5. EXAM REGISTRATION SYSTEM

Aim: To define problem statement and create UML system models for building Exam Registration system

Problem Statement:

Exam Registration system is used in the effective dispatch of registration form to all of the students. This system adopts a comprehensive approach to minimize the manual work and schedule resources, time in a cogent manner. The core of the system is to get the online registration form (with details such as name, reg. no etc.,) filled by the student whose testament is verified for its genuineness by the Exam Registration System with respect to the already existing information in the database.

Requirements Specification:

From the above problem statement, even without doing any deep analysis, we might easily identify some of the basic functionality of the system: New user registration, user login, fill registration, view registration, update, pay fee, download hall ticket, and user logout.

System Modeling with CASE tool (Star UML):

Star UML tool is used to transform requirements into System models and refine them into designs. A new model is opened and renamed as ExamRegistration. The following models are created to represent the requirements mentioned above.

Activity Diagram:(http://vlabs.iitkgp.ac.in/se/6/theory/)

Activity diagrams fall under the category of behavioural diagrams in Unified Modeling Language. It is a high level diagram used to visually represent the flow of control in a system. It has similarities with traditional flow charts. However, it is more powerful than a simple flow chart since it can represent various other concepts like concurrent activities, their joining, and so on.

Components of an Activity Diagram

Below we describe the building blocks of an activity diagram.

Activity

An activity denotes a particular action taken in the logical flow of control. This could simply be invocation of a mathematical function, alter an object's properties

and so on. An activity is represented with a rounded rectangle. A label inside the rectangle identifies the corresponding activity.

There are two special type of activity nodes: initial and final. They are represented with a filled circle, and a filled in circle with a border respectively. Initial node represents the starting point of a flow in an activity diagram. There could be multiple initial nodes, which means that invoking that particular activity diagram would initiate multiple flows.

A final node represents the end point of all activities. Like an initial node, there could be multiple final nodes. Any transition reaching a final node would stop all activities.

Flow

A flow (also termed as edge, or transition) is represented with a directed arrow. This is used to depict transfer of control from one activity to another, or to other types of components, as we will see below. A flow is often accompanied with a label, called the guard condition, indicating the necessary condition for the transition to happen.

Decision

A decision node, represented with a diamond, is a point where a single flow enters and two or more flows leave. The control flow can follow only one of the outgoing paths. The outgoing edges often have guard conditions indicating true-false or if-then-else conditions.

Merge

This is represented with a diamond shape, with two or more flows entering, and a single flow leaving out. A merge node represents the point where at least a single control should reach before further processing could continue.

Fork

Fork is a point where parallel activities begin. For example, when a student has been registered with a college, he can in parallel apply for student ID card and library card. A fork is graphically depicted with a black bar, with a single flow entering and multiple flows leaving out.

Join

A join is depicted with a black bar, with multiple input flows, but a single output flow. Physically it represents the synchronization of all concurrent activities. Unlike a merge, in case of a join all of the incoming controls **must be completed** before any

further progress could be made. For example, a sales order is closed only when the customer has receive the product, **and** the sales company has received it's payment.

Note

UML allows attaching a note to different components of a diagram to present some textual information. The information could simply be a comment or may be some constraint. A note can be attached to a decision point, for example, to indicate the branching criteria.

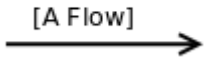
Partition

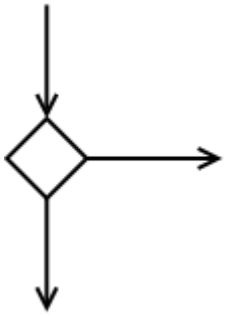
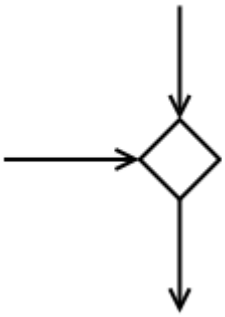
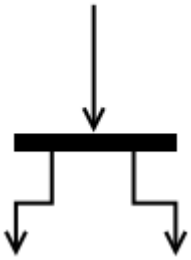
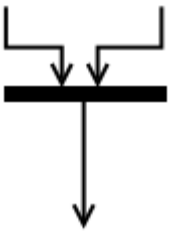
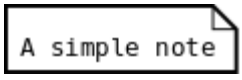
Different components of an activity diagram can be logically grouped into different areas, called partitions or swimlanes. They often correspond to different units of an organization or different actors. The drawing area can be partitioned into multiple compartments using vertical (or horizontal) parallel lines. Partitions in an activity diagram are not mandatory.

