UNIT –I

Introduction to OOAD – What is OOAD? – What is UML? What are the Unified process(UP) phases - Case study – the NextGen POS system, Inception -Use case Modeling - Relating Use cases – include, extend and generalization

1.1 What is Analysis and Design?

Analysis: "Do the right thing"-it emphasis an investigation of the problem and requirements, rather than solution. There are 2 types

:a) Requirement analysis
 b) Object oriented analysis.

Design: "Do the thing right"-it emphasis a conceptual solution (in software & hardware) that fulfill the requirement rather than its implementation.

1.2 What is Object Oriented Analysis & Design (OOAD)?

Object Oriented Analysis (OOA): During OOA there is an emphasis of finding and describing objects or concepts in the problem domain. E.g.: Book Bank is the problem domain. During OOA the concept lie book category, book medium, book detail can be found and described.

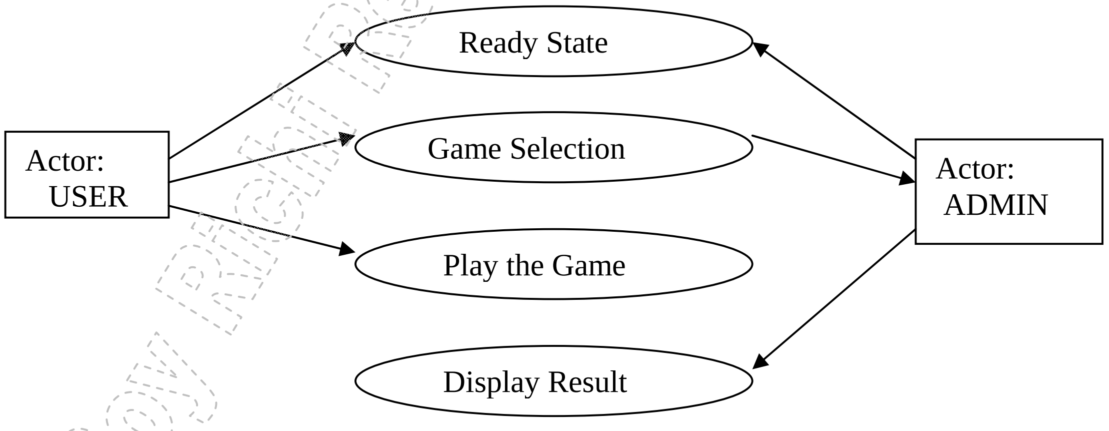
Object Oriented Design (OOD): During OOD there is an emphasis on designing software objects, and how they collaborate to fulfill the requirements .i.e. in this we can discuss about attributes & methods. For the book bank domain, the attributes & methods are title, book id, author, publication, year of publication, addBook(), display(), deleteBook().

1.3 An Example

a) Defining Use Cases:

1. Use case deals with analysis.
2. Use case is the interaction between the user and the system.
3. Use case or scenario for understanding system requirements.
4. These are simply written stories.
5. These are not an object oriented artifact.
6. Requirement analysis may include stories or scenarios of "How people use the application".
7. These are described as text, easy to read with a clear flow of events to follow.
8. Use case model captures the goal of the user & the responsibility of the system to its user.

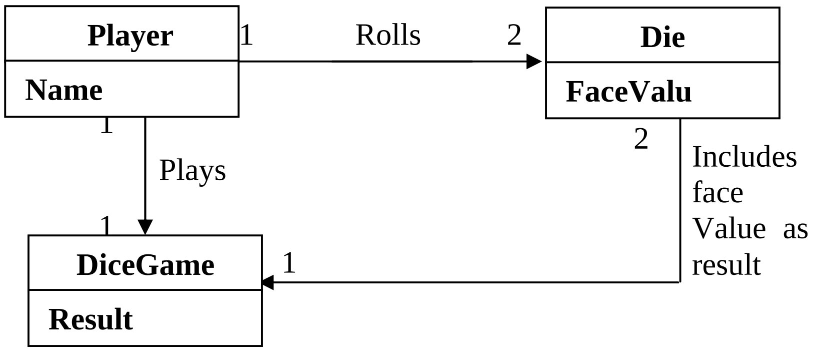
E.g. " Use case diagram for die game"



b) Domain Model: (Conceptual Object Model)

1. OOA is concerned with the description of the domain from perspective of the objects.
2. Identification of the concepts, attributes and associations that are considered noteworthy(significant).
3. The result can be expressed in a domain model which shows the noteworthy domain concepts or objects.
4. This model illustrates the important concepts, player, die, & die game, with an association & attributes. It is a visualization of concepts. This is also called as conceptual object model.

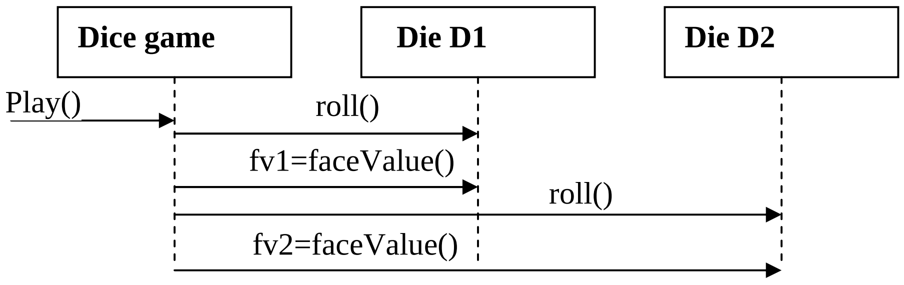
E.g.: Die game



c) Assign Object Responsibilities and Draw Interaction Diagram(Sequence diagram):

1. OOD is concerned with defining the software objects (i.e.) their responsibilities and colorations.
2. The common notation to illustrate these coloration is the sequence diagram(a kind of UML interaction diagram).
3. This diagram shows the flow of messages between software objects by invoking a method.

