Object Oriented System Design (KCS-054)

Content

Object Oriented Programming

- Object-oriented programming (OOP) is a computer programming model that organizes software design around data, or objects, rather than functions and logic.
- Word **object-oriented** is the combination of two words i.e. **object** and **oriented**. The dictionary meaning of the object is ‘an article or entity that exists in the real world’. The meaning of oriented is ‘interested in a particular kind of thing or entity’.
- Object oriented programming uses the concept of object and classes to model the real world entity.

Object Oriented Programming

- Object Oriented Programming as an approach that provides a way of modularizing programs by creating partitioned memory area for both data and functions that can be used as templates for creating copies of such modules on demand.

- It is an engineering approach to build software.

Features of Object Oriented Programming

- Emphasis on data rather than procedure.
- Data is hidden and can not be accessed by external functions.
- New data can be easily added.
- It follow the Bottom-up approach.

OBJECT

- Objects are the basic run time entities in an object oriented system. They may represent a person, a place, a bank account, a table of data or any item that the program has to handle.

- It is a real-world entity that has id, attributes, behavior, and properties.

- It is referred to as an instance of the class. It contains member functions, variables that we have defined in the class.

Objects

- **Objects have three responsibilities:**

What they know about themselves – (e.g., Attributes)

What they do – (e.g., Operations)

What they know about other objects – (e.g., Relationships)

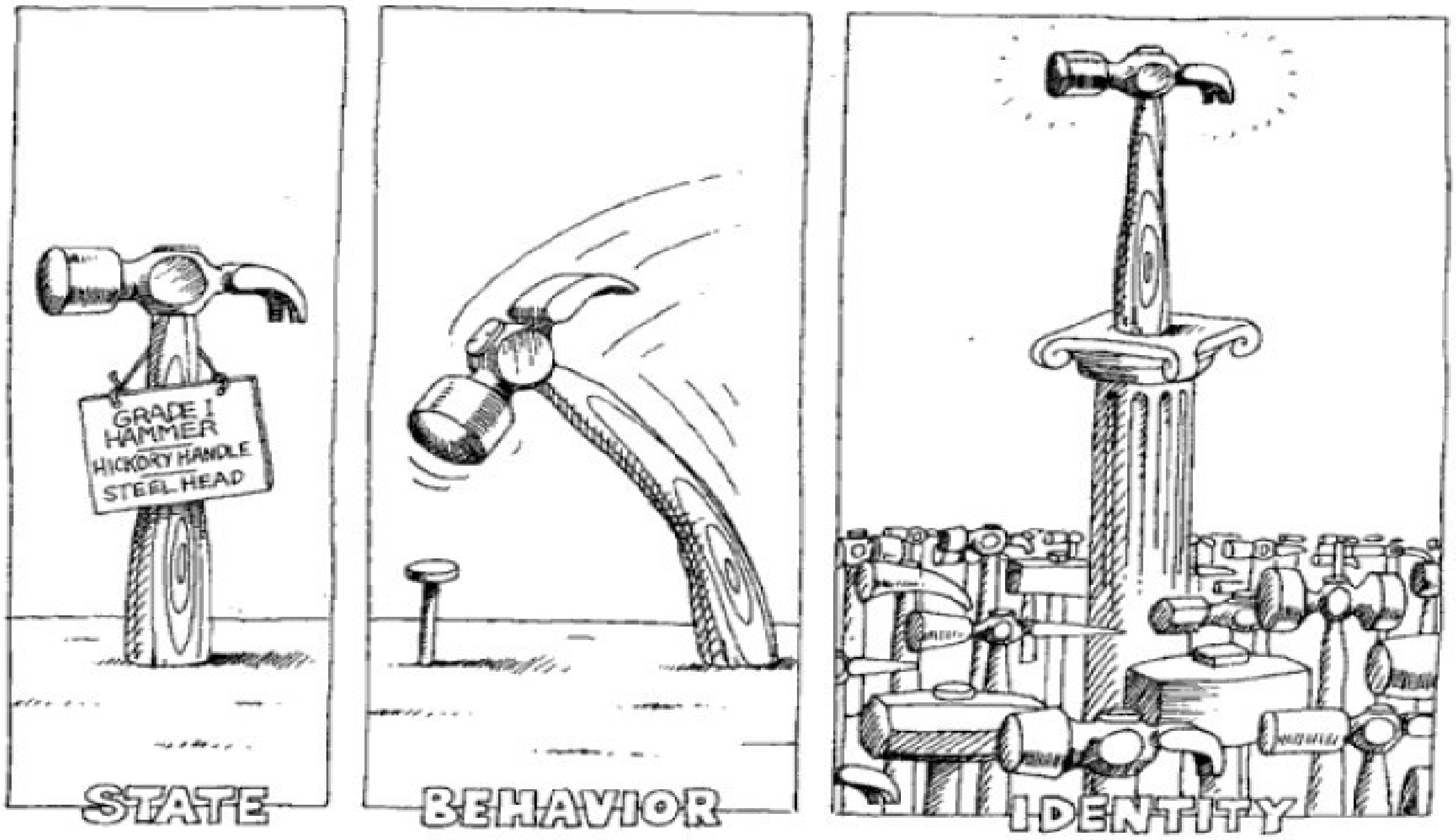
Objects

- An object has:
  - *state* - descriptive characteristics
  - *behaviors* - what it can do (or what can be done to it)
- The state of a bank account includes its account number and its current balance
- The behaviors associated with a bank account include the ability to make deposits and withdrawals
- Note that the behavior of an object might change its state

Objects

- Software objects model read-world objects or abstract concepts
  - dog, bicycle, Bank account
- Real-world objects have states and behaviors
  - Dogs' states: name, color, breed, hungry
  - Dogs' behaviors: barking fetching
  - Bicycle ‘s State :
  - Bicycle’ s behavior :
- Other examples of objects are:
  - myself – an instructor object.
  - you – a student object
  - this room – a room object
  - this university
  - your car, etc.

Object	Identity	Behavior	State
A person	‘ABC’	Speak, Walk, Read	Studying , Resting , Qualified
A book	Book with title Object Oriented System Design	Add Exemplar	Rent, Available , Reserved
A sale	Sale No 0015 With date	Send Invoiced(), Cancel().	Invoiced, Cancelled



An object has state, exhibits some weii-defined behavior, and has a unique identity.

Benefits of OOPs

- Code Reusability
- Code Flexibility
- Effective problem solving
- Modularity for easier troubleshooting
- Lower cost of development

Difference between OOPs and POP

Sr. No.	Key	OOP	POP
1	Definition	OOP stands for Object Oriented Programing.	POP stands for Procedural Oriented Programming.
2	Approach	OOP follows bottom up approach.	POP follows top down approach.
3	Division	A program is divided to objects and their interactions.	A program is divided into functions and they interacts.
4	Inheritance supported	Inheritance is supported.	Inheritance is not supported.
5	Access control	Access control is supported via access modifiers.	No access modifiers are supported.
6	Data Hiding	Encapsulation is used to hide data.	No data hiding present. Data is globally accessible.
7	Example	C++, Java	C, Pascal

Structure of Object Oriented Programming

The structure, or building blocks, of object-oriented programming include the following:

.

- **Classes** are user-defined data types that act as the blueprint for individual objects, attributes and methods.

- **Objects** are instances of a class created with specifically defined data. Objects can correspond to real-world objects or an abstract entity. When class is defined initially, the description is the only object that is defined.

Structure of Object Oriented Programming

- **Methods** are functions that are defined inside a class that describe the behaviors of an object. Each method contained in class definitions starts with a reference to an instance object. Additionally, the subroutines contained in an object are called instance methods.

- Programmers use methods for reusability or keeping functionality encapsulated inside one object at a time.

- **Attributes** are defined in the class template and represent the state of an object. Objects will have data stored in the attributes field. Class attributes belong to the class itself.

What is Object-Orientation about?

