Object-Oriented Analysis and Design

Systems development is a systematic process that includes phases such as planning, analysis, design, deployment, and maintenance.

- Systems analysis
- Systems design

Systems Analysis

It is a process of collecting and interpreting facts, identifying the problems, and decomposition of a system into its components.

Systems Design

It is a process of planning a new business system or replacing an existing system by defining its components or modules to satisfy specific requirements.

System Analysis and Design (SAD) mainly focuses on −

- Systems
- Processes
- Technology

What is a System?

It is an organized relationship between any set of components to achieve some common cause or objective.

Constraints of a System

A system must have three basic constraints –

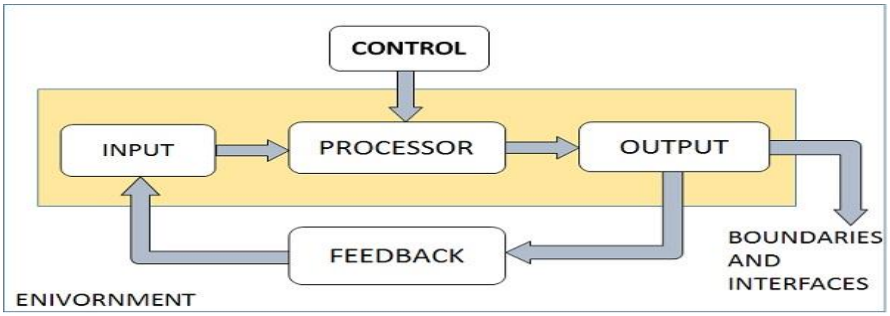
- A system must have some **structure and behavior** which is designed to achieve a predefined objective.
- **Interconnectivity and interdependence** must exist among the system components.
- The objectives of the organization have a higher priority than the objectives of its subsystems.

For example, traffic management systems, payroll systems, automatic library systems, and human resources information system.

Properties of a System

- Organization - Organization implies structure and order.
- Interaction - It is defined as the components operating with each other.
- Interdependence – It means how the components of a system depend on one another.
- Integration - It is concerned with how a system components are connected together.
- Central Objective - The objective of system must be central. It may be real or stated.

Elements of a System



Types of Systems

Physical or Abstract Systems

- Physical systems are tangible entities.
- It may be static or dynamic in nature.
  Eg;- Static - desks and chairs
  Dynamic - A programmed computer (programs, data, and applications can change according to the user's needs.)
- Abstract systems are non-physical entities or conceptual that may be formulas, representations, or models of a real system.

Open or Closed Systems

- An open system must interact with its environment.
- It receives inputs from and delivers outputs to the outside of the system.
- A closed system does not interact with its environment.
- It is isolated from environmental influences.
- A completely closed system is rare in reality.

Adaptive and Non-Adaptive System

- Adaptive System responds to the change in the environment in a way to improve their performance and to survive.
  Eg:- human beings, animals.
- Non Adaptive System is the system which does not respond to the environment.
  Eg:- machines.

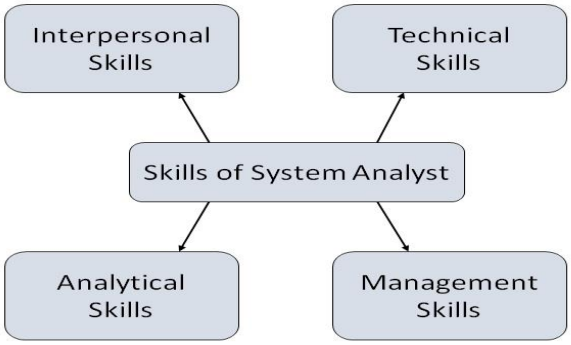
Permanent or Temporary System

- Permanent System persists for a long time.
  Eg:- business policies.
- Temporary System is made for a specified time and after that they are demolished.
  Eg:- A DJ system

Natural and Manufactured System

- Natural systems are created by the nature.
  Eg:- Solar system, seasonal system.
- Manufactured System is the man-made system.
  Eg:- Rockets, dams, trains.

Attributes of a Systems Analyst



Interpersonal Skills

- Interface with users and programmer.
- Facilitate groups and lead smaller teams.
- Managing expectations.
- Good understanding, communication, selling and teaching abilities.
- Motivator having the confidence to solve queries.

Analytical Skills

- System study and organizational knowledge
- Problem identification, problem analysis, and problem solving
- Sound commonsense
- Ability to access trade-off
- Curiosity to learn about new organization

Management Skills

- Understand users jargon and practices.
- Resource & project management.
- Change & risk management.
- Understand the management functions thoroughly.

Technical Skills

- Knowledge of computers and software.
- Keep abreast of modern development.
- Know of system design tools.
- Breadth knowledge about new technologies.