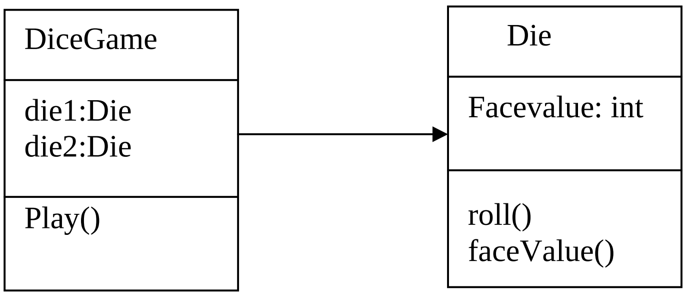
E.g.: Dice game



d) Design Class Diagram:

1. The static view of class definition is usefully shown with the design class diagram.
2. This diagram illustrates the attributes & methods of the classes.
3. Design class diagram is not same as the domain model, but some class names & contents are similar.

E.g.: Dice Game



1.4 Unified Modeling Language(UML):

- These languages for specifying, construction, visualization and documenting the software system and its components.
- It is a graphic language with set of rules and semantics.
- The rules and semantics of the models are expressed in English or in the form known as object constraint language.
- Model Driven Architecture & UML meta model describes the semantics of the modeling elements.

1.4.1 Ways to apply UML:

There are 3 ways.

- **UML as Sketch**
- **UML as Blueprint**
- **UML as Programming Language**

i) UML as Sketch:

Informal & incomplete diagrams are created to exploit the difficult parts of the problems or domain phase, by exploiting the power of visual languages.

ii) UML as Blueprint:

Detailed design diagram to domain space for either for

a) Code Generation(Forward Engineering):

Before programming detailed diagrams can provide diagrams for code generation either manually or automatically with the tools. When diagrams are used for some code and other codes is develop by the programmer by coding.

b) Reverse Engineering (to visualize & better understanding the existing code in the UML diagram)

A UML tool reads the source coding or binary's and generates UML packages class & sequential diagrams.

These blueprints can help the reader to understand the pictures elements, structure & collaborations.

iii) UML as Programming Languages:

The programmer works in UML programming language can generate the executable code automatically without stating the diagram it is not possible to modify the coding. UML requires to practical way to diagram all the behaviors or logic using interaction or state chart diagram.

UML as programming language is under development stage, in terms of theory, tool robustness and reusability.

Agile Modeling:

This emphasis the UML as sketch. This is the common way to apply the UML & the investment time is short.

1.4.2 Three perspective to apply UML:

1. Conceptual (Theoretical) perspective
2. Specification (software) perspective
3. Implementation(software) perspective

- **Conceptual Perspective:**

  Diagrams are interpreter as describing things in the situation of the real world or domain of interest.

  It is a raw UML class diagram notation used to visualize the real world concepts.
- **Specification Perspective:**

  Using the same notation as in the conceptual perspective, draw the diagrams which describe software abstractions or components with specification & interfaces, which do not depend on the particular language.

  It is a raw UML class diagram notation used to visualize the software elements.
- **Implementation Perspective:**

  Diagram describes software implementation in a particular technology (i.e.) languages such as java…

1.4.3 Meaning of "Class" in Different Perspective:

In raw UML the rectangular boxes are called classes which comprises variety of phenomenon such as Physical Things, Software things, abstract concepts, events etc.,

In raw UML, alternative terminology superimposes.

E.x.

In Unified Process (UP), when UML boxes are drawn in the domain model, they are called *domain concepts or conceptual class*. The domain module shows the *conceptual perspective*.

In the UP, when UML boxes are drawn in the design model, they are called *design classes*. The design model shows the specification or implementation perspective which is desired by the modeler.

In order to keep things clear, the class related terms are discussed as follows,

a) Conceptual class:

It refers real world concepts or things for a domain. The UP domain model contains conceptual classes, on a conceptual perspective.

b) Software Class:

A class representing a specification or implementation perspective of a software component.

c) Implementation Class:

A class implemented in a specific object oriented language, such as C++, JAVA, C#, Smalltalk.

1.5 What is the Unified Process(UP)?

- A Software development process describes an approach to building, deploying and maintaining the software.
- **The UP is a popular *iterative software development process* for building** Object Oriented System.
- The rational UP is a detailed refinement of the UP, which has been often adopted.
- UP is *common* & promotes widely recognize the best practices.
- It is useful for IT professionals to know it & the students who are entering the workforce to be aware of it.
- UP is very flexible & open and encourages skillful practices from other iterative methods such as Extreme Programming (XP), Scrum etc.

E.g.: XP's test drive & development, refractory & continuous integration practices can be fit within an UP project.

- Introducing UP is not mean to downplay the value of these methods and it is quite opposite.
- UP encourages the clients to understand and adopt the blend of useful techniques from several iterative methods and not making the mentality of ***"my method is better than your method".***
- UP combines commonly adopted best practices such as an iterative lifecycle and risk-driven development.

