



IQRA NATIONAL UNIVERSITY

Summer 2020 Final-Term Examination

Course Name	Max. Marks	Max. Time	Date	Instructor
OOSAD/OOAD	50	9am to 1pm	23 th September, 2020	M. Fahimullah

Note: Submit paper along with your ID in pdf format

Name : shayan khan ID : 6833

Subject : OOSAD Submitted To : Sir Fahimullah

Class : BS(SE)

Q1. Describe different elements of a use case diagram. (5)

Answer: Use case diagram is a subset of various behaviour diagrams. Use case diagrams are used to provide concrete examples of the elements which are supposed to implement. It is used to analyze objects.

The following are the elements of the use case diagrams:

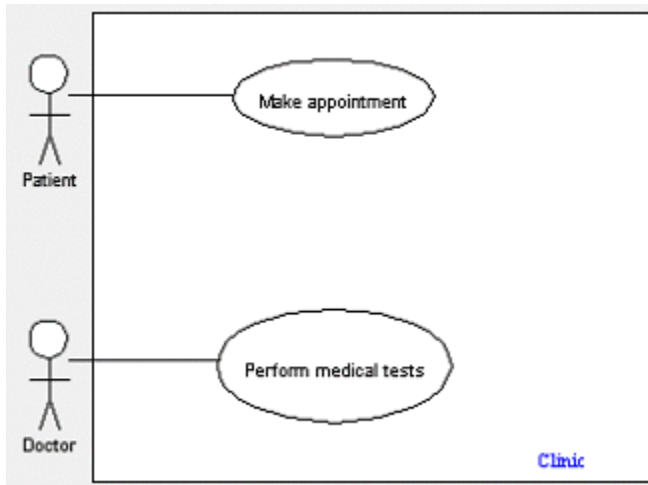
Actors: An actor is one of the entities who perform certain actions. These roles are the actual business roles of the users in given system. An actor interacts with a use case of the system. For example, for a banking system, a customer is one of the actors.



Use Case: A use case is a use case diagram of UML represents a business functionality that is distinct. The use case should list the discrete business functionality that is specified in the problem statement. Every business functionality is a potential use case.



System boundary: A system boundary defines the scope of the system. The systems that use cases also need to be defined in the limits of the system. The system boundary is shown as a rectangle that spans all use cases of the system



Q2. What is the purpose of a fork node? Provide example. (5)

Answer: For automation interface purposes, a Fork Node is a Control Node that has its Node Type set to Fork Node. A Fork Node splits a flow into multiple concurrent flows. A Fork Node has one incoming flow and multiple outgoing flows.

In addition to the standard properties a Fork Node has these properties:

Visibility

Owned by

Activity


Structured Activity Node— Applies only when the Fork Node is scoped directly to a Structured Activity Node.

Owns

Comment

Constraint

Control Flow: The Control Flow is owned jointly by Fork Node and the associated item. The access permissions you have to a Control Flow are determined by the access permissions you have to its source item.


 Dependency — The Dependency is owned jointly by the Fork Node and the other associated item. The access permissions you have to a Dependency are determined by the access permissions of the dependent item.

IO Flow — The IO Flow is owned jointly by the Fork Node, the IO Flow's other linked item and the IO Flow's IO Item.

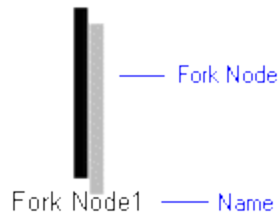
Object Flow— The Object Flow is owned jointly by Fork Node and the associated item. The access permissions you have to an Object Flow are determined by the access permissions you have to its source item.

For Example:

Create a Fork Node through a Modeler explorer pane or an Activity Diagram:

- In a Modeler pane, right-click an Activity or Structured Activity Node, point to New, point to Control Node, and then click Fork Node.
- On an Activity Diagram, click the  Fork Node toolbar button, and then click in free space or inside an Activity Partition, Interruptible Activity Region or Structured Activity Node.

When used on an Activity Diagram, a Fork Node's notation is as follows.



The View Options on an Activity Diagram allow you to show or hide the Name, and orientate the Fork Node vertically or horizontally. In addition, the view options allow you to dock the Fork Node onto the boundary of an Activity Partition. The view options are set through the Control Node entry. See [Control node view options - activity diagram](#).

