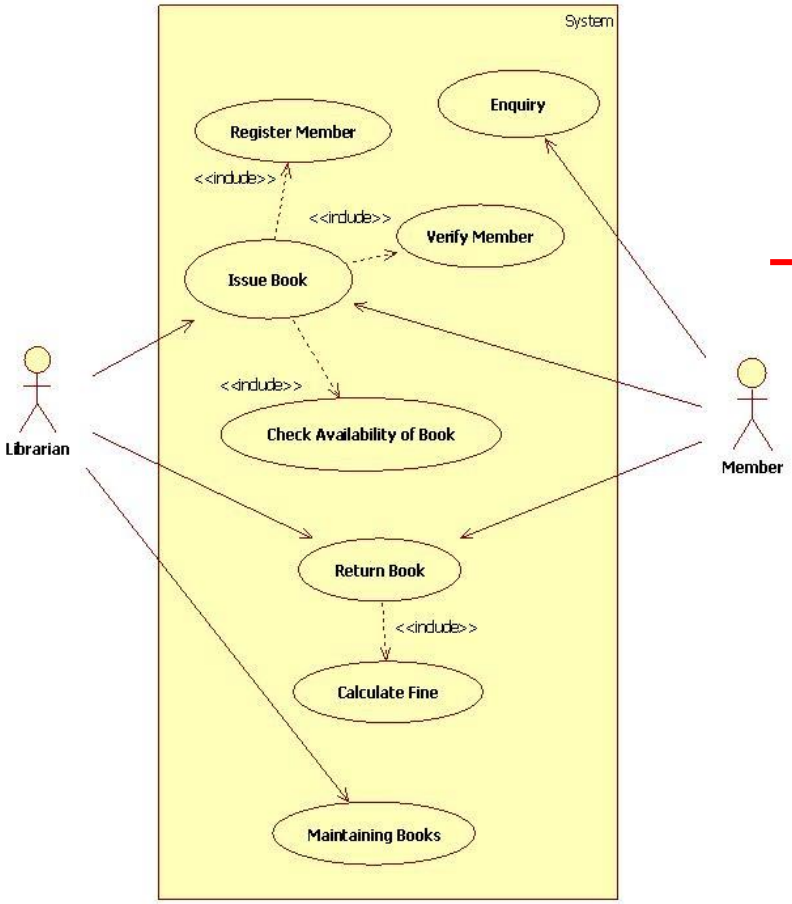
OOAD (Object Oriented Analysis and Design)

Unit 2 :- Elaboration

Use Case to Code




```
Class Librarian
{
        public void issueBook(Book b)
        {
                Book b1 =new Book(b);
                b1.addBook();
        }
}
Class Book
{
        public void addBook();
}
```

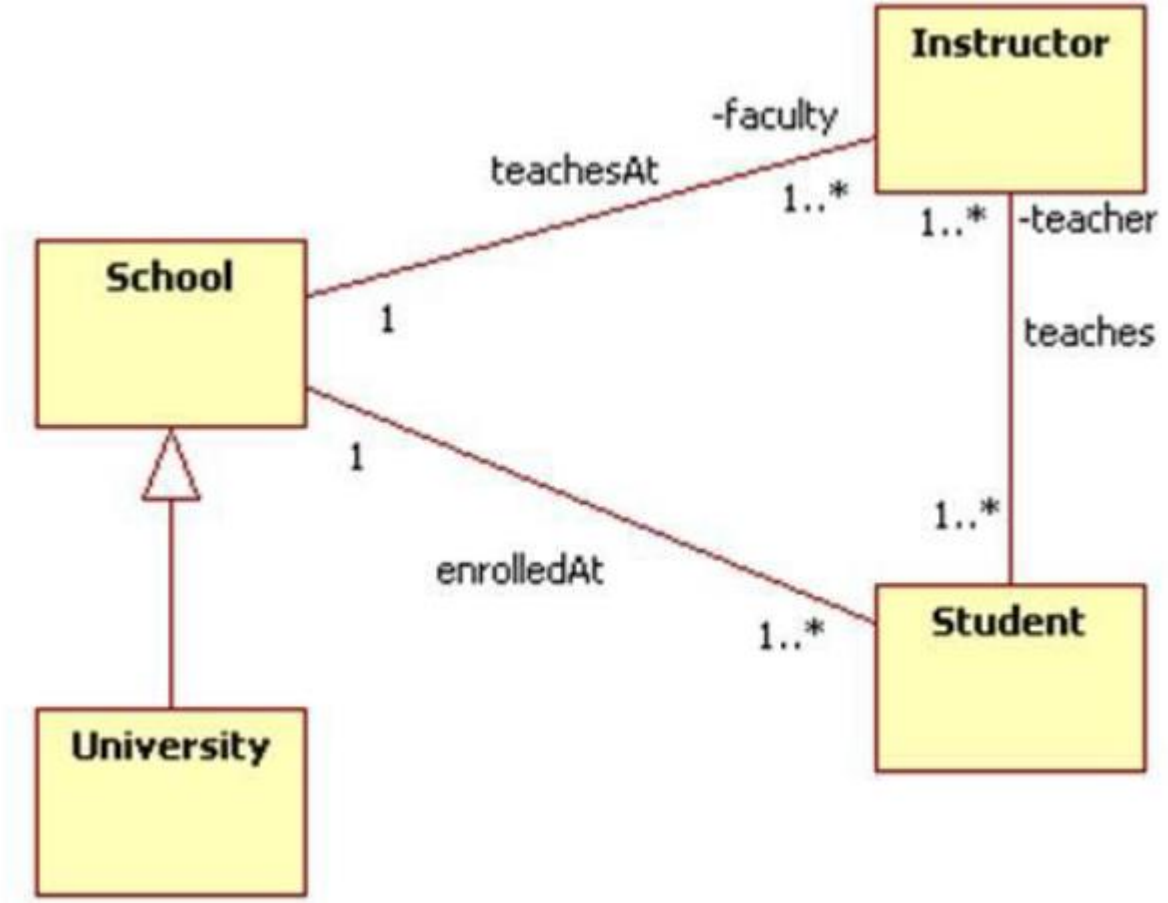
Domain Model

- A domain is a collection of related concepts, relationships, and workflows.
- A domain model is a package containing class and activity diagrams.
- In software engineering, a domain model is a conceptual model of the domain that incorporates both behavior and data
- Is not a description of software objects instead it is a visual representation of conceptual classes in a problem domain

Domain Model…

- **Features of a domain model**
  - Domain classes – each domain class denotes a type of object.
  - Attributes – an attribute is the description of a named slot of a specified type in a domain class; each instance of the class separately holds a value.
  - Associations – an association is a relationship between two (or more) domain classes that describes links between their object instances. Associations can have roles, describing the multiplicity and participation of a class in the relationship.
  - Additional rules – complex rules that cannot be shown with symbolically can be shown with attached notes.

Domain Model (Example)…



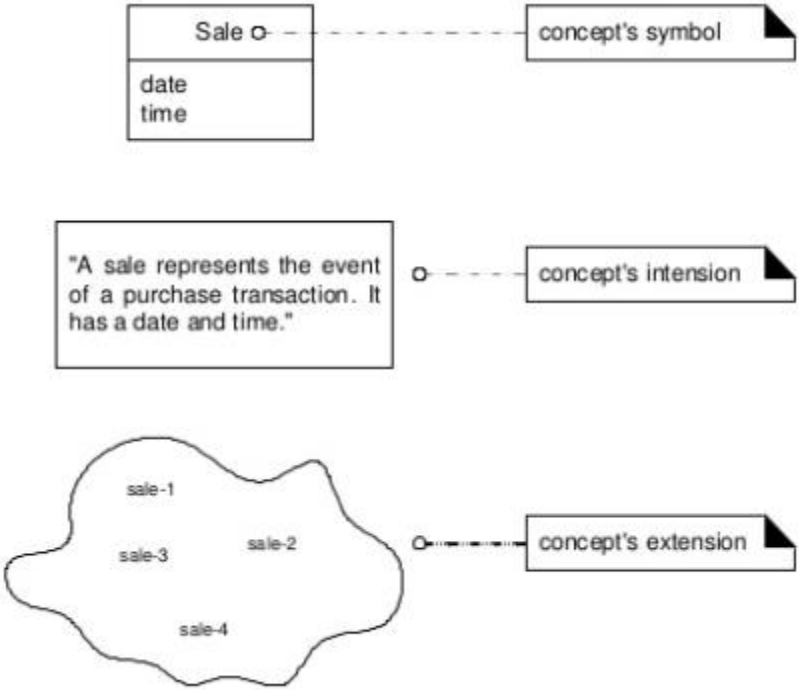
How to create a domain model?