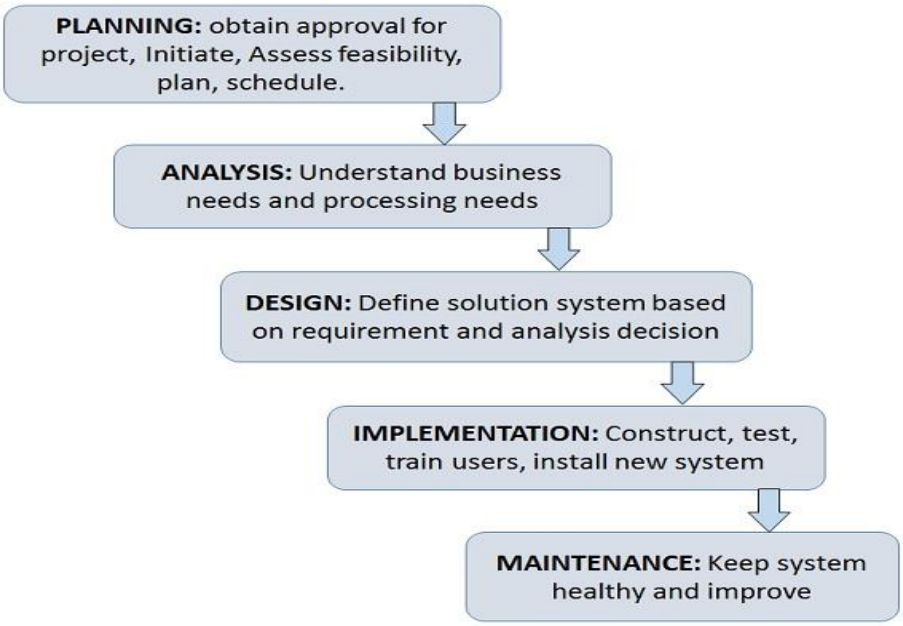
System Development Life Cycle (SDLC)

System Development Life Cycle (SDLC) is a conceptual model which includes policies and procedures for developing or altering systems throughout their life cycles.

1) Feasibility Study / Planning
2) Requirements Capture and Analysis
3) Design
4) Implementation
5) Testing
6) Maintenance

Phases of SDLC

Systems Development Life Cycle is a systematic approach

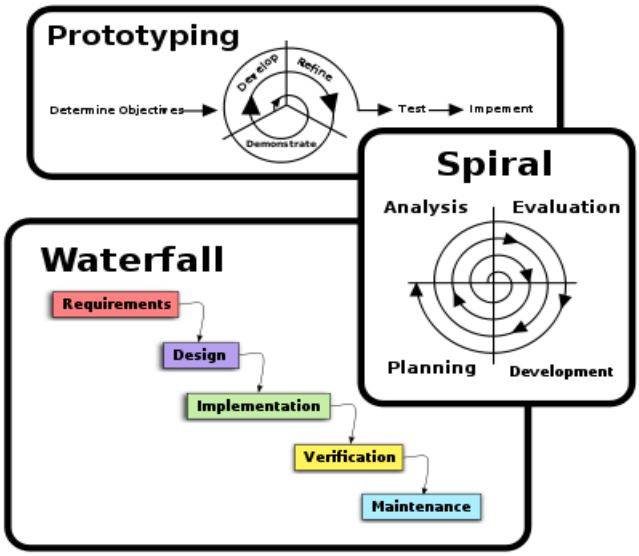


Software Process Models

A software process model is a standardized format for planning, organizing, and running a development project.

A model will define the following:

- The tasks to be performed
- The input and output of each task
- The pre and post-conditions for each task
- The flow and sequence of each task



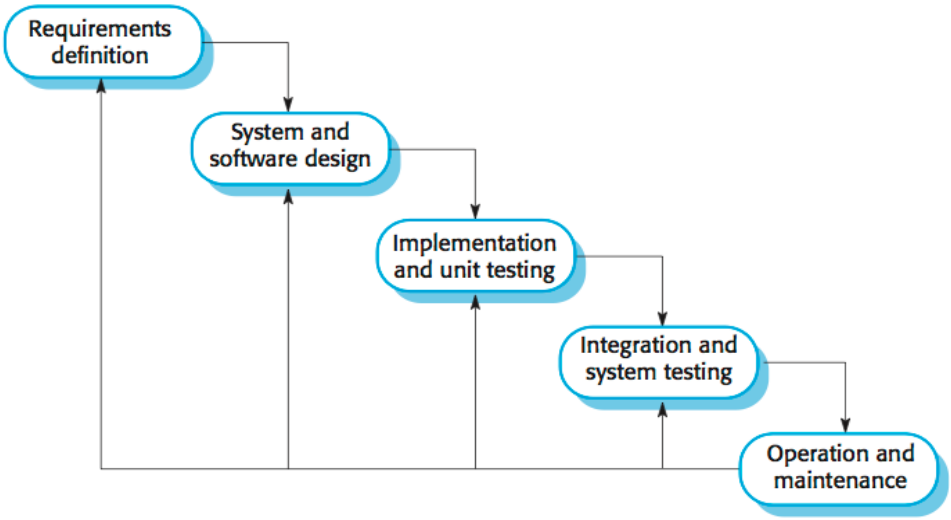
- Waterfall model
- Iterative model
- Prototype model
- Spiral model
- Rapid Application Development (RAD) model
- Unified Process (UP) model