- One of the key challenges faced by Computer Scientist is how to handle complexity.
- Two main concepts used to manage complexity are *Modularity* and *Abstractions*.
  - **Modularity** means breaking a large system up into smaller pieces until each peace becomes simple enough to be handled easily.
  - **Abstraction** focus on essential aspects of an application while ignoring details.
- Over the years, computer scientists have developed a number of approaches to achieve modularity and abstraction.
- The latest of these approaches is *Object-Orientation* or *OO* for short.
- The key concept in OO is of course *Object*,

Object-Oriented

- Software is organized as a collection of discrete objects that incorporate both State and behavior.
- **Four aspects (characteristics) required by an OO approach are -**

  Identity

  Classification

  Polymorphism

  Inheritance

Object Identity

- An Object is a real-world element in an object oriented environment that may have a physical or conceptual existence.
- Object identity is a fundamental object orientation_concept.
- With object identity, objects can contain or refer to other objects.
- Identity is a property of an object that distinguishes the object from all other objects in the application.

- There are several forms of identity:

1- value: A data value is used for identity (e.g. the primary key of a tuple in a relational databases)

2- **name:** A user-supplied name is used for identity.(e.g. file name in a file system).

3-built-in: A notion of identity is built-into the data model or programming languages.

Object	Identity	Behavior	State
A person	‘ABC’	Speak, Walk, Read	Studying , resting , qualified
A book	Book with title Object Oriented System Design	Add Exemplar	Rent, Available , Reserved
A sale	Sale No 0015 With date	Send Invoiced(), Cancel().	Invoiced, Cancelled

Classification

- It means that objects with same data structure (attribute) and behavior (operations) are grouped into a class.
- Each object is said to be an instance of its class.
- A *class* is simply a representation of a type of *object*. It is the blueprint/ plan/ template that describe the details of an *object*.
- A class is the blueprint from which the individual objects are created.

Basic Principle of Object Oriented programming

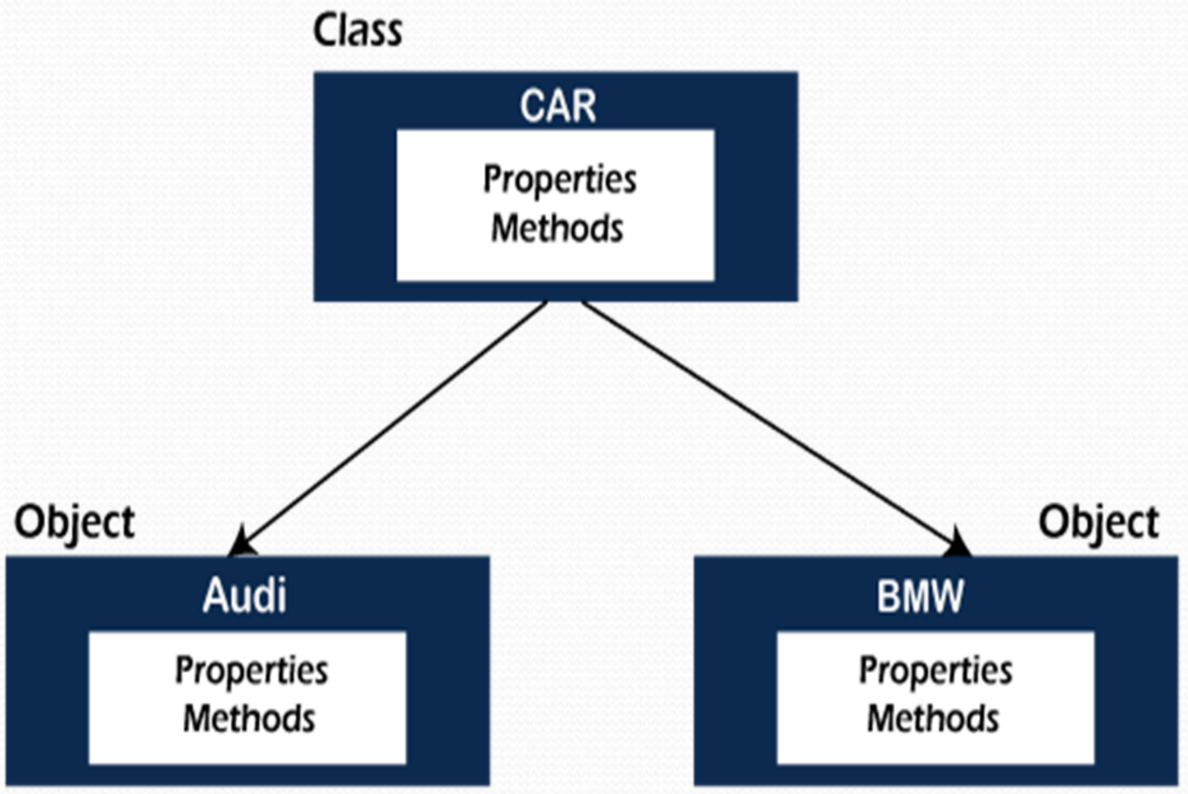
The major concept of an OOPs is defined below.

- Object
- Class
- Data Abstraction
- Encapsulation
- Inheritance
- Polymorphism

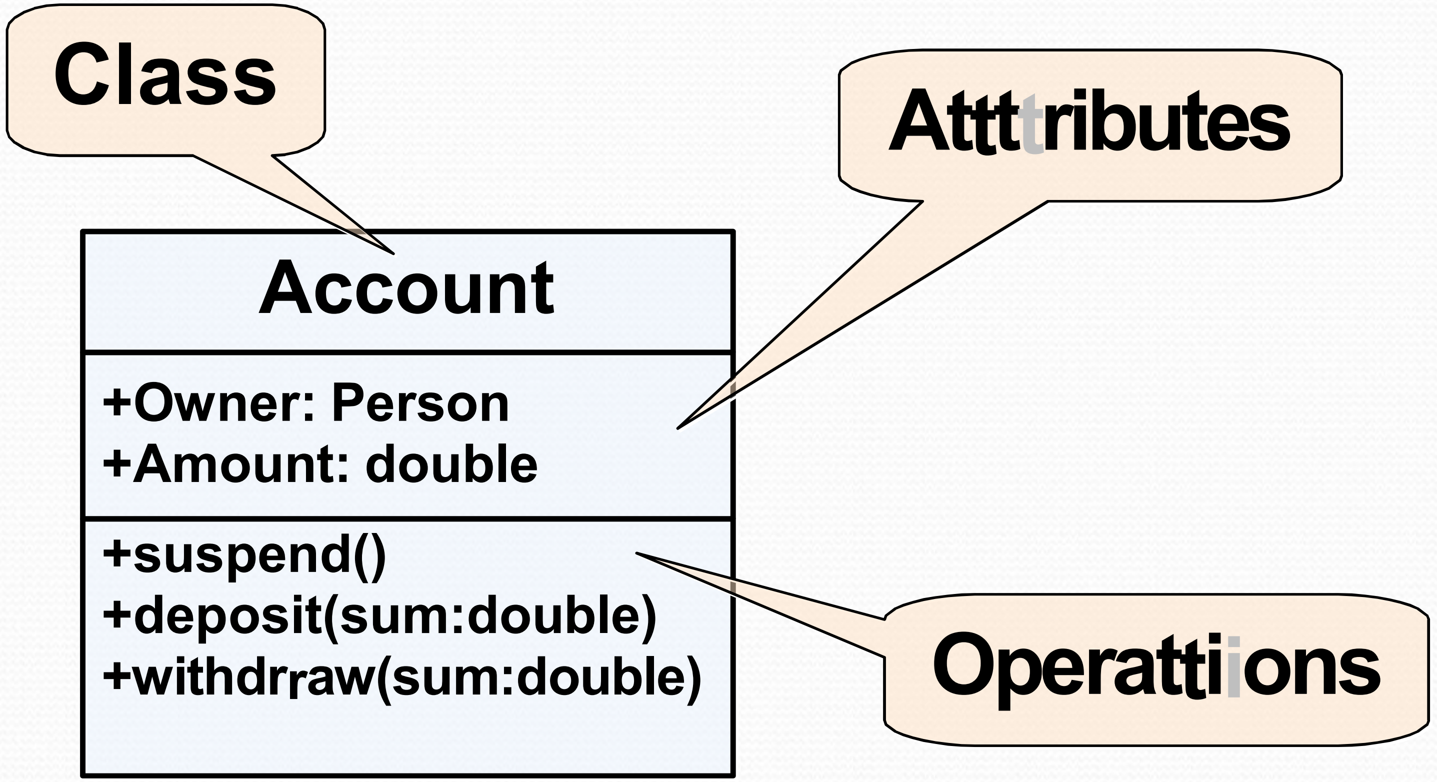
Class

- A class is a blueprint or template of an object. It is a user-defined data type.
- The entire set of data and code of an object can be made of a user defined data type with the help of a class.
- In fact, Objects are variables of the type class. Once a class has been defined, we can create any number of objects belonging to that class.
- Inside a class, we define variables, constants, member functions, and other functionality.