On an Activity Diagram, you can populate a Fork Node's missing Activity Flows, Comments and Constraints: right-click the Fork Node, point to Populate, and then click the appropriate command.

In the Dictionary pane, Fork Nodes are listed in the UML\Control Nodes folder.

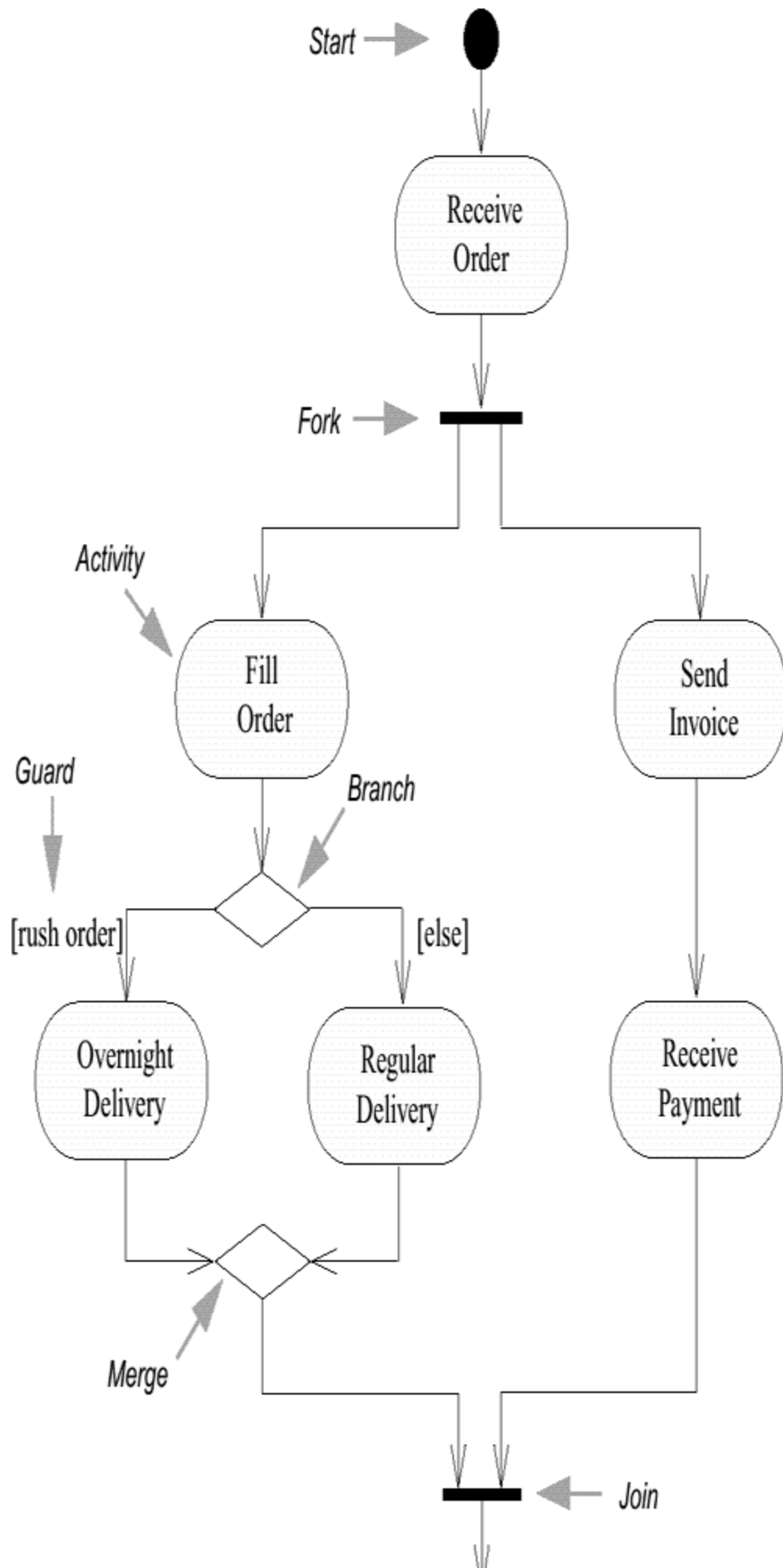
In the Modeler panes, a short-cut symbol on the Fork Node's icon indicates that the item is a stub.