Waterfall Model

The waterfall model is a **sequential, plan driven-process where you must plan and schedule all your activities before starting the project.**

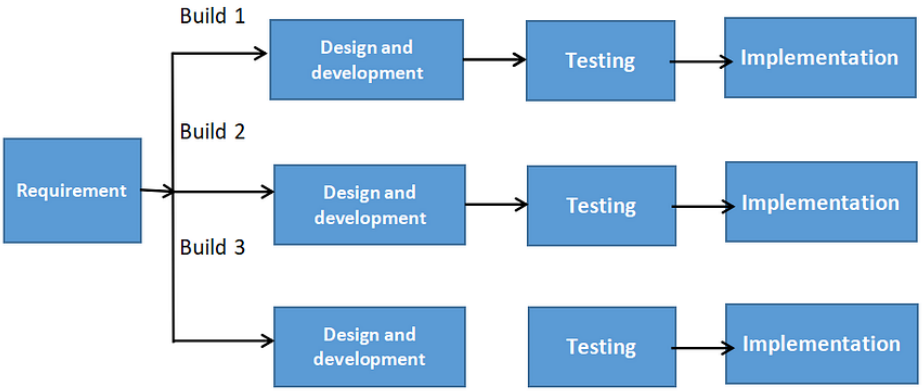
It has the following phases:

- Requirements
- Design
- Implementation
- Testing
- Deployment
- Maintenance



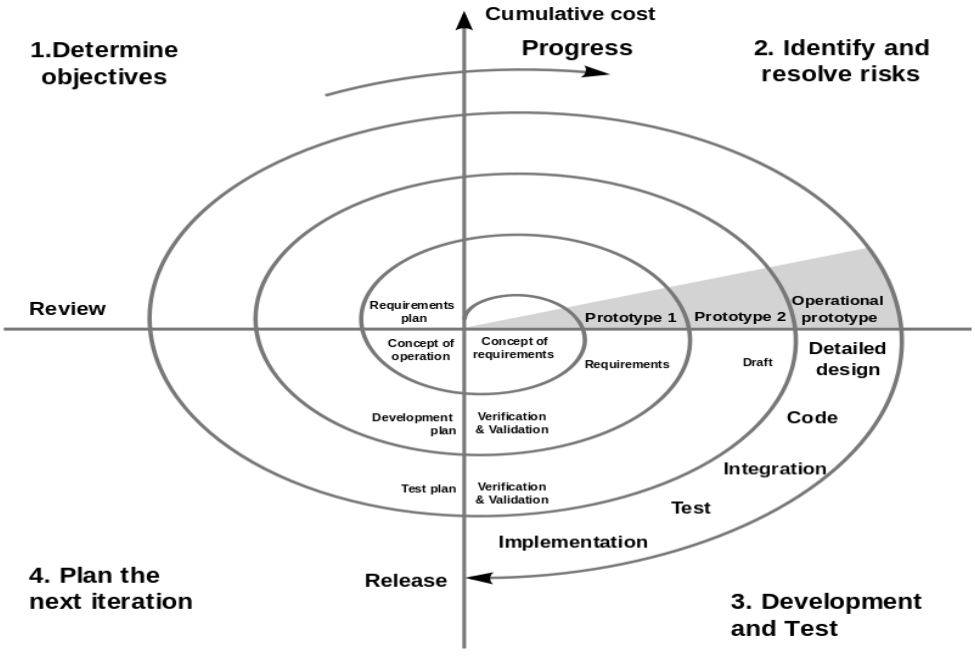
Iterative model

The iterative development model develops a system by **building small portions** of all the features. This helps to meet the initial scope quickly and release it for feedback.



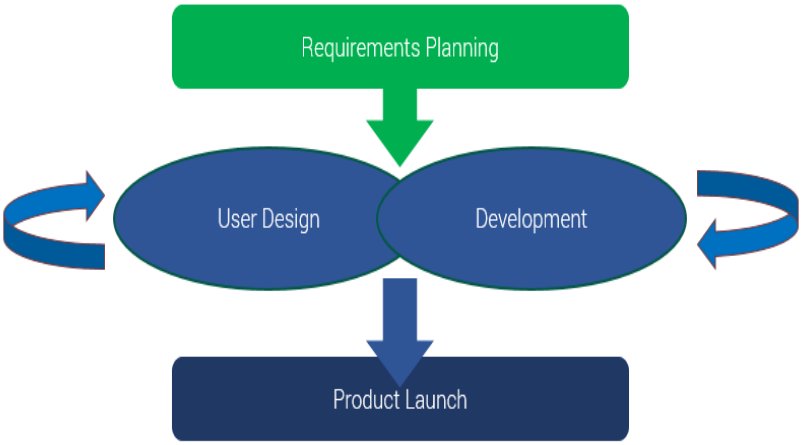
Spiral Model

- The spiral model is a **risk-driven iterative software process model.**
- The spiral model delivers projects in loops.



