The Concept Of Object-Orientation

- It's not a language itself but a set of concepts that is supported by many languages
- In an object-oriented language, this one large program will instead be split apart into self contained objects, almost like having several mini-programs, each object representing a different part of the application.
- And each object contains its own data and its own logic, and they communicate between themselves And each object contains its own data and its own logic, and they communicate between themselves.
- They represent things like employees, images, bank accounts, spaceships, asteroids, video segment, audio files, or whatever exists in your program

OOAD

- OOA- Identify the objects
- OOD- Relationships of identified objecs and model them

Object-Oriented Analysis And Design (OOAD)

It's a structured method for analyzing, designing a system by applying the object-orientated concepts, and develop a set of graphical system models during the development life cycle of the software.

OOAD In The SDLC

- The software life cycle is typically divided up into stages going from abstract descriptions of the problem to designs then to code and testing and finally to deployment.
- The earliest stages of this process are analysis (requirements) and design.
- The distinction between analysis and design is often described as "what Vs how"

Analysis

- In analysis developers work with users and domain experts to define what the system is supposed to do. Implementation details are supposed to be mostly or totally ignored at this phase.
- The goal of the analysis phase is to create a model of the system regardless of constraints such as appropriate technology.
- This is typically done via use cases and abstract definition of the most important objects using conceptual model.

Design

- The design phase refines the analysis model and applies the needed technology and other implementation constrains.
- It focuses on describing the objects, their attributes, behavior, and interactions.
- The design model should have all the details required so that programmers can implement the design in code.

Object-Oriented Analysis

In the object-oriented analysis, we have:-

- **Elicit requirements:** Define what does the software need to do, and what's the problem the software trying to solve.
- **Specify requirements:** Describe the requirements, usually, using use cases (and scenarios) or user stories.
- **Conceptual model**: Identify the important objects, refine them, and define their relationships and behavior and draw them in a simple diagram.

We're not going to cover the first two activities, just the last one. These are already explained in detail in Requirements Engineering

Object-Oriented Design

- The analysis phase identifies the objects, their relationship, and behavior using the conceptual model
- While in design phase, we describe these objects , their attributes, behavior, and interactions.
- The input for object-oriented design is provided by the output of object-oriented analysis. But, analysis and design may occur in parallel, and the results of one activity can be used by the other

In the object-oriented design, we have:-

- Describe the classes and their relationships using class diagram.
- Describe the interaction between the objects using sequence diagram.
- Apply software design principles and design patterns.
- Describing the interactions between those objects lets you better understand the responsibilities of the different objects, the behaviors they need to have.

Complexity

The larger the number of components and relationships between them, higher will be the complexity of the overall system.

Grady Booch

The function of good software is to make the complex system, but appear to be simple.

Simple

- Largely forgettable applications
- Very limited purpose
- Afford to throw them away

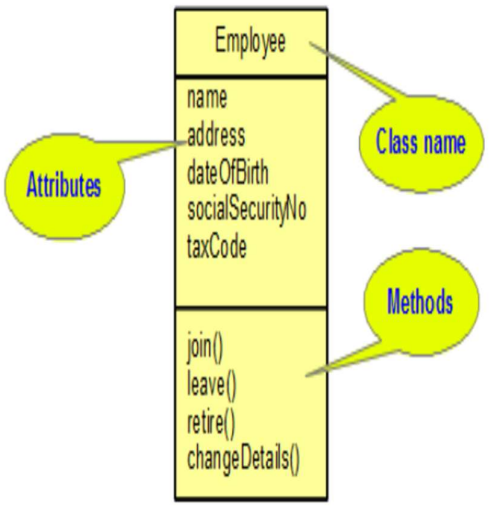
Object oriented methodologies

0bject-oriented development:- Object-oriented design is part of object-oriented development where an object-oriented strategy is used throughout the development process.