Q3. What is the difference between conditional branch and fork in an activity diagram? (5)

Answer: Conditional behavior is delineated by branches

A **branch** has a single incoming transition and several guarded outgoing transitions. Only one of the outgoing transitions can be taken, so the guards should be mutually exclusive. Using [else] as a guard indicates that the "else" transition should be used if all the other guards on the branch are false.

In Figure 1, after an order is filled, there is a branch. If you have a rush order, you do an overnight delivery; otherwise, you do a regular delivery.

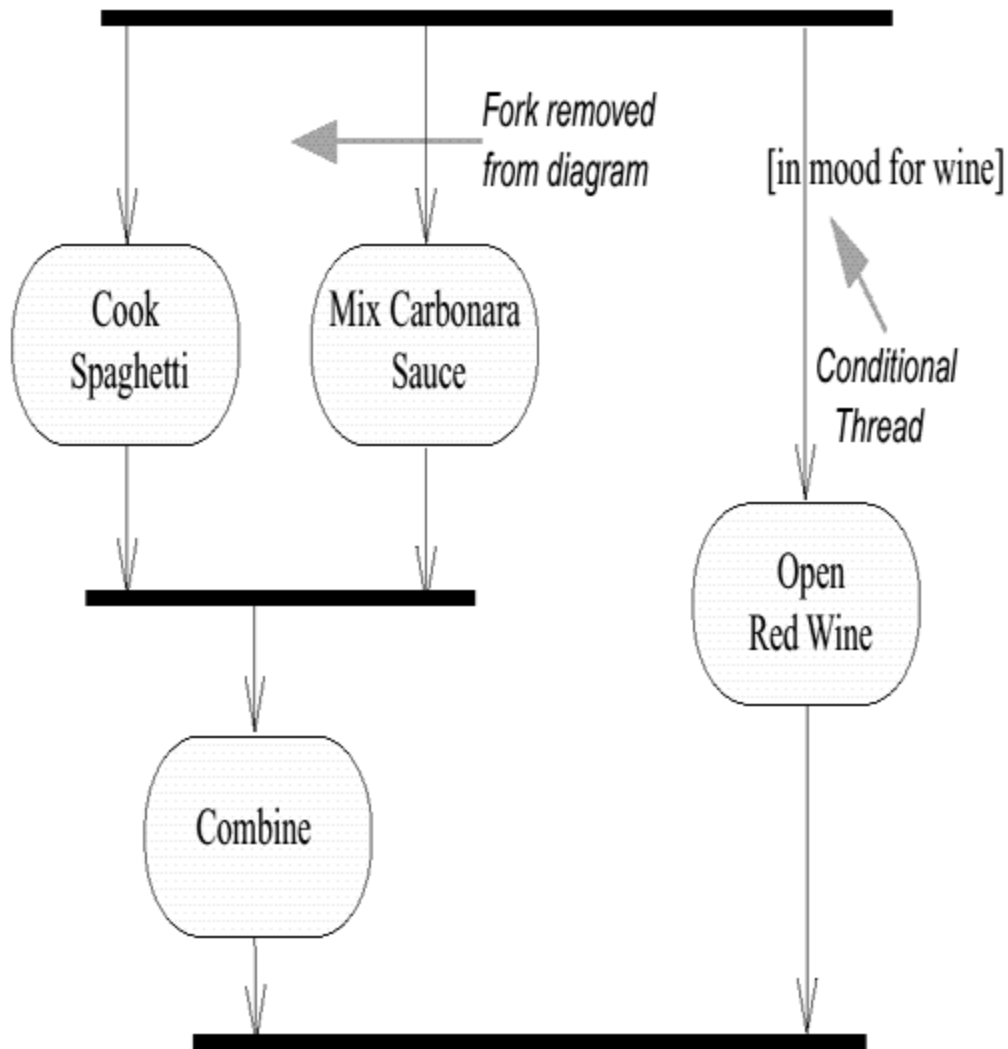


A **fork** has one incoming transition and several outgoing transitions. When the incoming transition is triggered, all of the outgoing transitions are taken in parallel. Thus, in Figure 1, after you receive an order, you fill the order and send the invoice in parallel.