RAD Model

- The Rapid Application Development (RAD model) is based on iterative development and prototyping with little planning involved.
- You develop functional modules in parallel for faster product delivery.
- It involves the following phases:
  - ❖ Business modeling
  - ❖ Data modeling
  - ❖ Process modeling
  - ❖ Application generation
  - ❖ Testing and turnover

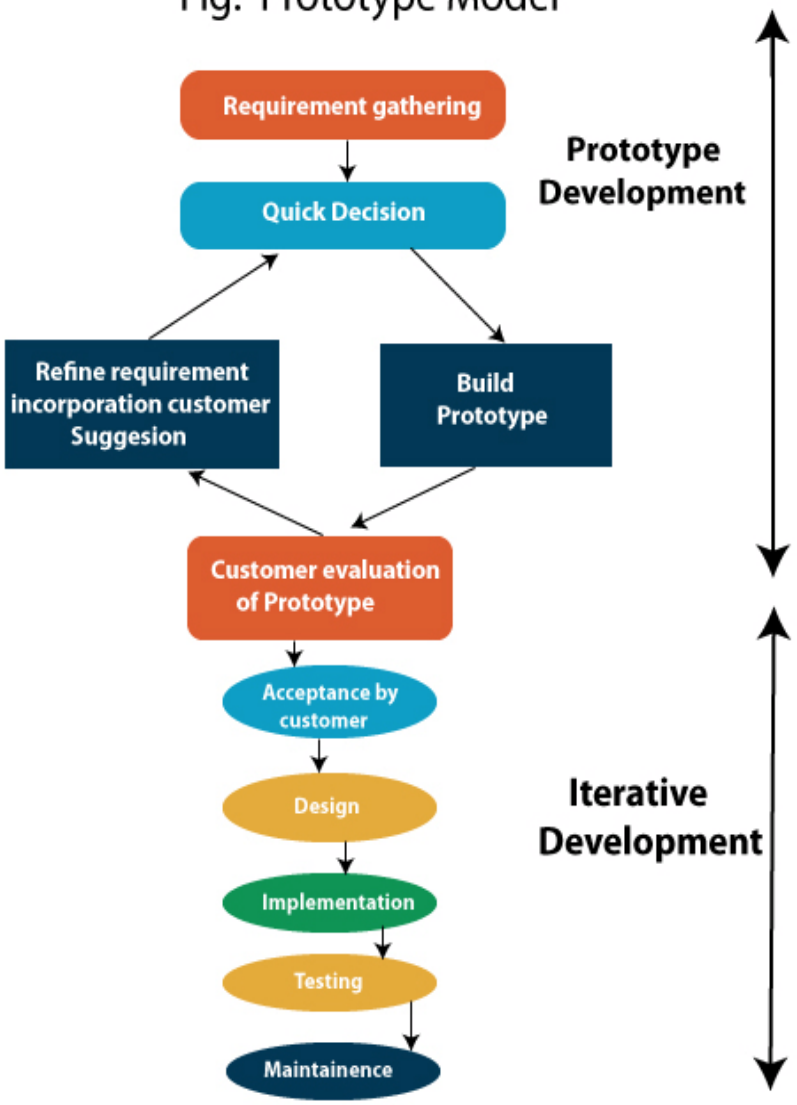


Types of Prototyping Models

There are four types of Prototyping Models, which are described below.

- ❖ Rapid Throwaway Prototyping
- ❖ Evolutionary Prototyping
- ❖ Incremental Prototyping
- ❖ Extreme Prototyping

Fig: Prototype Model



Object and Class

Object

- Object is a combination of data and logic that represent some real-world entity.
- An object has a state and behavior.
- An object is a unique programming entity that has attributes to describe it and methods to retrieve/ set attribute values.

An object has a state (Attribute)

- An object contains both data and methods that manipulate that data.
  - The data represent the state of the object.
  - Data can also describe the relationships between this object and other objects.

Eg:- **A Checking Account might have**

- A balance
- A owner

An object has a behavior (Method)

- An object contains both data and methods that manipulate that data.
  - An object is active, not passive, it does things.
  - An object is responsible for its own data.