Object-oriented analysis:- It is concerned with developing an objectoriented model of the application domain. The objects in that model reflect the entities and operations associated with the problem to be solved.

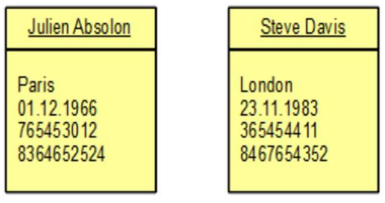
Object-oriented design:- It is concerned with developing an objectoriented model of a software system to implement the identified requirements.

Object-oriented programming:- It is concerned with implementing a software design using an object-oriented programming language, such as Java.



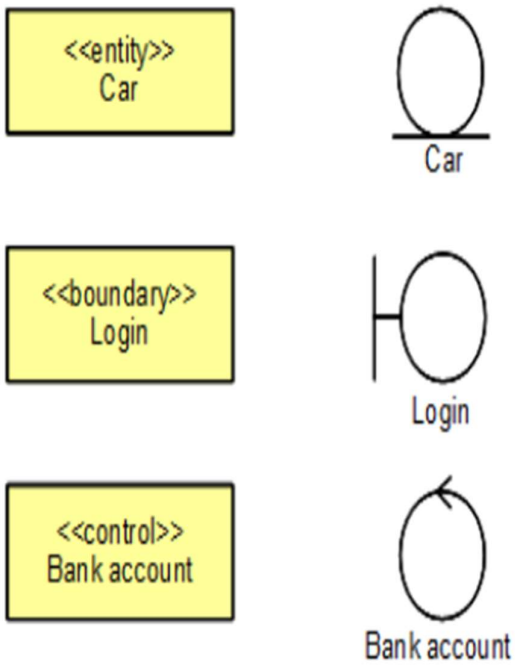
In the UML, an object class is represented as a named rectangle with two sections. The object attributes are listed in the top section. The operations that are associated with the object are set out in the bottom section. Figure illustrates this notation using an object class that models an employee in an organisation. The UML uses the term operation.

- The class Employee defines a number of attributes that hold information about employees including their name and address, social security number, tax code, and so on.



UML supports stereotypes, which are an inbuilt mechanism for logically extending or altering the meaning, display, characteristics or syntax of a basic UML model element such as class and object. Extending classes in UML the next stereotypes are often used:

- <<boundary>> stereotype. A boundary class represents a user interface.(service collaborator)
- <<entity>> stereotype. Entity classes represent manipulated units of information.(model)
- <<control>> (controller)mediate between boundary and entity. glue between boundary and entity elements.
- UML representation of stereotypes applied to classes Car, Login and Bank account are shown in Figure together with an icon for their default representation.



Four rules apply to their communication-

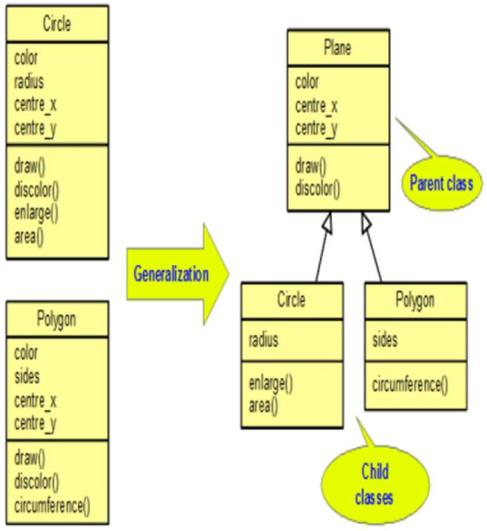
1. Actor can only talk to boundary objects.
2. Boundary objects can only talk to controller and actors.
3. Entity objects can only talk to controllers.
4. Controllers can talk to boundary objects and entity objects, and to other controllers,but not to actors