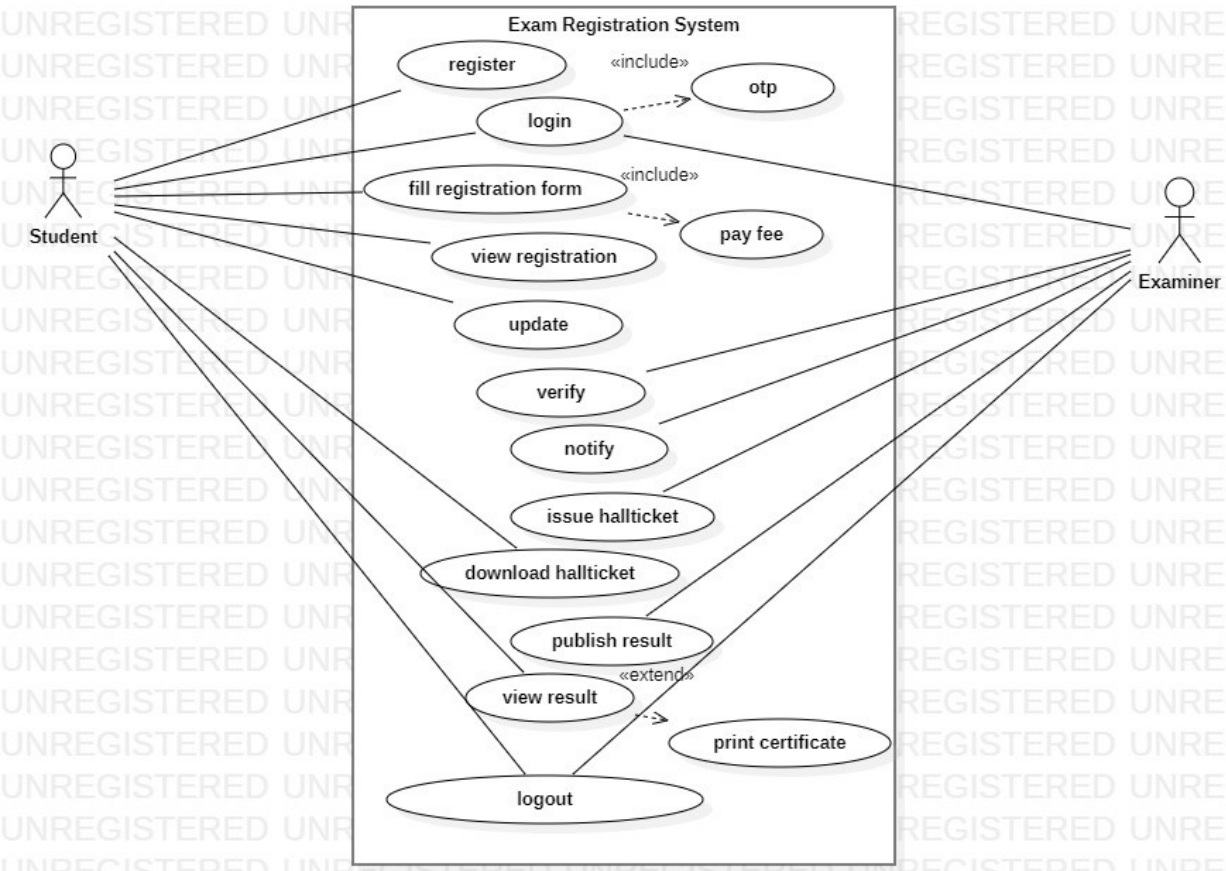
Guidelines for drawing an Activity Diagram

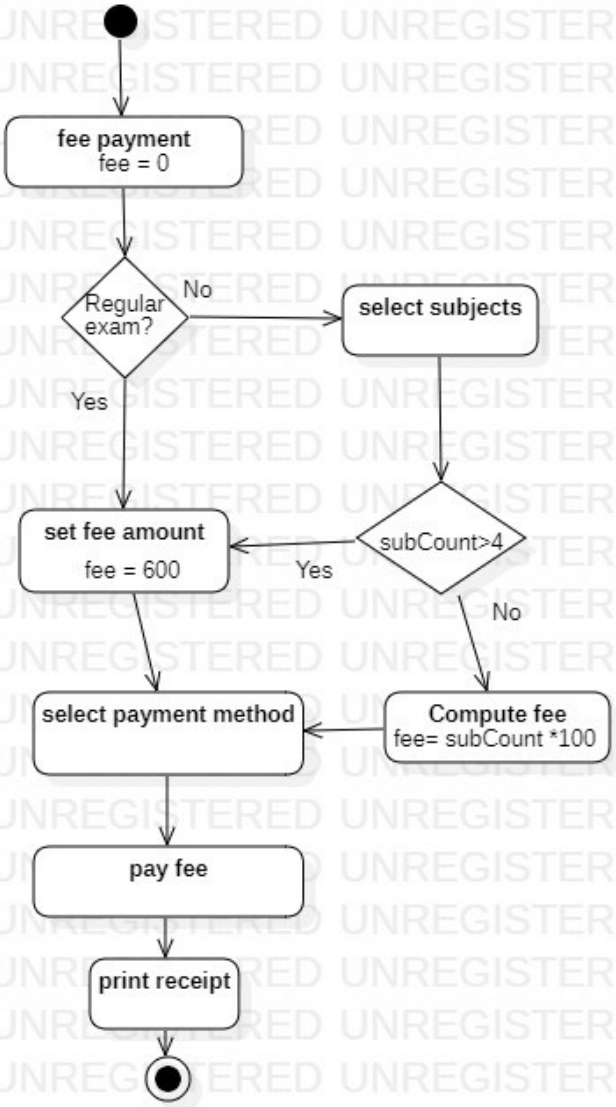
The following general guidelines could be followed to pictorially represent a complex logic.