Example:- **Computer Object**

- Computer in the room has **states** like,
  On, Off, Computer type
- **Behaviors** such as,

  Switching on/off, switching modes

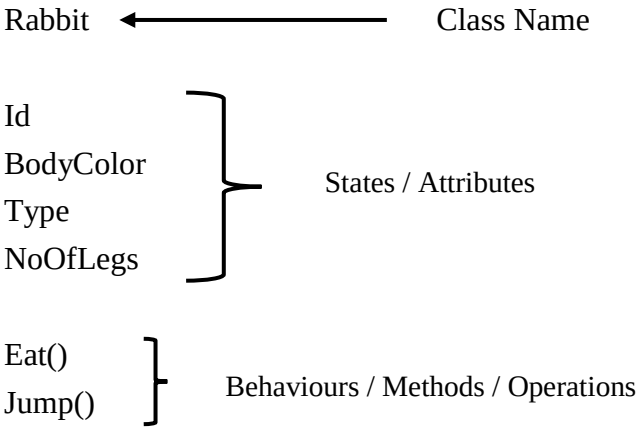
Properties of an Object

- State
- Behavior
- Identity

Class

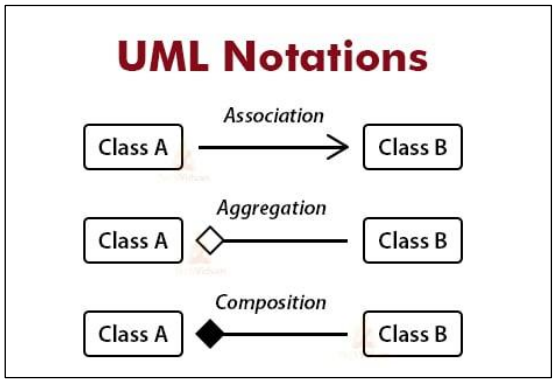
- Class is a template for a set of objects that share a common structure and a common behavior.
- A class is used to create an object-instances. (A class is a blueprint from which individual objects are created.)
- A class is a template used to create multiple objects with similar features.

Example



Relationships among objects

- ❖ Association
- ❖ Aggregation
- ❖ Composition



Association

- The most general relationship.
- An Association is a channel between classes through which messages can be sent.
- Association relationship is a way of describing what a class knows about and holds a reference to another class.
- **Association is a simple structural connection between classes and is a relationship. (where all objects have their own lifecycle and there is no owner)**

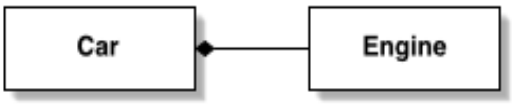
  Eg: Teacher and Student

- ➢ Reflexive Association is **an association between instances of the same class**.
- ➢ Multiplicity in Association **shows the number of objects from one class that relate to a number of objects in an associated class.**

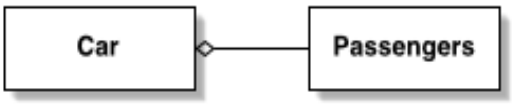
Inducator	**Meaning**
0..1	zero or one
1	only one
0..*	zero or more
1..*	one or more
n	only n (Where n>1)
0..n	zero to n (Where n>1)
1..n	one to n (Where n>1)

Aggregation

- ❖ Aggregation **is a specialized form of association.** ***(Where all object has their own lifecycle own but there is ownership)***
- ❖ An aggregation is a **special type of association in which objects are assembled or configured together to create a more complex object.**

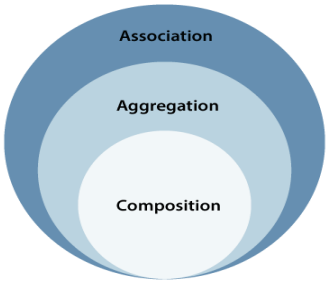


Composition: every car has an engine.



Aggregation: cars may have passengers, they come and go

Composition



- ❖ Composition **is a specialized form of aggregation. (Strong type of Aggregation)**

	Association	Aggregation	Composition
Owner	No owner	Single owner	Single owner
Life time	Have their own lifetime	Have their own lifetime	Owner's lifetime
Child object	Child objects all are independent	Child objects belong to a single parent	Child objects belong to a single parent
Diamond	-	Open diamond	Solid diamond

Hierarchies in real world…

- Hierarchy is a ranking or ordering of abstractions.
- In real world, two main types of Hierarchies are present.
    1. "**Is –a**" Hierarchy
    2. "**part-of**" Hierarchy

"Is –a" Hierarchy

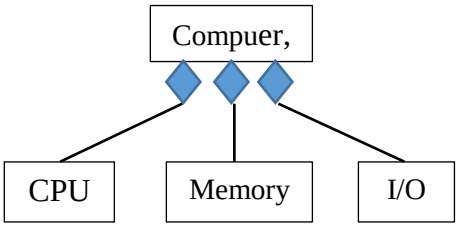
- Corresponds to "is-a" hierarchies in real world.
- Facilitates code re-use.
- Mainly identified as "is-a" relationship.
    - ***Gold fish is a kind of fish***
- In java, Inheritance is implemented by placing the parent class name with 'extends; keyboard.
    - **Class GoldFish Extends Fish**

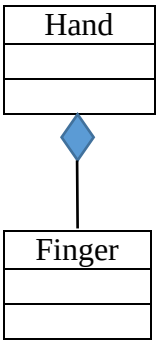
"part-of" Hierarchy

- real world objects are created by combining parts.

