1.

The roles of requirements in architectural design:

It is a transformational step where the requirements are the input and become the reasons for how the software will be structured. These requirements affect the architecture choice for the system and / or. The high-level requirements (software elements) require interaction, which is defined in the architectural design of the system.

The role of requirements in detail design:

They are more direct than they in architectural design. The ratio is 1:1 between functional requirements and modules in detail design.

2.

Aggregation is a special kind of association. One of those kinds corresponds with the “part-of” association. For instance, an engine is part of a vehicle, or a name is part of a customer. Another version of aggregation is composition. This version corresponds with the “made-of” association. In such cases, the subordinate subject can’t participate in any other association and is the responsibility of the containing class. An example an organ system which is made of organs.

3.

When we employ the technique of generalization, we are simplifying the design by keeping only the essentials and delaying the considerations of detail until a later time. For instance, we should only create subclasses when there is a need to incorporate additional behavior or attributes. In OO, moving from specific objects to a general class and focusing on it.

4.

State transition diagrams are diagrams representing information that concern the states of an object and the allowed state transitions.

UML sequence diagrams are diagrams that illustrate the flow of messages from one object to another and the sequence in which those messages are processed.

5.

Process view: represents the run-time processes and how they communicate with each other

Subsystem decomposition view: represents the modules and subsystems, joined with export and import relationships

Physical architecture view: represents the mapping of the software to hardware. This assumes a system that runs on a network of computers and depicts which processes, tasks, and objects are mapped to which nodes

6. Describe the difference between low-fidelity and high-fidelity prototyping in the design of the interface. Choose one and give the reasons why you would show this client this prototype.

A low-fidelity prototype is a simple mock-up sketch of the target product (handwritten).

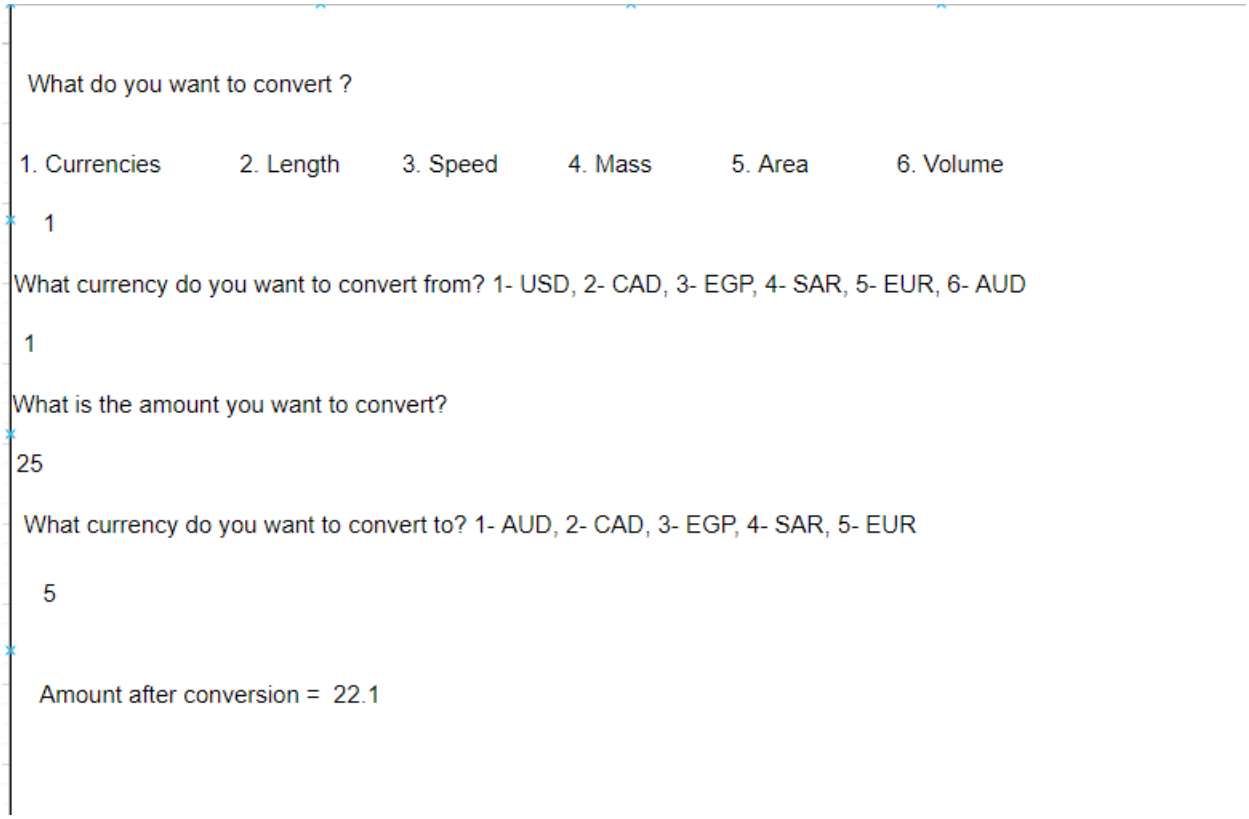
A high-fidelity prototype is a detailed mock-up resembling and behaving close to the final product (done with one or more screen design software).