- **Communication allowed**

	Entity	Boundary	Control
Entity	x		x
Boundary			x
Control	x	x	x

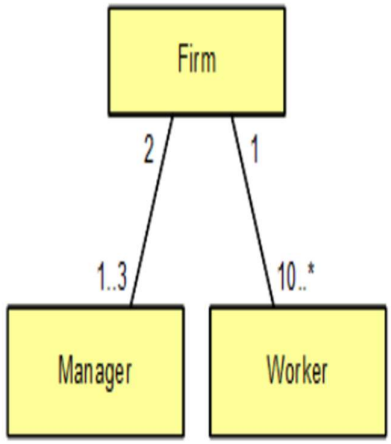
Generalisation

- Object classes can be arranged in a generalisation or inheritance hierarchy that shows the relationship between general and more specific object classes. The more specific object class is completely consistent with its parent class but includes further information. In the UML, an arrow that points from a class entity to its parent class indicates generalisation. In object-oriented programming languages, generalisation is implemented using inheritance. The child class inherits attributes and operations from the parent class.



Association

Objects that are members of an object class participate in relationships with other objects. These relationships may be modelled by describing the associations between the object classes. In the UML, associations are denoted by a line between the object classes that may optionally be annotated with information about the association. Association is a very general relationship and is often used in the UML to indicate that either an attribute of an object is an associated object or the implementation of an object method relies on the associated object.



Characteristics of Objects

- An object has identity (each object is a distinct individual).
- An object has state (it has various properties, which might change).
- An object has behavior (it can do things and can have things done to it)

For example, you can think of your bank account as an object, but it is not made of material. (Although you and the bank may use paper and other material in keeping track of your account, your account exists independently of this material.) Although it is not material, your account has

properties (a balance, an interest rate, an owner) and you can do things to it (deposit money, cancel it) and it can do things (charge for transactions, accumulate interest)

Object

An object is a real-world element in an object–oriented environment that may have a physical or a conceptual existence.

Each object has:-

- **Identity** that distinguishes it from other objects in the system.
- **State** that determines the characteristic properties of an object as well as the values of the properties that the object holds.
- **Behavior** that represents externally visible activities performed by an object in terms of changes in its state.

An object may have a physical existence, like a customer, a car, etc.; or an intangible conceptual existence, like a project, a process, etc.

Fundamental Concepts of Object Orientetion

- Class
- Objects
- Data Abstraction
- Encapsulation
- Inheritance
- Polymorphism
- Message Passing

Class

A class is a user-defined data type. It consists of data members and member functions, which can be accessed and used by creating an instance of that class. It represents the set of properties or methods that are common to all objects of one type. A class is like a blueprint for an object.

Object

It is a basic unit of Object-Oriented Programming and represents the real-life entities. An Object is an instance of a Class. When a class is defined, no memory is allocated but when it is instantiated (i.e. an object is created) memory is allocated. An object has an identity, state, and behavior.

Data Abstraction

Data abstraction is one of the most essential and important features of object-oriented programming. Data abstraction refers to providing only essential information about the data to the outside world, hiding the background details or implementation.

Encapsulation

Encapsulation is defined as the wrapping up of data under a single unit. It is the mechanism that binds together code and the data it manipulates. In Encapsulation, the variables or data of a class are hidden from any other class and can be accessed only through any member function of their class in which they are declared. As in encapsulation, the data in a class is hidden from other classes, so it is also known as data-hiding.

Inheritance

Inheritance is an important pillar of OO(Object-Orientetion) . The capability of a class to derive properties and characteristics from another class is called Inheritance.

Polymorphism

The word polymorphism means having many forms. In simple words, we can define polymorphism as the ability of a message to be displayed in more than one form. For example, A person at the same time can have different characteristics. Like a man at the same time is a father, a husband, an employee.

Message Passing

It is a form of communication used in objectoriented programming as well as parallel programming. Objects communicate with one another by sending and receiving information to each other. A message for an object is a request for execution of a procedure and therefore will invoke a function in the receiving object that generates the desired results. Message passing involves specifying the name of the object, the name of the function, and the information to be sent.