Eg:-

- CPU is a part of computer
- Memory is a part of computer
- I/O devices are a part of computer
- [CPU, Memory, I/O are parts. Computer is the whole]





- Hand is made up of fingers.
- A Finger is part of a Hand.
- A Hand has fingers.
- If the Hand is destroyed, so are the fingers.
- Fingers cant exit without hand.

Message Interactions

- In a system, objects work together.
- They do this by sending messages to one another.
- One object sends another a message to perform an operation.
- The receiving object performs that operation.

Eg:- A TV and a Remote

Visualizing relationships

- ✓ Association
- ✓ Aggregation
- ✓ Composition

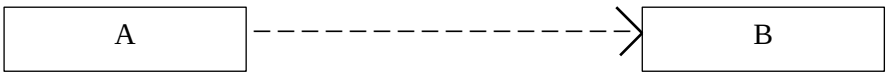
Multiplicity

- The numbers next to the ends of the lines denote multiplicity.
  - *The numbers of items that the one class is associated with the other class.*

- ❖ If the exact multiplicity is known, use it. (The 1 that appears in 8 places)
- ❖ If the range is known, use the range notation. (0..1 or 4..5)
- ❖ If the number is unspecified, use the asterisk. (*)
- ❖ If the range has upper limit unspecified. Combine the range notation with the asterisk notation. (2..*)

Dependency

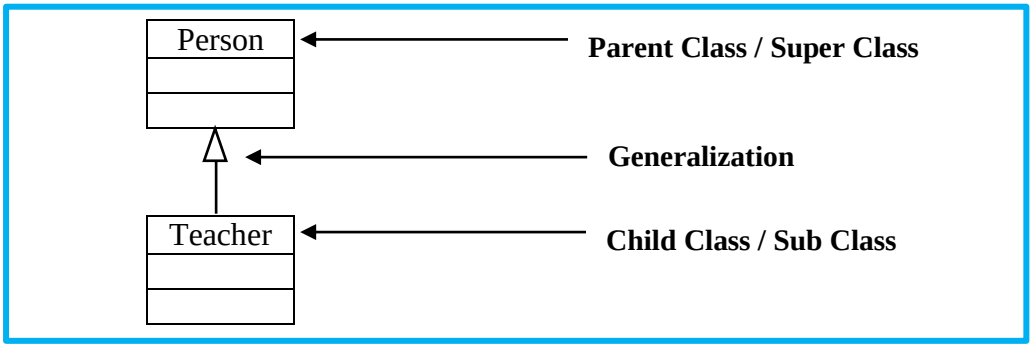
In UML, the dependency is rendered using a dashed directed line, which directed to the things or the class being depended on.



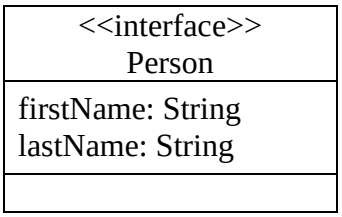
Class A depends on Class B

Generalization

- Parent class is generalization of the child class.
- Generalization is a relationship between a parent and child.
- This relationship also known as ‘is a’ or ‘is a kind of’ relationship.
  Eg:- - Teacher is a Person
  - Rose is a Flower
- A class can have zero, one or more parents.
- In java, a class is allowed to have only one parent.
- A class which **has no parents but has one or more children** is known Base Class.
- And a class with **no children** is known as a Leaf Class.

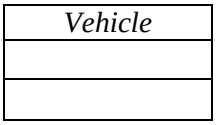


Visualizing interfaces



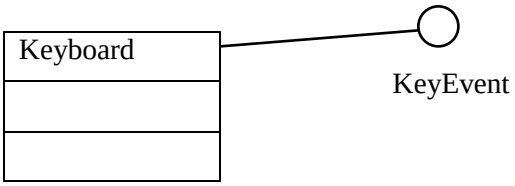
Abstract class

- Indicate the abstract class by writing its Name in italics.



Note

- A note is a graphical symbol for rendering comment attached to an element.
- In UML, note is rendered as a rectangle with a dog-eared corner, together with a textual or graphically comment.
- A note has no semantic impact on the meaning of the model.



Introduction to UML

When developing s/w systems we have to concentrate.

