Portfolio Project I

This portfolio is about object-oriented programming and software engineering with focus on how to implement classical OOP models and understand and apply software requirements. You can work on the portfolio in groups but be aware that the exam is individual, and you must be able to answer questions related to all parts of your portfolio. If you work in groups the members of the group must be named (with name and study number) on the first page, and only one of the group members need to submit on Moodle[1].

Portfolio 1 is in two separate parts.

In the first part of this portfolio, you should work on a case (real or imaginary). The main tasks in this part are to:

- establish a development environment,
- designing your application and
- plan the development of the application.

The development environment should as key components use the Kanban to manage the process and Git as version control. Clarify which other tools you expect to use in the development process.

The design should be done using UML, from use case diagrams to static and dynamic models describing the domain, your classes and the behavior of your application. This is clearly an iterative problem and should be solved by focusing first on the parts of the application that have the highest value for the customer (the product owner) and then iteratively adding more and more features. You need to consider the architecture, design patterns and design principles, and clearly describe what, why and where this have helped your design/process.

Your task is to start using Kanban as soon as possible, following the flow of adding tasks to the backlog, prioritize the tasks, do the task breakdown, solve tasks and validating the solutions. Obviously, in this part we should not do the implementation of the application, but the design process also need management. Your Kanban process can be done on paper, whiteboards, or whatever you like, but you need to use Github projects to report your progress, i.e. have timely updates of the progress to Github. Github should also be used to manage the version control repository, and you must setup an appropriate branching structure for the development/maintenance of your application. Describe how you can use Junit to validate some early components you plan to construct as part of the application.

To summarize the first part, the solution must include:

- a short introduction describing the application you want to develop
- documentation of your environment setup and your design
- description and decision of your choices (for the environment and your design)
- a short conclusion/summery

[1] The final submission to examen.ruc.dk is individual and should include the three portfolios and the final individual comments/reflections.

In the second part of this portfolio, you need to implement a small shapes model. Obviously, you could use the process management and develop environment from the first part for this part, too. It is a inheritance assignment and thus no visualization is needed.

The basic shapes that must be part of the shapes model is rectangles, triangles, and circles, but the model should be designed with extensibility in mind so other shapes can be added later without changing the basic model, i.e. lines, polylines, pentagons, etc. This should be done by use of inheritance and polymorphism, i.e. introduce an abstract superclass Shape, which define the common behavior. The shapes in the model may have different internal representations, e.g. circles could be represented by a center point and a radius, and triangles as three points, etc., but should have shared behavior. The basic common behavior that must be part of the model are the following:

- return the center of the shape
- compute the area of the shape
- compute the circumference of the shape
- indicate whether a point is inside the shape or not (e.g. the point could represent a mouse click)
- compute the Euclidean distance to another shape (center to center)

Future optional behavior is:

- tell if one shape is inside another shape
- compute overlaps between shapes
- move shapes within a two-dimensional space
- layout shapes without any overlaps

The task in this portfolio is to design the shape model with basic behavior, document the model in UML, and implement the model in Java. Feel free to extend with one or more of the future behaviors if you like or suggest other possible useful behavior, but these must be documented and implemented as well.

To summarize the second part, the solution must include

- An overall description of the shape model
- UML diagrams documenting representation (attributes) and behavior (methods) of the shape model
- An implementation of the shape system
- Unity tests verifying the behavior for all shapes.

The overall submission is limited to 5 pages[2].

[2] A page is defined as 2400 characters, including spaces. Use the word count in Microsoft Word to verify the size of your document.