I would choose to give a client a prototype based on who he or she is, but I will have both ready. I would like to give a high-fidelity prototype because it shows a lot of details on what will happen, and it will look more professional specially if they are the type to care a lot about the details. If they want to have a simple idea about what is going on, then I will mix both by using screen design software to create a simple sketch of the target product.

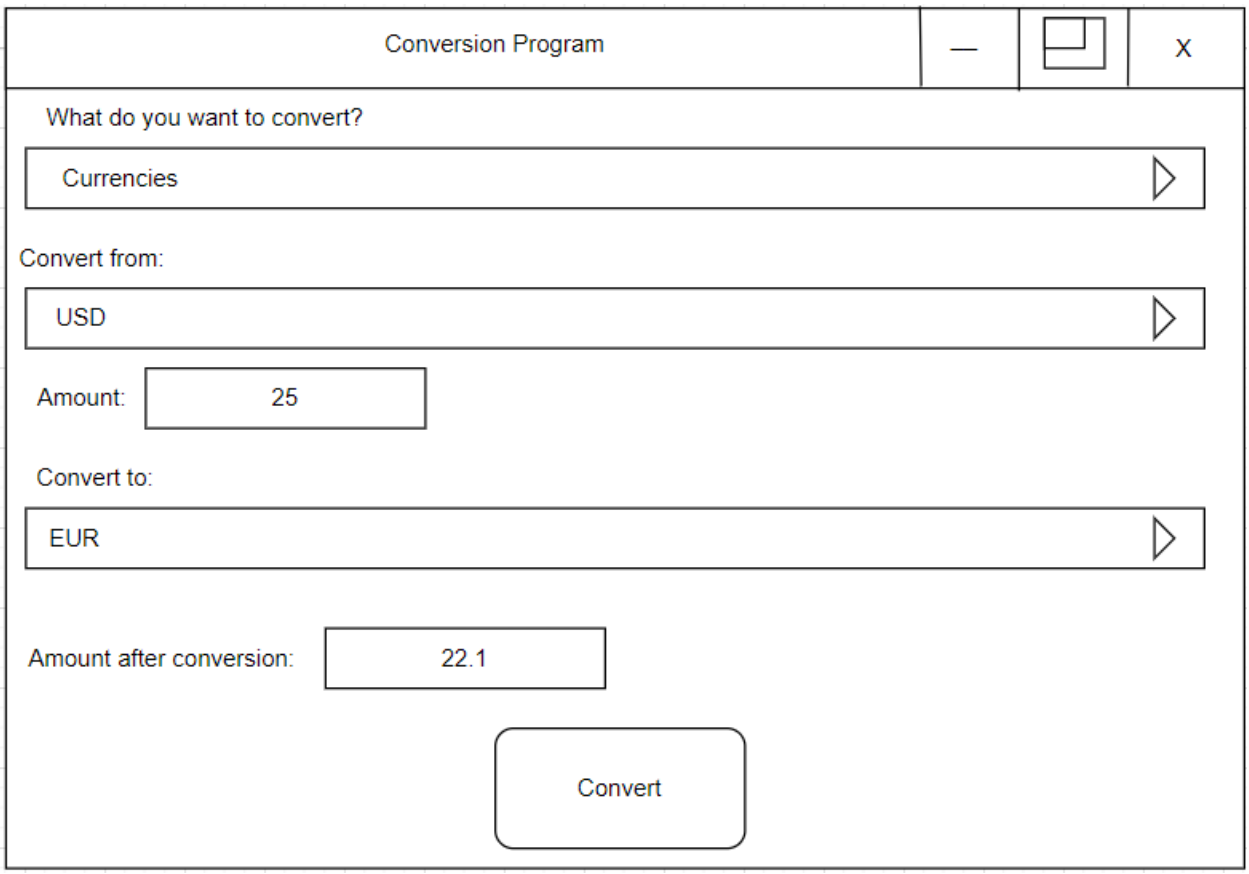
7. In the MVC style, what does model really model?

The model is responsible for sorting the data and for notifying the views whenever the data change. So, in short, it models data.