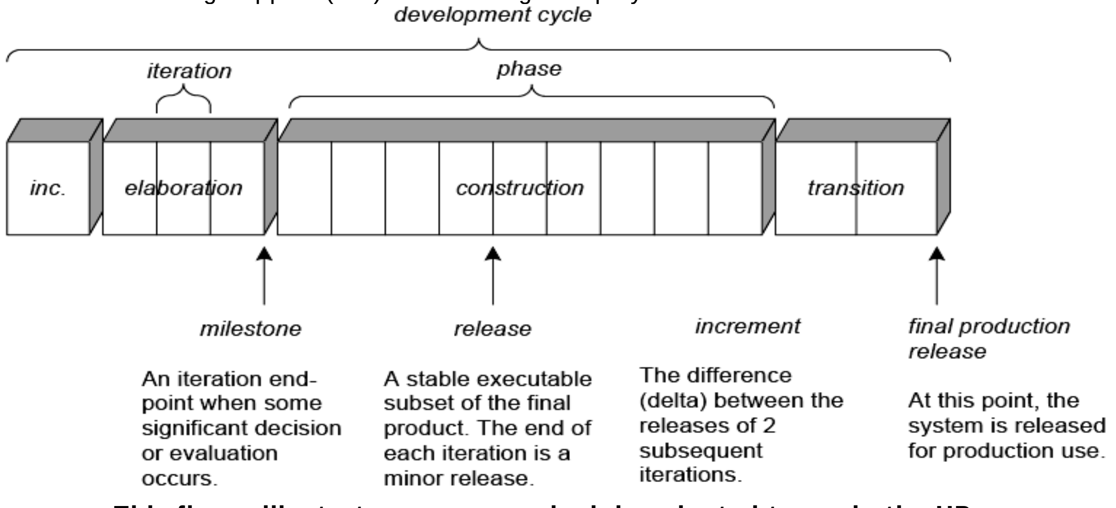
1.5.1 Why UP is used?

- UP is an iterative process for building OOAD projects.
- UP practices provide an example *structure* for how to do & thus how to explain OOAD.
- UP is flexible and can be applied in a light weight which includes best practices from other agile methods, such as XP or scrum.

1.5.2 What are the UP Phases?

The UP project organizes the work & iterations across the four major phases:

1. **Inception**:
   - Approximate vision
   - Business case
   - Scope
   - Wage estimate
2. **Elaboration**:
   - Refined vision
   - Iterative implementation of the code architecture.
   - Resolution of high risks.
   - Identification of most requirements & scope.
   - Most realistic estimate.
3. **Construction**:
   - Iterative implementation of the remaining lower risk & easier elements and preparation for deployment.
4. **Transition:**
   - Testing happens(i.e.) Beta testing & deployment.


development cycle
iteration
phase
inc.
elaboration
construction
transition
milestone
release
increment
final production release
An iteration end-point when some significant decision or evaluation occurs.
A stable executable subset of the final product. The end of each iteration is a minor release.
The difference (delta) between the releases of 2 subsequent iterations.
At this point, the system is released for production use.

This figure illustrates common schedule-oriented terms in the UP

Inception is not a requirement phase rather it is a feasibility phase, where enough investigation is done to support the decision to continue or stop.

Elaboration is also not the requirement phase or design phase rather it is a phase where, core architecture is iteratively implemented & high risk issues are mitigated.

The development cycle composed of many iterations, and it is ends in the release of a system in to production.

1.5.3 What are the UP Disciplines?

UP describes work activities such as writing a use case within the disciplines. Disciplines are a set of activities & related artifacts within requirement analysis.

Artifact:

In UP, Artifact is a general term for any work product such as code, web graphics, text document, database schema, diagram, models etc.

Types of Disciplines:

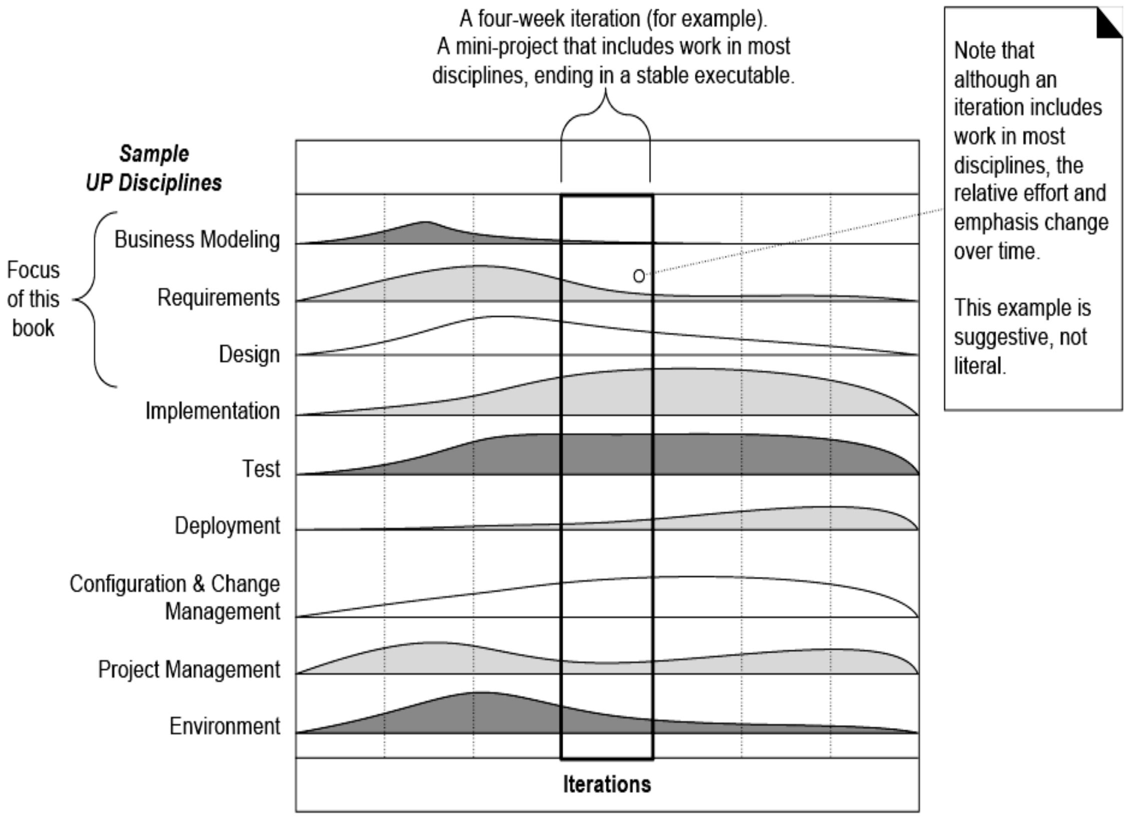
There are **9** disciplines in UP, they are,

- Business Modeling
- Requirements
- Design
- Implementation
- Test
- Deployment
- Configuration & change management
- Project management
- Environment

Among which three are discussed below for some of artifact.

