A picture containing shape

Description automatically generated

# Coursework 1 Specification

**CW1\_Specification\_CSI\_5\_OOP\_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-OOP |
| **Module Title** | Object Oriented Programming |
| **Lecturer** | Mike Child |
| **% of Module Mark** | 50% |
| **Distributed** | 25 October 2022 |
| **Submission Method** | Submit online via this Module’s Moodle site |
| **Submission Deadline** | 16:00, 25 November 2022 |
| **Release of Feedback & Marks** | Feedback and provisional marks will be available in the Gradebook on Moodle from 16 December 2022 |

## Coursework Aim:

## You are provided with a non-object-oriented program and are required to develop an object-oriented design for this program, with a clearly reasoned argument for the details of the design. You are then required to refactor the given code according to this design. The written report forms the main basis of assessment.

## Coursework Details:

|  |  |
| --- | --- |
| **Type:** | Written report |
| **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:** | * Work must be submitted as a Word document (.doc/docx) or a PDF * 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](https://libguides.lsbu.ac.uk/subjects/home) for guides and tips on referencing. |
| **Regulations:** | Make sure you understand the [University Regulations](http://www.lsbu.ac.uk/__data/assets/pdf_file/0008/84347/academic-regulations.pdf) 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:

* Select appropriate design notations, software development environments and programming languages.
* Read and understand object-oriented design documentation.
* Make effective use of technical documentation.
* Develop programs using the object-oriented approach.
* 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.

A description of the marking criteria and the detailed marking rubric that will be applied is included at the end of the assignment 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, the Oracle Java Tutorials and the Java API documentation are expected to be the main resources required for this assignment. You can find links to the Oracle Java documentation in the exercise documents.

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)** | Iannis Iatropolis / Kamal Thapa |
| **Moderated (CSI lead)** | George Ubakanma |
| **Signed off by (HoD/DHoD)** | [Name, date] |

----------------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] |

Assignment Specification

**Aim**

You are provided with a non-object-oriented program and are required to develop an object-oriented design for this program, with a clearly reasoned argument for the details of the design. You are then required to refactor the given code according to this design. The written report forms the main basis of assessment.

**Details**

This assignment concerns applying the basic principles of object-oriented programming to rewrite an existing *non*-object-oriented program.

You are provided with a Java program implemented in a single file (*Application1.java*). There is also another file called *ExifUtils.java* that you do not need to examine or concern yourself with for the assignment.

The program is intended to be used to organise collections of photographs and/or videos according to their dates. In order to determine the date for a given file, three techniques are supported:

1. Reading the embedded EXIF metadata (if there is any) to determine the original date of the photograph.
2. Examining the filename to see if any unambiguously recognisable date string can be found (for example *rosegarden-21.Jan.2020.jpg*).
3. Using the timestamp of the file.

The program allows you to use any or all of these techniques on a folder-by-folder basis, but if more than one technique is used they will always be tried in the order given above.

The program's inputs are as follows:

* a list of *target folders* containing photographs and possibly sub-folders containing photographs
* for each target folder, parameters that control whether the folder should be scanned recursively (entering into the sub-folders) or not, and which techniques should be used to determine the dates of the files (read the embedded EXIF metadata / parse the filename for a date string / use the file timestamp)
* an *output folder* to move the photographs to
* parameters to control the tree of folders to be created, that can consist of any combination of year folders, year-month folders, and year-month-day folders
* a *simulation* flag that determines whether the program will actually move the files (if simulation = false) or just print out the moves it would have made (if simulation = true).

You are provided with a ZIP archive containing a couple of folders of images to run the program on. When running the program you should do the following:

1. Extract the images from the ZIP file to a location of your choice (say C:\Users\yourname\oop).
2. Edit the main method of the program so that the commonParent variable is set to point to the place you put the image folders (for example C:\Users\yourname\oop).
3. Run the program in simulation mode first, to see what it is going to do (when testing you should probably stay in simulation mode most of the time).
4. Run the program for real and see if the PhotoOrganiserOutput folder is correctly created with a date based tree of folders inside it containing the organised pictures.