- Find the conceptual classes
- Draw them as classes in a UML class diagram
- Add associations and attributes

Conceptual Class

- An conceptual class is an idea, thing or object
- **Symbol**
  - Representing a conceptual class
- **Intension**
  - Definition of a conceptual class
  - Eg:- Customer may be a person that purchase good
- **Extension**
  - The set of examples to which the conceptual classes applies
  - Eg: the customer may be John, Jane, Tom

Conceptual Class…

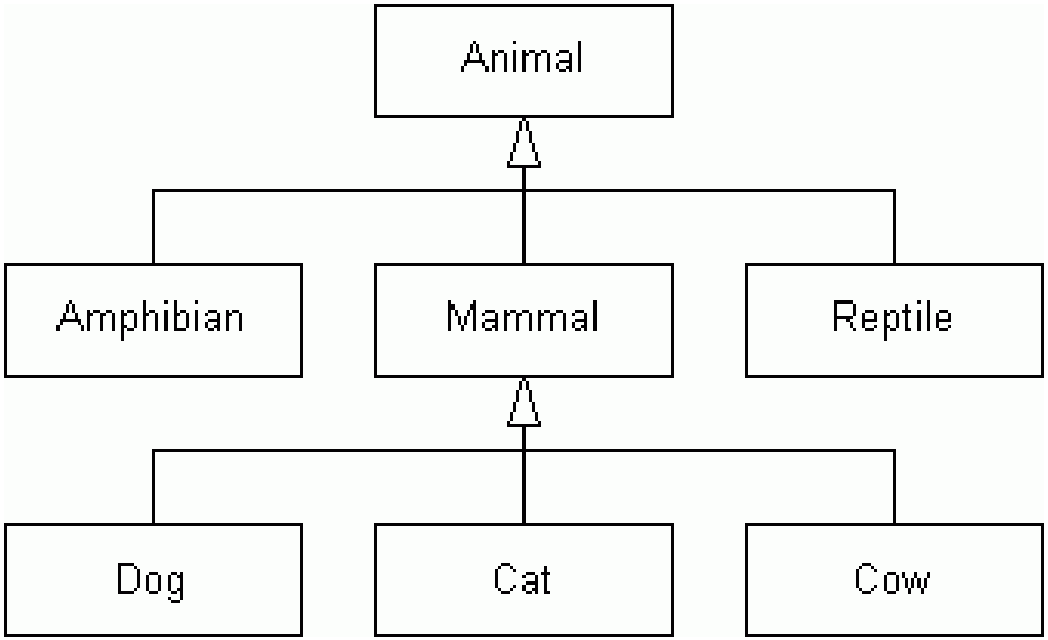


Strategies for finding conceptual class

- Reuse of modify existing model
- Use a category list
- Identify noun phrase

Finding conceptual class hierarchies

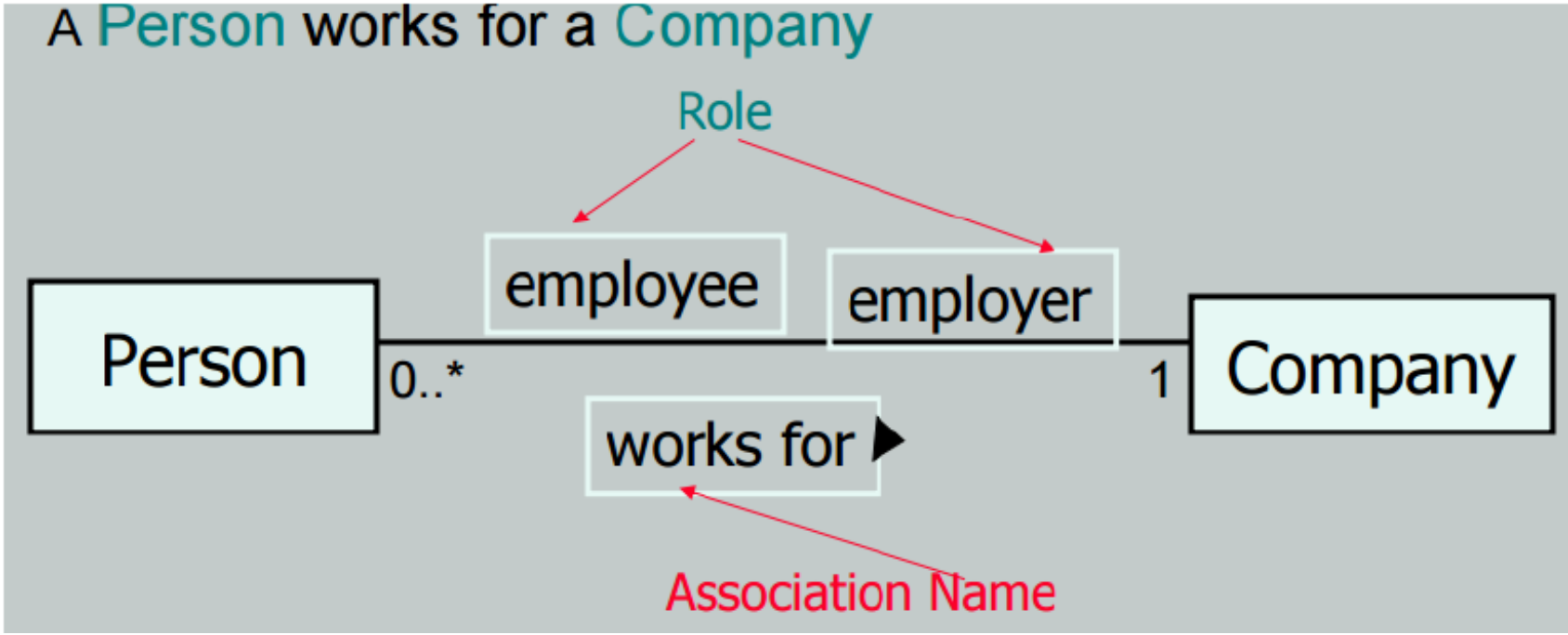
- Defining superclass and subclass



Association

- An association is an relationship between instances of types that indicates some meaningful and interesting connections
- Represented as a line between classes with an association name
- Associations are inherently bidirectional
- A link (relationship) between classes
- Each end of an association is called a ***Role***
- Role may be
  - Name
  - Multiplicity expression (*defines the number of instances of one class, that can be associated with one instance of other class*)