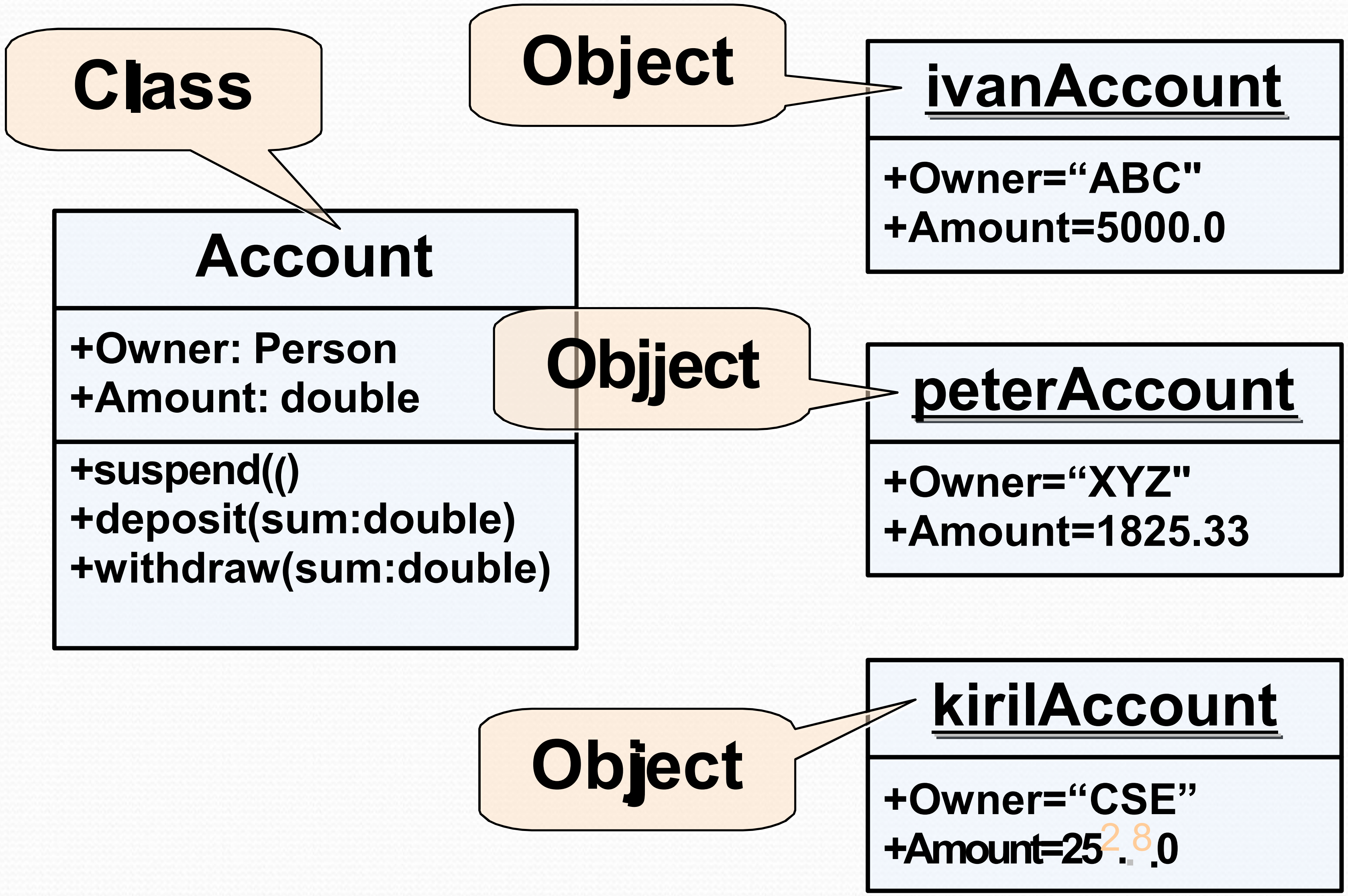
The following figure best illustrates the class and object in OOP.



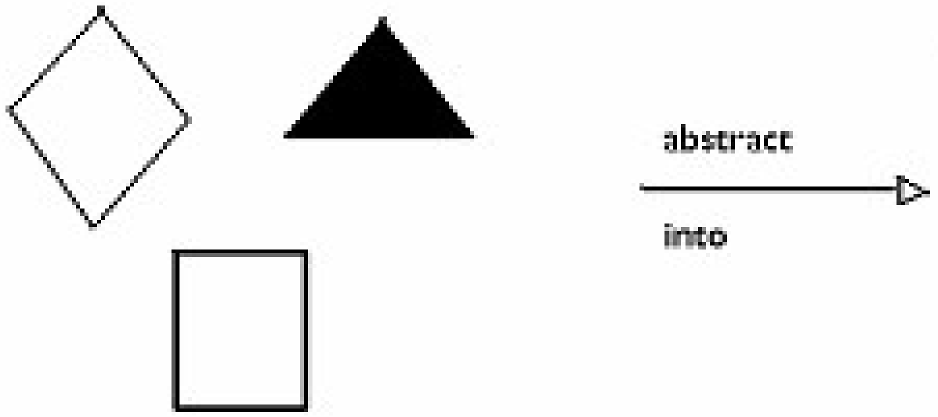
Classes – Example



Classes and Objects – Example



Eg:



Polygon class:

Attributes-

-vertices.

-border class.

-fill color.

Operations-

-draw.

-erase.

-move.

Bycycle class:

Attributes-

- frame size

-wheel size

no. of gears

-material

Operations-

-shift

-move

-repair

Data Abstraction(Information Hiding)

- Abstraction refers to the act of representing essential features without including the background details or explanations. since the classes use the concept of data abstraction ,they are known as Abstract Data Type (ADT)

- In other words, abstraction is a technique by which we can hide the data that is not required to a user.

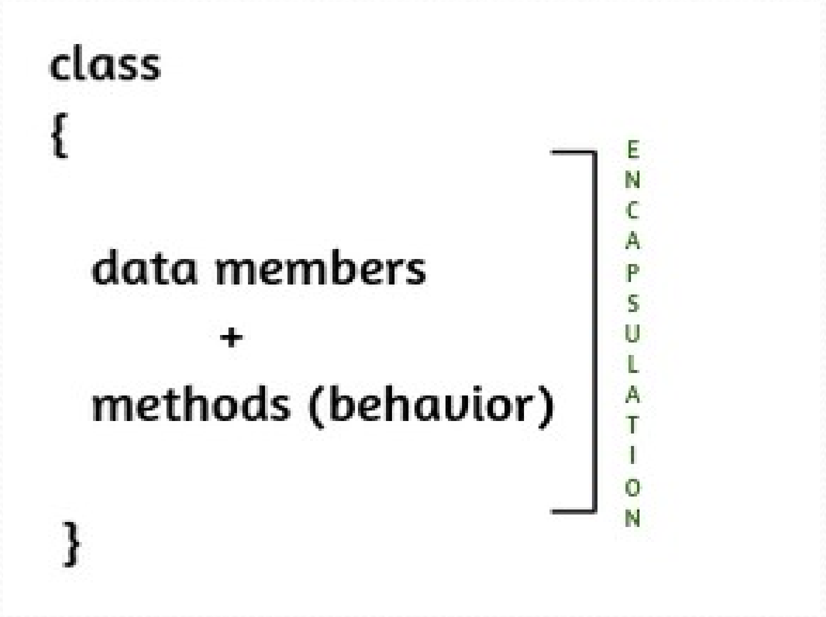
Example

ATM machine is a real-time example. We all use an ATM machine for cash withdrawal, money transfer, retrieve min-statement, etc. in our daily life.