1. **Business Modeling:**
   The domain model artifact, to visualize noteworthy concepts in the application domain.
2. **Requirements:**
   Use case model, vision, glossary specification supplementary specification are used to capture the fundamental & non-functional requirement.
3. **Design:**
   The design model artifact, to design the software objects.

Figure 1

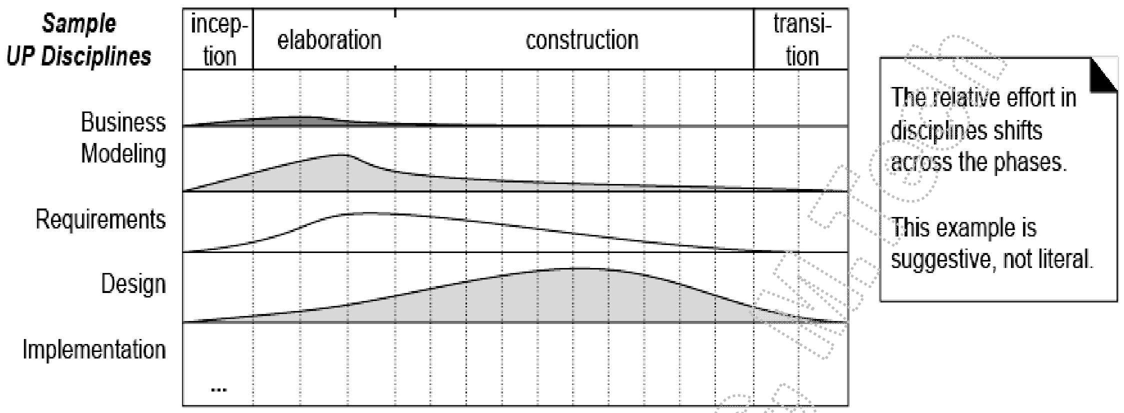


Implementation - It means programming and building the system, not deploying it.

Environment - This discipline refers to establishing the tools & customizing the process for this project (i.e.) setting up the tool & process environment.

1.5.4 What is the relationship between disciplines & phases in UP?

Figure 2



- From Figure 1, pointed out that during one iteration, the work goes on in all disciplines and the relative effort across those disciplines changes over time.
- Early iteration tend to apply greater relative emphasis to requirements and design, and later goes less, as the requirements and core design stabilize through a process of feedback, discussion and adaptation.
- This figure 2 illustrates that UP disciplines changing relative effort, with respect to UP phases. These relations are suggestive & not literal.
- For E.x.: In elaboration, the iterations tend to have a relatively high level of requirements and design work, as well as some implementation also.
- During construction, the emphasis is heavier on implementation and lighter on requirement analysis.

1.5.5 How to customize the Process in UP development Phase?

- The artifacts or practices in the UP are optional.
- Some UP practices & principles are invariant, such as iterative & risk driven development and continuous verification of validity.

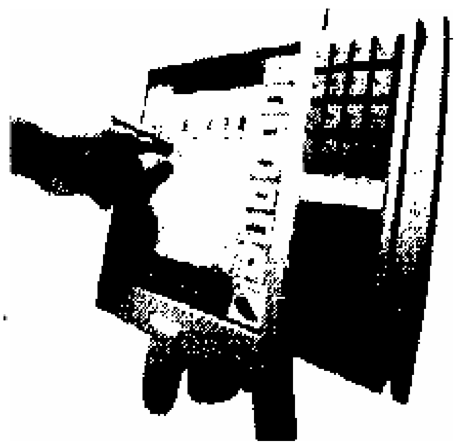
Figure 3 **The table shows the development case for the "NextGen Project"**

Discipline	Routine	Artifact	Inception	Elaboration	Construction	Transition
Business modeling	Agile modeling. Requirement workshop	Domain model		S		
Requirement	Requirement workshop, vision box exercise, dot voting	Use case model vision, Glossary supplementary specification	S S S	R R R		
Design	Agile modeling, test drive device	Design model, software architecture, Data model		S S S	R R R	
Implementation	::::::	::::::::::			:	: :

What is the Development case? The choice of practices and UP artifact for a project may be return in a short document in the development case. Figure 3 refers the development case for the next Gen System.

1.5.6 Case Study: "NextGen POS System"

- It is NextGen Point-Of-Sale System, which was developed by the object technologies. A "POS" system is a computerized application used to record sales and handle payments. It is used in retail store
- . It includes hardware component, such as computer and bar code scanner, printer and software to run the system.



- This system includes the interfaces to make various service application such as providing bill, third party tax calculator & inventory control.
- This system must be fault-tolerant (i.e.) even if remote services are temporarily unavailable ex. Inventory control, they must still having the capability to capture the sales & make the payment. So that the business is not crippled.
- A POS System must support multiple and varied client-side terminals and interfaces. E.g.: A thin client web browser terminal, touch screen input ,a regular personal computer with a JAVA swing GUI, wireless PDA's etc.
- A created commercial POS system will sell to the different clients with disparate (dissimilar) needs in terms of Business rule processing.
- Each client want a unique set of logic, to execute at certain predictable points in the scenario's of using the system, such as when a new sale is initiated (or) when a new line item is added.
- Therefore, we will need a mechanism to provide this flexibility & customization.
- By iterative development strategy, we can proceed through requirements, object oriented analysis, design and implementation.