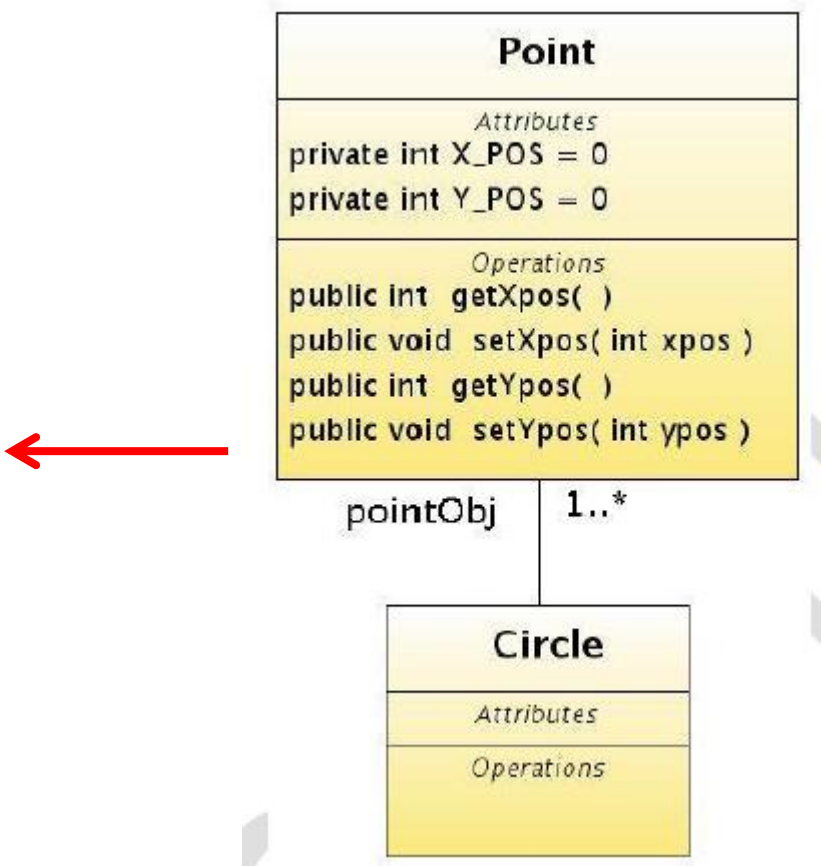
Association…



Association…

- **Multiplicity**
  - 1 → only one instance
  - 0..1 → zero or one instance
  - 0..* → zero or many instance
  - 1..* → one or many instance
  - 3,5,7 → exactly 3, 5 or 8

```
class Circle
{
        private Point objPoint;
}
class Point
{
        private int X_POS = 0;
        private int Y_POS = 0;
        public int getXPos()
        {
                return X_POS;
        }//so on…..
```



Aggregation and Composition

- **Aggregation** and **Composition** are subsets of association
- **Aggregation** implies a relationship where the child can exist independently of the parent
  - Example: Class (parent) and Student (child). Delete the Class and the Students still exist
- **Composition** implies a relationship where the child cannot exist independent of the parent
  - Example: House (parent) and Room (child). Rooms don't exist separate to a House

Aggregation and Composition (Example)…

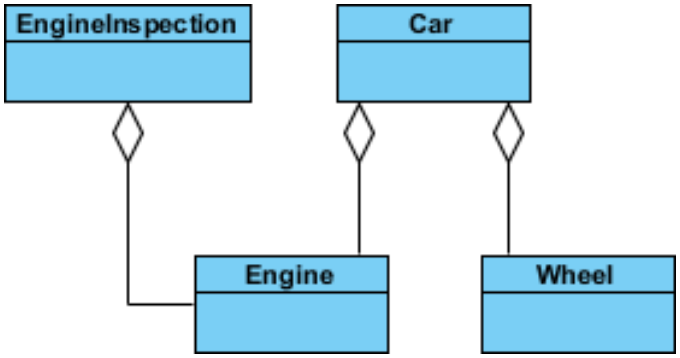


Fig (1) :- Aggregation

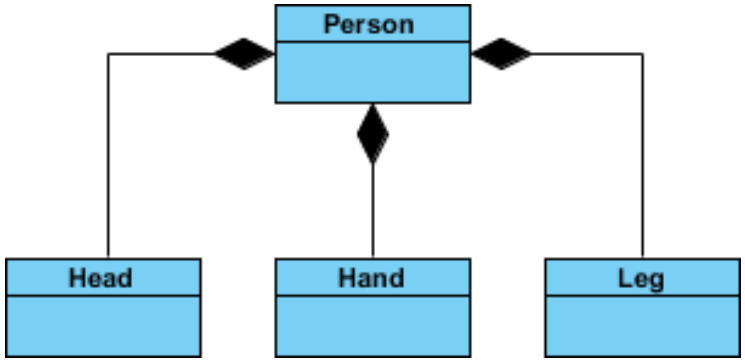
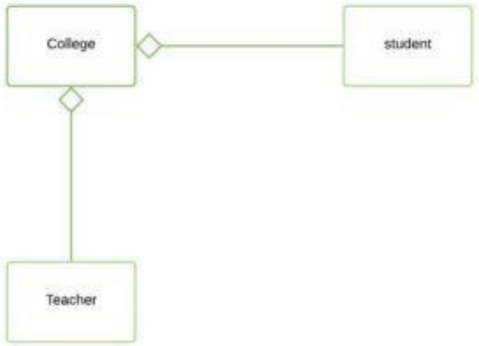


Fig (2):- Composition

Aggregation to Code




```
class Student
{
}
class Teacher
{
}
class College
{
        private List<Student> stds;
         private List<Teacher> tcher;
}
```

Composition to Code

```
class Engine
{
        public void work()
        {
        }
}
class Car
{
        private Engine eng;
        public Car(Engine end)
        {
        }
}
class CompositionTest
{
        public static void main(String arg[])
        {
                Engine e = new Engine();
                Car c = new Car(e);
        }
}
```

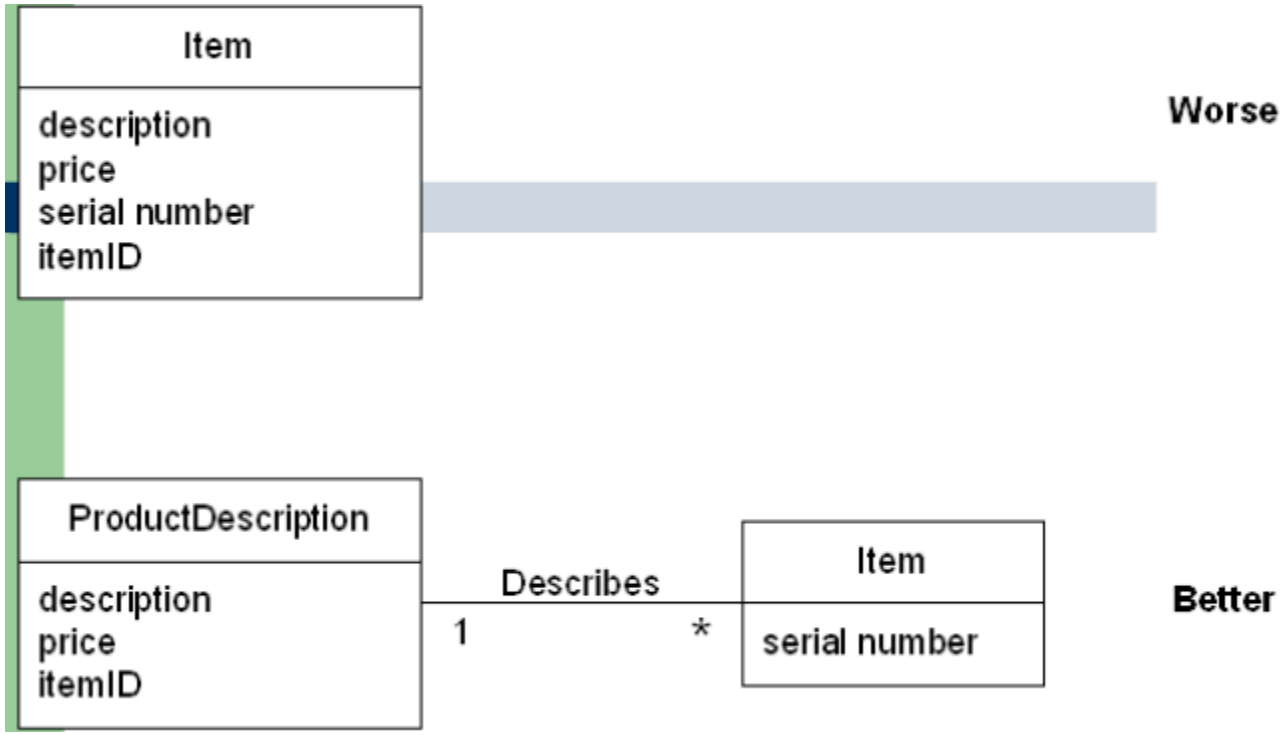
Attributes

- An attribute is a logical data value of an object
- You can also have derived attributes denoted by / before the name

Description Class

- A description class contains information that describes something else
- Example :- *ProductDescription* that records the price, picture and text description of an Item
- Does not represent an item, instead represents a description of an item
- Why
  - An item instance represents a physical item in a store
  - An item has a description, price, ID which are not recorded anywhere else
  - Everyone working in the store has amnesia
  - Every time an item is sold, a corresponding software instance of item is deleted
  - If we keep all the information in, say, a sales line item, once all of that item are sold, there is no record of what the item was
  - How does this relate to database design?