But we don’t know internally what things are happening inside ATM machine when you insert an ATM card for performing any kind of operation.

Encapsulation

Encapsulation is a mechanism that allows us to bind data and functions of a class into an entity. It protects data and functions from outside interference and misuse. Therefore, it also provides security. A class is the best example of encapsulation.



Example

Example1:

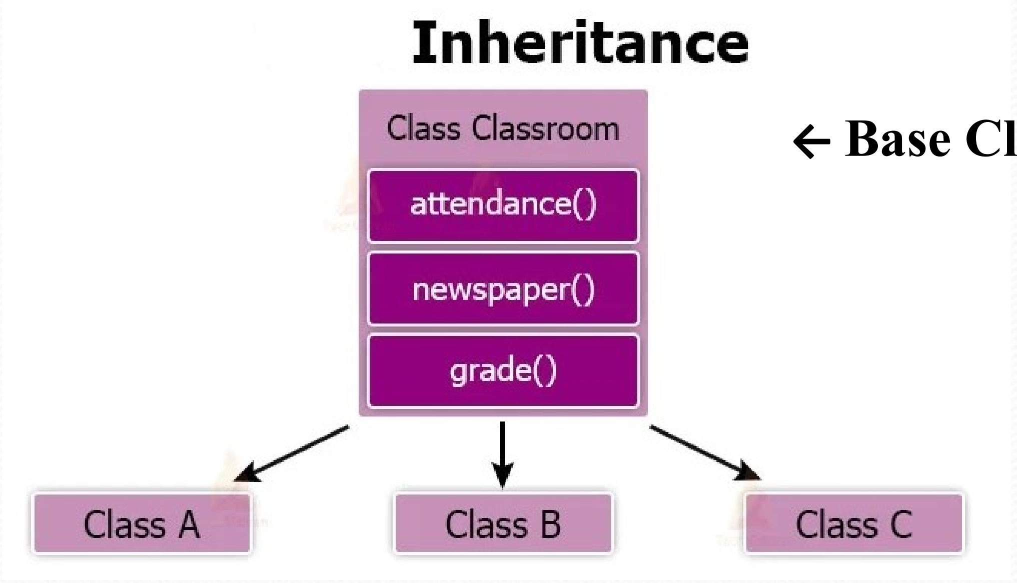
School bag is one of the most real examples of Encapsulation. School bag can keep our books, pens, etc.

Example2:

When you login into your email accounts such as Gmail, Yahoo Mail, or Rediffmail, there is a lot of internal processes taking place in the backend and you have no control over it.

Inheritance

The concept allows us to inherit or acquire the properties of an existing class (Base class) into a newly created class (Derived class). It is known as **inheritance**. It provides code reusability.



← Base Class or Super Class

← Derived Class or Sub class

Here Class A , Class B , Class C acquire the properties from its parent class i.e Classroom class .

Inheritance

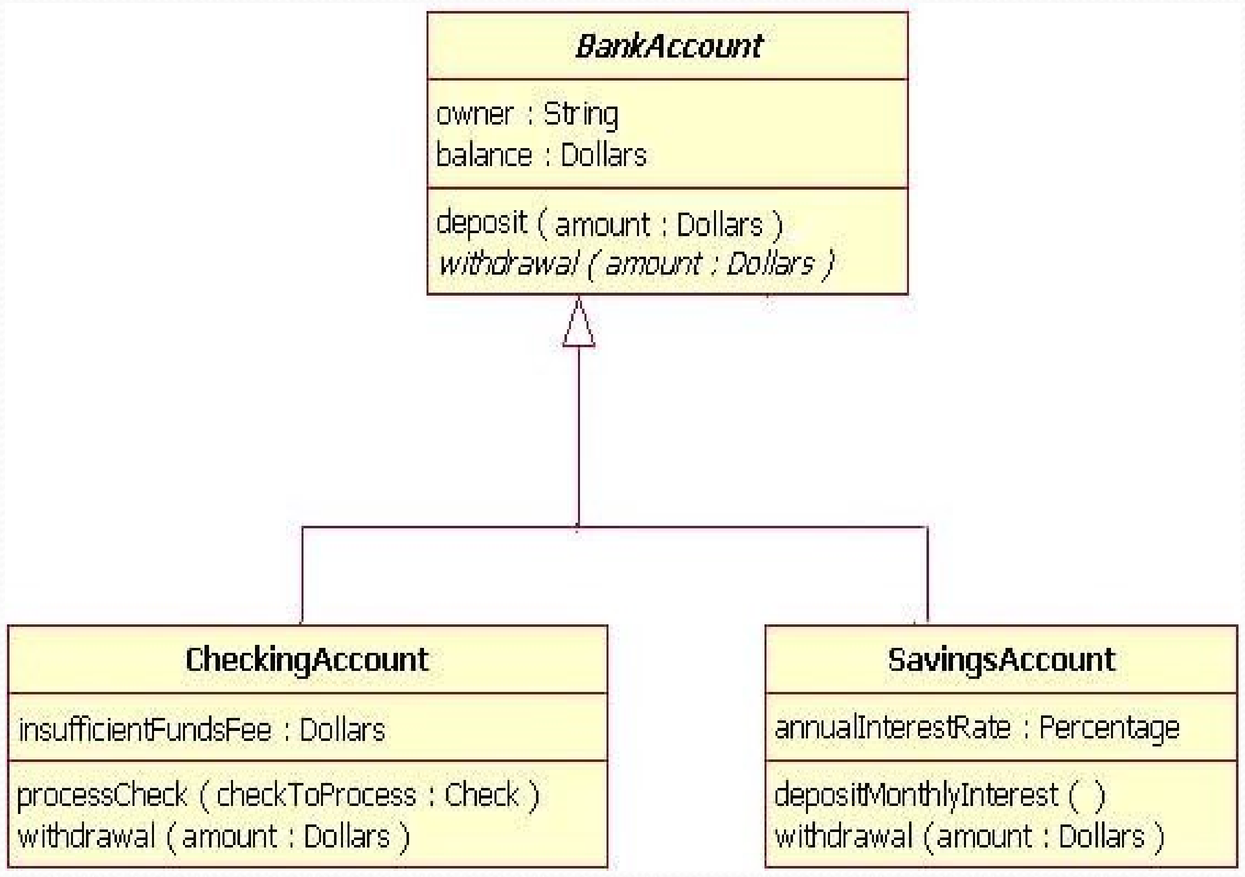
- It is the sharing of attributes and operations among classes based on a hierarchical relationship.
- Subclasses can be formed from broadly defined class.
- Each subclass incorporates or inherits all the properties of its super class and adds its own unique properties.

Example

- The real life example of inheritance is child and parents, all the properties of father are inherited by his son.

- We can also take the example of cars. The class ‘Car’ inherits its properties from the class ‘Automobiles’ which inherits some of its properties from another class ‘Vehicles’.