1.6 INCEPTION

- Inception means envision(imagine, predict) the product scope, vision and business case. It is the initial short step to establish a common vision and basic scope for the project.
- It will include analysis of the use cases & critical non-functional requirements, creation of a business case, and preparation of the development environment. So that programming can be start in the following elaboration phase.
- In inception phase of a UP Project the following questions are explored:
    (i) What is the vision & business case for this project?
    (ii) Feasible?
    (iii) Buy/Build or both?
    (iv) Rough unreliable range of cost?
    (v) Should we proceed or stop?
- Inception is to do enough investigation to form a rational & justifiable opinion about the feasibility of the new system.
- The purpose of inception case is not to define all the requirements or create believable estimate or plans, which will happen during elaboration phase.
- In inception phase, the most requirement analysis will occur in parallel with early production & testing.
- Thus, inception phase should be relatively short for most projects,(i.e.) 1 or few weeks long.

1.6.1 How long is inception?

- ❖ The aim of the inception is to establish some initial common vision and business case for the projects.
- ❖ Determine if it is feasible.
- ❖ Decide if it is worth, some serious investigation in elaboration, also it includes first requirement workshop, planning for the first iteration, and then quickly move on to elaboration phase.
- ❖ If it has been decided in advanced that the project will definitely be done, and it is clearly feasible, then the inception phase will be brief.

1.6.2 What Artifact may start in Inception? – Sample Inception artifacts

1. **Vision & Business Case**:
   - It describes high level goals & constraints, business case and provides how to execute it.
2. **Use-case Model**:
   - It describes the functional requirements.
   - During inception, the names of most use cases will be identified and 10% of the use-cases will be analysed in detail.
3. **Supplementary Specification**:
   - This describes non-functional requirements, which having the major impact on architecture.
4. **Glossary:**
   - Key domain terminology & data dictionary.
5. **Risk List & Risk Management Plan:**
   - It describes the risk & ideas for their mitigation(reduction, improve).
   - Risk List :
     1. Technical
     2. Business
     2. Resources
     3. Schedule(plan)
6. **Prototypes & Proof-of-concepts:**
   - To clarify the vision & validate the technical ideas.
7. **Iteration Plan:**
   - It describes what to do in the first elaboration iteration.
8. **Phase plan & software development plan:**
   - It describes low precision guess for elaboration. Discussion about software tools, people, education and other resources.
9. **Development Phase:**
   - It gives description for the customized UP steps & artifacts.

- ❖ These artifacts are partially completed in inception phase & they will be iteratively refined in sub-sequent iterations.
- ❖ In inception, the investigation & artifact content should be light.

For E.g.: Use case model artifact may list all the names of the use cases expected & actors, but describes only 10% of the use case in detail.

1.6.3 When you didn't understand Inception?

1. There is an attempt to define all the requirements.
2. Estimate or plans are expected to be reliable.
3. If you define the architecture(must be done in iterative method).
4. There is no business case or vision artifact.
5. All the use-cases were written in detail.
6. None of the use-cases where written in detail.
7. It is more than "a few" weeks long.

1.6.4 How much UML during inception?

1. The purpose of inception is to collect enough information
   - ❖ to establish a common vision.
   - ❖ Decide if moving forward is feasible.
   - ❖ If the project is worth some

2. Possibly simple UML use-case diagrams are drawn & not much diagramming is warranted.
3. In practice most UML diagramming will occur in the next phase elaboration.
4. In inception, there is more focus on understanding the basic scope and 10% the requirement are expressed mostly in text form.

1.7 Use-cases:

Use-case concept was introduced by Ivar Jacobson.

1. Use cases are text stories, used to discover and record requirements.
2. The use-case corresponds to a sequence of transaction, in which each transaction is invoked from outside the system, and engages internal objects to interact with one-another and with the systems surrounding.
3. Use-case defines what happens in the system, when the use-case is performed.
4. Essence the use-case model defines the outside (actors) and inside (use-case) of the system behavior.
5. The use cases are initiated by the actors & describe the flow of events that these actors set-off.

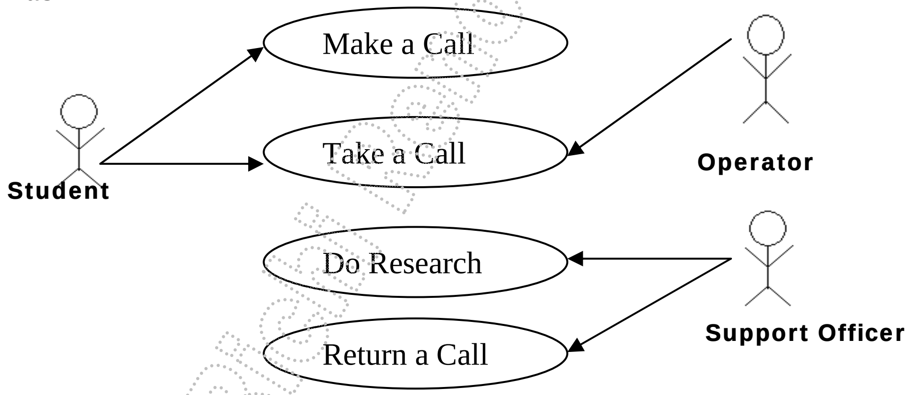
Actors: An actor is something with behavior, such as a person (identified by role), computer system, or organization; for example, a cashier.

Scenarios: A scenario is a specific sequence of actions and interactions between actors and the system under discussion; it is also called a **use case instance.**

1.7.1 Use case Diagram:

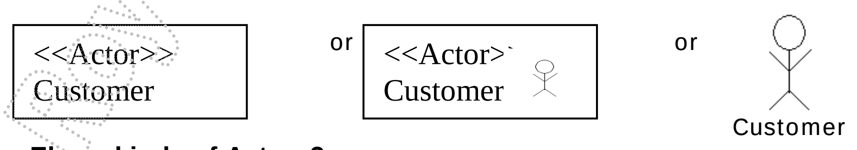
- This shows the relationship among the actors & use-case within a system.
- The use case diagram is a graph of actors, set of use cases enclosed by a system boundary, communication.
- Communication, association between the actors & the use cases, and generalization among the use case.
- Use-case is shown as an ellipse containing the name of the use case.
- The name of the use case can be placed below or the inside the ellipse.