Description Class…



UML Activity Diagrams

- Activity diagram is basically a flow chart to represent the flow form one activity to another activity
- The control flow is drawn from one operation to another and this flow can be sequential, branched or concurrent
- This flow can be sequential, branched, or concurrent
- Describe the dynamic aspects of the system
- Specify system behavior

UML Activity Diagrams…

- **Purposes**
  - Draw the activity flow of a system
  - Describe the sequence from one activity to another
  - Describe the parallel, branched and concurrent flow of the system

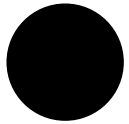
Component of Activity Diagram

- **Activity**
  - Is some task which needs to be done
  - Represented by rounded rectangle

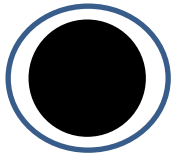
- **Transmission (Flow)**
  - When the activity of a state completes, flow of control passes to the next action
  - Represented by arrow symbol

Component of Activity Diagram…

- **Starting Node**
  - The source of flow of control
  - Represented by marked circle

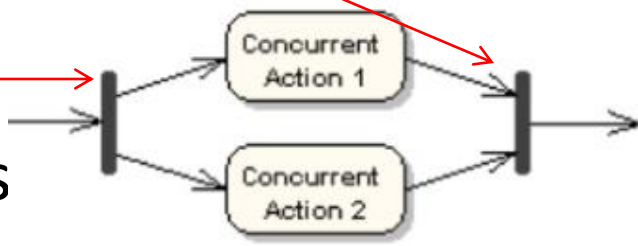


- **Ending Node**
  - Destination of flow of control
  - Represented by marked circle in a circle



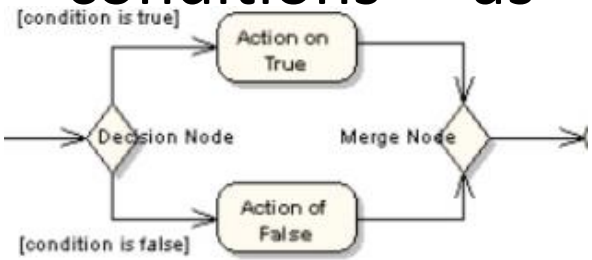
Component of Activity Diagram…

- **Join**
  - Join and Fork have the same notation (either a horizontal or vertical bar)
  - A black bar with several flows entering in it and one leaving from it
  - Denoted the end of parallel activities
- **Fork**
  - Beginning of the parallel activities



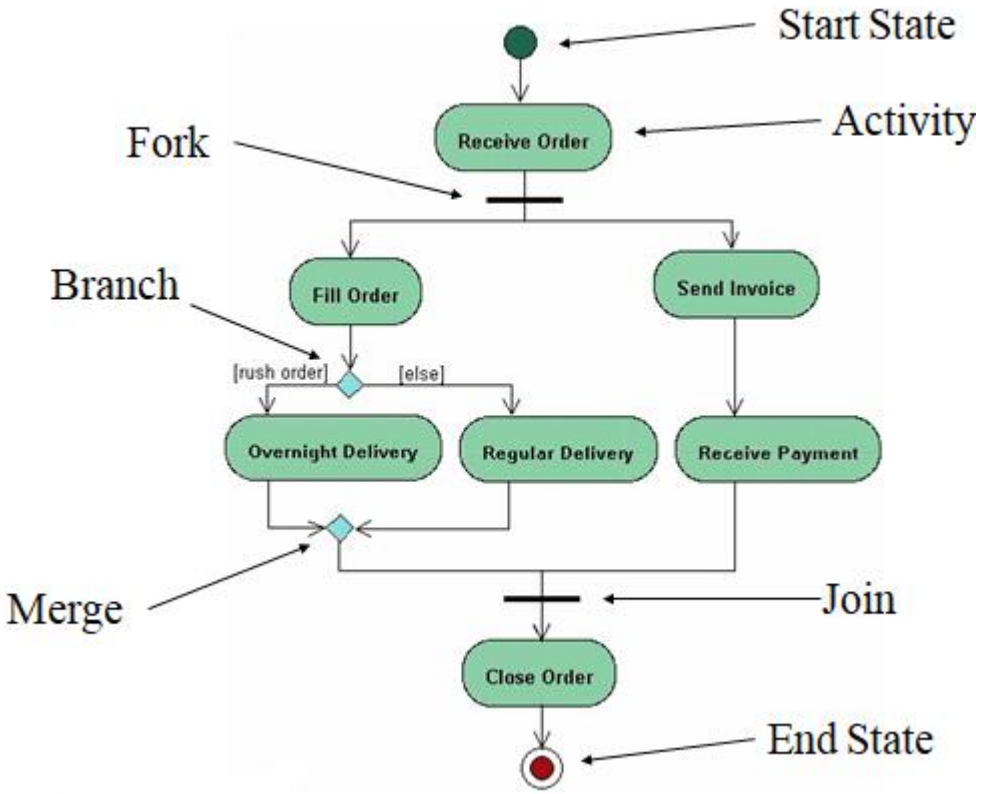
Component of Activity Diagram…

- **Decision**
  - A diamond with one flow entering and several leaving
  - The flow leaving includes conditions as TRUE/FALSE state



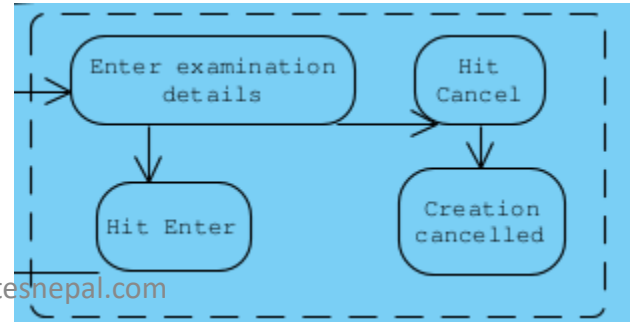
- **Merge**
  - A diamond with several flows entering and one leaving
  - The implication is that all incoming flows to reach this point until processing continues

Activity Diagram (Components and Example)



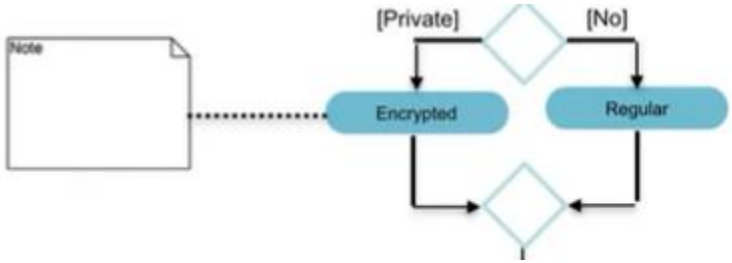
Component of Activity Diagram…

- **Flow Final**
  - Indicates that the process stop at this point
  - The circle with X 
- **Swim Lane**
  - A partition in activity diagram by means of dashed line, called swim lane
  - Each zone represents the responsibilities caried out by different actor