The diagram says that these activities can occur in parallel. Essentially, this means that the sequence between them is irrelevant. I could fill the order, send the invoice, deliver, and then receive payment; or, I could send the invoice, receive the payment, then fill the order and deliver you get the picture.

There is an exception to the rule that all incoming states on a join must have finished before the join can be taken. You can add a condition to a thread coming out of a fork. The result is a **conditional thread**. During execution, if the condition on a conditional thread is false, that thread is considered to be complete as far as the join is concerned. So in Figure 2, even if I don't feel like wine, I would still be able to eat my Spaghetti Carbonara. (I must confess, though, that I've never had to test this rule when executing this diagram!)

Figure 2



Q4. What is modeling? (5)

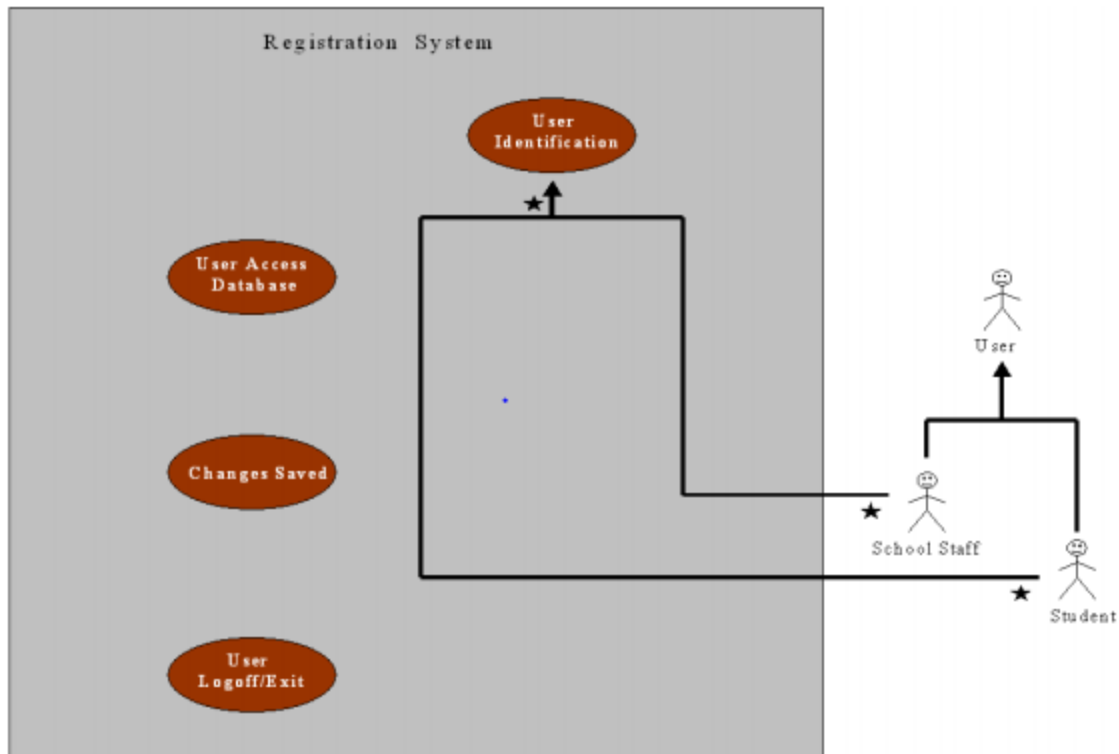
Answer: Modeling is one of the basic methods in empirical sciences. Generally speaking, it consists of the gradual construction of a cognitively use-ful – though simplified and idealized – image of described phenomena. As this image often takes the form of an abstract formal description – for example, a system of equations, or a set of logical formulas – modeling relies significantly on formal sciences such as mathematics, logic, or computer .

The following can be pointed to as typical examples of models constructed in empirical sciences: a) in physics – models of the atom, for example, the Bohr atomic model; b) in neurobiology – models of the neuron, for example, the McCulloch-Pitts linear neural model in psychology – computerized models of semantic memory, for example, Quillian's network model in cognitive science – partial models of mind, including a rich collection of rule-based reasoning models in science.