8. Your programming team has been given the project to write a command-line program for converting different units of measurement. Create (draw) the initial screens for the project.



9. Your programming team has been given the project to write a GUI program for converting different units of measurement. Create (draw) the initial screens for the project.



10. For each of the architectural styles mentioned in Chapter 7 of the textbook, find an example software system (which is not mentioned in the textbook) that uses the style. Briefly describe how the style is used in the system and the source of your information.

·Pipes and filters

Pipes are the connectors. They are used to transfer data from one filter to another.

Filters are what is used to filter the data received by pipes. An example is UNIX. In UNIX any link can be connected to any. So, the output of the one program will the input of another program in UNIX. This process is done by using filters and pipes.

·Event-driven

Even driven is a style in which system components react to externally generated events and communicate with other components through events. An example is a mouse click.

·Client-server

a style showing a clear demarcation between clients and servers, which reside on different nodes in a network. An example is Facebook or WhatsApp.

·Model-view-controller (MVC)

A popular way of organizing GUI programs that need to display several different views of data. The main idea is to separate the data from the display. In the original version, a controller took care of translating user input into appropriate messages for the view. Modern variations of this pattern use just the model and view classes. The model is responsible for storing the data and for notifying the views whenever the data change. The views register with the model, can modify the model, and respond to changes in the model by redrawing themselves. An example is the user interface of any program (the group project).

·Layered

A style in which components are grouped into layers, and the components communicate only with other components in the layers immediately above and below their own layer. A network layer is one of the things that first come to mind as an example.

·Database-centric

A style in which a central database and separate programs access the database. The programs communicate only through the database, not directly among themselves. A big advantage of this style is that it introduces a layer of abstraction for the database, which is usually called a database management system. An example is any system in which programs can only communicate with the database in order to communicate with other programs.

·Three-tier

A variation on the database-centric and client-server approaches that adds a middle tier between the clients and servers, implementing much of the business logic. The clients cannot access the database directly and must go through the middle tier. This way, the business logic gets implemented in one place, simplifying the system.

11. User interface design involves two aspects: actor-system data exchange/interaction and look-and-feel. Explain these two aspects and their differences. How separation-of-concerns software engineering principle applies to user interface design?

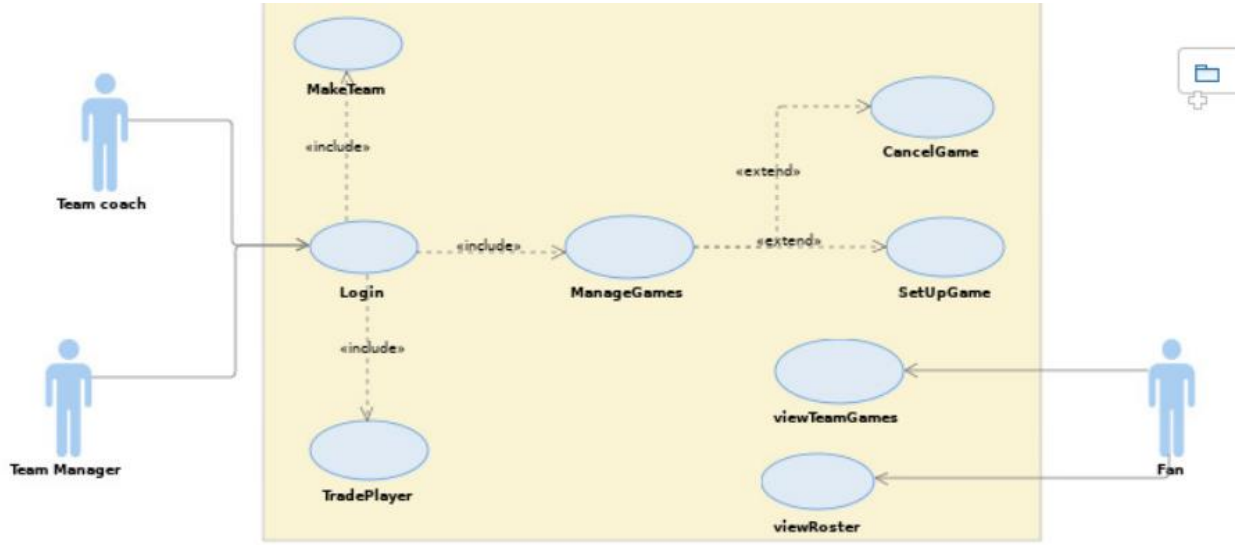
Actor-system interaction: the actor interaction is done by interfaces and the services or actions defined on a system or an actor port is a means to interact with actor data. Focuses on the interaction between actors.

Look-and-feel: is one of the user interface designs. The understandability of the interface leads to better look and feel. This is all about the interface of the design.