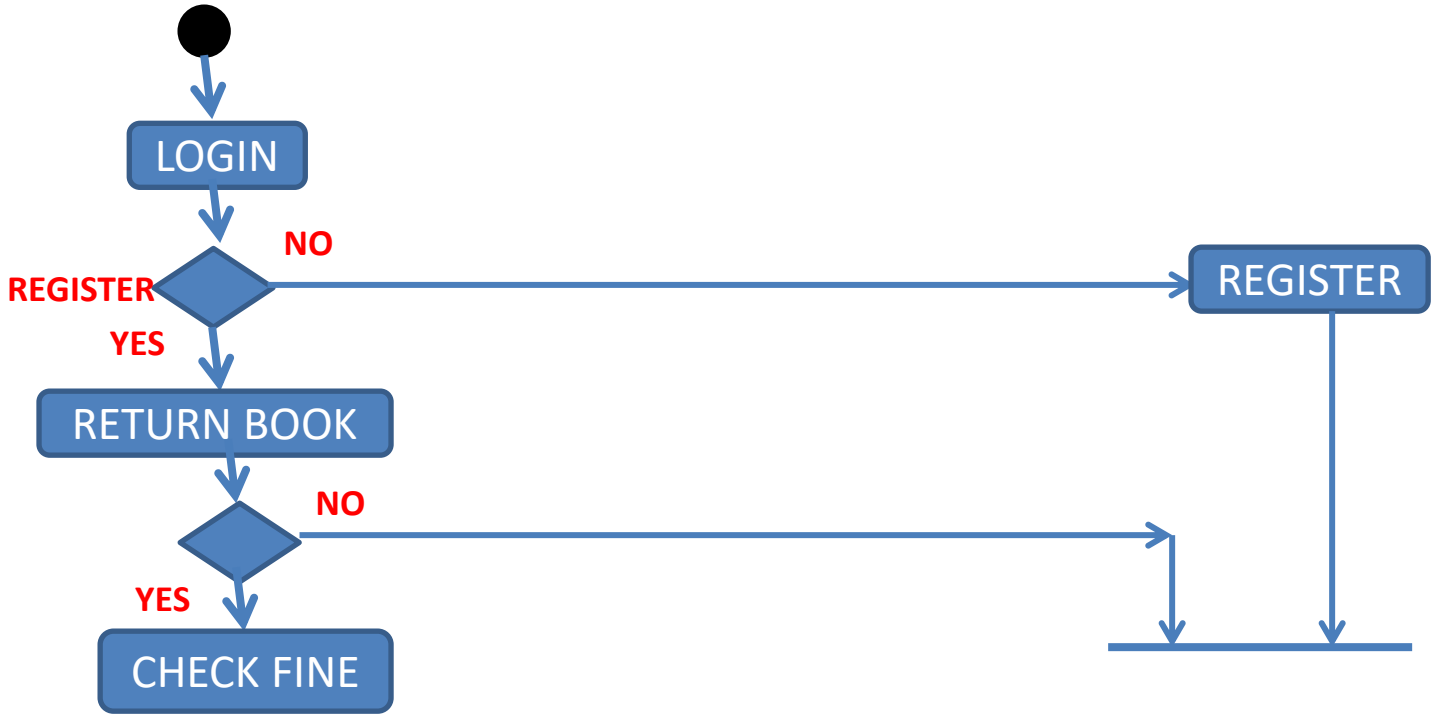
Component of Activity Diagram…

- **Notes**
  - Used to add relevant comments to elements

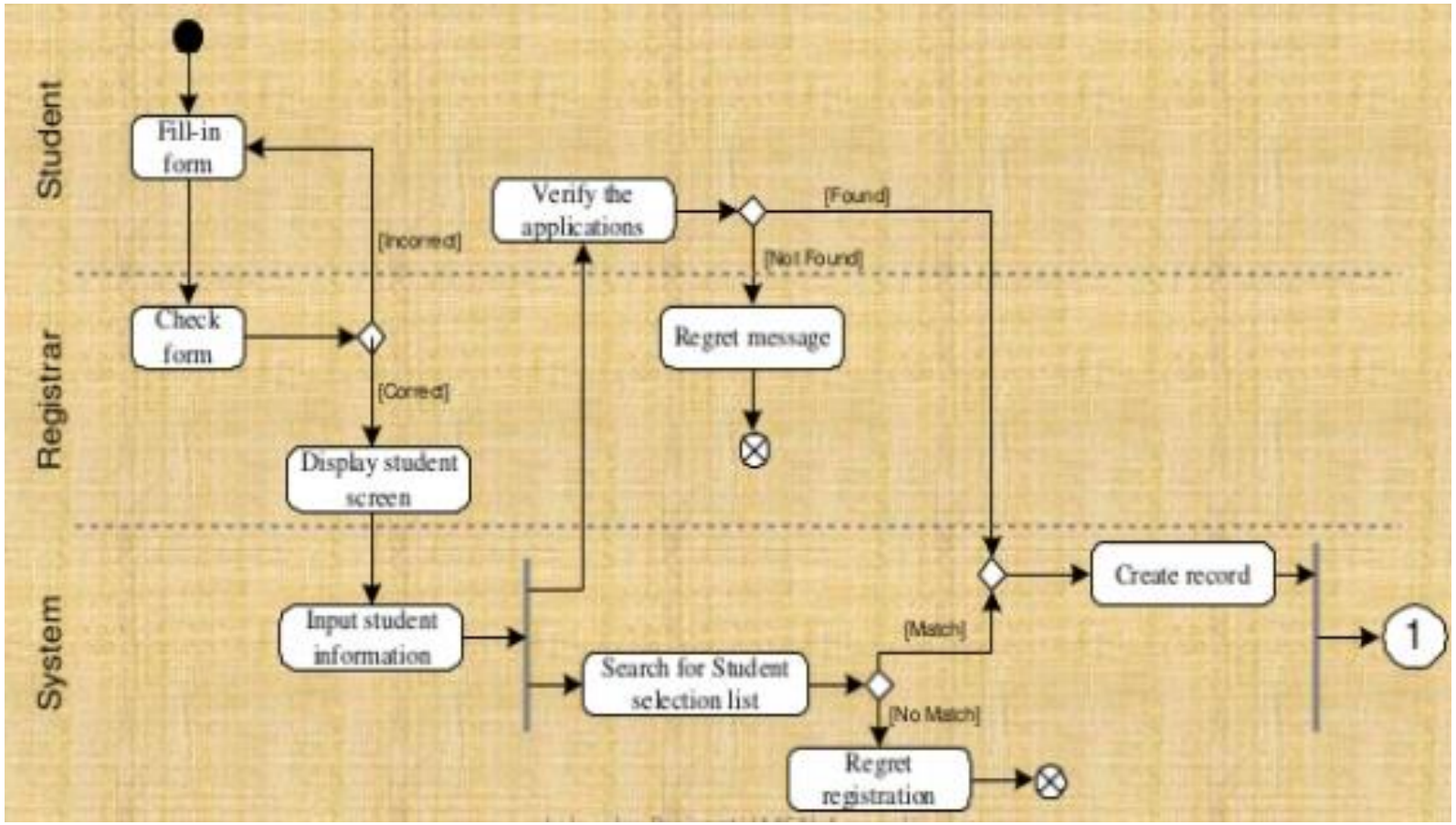


Activity Diagram (Components and Example)