Q5. Create USE CASE diagram for the following narrative: (15)

Create a Use Case diagram for an online university registration system. The system should enable the staff of each academic department to examine the courses offered by their department, add and remove courses, and change the information about them (e.g., the maximum number of students permitted). It should permit students to examine currently available courses, add and drop courses to and from their schedules, and examine the courses for which they are enrolled. Department staff should be able to print a variety of reports about the courses and the students enrolled in them. The system should ensure that no student takes too many courses and that students who have any unpaid fees are not permitted to register (assume that a fees data store is maintained by the university's financial office, which the registration system accesses but does not change).

Answer:

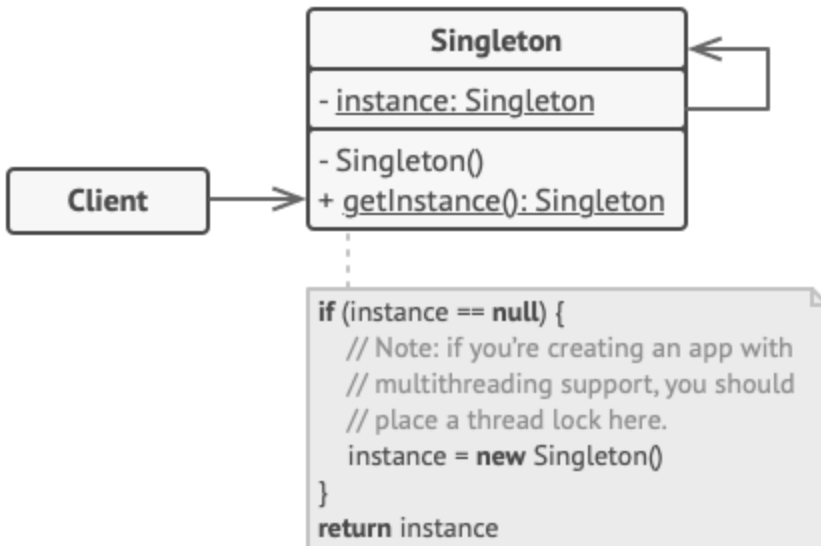


Q6. What is singleton pattern? Provide code structure and also uml structure.

(5)

Answer: In software engineering, the **singleton pattern** is a software design pattern that restricts the instantiation of a class to one "single" instance. This is useful when exactly one object is needed to coordinate actions across the system. The term comes from the mathematical concept of a singleton.

code structure



The **Singleton** class declares the static method `getInstance` that returns the same instance of its own class.

The Singleton's constructor should be hidden from the client code. Calling the `getInstance` method should be the only way of getting the Singleton object.

Pseudocode

In this example, the database connection class acts as a **Singleton**. This class doesn't have a public constructor, so the only way to get its object is to call the `getInstance` method. This method caches the first created object and returns it in all subsequent calls.

```

// The Database class defines the `getInstance` method that
// lets
// clients access the same instance of a database connection
// throughout the program.
class Database is
    // The field for storing the singleton instance should
    be
    // declared static.
    private static field instance: Database

    // The singleton's constructor should always be private
    to
    // prevent direct construction calls with the `new`
    // operator.
  