Inheritance

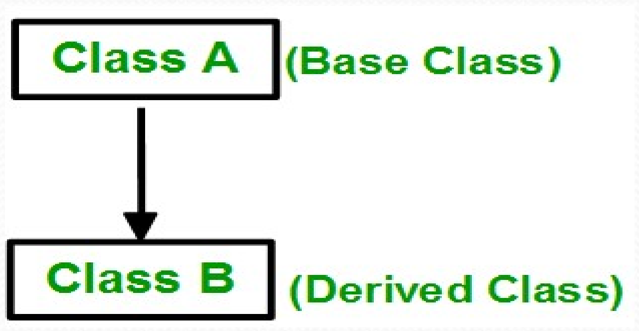


BankAccount
owner : String
balance : Dollars
deposit (amount : Dollars)
withdrawal (amount : Dollars)
CheckingAccount
insufficientFundsFee : Dollars
processCheck (checkToProcess : Check)
withdrawal (amount : Dollars)
SavingsAccount
annualInterestRate : Percentage
depositMonthlyInterest ( )
withdrawal (amount : Dollars)

Types of Inheritance

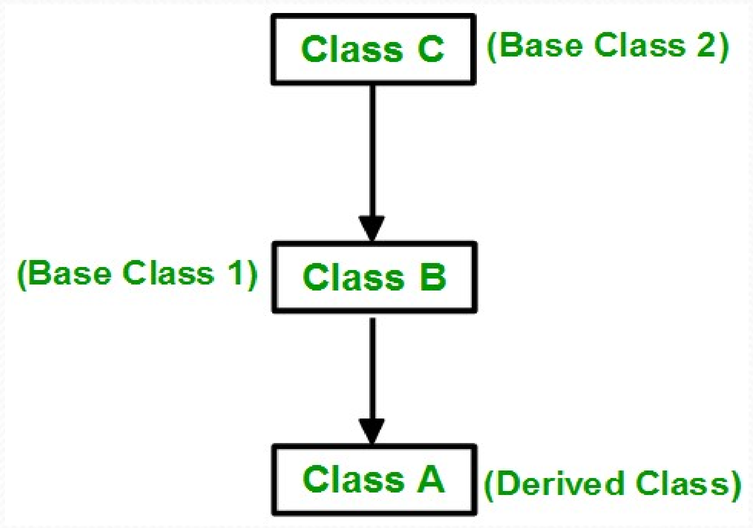
Single inheritance

In this inheritance, a derived class is created from a single base class.

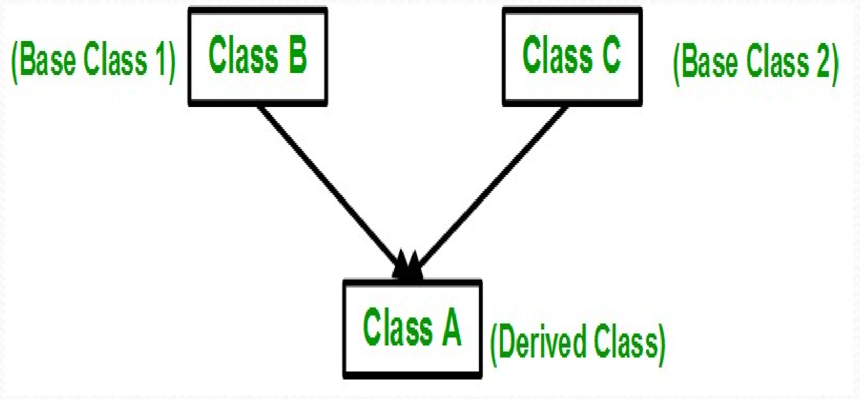


Multilevel inheritance

In this inheritance, a derived class is created from another derived class.

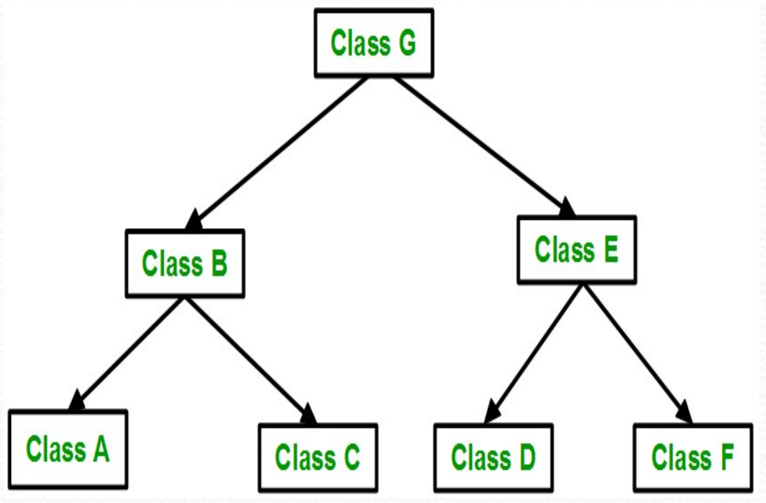


Multiple inheritance



In this inheritance, a derived class is created from more than one base class.

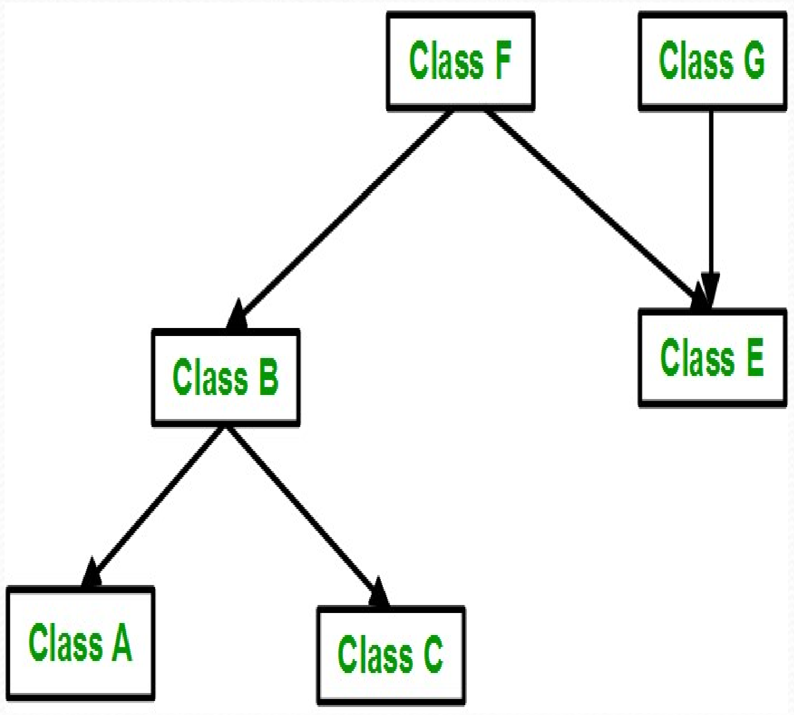
Hierarchical inheritance



In this inheritance, more than one derived class is created from a single base class and further child classes act as parent classes for more than one child class.