- Implementing the customer requested functionalities.
- Maintaining the quality of the developed systems.
- Manage the risk of failing the system.
- Implement the system timely with an efficient and effective use of resources.

Modeling

One diagram is better than thousand words!!

Models - useful for understanding problems, communicating with everyone involved with project, modeling enterprises, preparing documentation, designing programs and databases.

- ❖ Models are abstractions that describes the essentials of a complex problem or structure by filtering out nonessential details, thus making the problem easier to understand.
- ❖ A model is expressed in a medium that is convenient for working.

The importance of Modeling

- To capture and precisely express the customer requirements and the domain knowledge.
- To get overall architecture of the system beforehand.
- To capture the requirements in mutable form.
- To master complex systems.

Visual Modeling

- Helps organize, visualize and understand complexity.
- It the mapping of real processes of a system to a graphical representation.
- Is an abstraction that represent the essentials of a system, making the problem easier to understand.
- Is a proven and accepted engineering technique.
- Has a common vocabulary, the UML.

What is UML?

- UML is an acronym for **Unified Modeling Language**.
  - UML is therefore a language (A language is simply a tool for expressing ideas.)
- UML is a notation, not a methodology.
  - It can be used in conjunction with any methodology.
- UML has become a world standard.
  - Every information technology professional today needs to know UML.

Modeling language

- Most important part of a methodology.
- It is the key part for communication.
- Any natural language has Words, Sentences and Grammar.
- Modeling language has symbols, Diagrams and Connection Rules.

Unified Modeling Language

It most directly unifies the methods of,

- Booch
- Rumbaugt (OMT)
- Jacobson

Benefits of UML

1. Being independent of any particular programming language and development process its easy to understand.
2. To provide best engineering practices for large and complex system.
3. Encourage the growth of Object Oriented tools.
4. Provides a good way to gather requirements.
5. Improve software quality and decreases time and cost of the software.

Building Blocks of the UML

The vocabulary of UML consists of THREE kinds of building blocks.

- **Things**
- **Relationships**
- **Diagrams**

Things

1. **Structural Things -** static parts of the model that representing elements either in conceptual or physical

   Eg: class, interface, use case, component, node

2. **Behavioral Things –** it represent the dynamic parts of the model.

   Eg: interaction: passing messages among objects, and state: a particular state of an object that has a been taken as a response to an event.

3. **Grouping Things –** they are the organizational part of the UML models.

   Eg: Package

4. **Annotational Things –** they are explanatory things in the UML models.

   Eg: comments

Relationships

1. **Dependency –** It is a relationship between two objects, where state change of One Object (Independent thing) cause the state change of another object (dependent thing)
2. **Association –** It is a relationship between two structural things.
3. **Generalization –** The relationship when one class inherited from other is called Generalization / Specialization relationship.
4. **Realization –** When a class implement an interface, it is called realization relationship.

Diagram

- The UML consists of a number of graphical element that combine to form diagram
- The purpose of the diagrams is to present multiple views of a system.

✓ **Class Diagrams –** Shows set of classes, interfaces and collaborations and their relationship.

✓ **Object Diagrams –** Shows a set of objects and their relationship.

✓ **Use case Diagrams –** Shows a set of use cases and actors and their relationship.

✓ **Sequence Diagrams –** An interaction diagram that emphasizes the time-ordering of messages.

✓ **Communication/ Collaboration Diagrams -** An interaction diagram that emphasizes the structural Organization of the objects that send and receive messages.

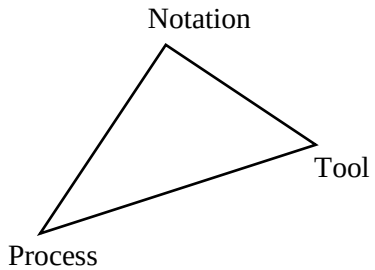
✓ **State chart Diagrams –** Address the dynamic view of a system. Shows a state machine consisting of states, transitions, events and activities.

✓ **Activity Diagrams –** A special kind of a statechart diagram that shows the flow from activity to activity within a system. Address the dynamic view of a system.

✓ **Component Diagrams –** Shows the organizations and dependencies among a set of components. Address the static implementation view of a system.

✓ **Deployment Diagrams –** Shows the configuration of run time processing nodes and the components live on them.

Components needed for a successful project



Triangle for Success

- ❖ You can learn a notation, but if you don't know how to use it (process), you will probably fail.
- ❖ You may have a great process, but if you cannot communicate the process (natation), you will probably fail.
- ❖ Lastly, if you cannot document the artefacts of your work (tool), you''ll probably fail.

The role of **Notation**

- Notation plays an important part in any model.
- It is the glue that holds the process together

Notation has three roles

1. It serves as the language for communicating decisions that are not obvious or cannot be inferred from the code itself.
2. It provides semantics that are rich enough to capture all important strategic decisions.
3. It offers a from, concrete enough for humans to reason and tools to manipulate.