```

```

private constructor Database() is
    // Some initialization code, such as the actual
    // connection to a database server.
    // ...

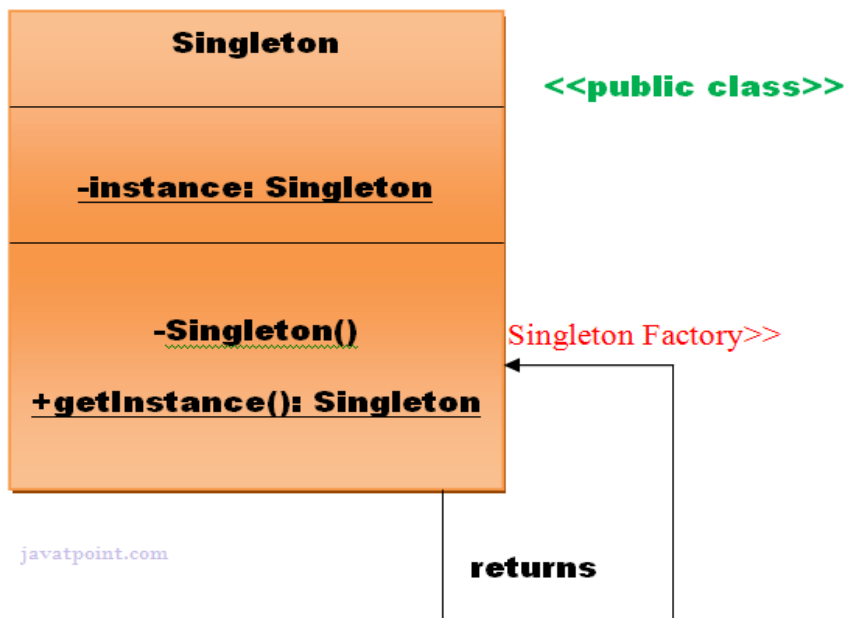
    // The static method that controls access to the
singleton
instance.
public static method getInstance() is
    if (Database.instance == null) then
        acquireThreadLock() and then
            // Ensure that the instance hasn't yet been
            // initialized by another thread while this
one
            // has been waiting for the lock's release.
            if (Database.instance == null) then
                Database.instance = new Database()
        return Database.instance

    // Finally, any singleton should define some business
logic
    // which can be executed on its instance.
public method query(sql) is
    // For instance, all database queries of an app go
    // through this method. Therefore, you can place
    // throttling or caching logic here.
    // ...

class Application is
    method main() is
        Database foo = Database.getInstance()
        foo.query("SELECT ...")
        // ...
        Database bar = Database.getInstance()
        bar.query("SELECT ...")
        // The variable `bar` will contain the same object
as
        // the variable `foo`.