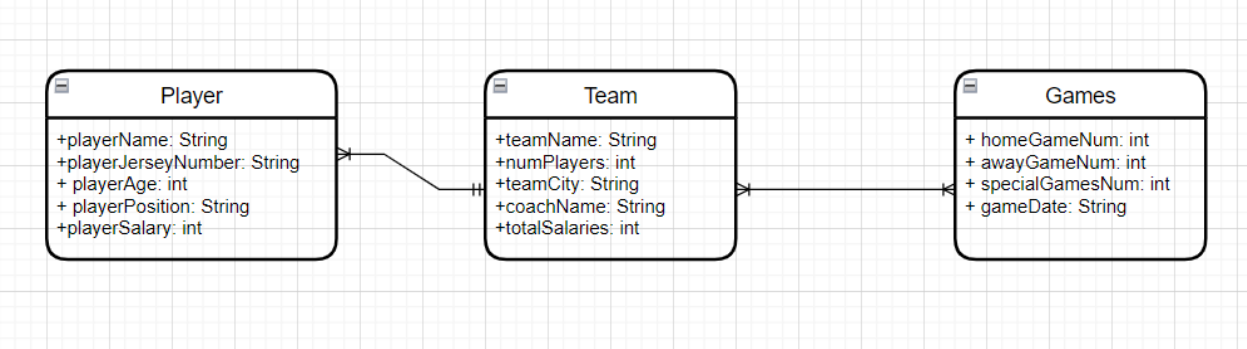
The software engineering principle is separation of concerns, it focuses on reducing the dependency among modules through the definition of clear interfaces.

12. Consider the case of a software system designed to keep track of team rosters and scheduled games for a sport league.

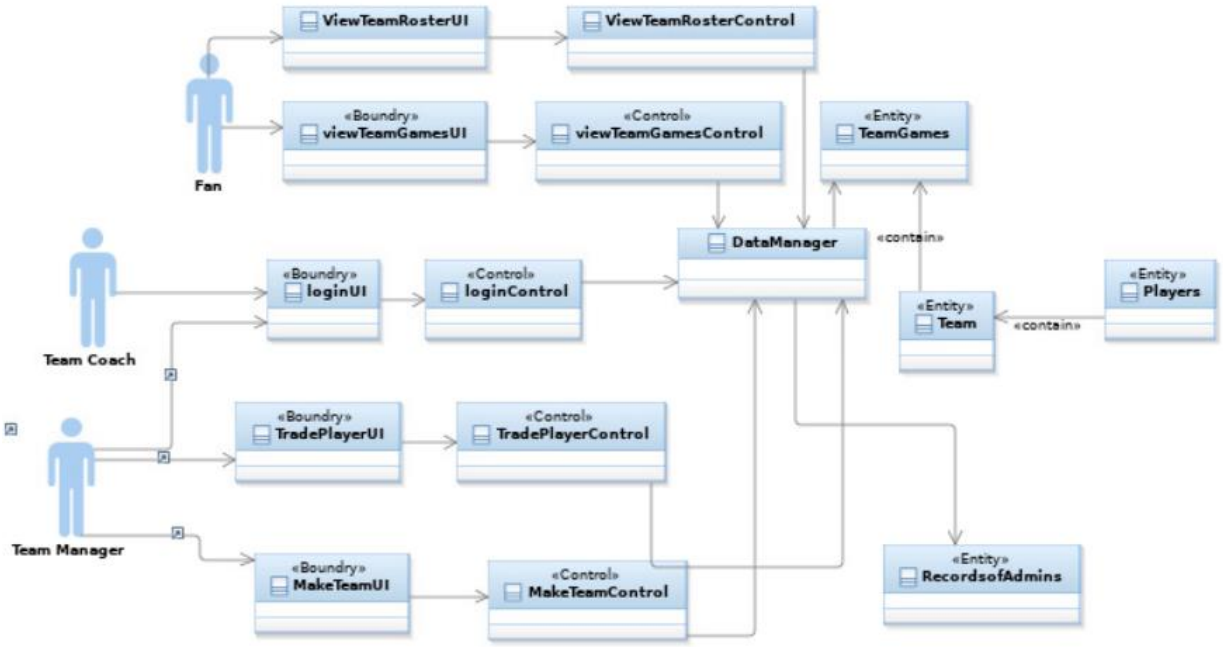
· Draw a UML use case diagram for the system



· Draw a UML class diagram to model the domain entities involved in the system.



· Draw a UML class diagram to show the analysis model or conceptual design of the system.



Note that you can draw the above diagrams on paper, scan them, and paste them, or using RSAD and copy/paste them, into your answer.