A picture containing shape

Description automatically generated

# Coursework 2 Specification

**CW2\_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** | 29 November 2022 |
| **Submission Method** | Submit online via this Module’s Moodle site |
| **Submission Deadline** | 16:00, 6 January 2023 |
| **Release of Feedback & Marks** | Feedback and provisional marks will be available in the Gradebook on Moodle from 27 January 2023 |

## Coursework Aim:

## You are provided with a working object-oriented program. You are required to analyse the design of this program and identify and explain the object-oriented techniques used in it. You are then required to add some additional functionality to the program, and then refactor some aspects of it to improve its 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 working object-oriented program. You are required to analyse the design of this program and identify and explain the object-oriented techniques used in it. You are then required to add some additional functionality to the program, and then refactor some aspects of it to improve its design. The written report forms the main basis of assessment.

**Details**

You are provided with a Java program that allows the user to load an image file and add text and shapes to the image. The various shapes can be adjusted in terms of size, position, colour and transparency and once the user is satisfied the image can be saved with additions in place.

**Using the program**

The following explains how to use the program:

1. Use the *open* item on the *file* menu to choose and load an image file.
2. Choose a shape from the *shape* menu.
3. Click on the image with the left mouse button and drag out a rectangle while holding the button down.
4. The shape will be drawn in its default state in the rectangle you have defined and this is now the selected shape.
5. Choose *configure* from the *selected shape* menu and adjust the colour, transparency and other properties of the shape until you are happy with it.
6. While the shape is selected, you can move the mouse with the left button down to move it, and move the mouse with the right button down to resize it.
7. To deselect the shape, click in an empty space on the picture.
8. To select a shape, click inside it.
9. You can add as many shapes as you want.
10. To delete a shape, select it and use *delete* from the *selected shape* menu.
11. You can save the image with the shapes added by using the *save* item on the *file* menu.

Familiarise yourself with how the program works and examine the source code of the application, and then proceed to your tasks below.

**Your Tasks**

**Task 1 - Write analysis of classes in the *shapes* package**

Examine the classes that are inside the ***shapes*** package and the packages inside it. In particular the classes that are immediately in ***shapes***, those in ***shapes.simple*** and those in ***shapes.compound***. Your task is to write a detailed analysis of these classes explaining how they are related, how they make use of each other and, to the best of your ability, the reason each of the classes exists - that is, what each class contributes to the application. You should include the interface **PaintableShape** in this analysis. Note that a UML class structure of these classes is shown below to help you with this. You can include this diagram or similar ones of your own creation if this helps to support the text of your analysis.

**Task 2 - Describe execution sequence of *addShape* method**

The *addShape* method in *ImagePane* is invoked when the mouse button is released after dragging out. You must describe the sequence of method calls, and the objects involved in them, from the beginning of this method to the end. You should use a sequence diagram to illustrate this, but it is your written description of the process that will be used to award credit. The diagram can be drawn by hand or generated by a tool, but in either case will contribute only to the report standards criteria.

**Task 3 - Create a factory class and refactor the application to use it**

You are required to create a *factory* class that is capable of instantiating and providing PaintableShape objects on demand. The factory should provide a method that gives a list of the names of all the available PaintableShape objects, and another method that provides a suitable PaintableShape object for each name. The PaintOn and ImagePane classes should then be modified so that they do not have any references to any shape type other than the interface PaintableShape and configure themselves entirely on the basis of what the factory class offers. This means that the range of shapes the application offers can be changed just by modifying the factory class and what it offers. Again you must document and explain the factory code and the changes to the existing classes you make in your report.