**Note that after running the program for real, you no longer have your input folders with images in to use again. You will need to delete the PhotoOrganiserOutput folder and re-extract the images from the ZIP file to run the program again.**

In its provided state, the main method has recursion and use file timestamps turned off for both target folders. You should experiment with adjusting these settings to see how the program works (use simulation mode for this).

**IMPORTANT**

The program requires the EXIF metadata library that is provided in the ***exifutil.jar*** java archive file. You will need to add this jar file to your IDE's classpath to compile and run the program successfully.

It is assumed that you watch the coursework demonstration video in order to find out how to do this and other details of the program's behaviour. This demonstration video is not an optional extra - it is part of the assignment specification.

**Things regarding object-oriented design to bear in mind (Advice!)**

* Classes are intended to group *related* variables together so that they can be treated as single (complex) value.
* Classes also allow functionality that *uses the variables the class contains* to be implemented as instance methods.
* Classes can use other classes as the type of the fields they declare.
* **Static** utility methods such as ***trim*** and ***splitByWordsAndNumbers*** in the given application do not necessarily benefit from object-oriented design - not everything needs to be changed.
* **Static** class-level variables such as those used in the given application are not desirable in an object-oriented design and you should try to get rid of them.
* (Advanced topic) An abstract parent class or interface can define a method for which multiple subclasses can provide differing implementations - different ways of doing the same thing.

**Your Task**

You are required to analyse the existing application for the purpose of refactoring it into an object-oriented design. In order to understand the program you may need to refer to technical documentation such as the Java language APIs or online tutorials.

You must identify potential classes that could be used to improve the code and make it object-oriented. You should discuss your thinking and give reasons for why you think each class would be beneficial. How many classes are required or useful is up to you, and will be affected by your design decisions, but it is suggested that four or five might be typical, although it might be more if polymorphism is employed as part of your design.

You are encouraged to describe multiple different approaches to the organisation of classes in your design discussion but you should settle on a single set of classes as your final proposal.

You should depict your design proposal using a UML class structure diagram, but this diagram should only be used to illustrate your **written description** of the design. The diagram by itself is not enough and will gain little credit.

You should then refactor the provided code into the classes you have described. You should then describe this code as part of your report, including the relevant code in the body of your report as you describe it. The code should be presented clearly legibly using a fixed-width font, proper indentation and single spaced lines.

The final version of the program should also be uploaded as the source files in their appropriate package folders packed into a zip file. To do this, you should typically pack the main *src* folder of your IDE's project into a zip file.

You must upload **exactly two files**: a word document containing your report and a *separate* zip file containing only the source code.

**Resources**

Java 8 API: <https://docs.oracle.com/javase/8/docs/api/>

Java 11 API: <https://docs.oracle.com/en/java/javase/11/docs/api>

Oracle tutorials: <https://docs.oracle.com/javase/tutorial>

All work will be submitted to the TurnItIn plagiarism detection service. Your work should be entirely your own. Collaboration between students is not permitted. You are reminded that if another student submits some or all of your work without your permission you will be held responsible for this, as the University cannot tell who actually did the work; this is treated as collaboration and each student that is involved will be held accountable. For this reason **you are strongly advised *not* to share your work with other students**, and ***not* to publish parts of your work in progress on the VLE forum when seeking advice from tutors**.

**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. A detailed rubric is shown on the next page, but the main criteria are as follows:

1. **Subject Knowledge (35%)**

*Understanding and application of subject knowledge. Contribution to subject debate.*

Mainly assessed by your written explanation of your code and correct use of diagrams and program syntax.

1. **Critical Analysis (15%)**

Assessed by the rationale you give for your design approaches and their contrast to alternative approaches, and your evaluation of the finished program.

1. **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 together with its documentation, bearing in mind that code that you do not discuss in your narrative will be given very little credit (as the implication is that you do not understand what it is and what it does if you did not discuss it). Also the extent to which it fulfils the application requirements and any documented ways in which it enhances or extends upon it.

1. **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.

1. **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, including clarity of writing, presentation, and properly addressing the assignment specification.

**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. |