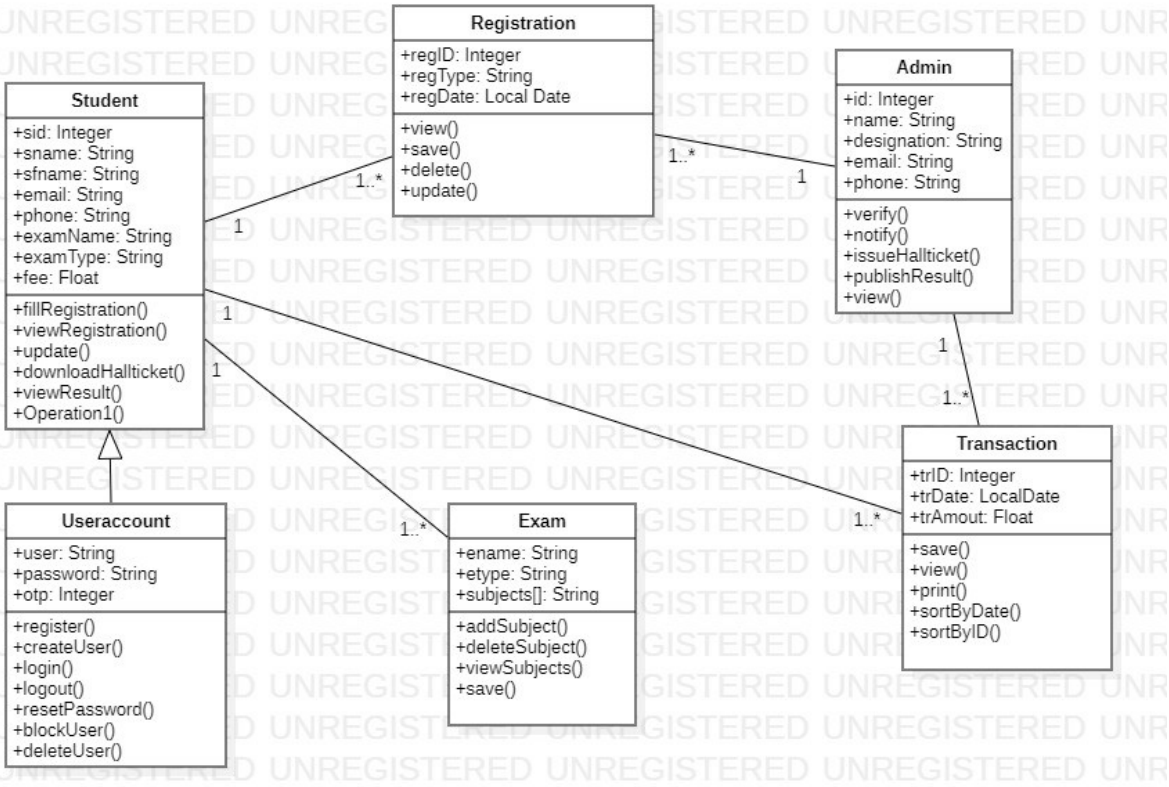
- Identify tiny pieces of work being performed by the system
- Identify the next logical activity that should be performed
- Think about all those conditions that should be made, and all those constraints that should be satisfied, before one can move to the next activity
- Put non-trivial guard conditions on the edges to avoid confusion

Component	Graphical Notation
Activity	
Flow	

Component	Graphical Notation
Decision	
Merge	
Fork	
Join	
Note	


Exam Registration System
register
«include»
otp
login
fill registration form
«include»
pay fee
view registration
update
verify
notify
issue hallticket
download hallticket
publish result
«extend»
view result
print certificate
logout
Student
Examiner


fee payment
fee = 0
Regular exam?
No
Yes
select subjects
subCount>4
Yes
No
set fee amount
fee = 600
Compute fee
fee= subCount *100
select payment method
pay fee
print receipt


Registration
+regID: Integer
+regType: String
+regDate: Local Date
+view()
+save()
+delete()
+update()
Admin
+id: Integer
+name: String
+designation: String
+email: String
+phone: String
+verify()
+notify()
+issueHallticket()
+publishResult()
+view()
Student
+sid: Integer
+sname: String
+sfname: String
+email: String
+phone: String
+examName: String
+examType: String
+fee: Float
+fillRegistration()
+viewRegistration()
+update()
+downloadHallticket()
+viewResult()
+Operation1()
Transaction
+trID: Integer
+trDate: LocalDate
+trAmout: Float
+save()
+view()
+print()
+sortByDate()
+sortByID()
Useraccount
+user: String
+password: String
+otp: Integer
+register()
+createUser()
+login()
+logout()
+resetPassword()
+blockUser()
+deleteUser()
Exam
+ename: String
+etype: String
+subjects[]: String
+addSubject()
+deleteSubject()
+viewSubjects()
+save()
1..*
1
1..*
1
1
1
1
1..*
1..*
1..*