Hybrid Inheritance

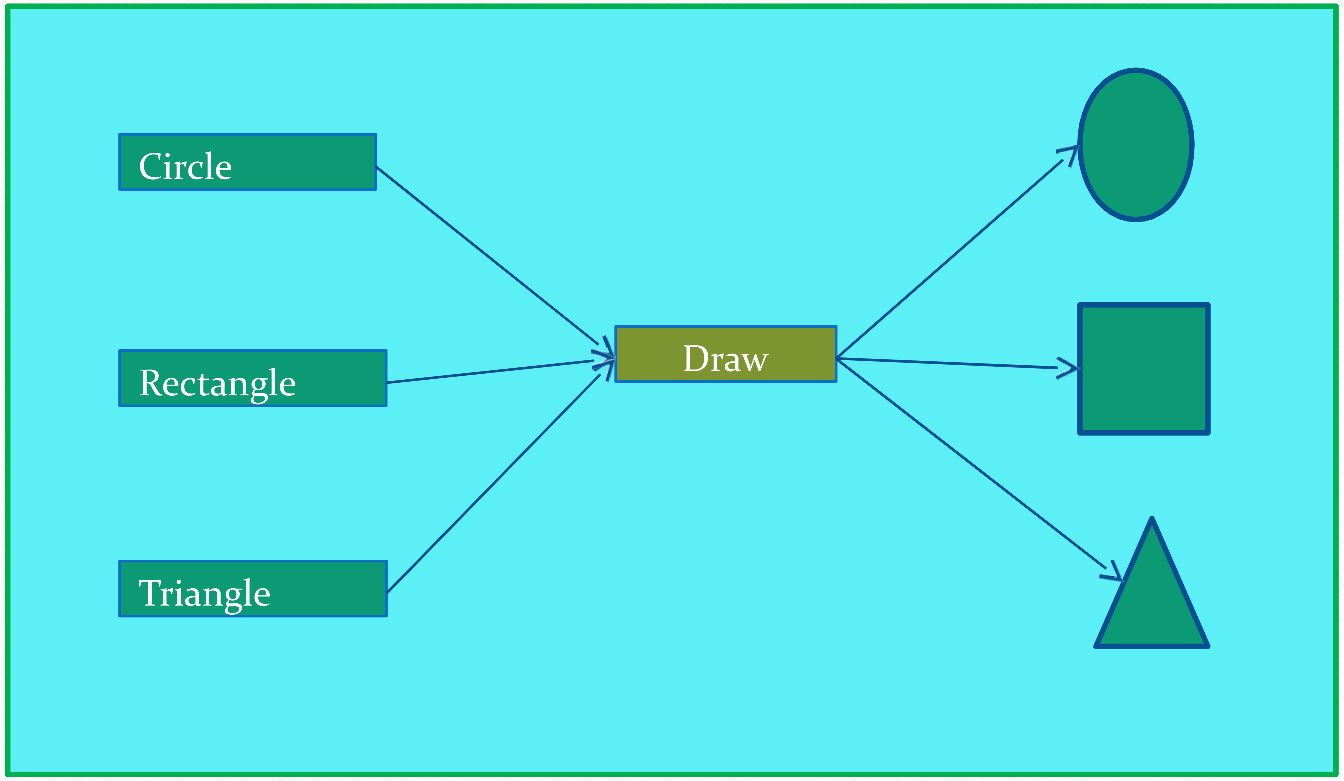
This is a combination of more than one inheritance. Hence, it may be a combination of Multilevel and Multiple inheritance or Hierarchical and Multilevel inheritance Hierarchical



Polymorphism

- The word **polymorphism** is derived from the two words i.e. **ploy** and **morphs**. Poly means many and morphs means forms.
- It means that the same operation (i.e. action or transformation that the object performs) may behave differently on different classes.
- Ability of different objects to response same message in different ways.
- It allows us to create methods with the same name but different method signatures. It allows the developer to create clean, sensible, readable, and resilient code.

Polymorphism



Types Of Polymorphism

polymorphism is mainly divided into two types:

Compile-time Polymorphism

In compile-time polymorphism, a function is called at the time of program compilation. We call this type of polymorphism as early binding or Static binding.

Function overloading and operator overloading is the type of Compile time polymorphism.

Runtime Polymorphism

Runtime polymorphism is also known as dynamic polymorphism or late binding. In runtime polymorphism, the function call is resolved at run time.

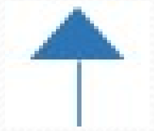
Function overriding and Virtual Function is a part of runtime polymorphism.

What is Generalization ?

- The process of extracting common characteristics from two or more classes and combining them into a generalized superclass is called Generalization. The common characteristics can be attributes or methods.

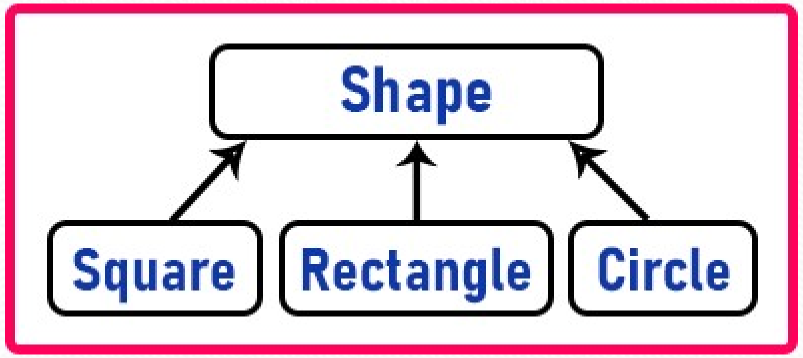
- The Generalization is the process that does the grouping entities into broader or distributed categories based on certain common attributes. All the common attributes bind together to form a higher-level component or element is called a generalized entity.

- Generalization is represented by a triangle followed by a line.



- **Examples:**

We have 3 classes that are Square, Rectangle, and Circle. Then for all of these, we are defining one class that is Shape. And Square, Rectangle, and Circle are inheriting from Shape.



- Among these classes which one is existing first? We know all the shapes i.e. square, rectangle and circle really exist in the real world. So, we are calling them shapes.

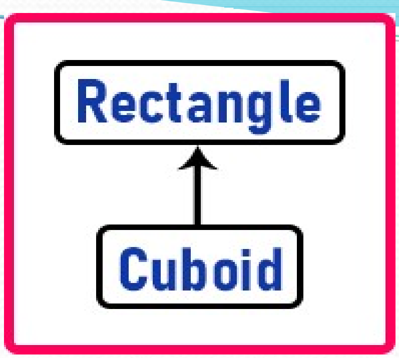
- Do you think Shape really exists in the real world? No. It is just a virtual term. If I asked you to show me some shape then you'll be drawing a rectangle, a circle, a square, a Rhombus, etc. but you cannot show me just a shape. So, the shape is what you're drawing. The Shape is a term or generic term or generalized term.

What is Specialization?

- Specialization is the reverse process of Generalization means creating new subclasses from an existing class.

- Specialization is the process of dividing a parent-level entity into narrower categories accordingly to all the possible child categories. By having the behavior of the opposite of the generalization process, specialization requires the separation of entities based on certain uncommon attributes.

Example



- Rectangle and Cuboid really exist in the world. Can you show me a rectangle? Yes, you can show me some shapes that is a rectangular or a paper that is rectangular or windowpane. So, rectangles exist in the real world.
- And what about 'Cuboid'? Yes, it also exists in the real world. Any box shape thing is a cuboid. So, these two things exist in the real world. Now in this. if we say, which was existing first in our example? The Rectangle was existing. From there we have derived a Cuboid. So, the Rectangle class is already existing and we have defined a new class with extra features so we have a **specialized class** that is Cuboid. This is a specialization

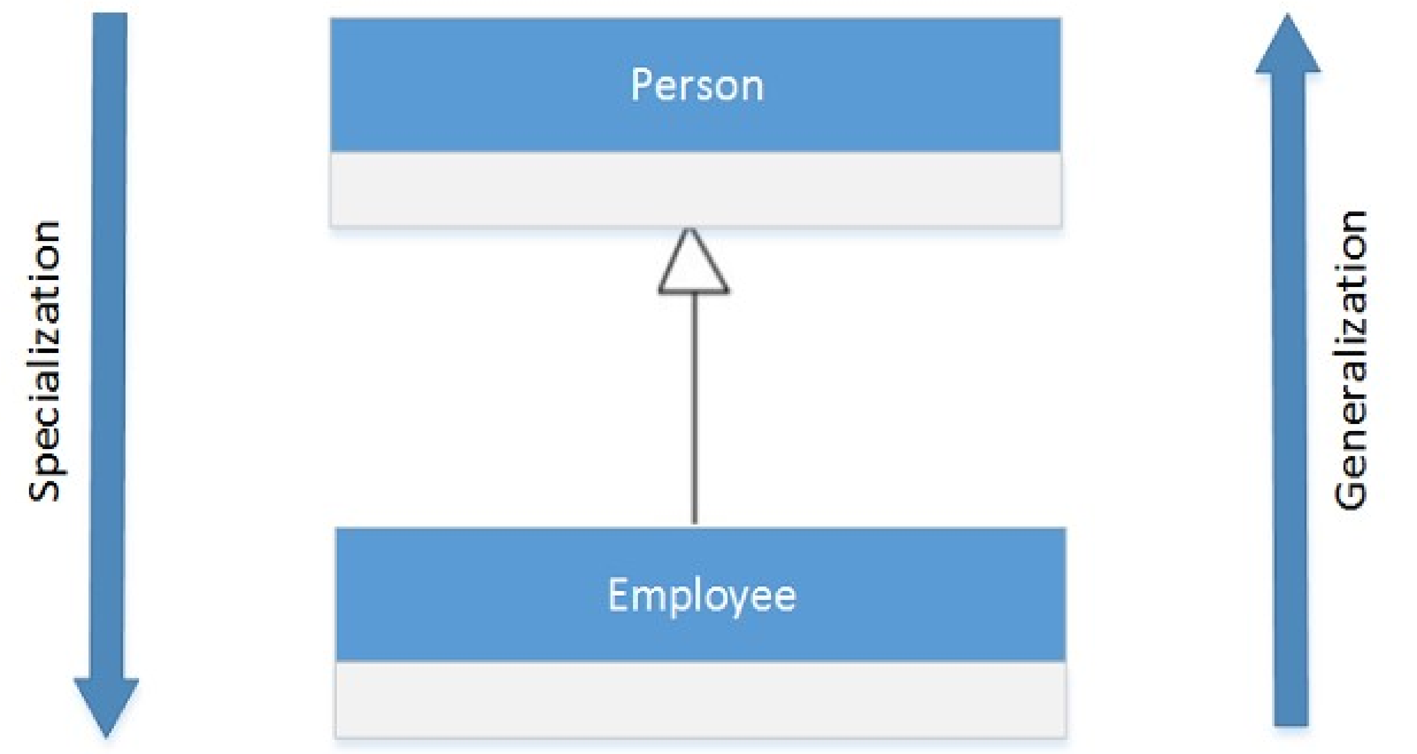
Difference between Generalization and Specialization