Unified Modeling Language (UML)

- Unified Modeling Language (UML) is a general purpose modeling language. The main aim of UML is to define a standard way to visualize the way a system has been designed. It is quite similar to blueprints used in other fields of engineering.
- UML is not a programming language
- The Object Management Group (OMG) adopted Unified Modeling Language as a standard in 1997. Its been managed by OMG ever since. International Organization for Standardization (ISO) published UML as an approved standard in 2005.

Role of UML in OO Design

- UML is a modeling language used to model software and non-software systems.
- If we look into class diagram, object diagram, collaboration diagram, interaction diagrams all would basically be designed based on the objects.
- Hence, the relation between OO design and UML is very important to understand. The OO design is transformed into UML diagrams according to the requirement.

Do we really need UML?

- Complex applications need collaboration and planning from multiple teams and hence require a clear and concise way to communicate amongst them.
- Businessmen do not understand code. So UML becomes essential to communicate with non programmers essential requirements, functionalities and processes of the system.
- A lot of time is saved down the line when teams are able to visualize processes, user interactions and static structure of the system.
- UML is linked with object oriented design and analysis.

Diagrams in UML can be broadly classified as:-

- Structural Diagrams
- Behavior Diagrams

They will be discussed in Depth in further units.

RUP and its Phases

Rational Unified Process (RUP) is a software development process for object-oriented models. It is also known as the Unified Process Model. It is created by Rational corporation and is designed and documented using UML (Unified Modeling Language). This process is included in IBM Rational Method Composer (RMC) product. IBM (International Business Machine Corporation) allows us to customize, design, and personalize the unified process.

RUP is proposed by Ivar Jacobson, Grady Bootch, and James Rambaugh. Some characteristics of RUP include use-case driven, Iterative (repetition of the process), and Incremental (increase in value) by nature, delivered online using web technology, can be customized or tailored in modular and electronic form, etc. RUP reduces unexpected development costs and prevents wastage of resources.

Phases of RUP

There are total 5 phases of life cycle of RUP:-

Inception –

- Communication and planning are main.
- Identifies Scope of the project using use-case model allowing managers to estimate costs and time required.
- Customers requirements are identified and then it becomes easy to make a plan of the project.

Elaboration—

- Elaboration means describing something in more detail.
- Here we go into more detail of 1s t phase.
- Detailed evaluation, development plan is carried out and diminish the risks.
- Revise or redefine use-case model (approx. 80%), business case, risks.

Construction –

- Here we develop and complete the project based on the data we get from previous stages.
- System or source code is created and then testing is done.
- Coding takes place.

Transition –

- Final project is released to public.
- Project is developed and completed.
- Remove the bug from project based on customer feedback
- Transit the project from development into production.

Production –

- Final phase of the model.
- Project is maintained and updated accordingly.

Six best practices of rational unified process(RUP):-

Develop iteratively

It is best to know all requirements in advance; however, often this is not the case. Several software development processes exist that deal with providing solutions to minimize cost in terms of development phases.

Manage requirements

Always keep in mind the requirements set by users.

Use components

Breaking down an advanced project is not only suggested but in fact unavoidable. This promotes ability to test individual components before they are integrated into a larger system. Also, code reuse is a big plus and can be accomplished more easily through the use of object-oriented programming

Model visually

Use diagrams to represent all major components, users, and their interaction. "UML" , short for Unified Modeling Language, is one tool that can be used to make this task more feasible.

Verify quality

Always make testing a major part of the project at any point of time. Testing becomes heavier as the project progresses but should be a constant factor in any software product creation.

Control changes

Many projects are created by many teams, sometimes in various locations, different platforms may be used, etc. As a result, it is essential to make sure that changes made to a system are synchronized and verified constantly. (See Continuous integration)