E.g.: Help Task



- An Actor is shown as a class rectangle with the label<<actor>> or the label & the stick figure or stick figure only, with the name of the actor below the figure.

E.g.:

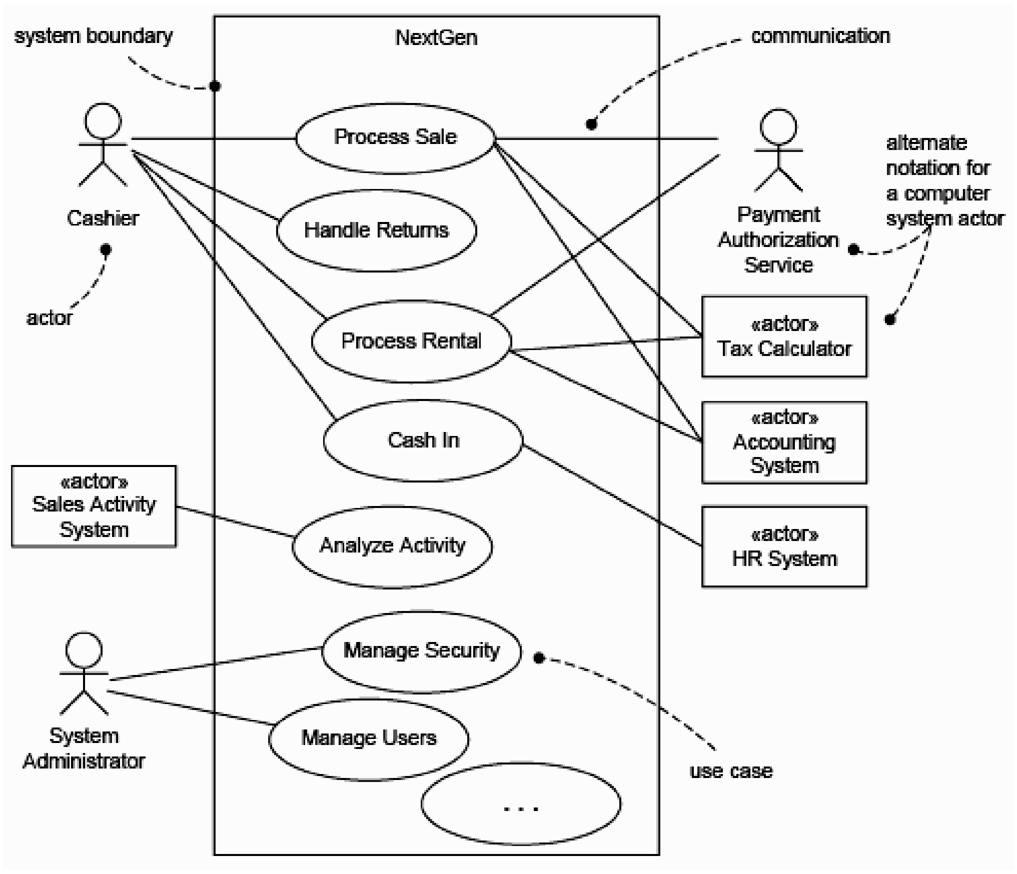


## What are the Three kinds of Actors?

1. Primary actor has user goals fulfilled through using services of the SuD(System under discussion). For example, the cashier.

2. Supporting actor provides a service (for example, information) to the SuD.
The automated payment authorization service is an example. Often a computer system, but could be an organization or person.

3. Offstage actor has an interest in the behavior of the use case, but is not primary or supporting; for example, a government tax agency.



## What are the Three Common Use Case Format?

Use cases are written in different formats, depending on need. In addition to the black-box versus white-box *visibility* type, use cases are written in varying degrees *of formalit*

- **brief**—terse one-paragraph summary, usually of the main success scenario. The prior *Process Sale* example was brief.
- **casual**—informal paragraph format. Multiple paragraphs that cover various scenarios. The prior *Handle Returns* example was casual.
- **fully dressed**—the most elaborate. All steps and variations are written in detail, and there are supporting sections, such as preconditions and success guarantees.

## How to find Use Cases?

Use cases are defined to satisfy the user goals of the primary actors. Hence, the basic procedure to find the use case is:

1. Choose the system boundary. Is it just a software application, the hardware and application as a unit, that plus a person using it, or an entire organization?
2. Identify the primary actors - those that have user goals fulfilled through using services of the system.
3. Identify the goals for each primary actor.
4. Define use cases that satisfy user goals; name them according to their goal.

1.7.2 Use Case Modeling:

1. Use cases are text document and not diagrams, where use case modeling is primarily an act of writing texts and not drawing diagrams.
2. Use case model is the model of the system functionalities & environment.

3. It is not only the requirement artifact in UP, but it provides useful things for requirement analysis like supplementary specification, glossary, vision & business rules.