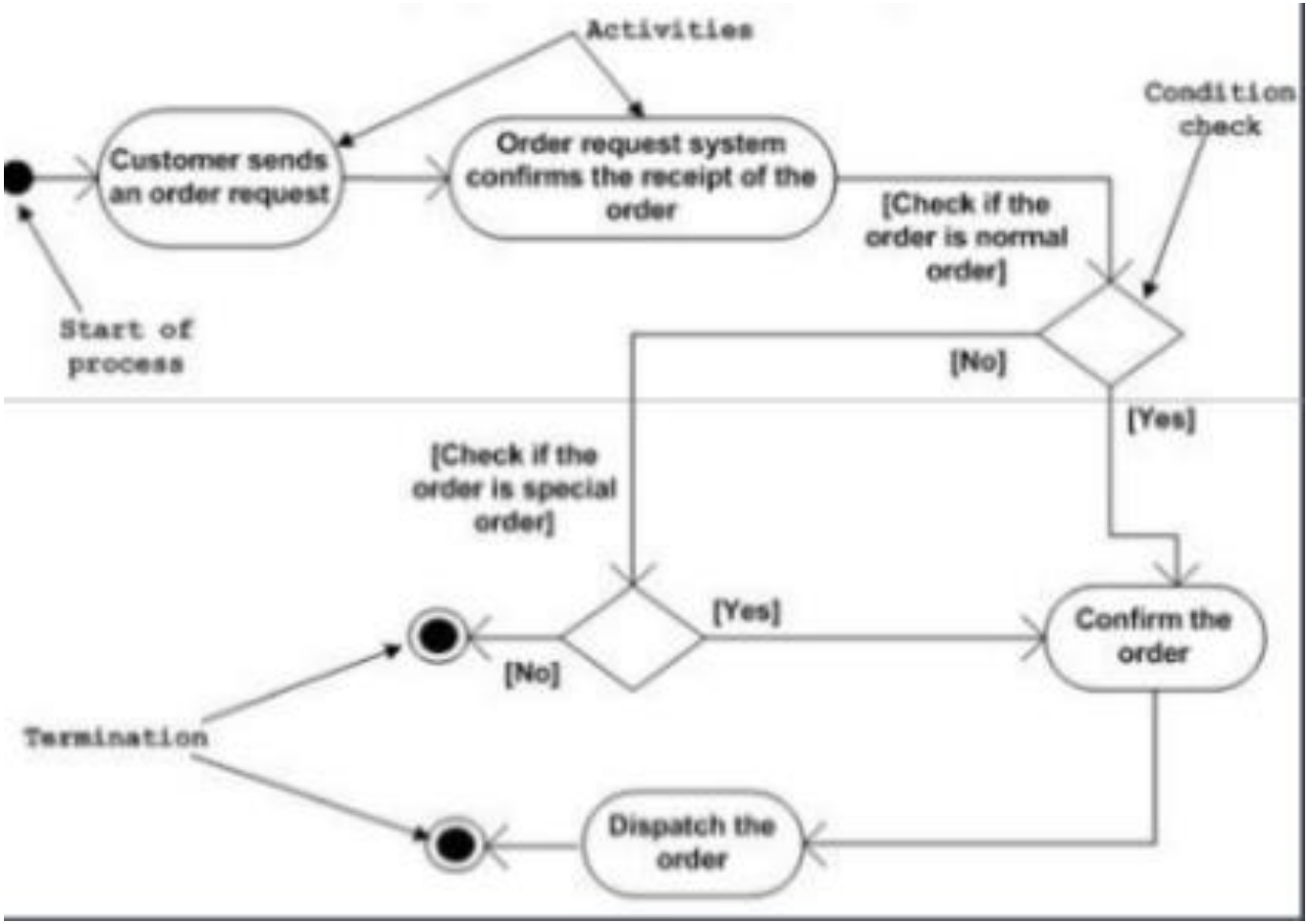
- **Borrow Book**



Activity Diagram (Components and Example)



Activity Diagram (Components and Example)

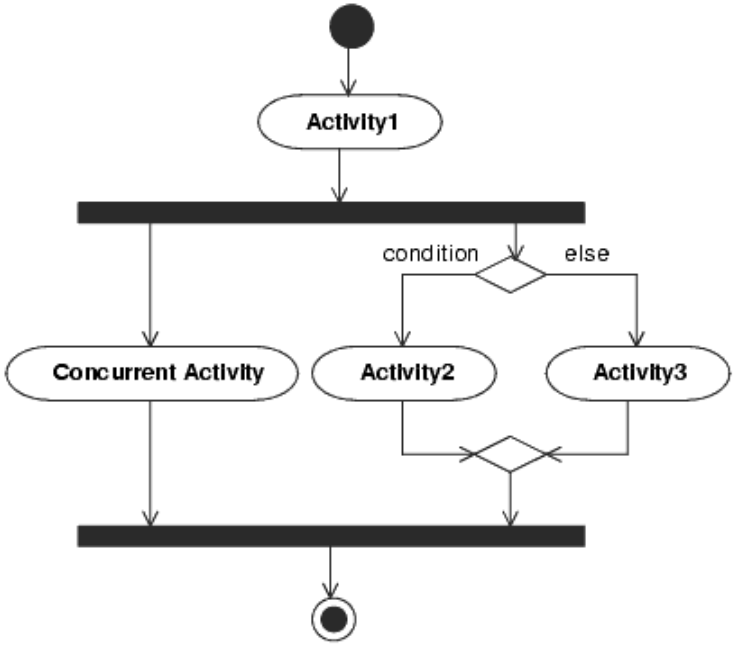


Component of Activity Diagram…

- **Join VS Merge**
  - Join synchronizes two inflows and produces a single outflow
  - The outflow from a join cannot execute until all inflows have been received
  - A merge passes any control flows straight through it
  - If two or more inflows are received by merge, the action pointed to by its outflow is executed two or more times

Component of Activity Diagram…

- Join VS Merge
  - ***Activity 2*** and ***Activity 3*** are our **alternate flows** and only **one of which** will arrive. And they are **not** synchronize incoming.
  - However, the ***Concurrent_Activity*** and result of decision between ***Activity 1*** and ***Activity 2*** (that merged into one output) are synchronize incoming concurrent flows. The join **waits for both** to perform and continue.



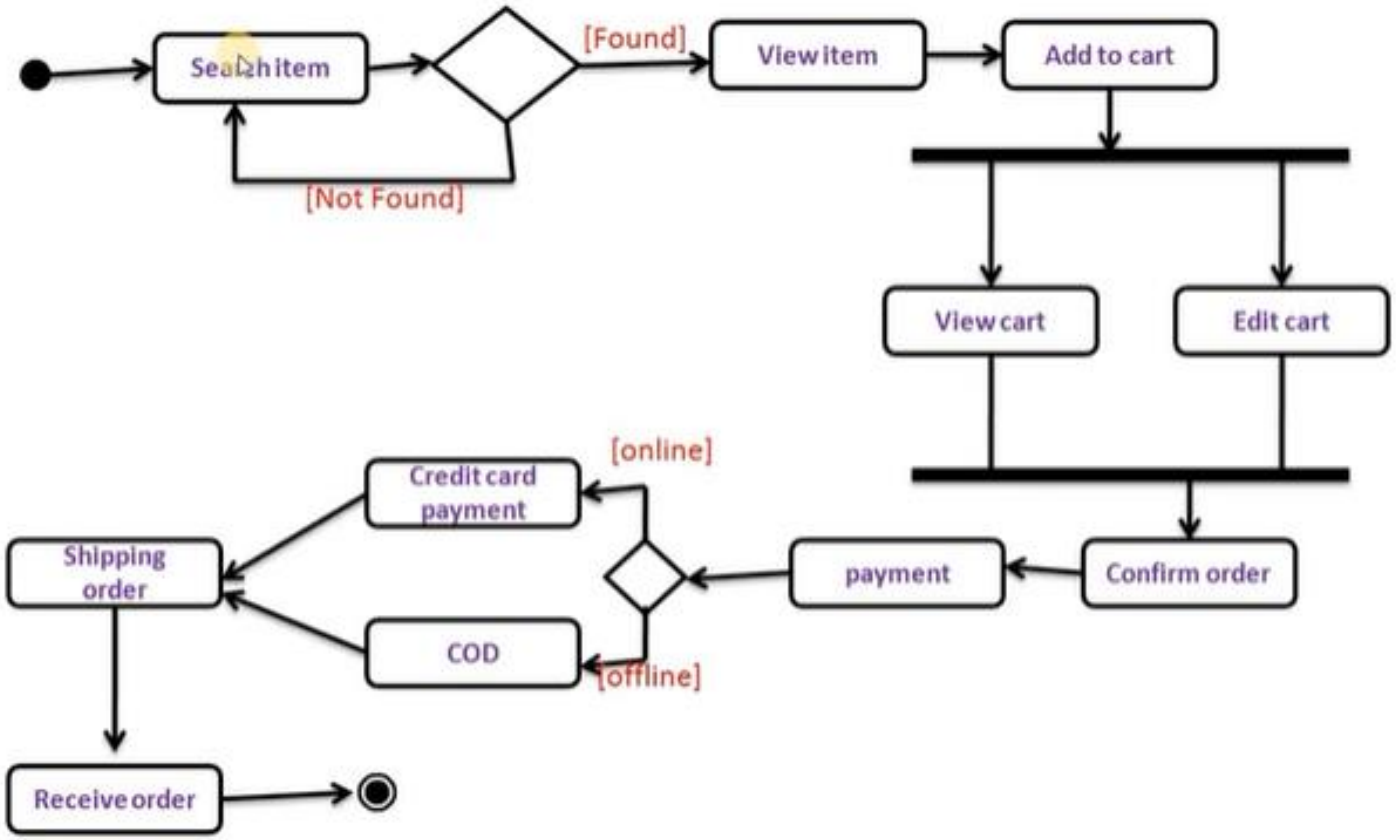
Drawbacks of Activity Diagram

- Activity diagram tell us about what is happening, but not who does what
- Does not specify the relation between class and activity