- Generalization works in a bottom-up manner while the specialized works complete ever than the generalization where it follows the top-down approach

- The entity of the higher-level must always have the lower-level entity in Generalization whereas, in Specialization, the entities of the top-level may not consist of the entities of the lower level periodically.

- Generalization can be defined as a process where the grouping is created from multiple entity sets, and the Specialization takes a sub-set of the higher level entity and it formulates a lower-level entity set.

- Generalization helps to reduce the schema of the data by unifying the multiple components. and in Specialization, it expands the schema by multiplying the multiple components
- Generalization leads to a reduction in the overall size of a schema and focuses on the specialization, on the other side, Specialization increase the size of schemas



The Employee class extends the Person class (Inheritance)

Template/ Generosity

It is a process that allows a function or a class to work with different data types.

That type of function is called template function

What is Modeling

- Modeling consists of building an abstraction of reality.
- Abstractions are simplifications because:
    - They ignore irrelevant details and
    - They only represent the relevant details.
- What is *relevant* or *irrelevant* depends on the purpose of the model.

What is a Model

A model is a simplification of reality.

A model may provide

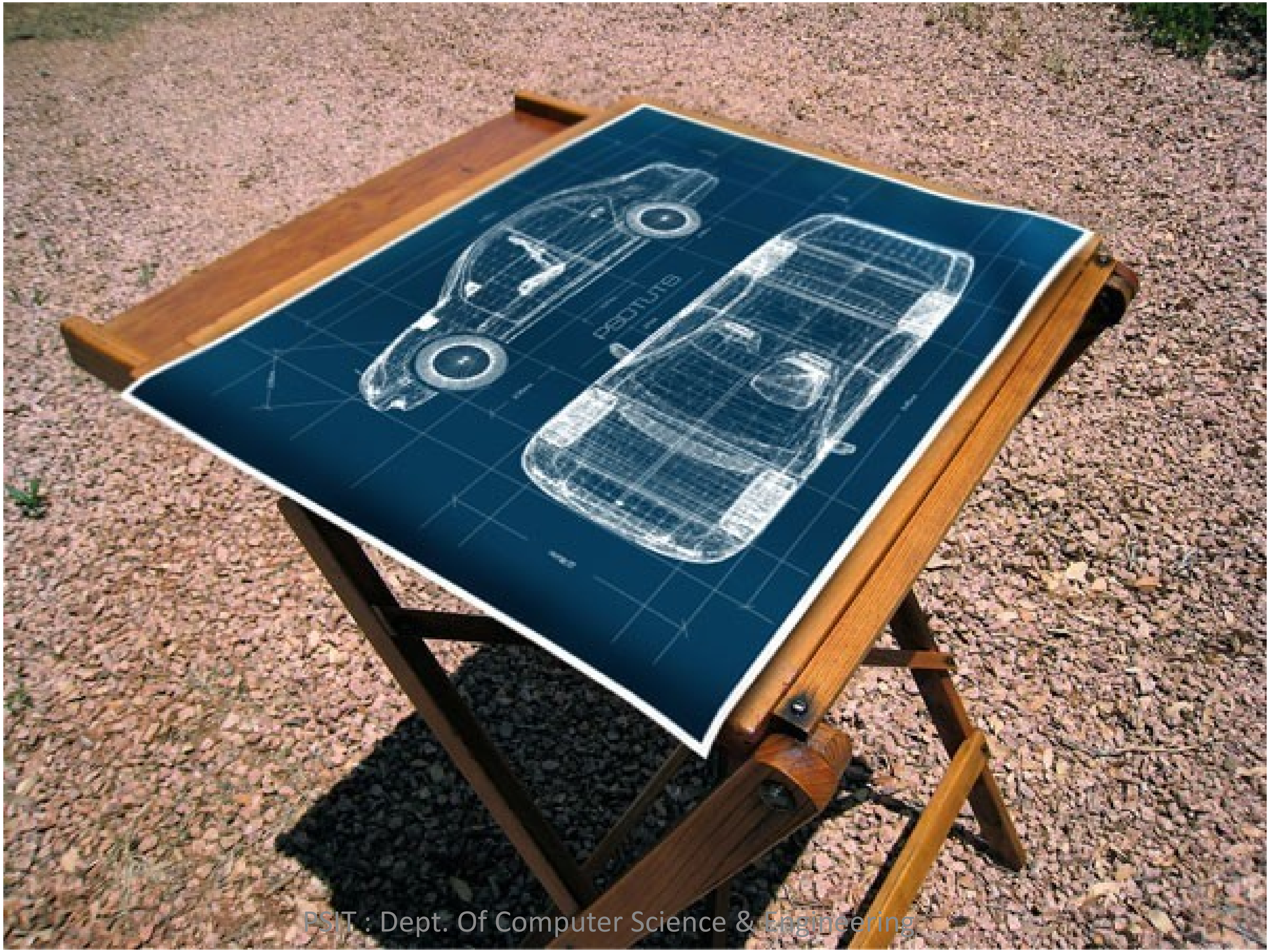
- blueprints of a system
- Organization of the system
- Dynamic of the system