1.7.3 Use Case Relationship:

i) Communication:

- An Association relationship may exists between an actor and a use case is referred as a communicate association, since it represents a communication between an actor and a use case.
- An association may be navigable in both direction.(i.e.) actor to use case & use case to actor or it may be navigable in only one direction(i.e.) actor to use case or use case to actor.
- The navigation direction of an association represents who is initiating the communication.

ii) Between Use cases:

- There are two types of relationship that may exist between use cases.

a) Include Relationship:

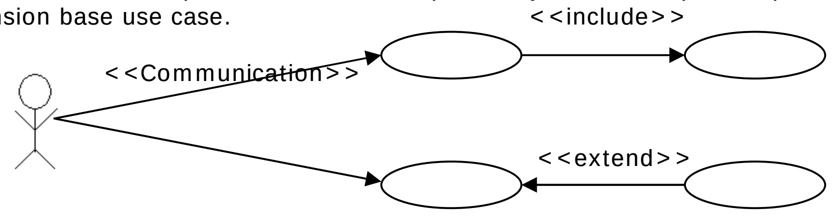
- o Multiple use case may share the same functionality. This functionality is placed in a new separate use case rather than documenting it in every use case that needs it.
- o Include relationship are created between this new use case and any other use case that "USES" its functionality.
- o An include relationship is drawn as a dependency relationship that points from the base use case to the used use case.

b) Extend Relationship:

An Extend relationship is used to show

I. optional behavior
II. behavior (i.e.) run only under certain conditions. E.g. Triggering an alarm in a bank
III. several different flows that may be run based on actor selection.

- o An Extend relationship is drawn as a dependency relationship that points from the extension base use case.



iii) Stereotype:

1. In UML, a new concept ,"Stereotype" are involved, which provides the capability of extending the basic modeling elements to create new elements
2. Stereotype names used to create the needed use case relationship.
3. These names are included between guillemets (<< >>) and placed along the relationship line.
4. The Stereotype<<communication>> may be added to an association to show that the association is a communicate association. This is optional one, since this type of association is between the actor and a use case.
5. Include & extend relationship must use Stereotype, since they are both represented by a dependency relationship.

1.8 UML DIAGRAMS

1. Class Diagram(static)
2. Use case Diagram
3. Behaviour Diagram(dynamic)
   a. Interaction Diagram
      i. Sequence Diagram
      ii. Collaboration Diagram
   b. State Chart Diagram
   c. Activity Diagram
4. Implementation Diagram
   a. Component Diagram
   b. Deployment Diagram

1.8.1 Class Diagram:

It is a collection of *Static modeling elements* such as classes, interfaces, collaborations & their relationship, connected as a graph to each other and to their contents.

a. Class Notations:

A class is drawn as a rectangle with 3 components separately by horizontal lines. The top main compartment holds the class name, the middle compartment holds the attributes & the bottom compartment having the list of operation.

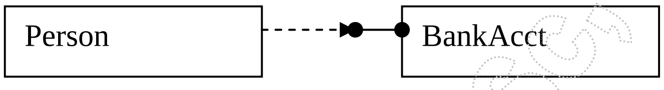
A stylish conversion of UML is to use an Italic font for an ***abstract classes*** & Roman font for concrete class.

b. Object Diagram:

A static object diagram is an instance of a class diagram. The notation for object diagram & class diagram are same. The class diagram can contains object, so class diagram with objects & no classes is an object diagram.

c. Class Interface Diagram:

It is used to describe the externally visible behavior of a class.(i.e.) an operation with public visibility.



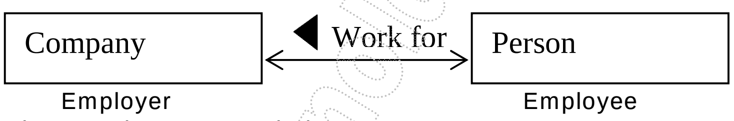
Identifying class interfaces is an design activity. The UML notation for an interface is a small circle with the name of the interface connected to the class.

A class that requires the operation in the interface may be attached to the circle by a [hashed arrow].-------▶

d. Binary Association Notation:

Binary association is drawn as a *solid path connecting two classes* (or) both ends may be connected to the same class.

E.g.

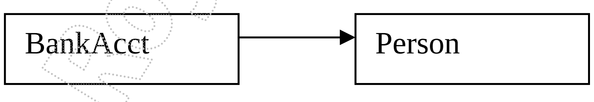


1. An association may have a association name.
2. Association name may have a optional black triangle in it.
3. The point of the triangle indicating the direction in which to read the name.
4. The association technically refers the binary association which is drawn as a solid line connecting two class symbols.

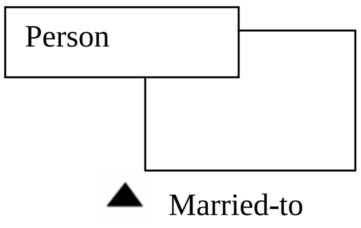
e. Association Role:

1. A simple association, the technical term for it is binary association, which is drawn as a solid line connecting two class symbols.
2. The end of an association, where it connects to a class, shows the association role.
3. The association role is part of the association and not a part of the class.
4. Each association has two or more roles to which it is connected.
5. The association is navigable in one direction which is indicated by an arrow.
6. An arrow may be attached to neither, one or both ends of the path.
7. Arrows could be shown whenever the navigation is supported in given direction.

E.g.1 Association is navigable in below example from **Bankaccount to person** & not the reverse.

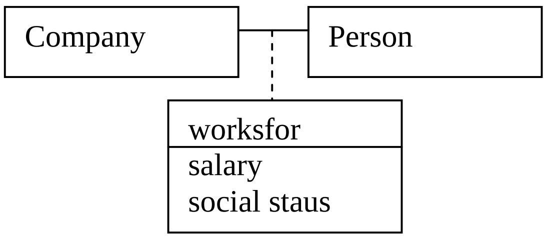


E.g.2

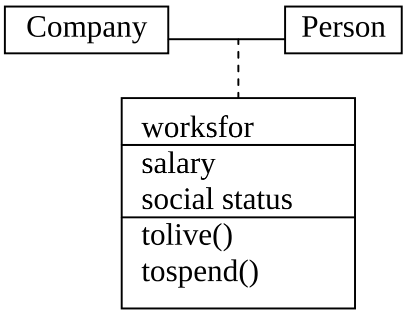


e. Association Class:

1. An association class is an association that also has class properties.



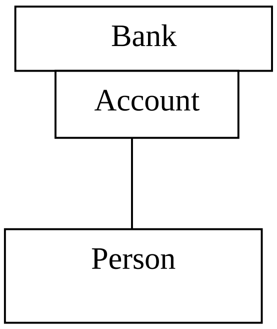
2. An association class is shown as a class symbol, attached by a dashed line, to an association path.



