analyse the application code and *explain* every aspect of the behaviour you have documented above. Try to be as detailed as possible in doing this and make sure you identify the significance of threads and the event dispatch architecture in this.

3. Modify the program so that it works correctly, in particular ensuring that:

    - The "searching" process displays all routes that are tested and not just the last few - note that the application prints out how many "Paint calls" were made (to actually paint a route on the screen) and how many "Update calls" were made (the number of routes tried in the search process) and these two numbers should be the same.
    - The "replay" process also displays all the routes properly
    - The cancel button works to cancel both the search and replay processes, and allows the user to continue with a new search after cancellation.
    - That all accesses to components belonging to the swing framework are made only on the Swing event dispatch thread (your program may work properly without doing this, but it is not correct without doing this).

- That all data passed between threads can be guaranteed free of memory-consistency problems (again your program may work properly without doing this, but it is not correct without doing this).
- That the main window close button cancels all the background threads before closing the window so that the program does not continue running after the window has been closed.

The whole process of modifying the program must be clearly documented, explaining your reasoning and the details of your code. You must include the relevant code in the body of your report while discussing it. You should describe the modifications you made in the order that you did them, and discussing any issues or problems you had as you went along. You should explicitly refer to lecture, lab and other technical materials that you used to find out how to solve specific problems as you go along. You should also discuss alternative approaches you considered and rejected and explain why.

Make sure you clearly explain all issues of threading, including the behaviour of the event dispatch architecture, ensuring events occur in the required sequence, and all possible memory-consistency problems.

Note: It is intended that you modify the existing code rather than extensively rewrite the program because (a) identifying the specific issues in the code as given is an important part of the assignment and (b) this involves less work. In particular, while there exist utility classes in Java that are explicitly intended to help write background threads for GUI applications, making use of them in this case is likely to involve a lot more complexity than a "simple fix" would. Should you choose to make significant changes to the design of the program, you should also explain what a simple direct fix would consist of and why you have chosen to do things differently.

Provided files:

- **TSPUi.java** - the UI application that provides the window and has a main method.
- **ImagePanel.java** - an extension of JPanel that displays an image and also tracks the number of times the image is painted for debugging purposes.
- **TSP.java** - a class that does the work of finding every possible route and which emits information about them to a TSPListener object. This class has a main method for demonstration purposes which just outputs to the command line.
- **TSPRoute.java** - a class that encapsulates a specific route around the cities and the total distance that route consists of. Instances are comparable (thus sortable) on the basis of this distance.
- **TSPListener.java** - an interface for objects that want to listen to the TSP class output. There are two methods, one called for each route tested, and one called at the end with the best route found.
- **SysOutListener.java** - an implementation of TSPListener that simply prints output on the command line. This is used by the TSP class' main method for demonstration purposes.
- **UIListener.java** - an implementation of TSPListener that is used by TSPUi to listen to an instance of TSP and forward updates to the TSPUi methods that display the routes. It also records the number of update calls that are made, to compare to the number of paint calls that ImagePanel reports (they should be the same).

IMPORTANT NOTE

This application needs to update Swing components from an independent worker thread in such a way that each update results in an actual paint operation. To get this right, and understand how it works, is very challenging and actually requires a significant understanding of how Swing handles repaint() requests. It also requires making the updating thread avoid saturating the swing event queue with update requests - essentially by waiting for them to be completed. (Note that saturating the event queue with update requests would make cancellation impossible!) A discussion of the issues involved is an opportunity to demonstrate exceptional understanding. If you are unable to achieve the appropriate results, a discussion of what you tried and why you thought it might help will also be of great value.

Report

Your report needs to address the tasks detailed above and should be separated into two clear sections:

Part 1: Description of initial behaviour or program and analysis of the underlying causes in the code.
Part 2: Narrative explaining the modifications made to the program.

Deliverables

1. Written report as specified above.
2. Zip file containing your modified version of the source code in an easily compilable form.

The written report must not be contained in the zip file.

Marking Criteria

This assignment will be marked using an adaptation of the University's standardised marking criteria. It is important that you pay attention to the criteria that will be applied and address them in the text of your report.

Please note that the primary criterion being assessed in this assignment is your understanding of the material and your ability to analyse and reason about it. Submitting a working program is not sufficient to demonstrate this and will not in itself obtain a high grade.

Note that marks are awarded for the following main criteria:

1. **Subject Knowledge (35%)**
   Understanding and application of subject knowledge. Contribution to subject debate.

   Mainly assessed by your written documentation of the behaviour of the existing application, the analysis of the code and your explanation of your own code.

2. **Critical Analysis (15%)**
   Assessed by the rationale you give for your design approaches and their contrast to alternative approaches and your ability to reason about threading issues.

3. **Testing and Problem-Solving Skills (30%)**
   Design, implementation, testing and analysis of product/process/system/idea/solution(s) to practical or theoretical questions or problems.

   Assessed on the basis of the software you develop *and* document - that is, your solutions to the problem at hand. It is important to note that very little credit will be given to any code you provide without explanation.