Notation, Process, Tools

Componemts:-

- **Notation – UML**
- **Process – Rational Unified Process**
- **Tool – Star UML / Rational Rose**

Key Points

- **Visual modeling is a way of thinking about problems using models organized around real-world ideas.**
- **Notation, Process and Tool are Triangle for Success.**
- **UML is an acronym for Unified Modeling Language.**

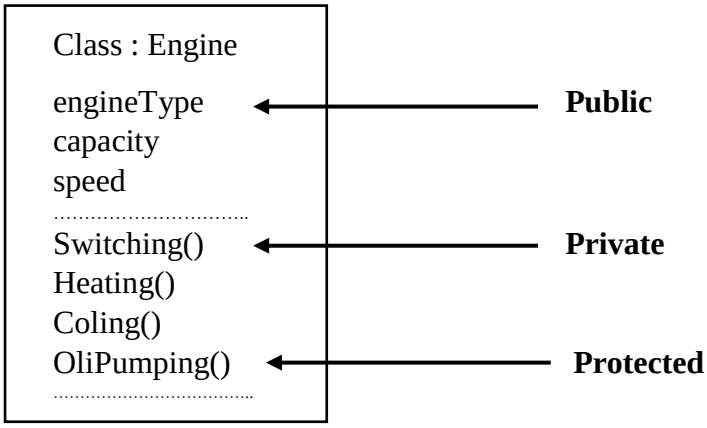
Concepts in OOP

- ✓ Encapsulation – (Access modifiers)
- ✓ Inheritance
- ✓ Abstraction – (Abstract class & methods, Interface)
- ✓ Polymorphism – (Overloading, Overriding)

Encapsulation – It is combining data and methods of a class in a single unit and hiding the implementation details from the user of the object. (Information hiding)

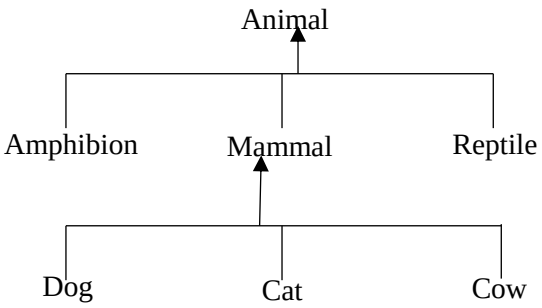
Access modifiers

1. **Private** – Attributes and methods with private access modifier are accessible only within the class. (outsider can not access)
2. **Public** – Attributes and methods with public access modifier can be accessed by anyone.
3. **Protected** – Attributes and methods with protected access modifier can be accessed only be the subclass (Children) of the class (Parent) or the classes in the same package.



Inheritance – It is the property whereby one class extends another class by including additional methods and/or variables.

- The original class is called the **superclass.**
- The extending class is called the **subclass.**
- Since a subclass contains all of the data and methods of the superclass plus additional resources, It is more specific.
- Since the superclass lacks some of the resources of the subclass, It is more general or abstract, than its subclasses.




```
Class Animal {
}
Class Mammal extends Animal {
}
Class Cat extends Mammal {
}
```

Abstraction – It is the act of representing essential features without giving background details or explanations.

- Can be achieved two ways.
  - *Abstract classes*
  - *Interfaces*

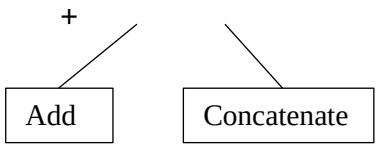
- Abstract class – A class which cannot be instantiated.
- Abstract method – It doesn't have an implementation.
- Interface – It is a collection of methods that can be used to implement varying behaviors.
  - It provides only a form, but no implementation.

```
public abstract class MyAddition {
        public abstract void addNumbers() ;
        //abstract method
}
public class MyNewAddition extends MyAddition {
        public void addNumbers() {
            System.out.println ("My new addition method.") ;
        }
}
```

Polymorphism

- It means that ability to take more than one form.
- It allows objects having different internal structure to share the same external interface.

Eg:- + *operater may exhibit different behavior in different instances.*



Overloading	Overriding
Must have at least **two methods by same name in same class.**	Must have at least **one methods by same name in both Parent and Child class.**
Must have **different number of Parameters.**	Must have **same number of Parameters.**
If number of parameters are same then must have **different types of parameters.**	**Types of parameters also must be same.**
Overloading known as **compile time** polymorphism.	Overriding known as **run time** polymorphism.

Principles for Systems Development

Underlying Principles for Systems Development

1) Get the system users involved.
2) Use a problem-solving approach.
3) Establish phases and activities.
4) Document throughout development.
5) Establish standards.
6) Manage the process and projects.
7) Justify systems as capital investments.
8) Don’t be afraid to cancel or service scope.
9) Divide and conquer.
10) Design systems for growth and change.