3. The name in the class symbol and the name string attached to the association path are the same. (i.e.) the name shown on the path.
4. If an association class has attributes & no operations, it emphasis its “Association Nature”.
5. If an association class has operations and attributes, then the name is placed in the class rectangle to emphasis its “Class Nature”.

e. Qualifier(Association attributes):

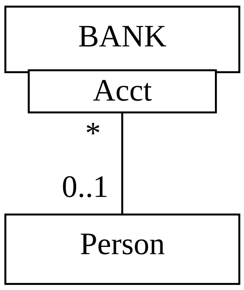
1. Qualifier is an Association attribute
   E.g.



   Person object may be associated with the **Bank** object. An attribute of this association is the account. The account is the qualifier of this association.
2. The UML notation of qualifier is shown as the small rectangle than the class rectangle attached to the end of an association path.
3. The Qualifier rectangle is part of the association path, and not part of the class.

f. Multiplicity:

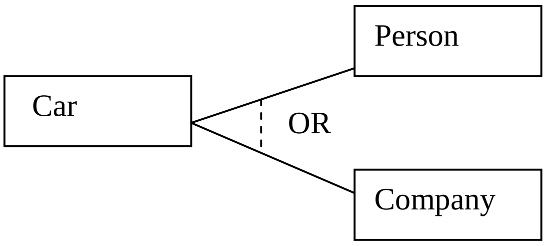
1. Multiplicity specifies the range of allowable associated class.
2. It is given for roles within associations, repetitions, parts within composition & other purposes.
3. The general format to represent the multiplicity is,
   Lower bound . . Upper bound
4. The terms lower bound and upper bounds are integer values specifying the range of integers.
5. “ * ” this symbol may be used for the upper bound to denote the unlimited upper bound.
   E.g.: (0..1),(0..7),(1..*)



g. OR Association:

1. An OR association indicates a situation in which only one of several associations may be instantiated at a time for any single object.
2. This is shown as a dashed line connecting two or more associations, all of which must have a class in common.

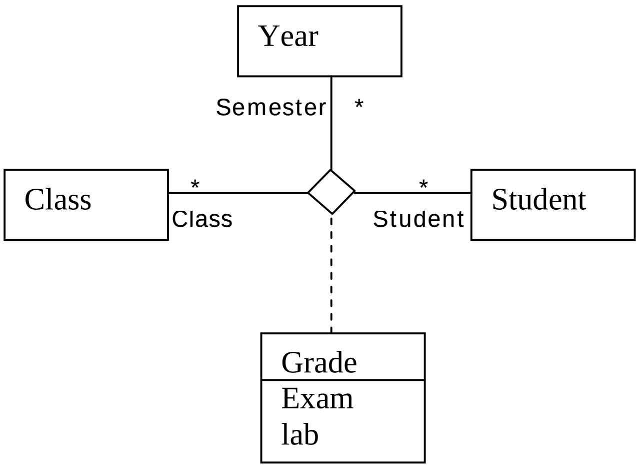
E.g.:



h. N-ary Association:

1. An N-ary association is an association among more than two classes.
2. An N-ary association is shown as a large diamond having paths to each participant class.

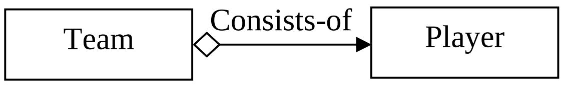
E.g.:



3. The role attachment may appear on each path as like a binary association.
4. Multiplicity may be indicated
5. Qualifiers & aggregation are not permitted.
6. An association class symbol may be attached to the diamond by a dashed line, indicating an N-ary association that has attributes, operation and associations.

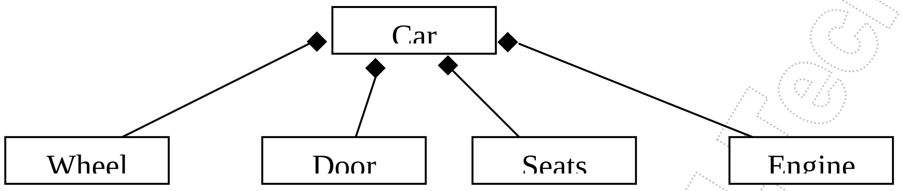
i. Aggregation

1. Form of association.
2. Hollow diamond is attached to the end of the path to indicate aggregation, but the diamond may not be attached to both end of the line.



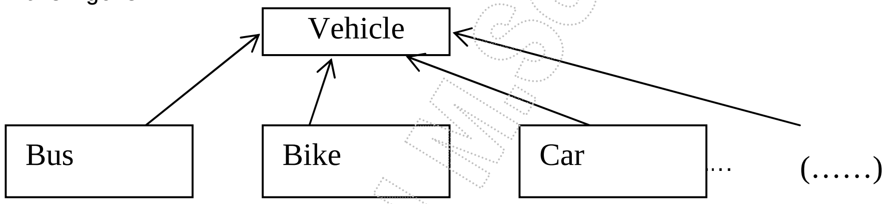
j. Composition:

1. Composition is a form of aggregation with strong ownership to represent the component of the complex object.
2. It is also referred as "a part of" whole relationship.
3. The UML notation is the solid diamond at the end of the path.



k. Generalization:

1. It is the relationship between the more general class & the more specific class.
2. Generalization is displayed as directed line, with a closed, hollow arrow head at the super class end. It is known as separate target style.
3. Ellipse(..) indicates that the generalization is incomplete and more sub-classes exist ,that are not shown in the figure.


Vehicle
Bus
Bike
Car
…
(……)