Activity Diagram (Exercise)

- **Online Shopping Process**
  - Online customer can browse or search items, view specific item, add it to shopping cart, view and update the shopping cart, check out
  - User can view shopping cart at any time
  - Check out is assumed to include user registration and login

Activity Diagram (Exercise)…



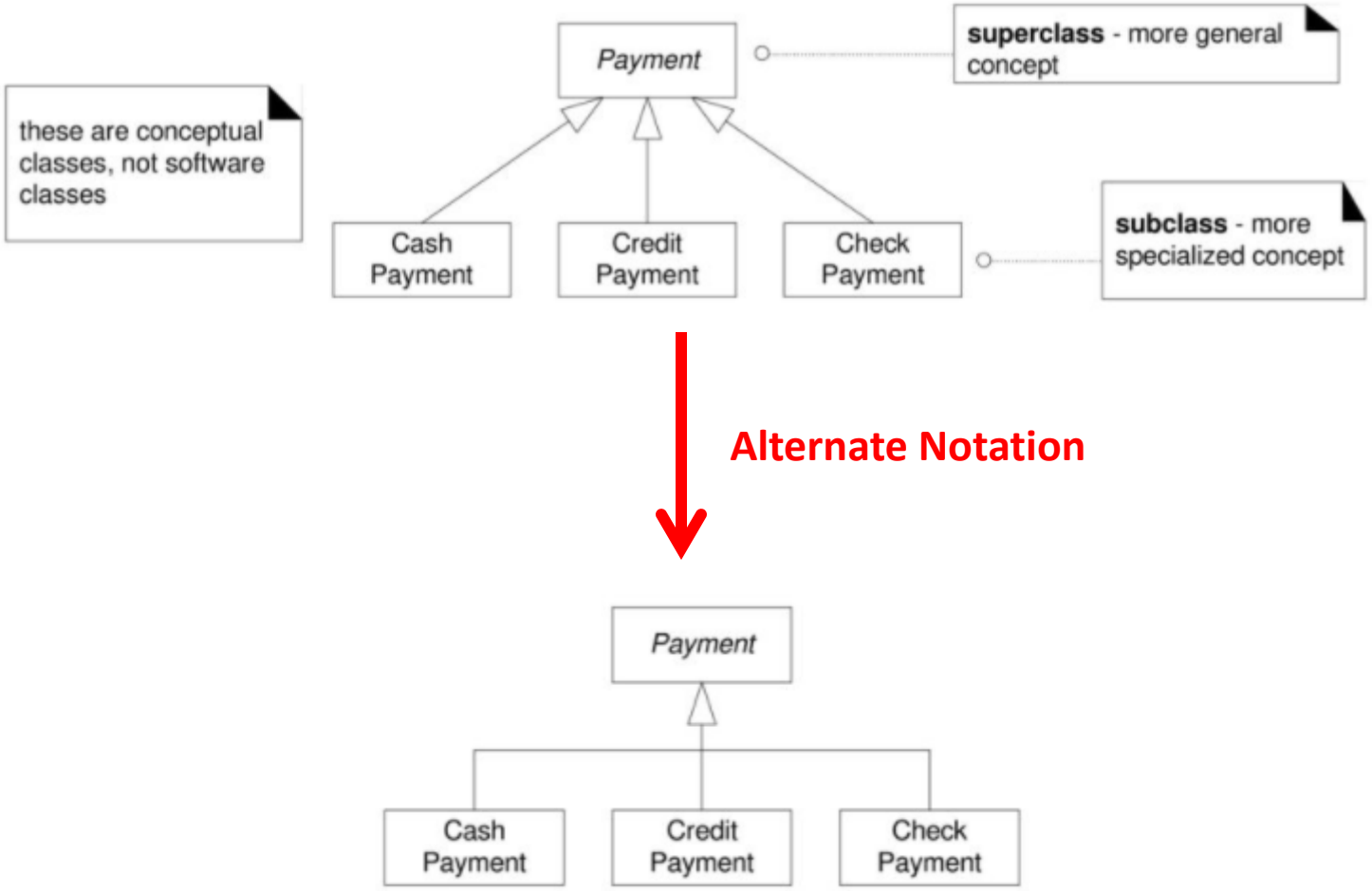
Activity Diagram (Exercise)…

- **Ticket Vending Machine**
  - Activity is started by Commuter actor who needs to buy a ticket. Ticket vending machine will request trip information from Commuter. This information will include number and type of ticket, eg whether it is a monthly pass, one way or round ticket, route number, destination number
  - Based on these information, ticket vending machine will calculate payment due and request payment option (cash, card).
  - If card, Bank will participate in this activity

Domain Modeling Refinement

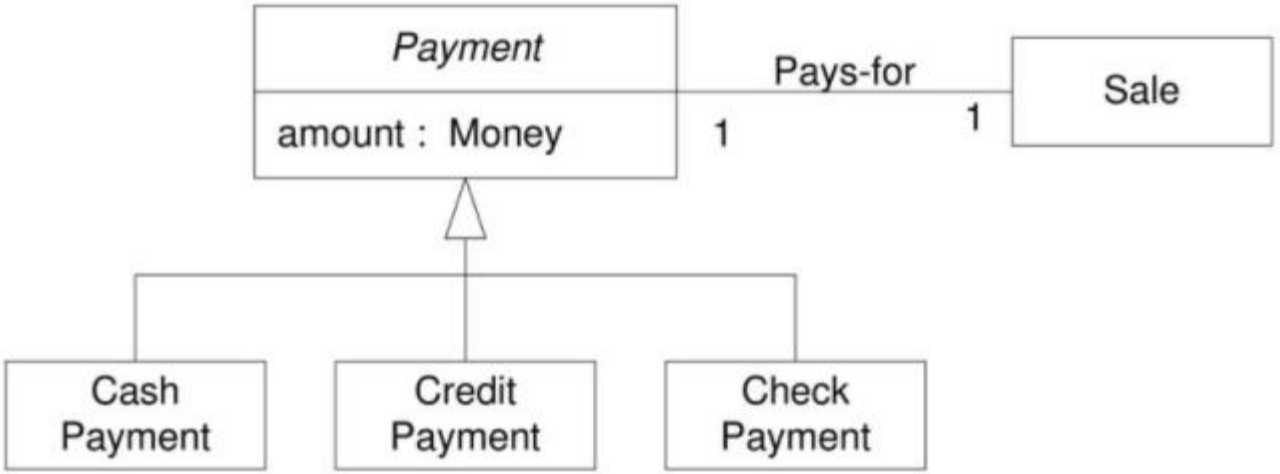
- Notations extensions

Generalization in Domain Model

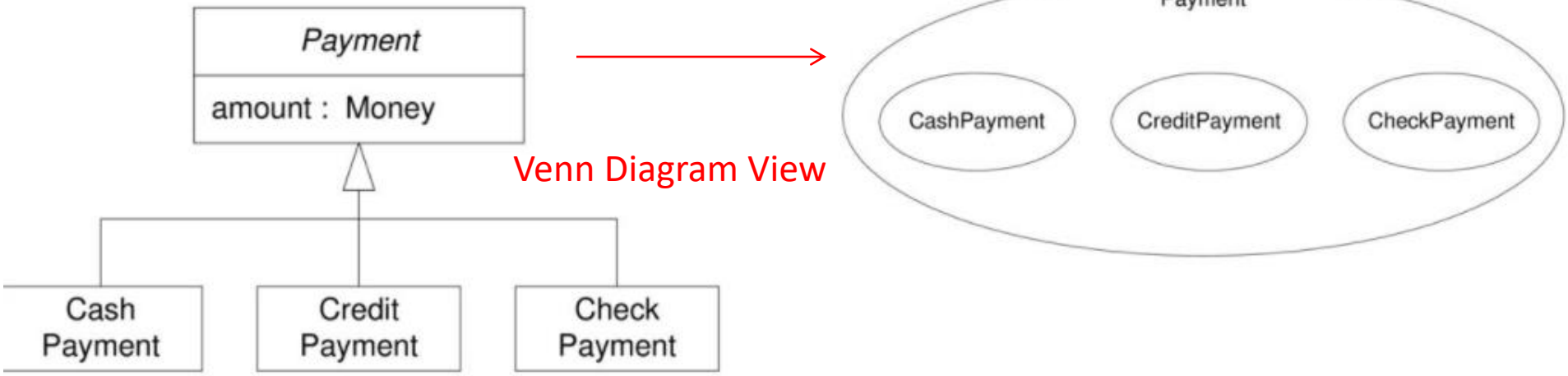


Conceptual Superclasses and Subclasses

- All members of sub class must be members of superclass
- 100% superclass definition shall apply to the subclass (100% rule)
  - Attribute and associations
- Subclass “is – a” superclass
  - Employee is a person