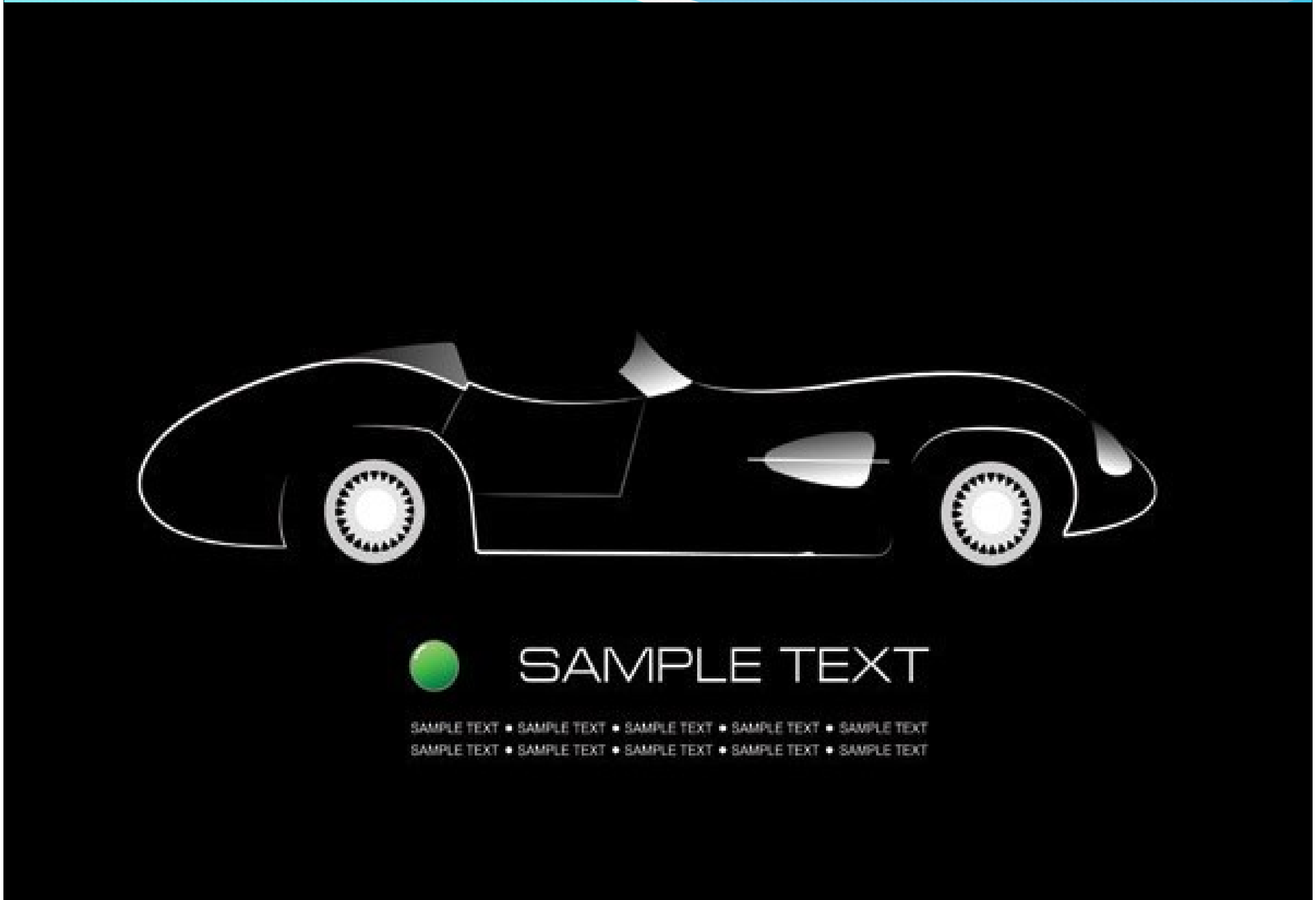
MODEL

- **A model is an abstraction, before building any system a prototype may be developed**. The main purpose of model is for understanding of the system.

- Designer build different kinds of models for various purposes before constructing things.

- For example car , airplane, blueprints of machine parts, Plan for house construction etc., Models serve many purposes




SAMPLE TEXT
SAMPLE TEXT • SAMPLE TEXT • SAMPLE TEXT • SAMPLE TEXT • SAMPLE TEXT
SAMPLE TEXT • SAMPLE TEXT • SAMPLE TEXT • SAMPLE TEXT • SAMPLE TEXT

Importance of modeling:

Models help us to visualize a system as it is or as we want it to be.

Models permit us to specify the structure or behavior of a system.

Models give us a template that guides us in constructing a system.

Models document the decisions we have made.

Purpose of Modeling

Designers build many kinds of models for various purposes before constructing things.

Models serve several purposes –

- Testing a physical entity before building (simulation)
- Communication with customer
- Visualization
- Reduction of complexity
- Better understanding of the problem

Purpose of Modeling

Communication with customers

Architects and product designers build models to show their customers. Mock ups are demonstration products that imitate some or all of the external behaviour of a system.

Purpose of Modeling

Visualization

Storyboards of movies, television shows and advertisements let writers see how their ideas flow. They can modify awkward transitions, and unnecessary segments before detailed writing begins.

Purpose of Modeling

Reduction of complexity

The main reason for modeling is to deal with systems that are too complex to understand directly. Models reduce complexity by separating out a small number of important things to deal with at a time.

Three models

We use three kinds of models to describe a system from different view points.

1. **Class Model for the objects in the system & their relationships.**

2. **State model—for the life history of objects.**
   Show how systems behave internally

3. **Interaction Model—for the interaction among objects.**
   Show the behaviour of systems in terms of how objects interact with each other

Principles of modelling

Four basic principles of modeling.

1. The choice of what models to create has a profound influence on how a problem is attacked and how a solution is shaped.

2. Every model may be expressed at different levels of precision.

3. The best models are connected to reality.

4. No single model is sufficient. Every non-trivial system is best approached through a small set of nearly independent models.

Principles of modeling:

1. ***The choice of what models to create has a profound influence on how a problem is attacked and how a solution is shaped.***
   - “choose your models well”.
   - The right models will brilliantly illuminate the most wicked development problems.
   - The wrong models will mislead you, causing you to focus on irrelevant issues.

2. ***Every model may be expressed at different levels of precision.***
   - The best kinds of models are those that let you choose your degree of detail.
   - An analyst or an end user will want to focus on issues of what;
   - A developer will want to focus on issues of how.
   - Both of these stakeholders will want to visualize a system at different levels of detail at different times.

3. The best models are connected to reality.

- In software, the Achilles heel (weak point) of structured analysis techniques is the fact that there is a basic disconnect between its analysis model and the system's design model.

- In object-oriented systems, it is possible to connect all the nearly independent views of a system into one semantic whole.

4. No single model or view is sufficient. Every nontrivial system is best approached through a small set of nearly independent models with multiple viewpoints.

- To understand the architecture of such a system, you need several complementary and interlocking views.
- A use case view (exposing the requirements of the system),
- A design view (capturing the vocabulary of the problem space and the solution space).
- An interaction view (showing the interactions among the parts of the system and between the system And the environment).
- A deployment view (focusing on system engineering issues).

Object Oriented Modeling

In software, there are several ways to approach a model. The two most common ways are

An algorithmic perspective

An object-oriented perspective.

In **Algorithmic approach,** the main building block of all software is the procedure or function.

This view leads developers to focus on issues of control and the decomposition of larger algorithms into smaller ones.

As requirements change (and they will) and the system grows (and it will), systems built with an algorithmic focus turn out to be very hard to maintain.

The contemporary view of software development takes an object-oriented perspective.

In this approach, the main building block of all software systems is the object or class.

Simply put, an object is a thing, generally drawn from the vocabulary of the problem space or the solution space.

Every object has identity (you can name it or otherwise distinguish it from other objects), state (there's generally some data associated with it), and behavior (you can do things to the object, and it can do things to other objects as well).

Ex. Consider simple three-tier architecture for a billing system, involving a user interface, business services, and a database.

- In the user interface, you will find concrete objects, such as buttons, menus, and dialog boxes.
- In the database, you will find concrete objects, such as tables representing entities from the problem domain, including customers, products, and orders.
- In the middle layer, you will find objects such as transactions and business rules, as well as higher-level views of problem entities, such as customers, products, and orders.

Object Oriented Modelling

- The object oriented modeling is a way of constructing visual models based on real world object.
- Modeling helps in understanding the problems, develop proper documents and producing well designed program.
- Object-oriented modeling (OOM) is the construction of objects using a collection of objects that contain stored values of the instance variables found within an object.

In OOM the modeling passes through the following processes:

- System Analysis
- System Design
- Object Design
- Final Implementation.

- **System Analysis:** In this stage a statement of the problem is formulated and a model is build by the analyst in encouraging real-world situation. This phase show the important properties associated with the situation.

- **System Design:** At this stage, the complete system architecture is designed. This is the stage where the whole system is divided into subsystems, based on both the system analysis model and the proposed architecture of the system.

- **Object Design:** At this stage, a design model is developed based on the analysis model which is already developed in the earlier phase of development.

- **Final Implementation:** At this stage, the final implementation of classes and relationships developed during object design takes place a particular programming language, database, or hardware implementation (if needed).