```

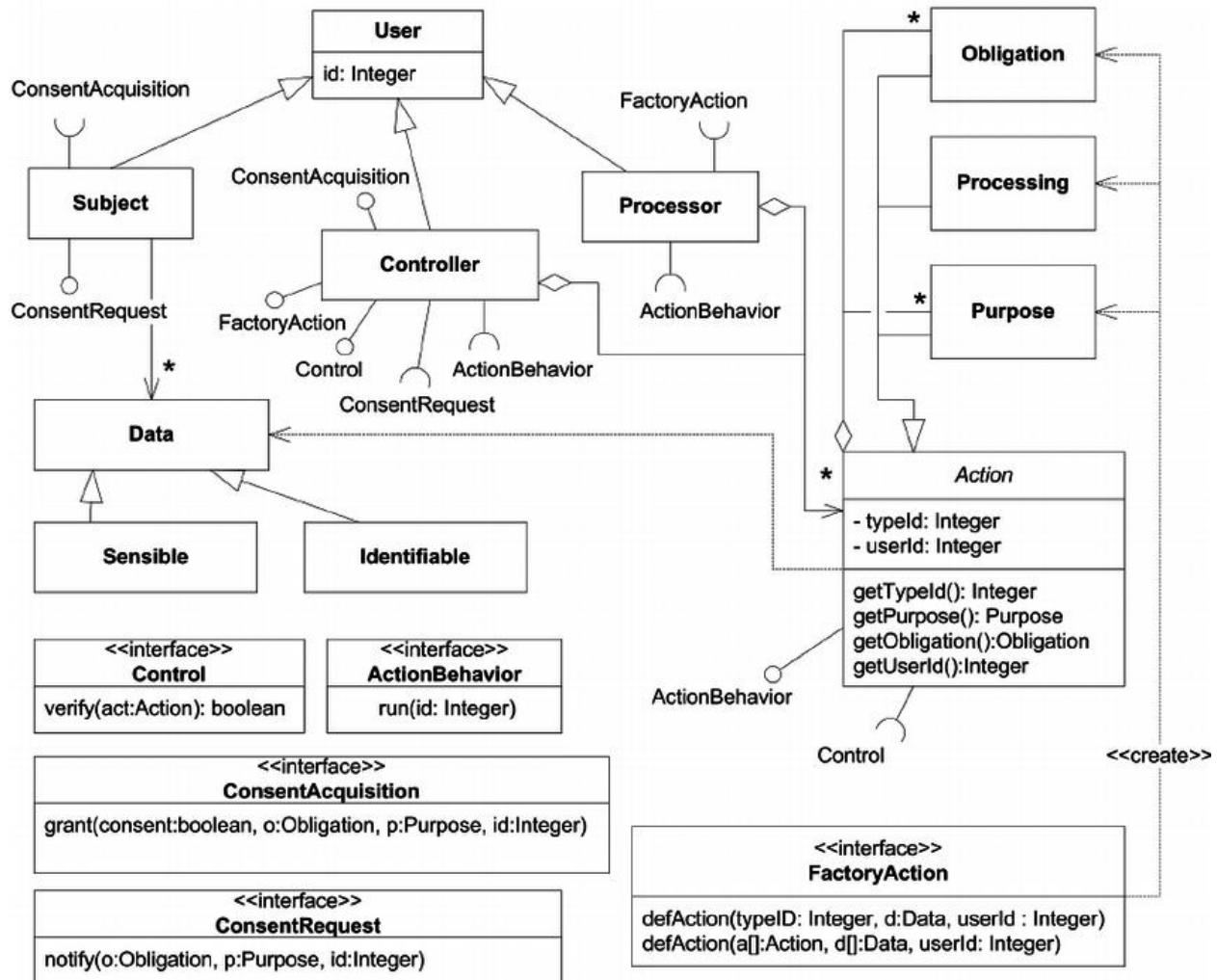
Uml of Singleton design pattern



Q7. What is privacy pattern? Provide uml structure. (5)

Inspired by the design patterns of object-oriented software architecture, we offer an initial set of "privacy patterns". Our intent is to describe the most important ways in which software systems can offer privacy to their stakeholders. We express our privacy patterns as class diagrams in the UML (Universal Modelling Language), because this is a commonly-used language for expressing the high-level architecture of an object-oriented system. In this initial set of privacy patterns, we sketch how each of Westin's four states of privacy can be implemented in a software system. In addition to Westin's states of Solitude, Intimacy, Anonymity, and Reserve, we develop a privacy pattern for an institutionalised form of Intimacy which we call Confidence.

UML structure



Q8. What is observer pattern? Provide generic uml structure and provide example with uml structure (5)

Answer: The **observer pattern** is a software design pattern in which an object, called the **subject**, maintains a list of its dependents, called **observers**, and notifies them automatically of any state changes, usually by calling one of their methods.

It is mainly used to implement distributed event handling systems, in "event driven" software. In those systems, the subject is usually called a "stream of events" or "stream source of events", while the observers are called "sink of events". The stream nomenclature simulates or adapts to a physical setup where the observers are physically separated and have no control over the emitted events of the subject/stream-source. This pattern then perfectly suits any process where data arrives through I/O, that is, where data is not available to the CPU at startup, but can arrive "randomly" (HTTP requests, GPIO data, user input from

keyboard/mouse/..., distributed databases and blockchains, ...). Most modern languages have built-in "event" constructs which implement the observer pattern components. implementations will use background threads listening for subject events and other support mechanism from the kernel (Linux epoll, ...).

The Observer design pattern is one of the twenty-three well-known "Gang of Four" design patterns that describe how to solve recurring design problems to design flexible and reusable object-oriented software, that is, objects that are easier to implement, change, test, and reuse.

A UML diagram is a diagram based on the UML (Unified Modeling Language) with the purpose of **visually representing a system** along with its main actors, roles, actions, artifacts or classes, in order to better understand, alter, maintain, or document information about the system.

Generic UML Structure:

UML diagrams, in this case, are used to communicate different aspects and characteristics of a system. However, this is only a top-level view of the system and will most probably not include all the necessary details to execute the project until the very end.