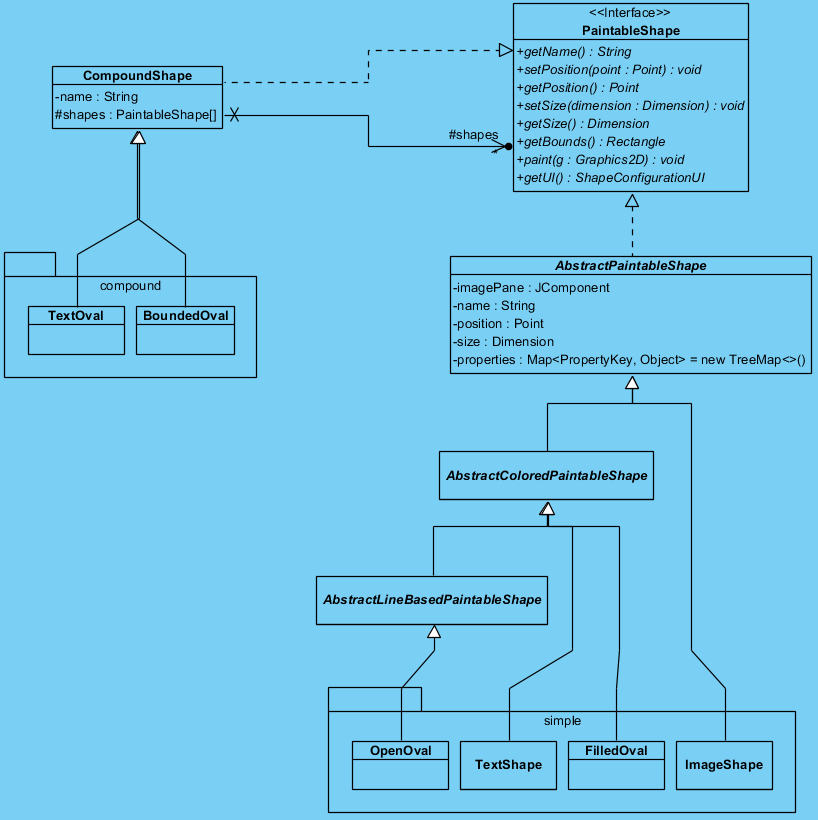
**Task 4 - Implement additional classes for the factory to create**

You are now required to implement some additional shapes for the program; rectangular equivalents for the existing oval classes, that is: FilledRectangle, OpenRectangle, BoundedRectangle and TextRectangle. These classes can be implemented largely by replicating the existing code with minor adjustments. They should then be integrated into the application solely by modifying your factory class. This task is to demonstrate why the factory class is a good thing to have! You must document the changes you have made and the classes you have defined with explanatory text in your report. See the notes below for some guidance regarding this.

**Advice and technical notes**

* The program in this assignment and elements of its design are similar to some of the code used in the tutorial exercises - this should help you to understand it, but be aware that they are not the same.
* The OpenOval class uses a more sophisticated drawing mechanism than the FilledOval class. The Graphics2D class has a method to draw arbitrary **Shape** objects using an abitrary **Stroke** object, where the **Stroke** object defines the thickness of the line used (amongst other things). This **Shape** class is part of the java graphics libraries and should not be confused with the PaintableShape type used in this program (or the shape classes used in the tutorials). OpenOval uses an instance of Ellipse2D as the shape to draw (instead of using the more basic drawOval method) - you should be able to find the line of code that reads:  
  **new** Ellipse2D.Double(0, 0, size.width, size.height)  
  to draw a rectangle using the same approach you can simply use:  
  **new** Rectangle2D.Double(0, 0, size.width, size.height)  
  instead. See <https://docs.oracle.com/javase/tutorial/2d/index.html> for more details about this.

**UML Class structure diagram for the classes to be analysed for Task 1:**



**Report structure**

Your report should have headings for each of the tasks (that is, Task 1, Task 2 etc.)

**NO OTHER HEADINGS ARE REQUIRED.**

You do **not** need to write an introduction, background information or any conclusion, and doing so will impact your marks negatively.

You *should* include personal reflections and your thought processes within the written sections of the report.

The final version of the program should 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.

AS USUAL, YOU SHOULD EXPECT THE VLE TO ISSUE A CONFUSING ERROR MESSAGE COMPLAINING ABOUT THE zip FILE, BUT THERE IS NO NEED TO WORRY ABOUT THIS AS LONG AS THE FILE HAS BEEN SAVED ON THE SERVER.

**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 both the given code and your own code and correct use of diagrams and program syntax.

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

Assessed on the basis of your analysis of the given source code and 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. |