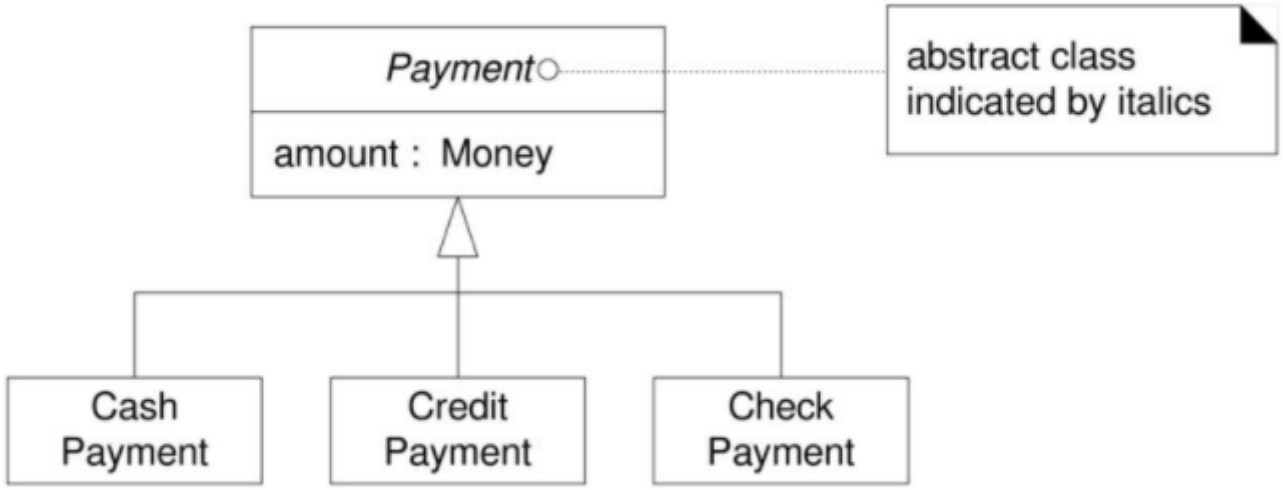


Conceptual Superclasses and Subclasses…



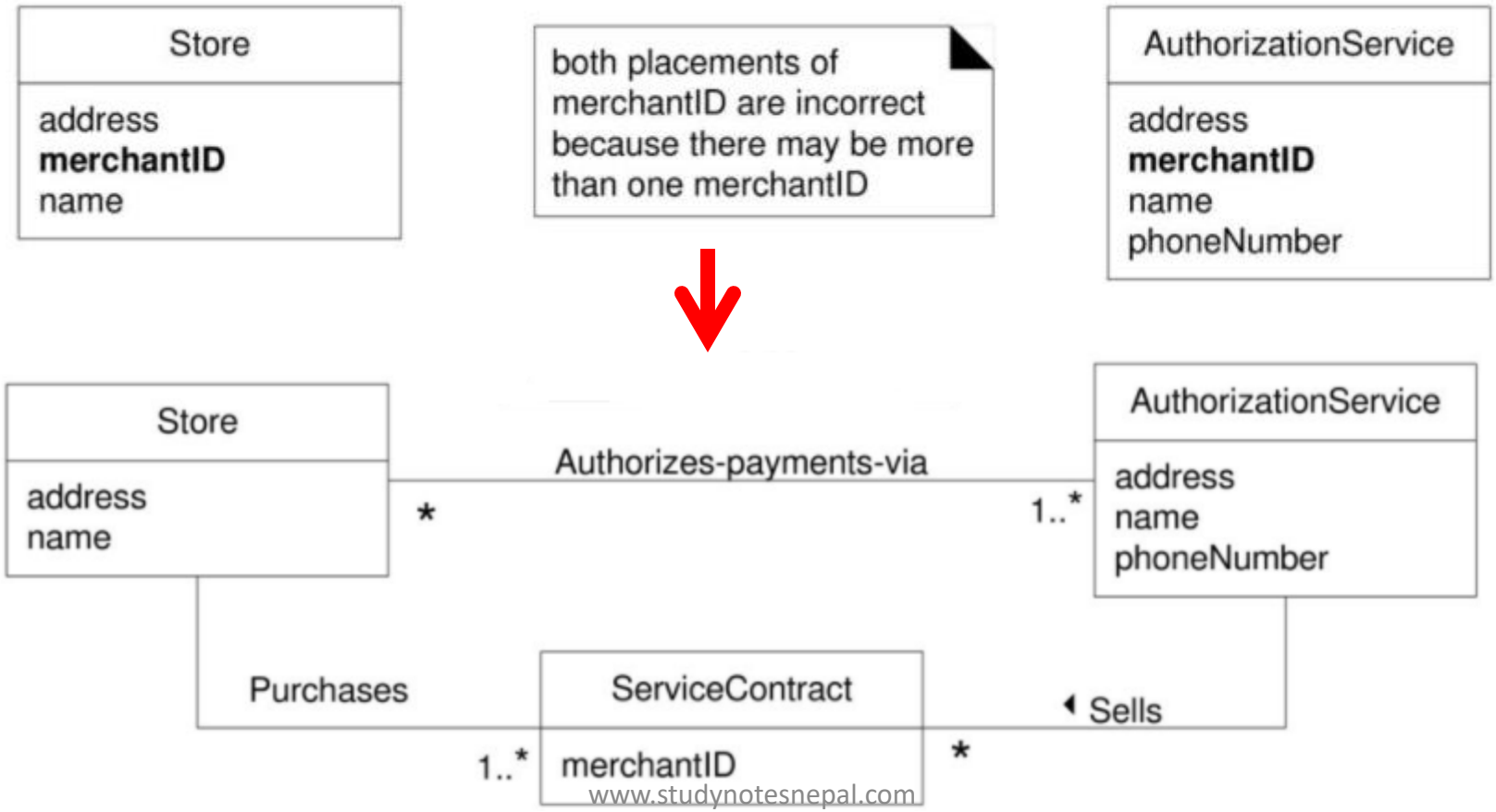
No other types of payment is possible on this domain

Abstract Class Notation in UML

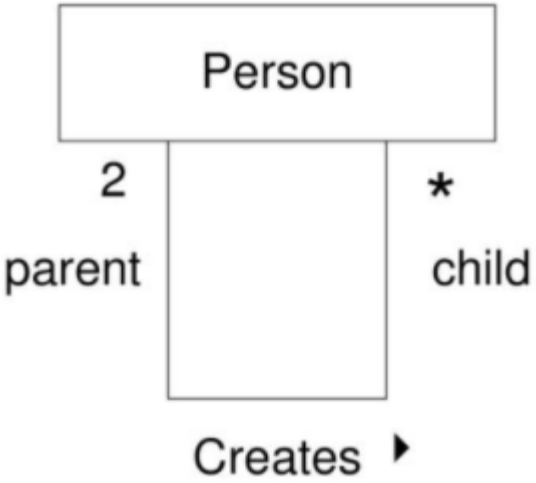


Many to Many Associations

- If a class C can simultaneously have multiple values for attribute A, put these in separate class



Reflexive Associations



End of Session