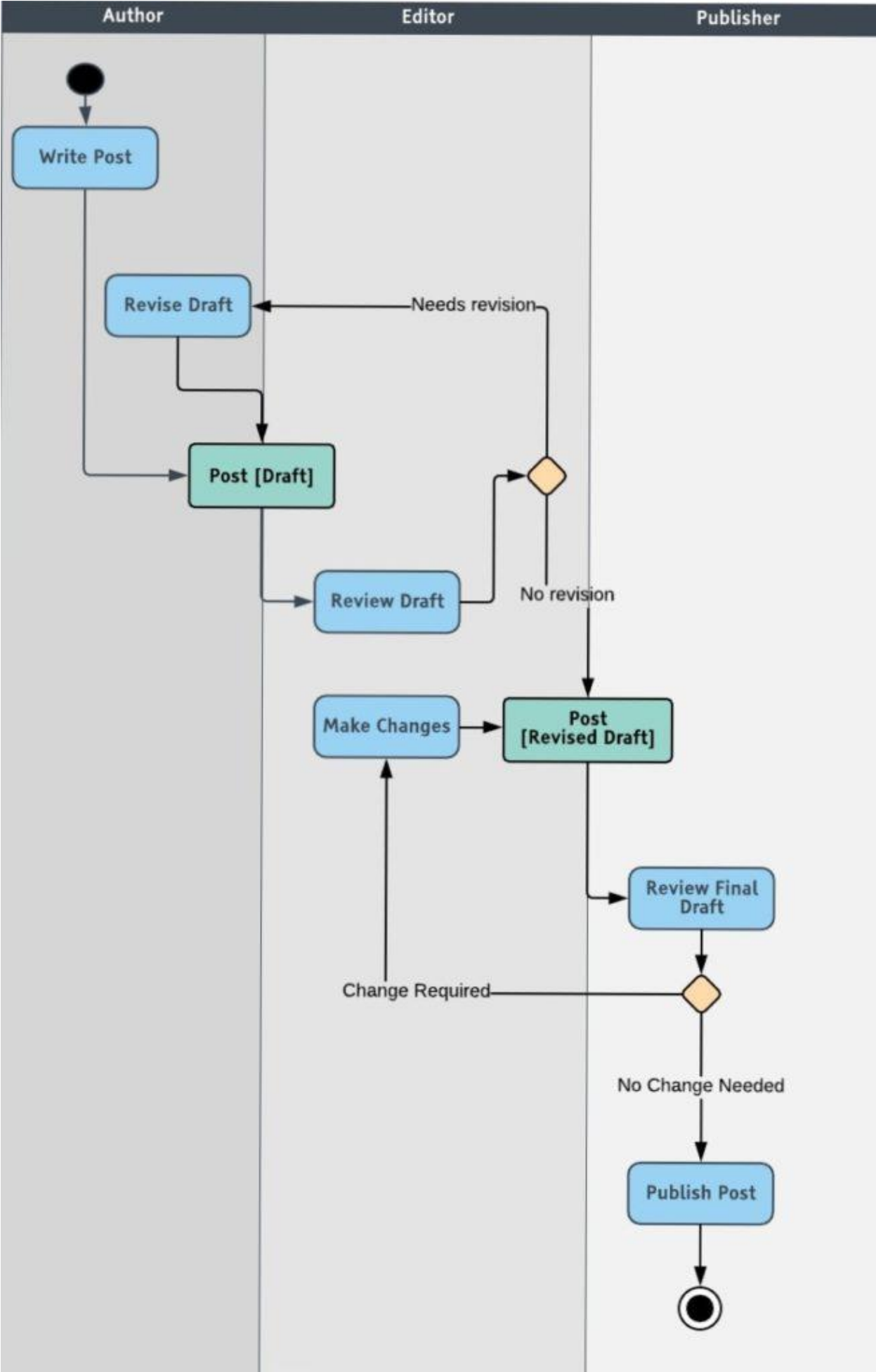
- **Forward Design** – The design of the sketch is done before coding the application. This is done to get a better view of the system or workflow that you are trying to create. Many design issues or flaws can be revealed, thus improving the overall project health and well-being.
- **Backward Design** – After writing the code, the UML diagrams are drawn as a form of documentation for the different activities, roles, actors, and workflows.

PRACTICAL EXAMPLE

One practical adoption would be to visually represent the process flow for telesales through an activity diagram. From the point in which an order is taken as an input, to the point where the order is completed and a specific output is given.

Activity Diagram

Activity diagrams are probably the most important UML diagrams for doing business process modeling. In software development, it is generally used to describe the flow of different activities and actions. These can be both sequential and in parallel. They describe the objects used, consumed or produced by an activity and the relationship between the different activities. All the above are essential in business process modeling.



----- Good ☺ Luck -----