4. **Practical Competence (10%)**
   Skills to apply theory to practice or to test theory.

   Assessed on documented evidence of your use of technical documentation (for example Oracle's tutorials, Java documentation and course materials) in working out how to accomplish the assignment.

5. **Personal and Professional Development (10%)**
   Management of learning through self-direction, planning and reflection.

   Assessed on the basis of the quality of your submitted report, how well it addresses this assignment specification, clarity of writing and general presentation. Program code should be presented in a clearly legible fixed-width font. Screenshots should not be used to present code, only to illustrate the behaviour of the application if you feel it is necessary.

RUBRIC

Please note the criteria weightings and general interpretation shown in bold capitals under each criteria.

Criteria	Outstanding 100-80%	Excellent 79-70%	Very good 69-60%	Good 59-50%	Satisfactory 49-40%	Inadequate 39-30%	Very poor 29-0%
Subject Knowledge Understanding and application of subject knowledge. Contribution to subject debate. **CODE EXPLANATION 35%**	Shows sustained breadth, accuracy and detail in understanding key aspects of subject. Contributes to subject debate. Awareness of ambiguities and limitations of knowledge.	Shows breadth, accuracy and detail in understanding key aspects of subject. Contributes to subject debate. Some awareness of ambiguities and limitations of knowledge.	Accurate and extensive understanding of key aspects of subject. Evidence of coherent knowledge.	Accurate understanding of key aspects of subject. Evidence of coherent knowledge.	Understanding of **key aspects** of subject. Some evidence of coherent knowledge.	Some evidence of superficial understanding of subject. Inaccuracies.	Little or no evidence of understanding of subject. Inaccuracies.
Critical Analysis Analysis and interpretation of sources, literature and/or results. Structuring of issues/debates. **RATIONALE FOR DESIGN APPROACHES 15%**	Outstanding demonstration of critical analysis of the possible design strategies that could be used to meet the software requirements, and evaluation of the approaches chosen.	Excellent demonstration of critical analysis of the possible design strategies that could be used to meet the software requirements, and evaluation of the approaches chosen.	Very good demonstration of critical analysis of the possible design strategies that could be used to meet the software requirements, and evaluation of the approaches chosen.	Good demonstration of critical analysis of the possible design strategies that could be used to meet the software requirements, and evaluation of the approaches chosen.	Demonstration of critical analysis of the **key** possible design strategies that could be used to meet the software requirements, and evaluation of the approaches chosen.	Trivial demonstration of critical analysis of the possible design strategies that could be used to meet the software requirements, and evaluation of the approaches chosen.	Little or no critical analysis has been demonstrated.
Testing and Problem-Solving Skills Design, implementation, testing and analysis of **product/process/system/idea/solution(s)** to practical or theoretical questions or problems **IMPLEMENTATION 30%**	Outstanding implementation of all required software, with near perfectly organised, formatted and documented source code, and documented demonstration of runtime behaviour.	Excellent implementation of all required software, with well organised, formatted and documented source code provided	Competent implementation of all required software, with well organised, formatted and documented source code, and documented demonstration of runtime behaviour.	Implementation of all required software, with well organised, formatted and documented source code, and documented demonstration of runtime behaviour, with some missing/incorrect functionality or poor quality.	Implementation of most of the required software, with well organised, formatted and documented source code, and documented demonstration of runtime behaviour, with some missing/incorrect functionality or poor quality.	Implementation of only part of the required software, with well organised, formatted and documented source code, and documented demonstration of runtime behaviour, with some missing/incorrect functionality or poor quality.	Little or no functionality has been implemented.
Practical Competence Skills to apply theory to practice or to test theory **USE OF REFERENCE MATERIAL 10%**	Outstanding descriptions of factual information, programming techniques or theoretical explanations being found in technical or theoretical reference material.	Excellent explicit descriptions of all factual information, programming techniques or theoretical explanations that were found in technical or theoretical reference material.	Good explicit descriptions of all factual information, programming techniques or theoretical explanations that were found in technical or theoretical reference material.	Reasonable descriptions of most factual information, programming techniques or theoretical explanations that were found in technical or theoretical reference material.	Basic examples of the **main** factual information, programming techniques or theoretical explanations that were found in technical or theoretical reference material.	Some trivial examples of factual information, programming techniques or theoretical explanations being found in technical or theoretical reference material.	Little or no evidence of factual information, programming techniques or theoretical explanations being found in technical or theoretical reference material.
Personal and Professional Development Management of learning through self-direction, planning and reflection **REPORT QUALITY 10%**	Outstanding report organisation, structure, presentation, narrative voice and language.	Excellent report organisation, structure, presentation, narrative voice and language.	Very good report organisation, structure, presentation, narrative voice and language.	Good report organisation, structure, presentation, narrative voice and language.	Satisfactory report organisation, structure, presentation, narrative voice and language.	Poor report organisation, structure, presentation, narrative voice and language.	Report does not constitute a serious attempt at the assignment.