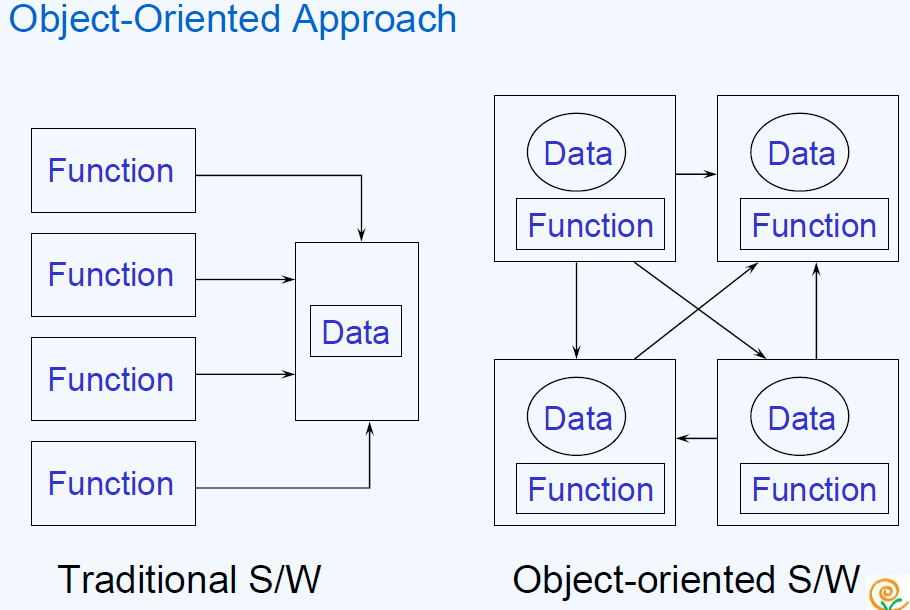
Benefits of OOP:

* Create manageable projects more easily.
* OOP provides facilities to create reusable objects
* Almost all big applications are written using the object oriented approach.
* Reusability: Prevent DRY (Don’t Repeat Yourself).
* Refactoring: All objects are small entities and contain its properties and methods as a part of itself. So refactoring is comparatively easier.
* Extensible: We can extend the object and create a totally new object that retains all the necessary properties and methods of the parent object from which it has been derived, and then expose new features.
* Maintenance: Very easy to merge the code.
* Efficiency: The concept of object oriented programming is actually developed for better efficiency and ease of development process. Because you first split your problem into a small set of problems and then find solutions for each of them, the big problem is solved automatically.

Why Object oriented Programming?

* Object oriented technology offers a new model where data and processes are integrated.
* Data and processes are packaged together to form an object.
* The object contains all the information and the data which is required to provide the functionality for which it is designed.



The Object-Oriented Paradigm

• Collects the data with their associated functions

• Collections communicate via messages

Message Communication:

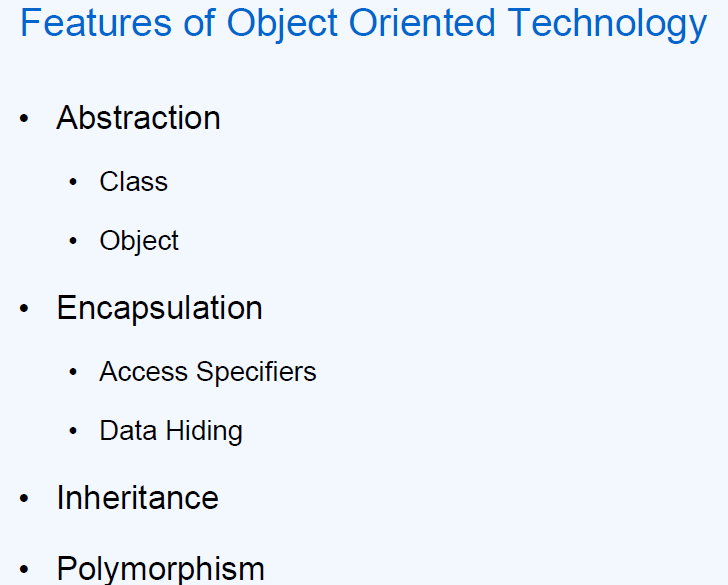
Object oriented programming consists of a set of objects that communicate with each other. The process of programming in an object oriented language therefore involves the following basic steps:

1. Creating classes that defines objects and their behavior.

2. Creating objects from class definitions.

3. Establishing communication among objects.

Objects communicate with one another by sending or receiving messages. The concept of message passing makes it easier to talk about building systems that directly model or simulate the real-world. A message for an object is a request for execution of a procedure/function. The receiving object will generate the desired result according to the procedure/function call.



• Abstraction -> It denotes the essential characteristics of an object that distinguish it from all other kind of objects and thus provide crisply defined conceptual boundaries, relative to the perspective of the viewer. It focuses on observable behavior of an object.

* Class -> It is a set of objects that share a common structure and a common behavior.
* Data Member
* Member Methods -> denotes the service that a class offers to its client.
* Object -> Object is an instances of a class. Thus an object is an entity that has attributes and provide certain operations that are defined for the particular object.

• Encapsulation -> Focuses upon the implementation that gives rise to the observable behavior.

* Access Specifies
* Data Hiding

• Inheritance -> Opens the possibility of extending and adapting object description and implementation without changing their source code.

• Polymorphism

* Static and Dynamic Polymorphism

Abstraction

* Focuses upon the essential elements and characteristics of some object, relative to the perspective of the viewer
* Abstraction means putting all the variables and methods in a class which are necessary for example – Abstract class and abstract method.
* Abstraction is the common thing.

Example

* If somebody in your college tell you to fill application form, you will fill your details like name, address, date of both which semester, percentage you get etc.
* If some doctor gives you an application to fill the details like name, address, date of birth, blood group, height and weight.
* See in the above example what is the common thing?
* Age, name, address so we can create the class which consist of common thing that is called abstract class.
* That class is not complete and it can inherit by other class.

Example

Take an entity as BOOK. Let us try to find out the different characteristics of the same entity from the perspective of the viewer.

• Let us take 2 cases where the same entity BOOK can be viewed differently:

• Library System

• In this case we will be focusing on Access Number, Book Name, Author Name

• Shopkeeper

• In this case we will be focusing on Item Number, Item Name, Price, and Quantity On Hand.

An object

• Is a unique, identifiable, self-contained entity that possesses operations and contains attributes?

• Possesses all the know-how and information it needs to perform the services for which it was designed

• Is a "black box" which receives and sends messages?

* Object is an instance of a class.

Every object:

* Contains data: The data stores information that describes the state of the object.
* Has a set of defined behavior. This behavior consist of all the things that the object "knows" how to do. These are the methods present inside the object.
* Has an individual identity. Each object is different from the other object even if they are instantiated from the same class.

A Class

* Is a software template that defines the methods and variables to be included in a particular kind of Object?
* Is a blue print used to create objects? As it is a blue print, at run time it will not occupy any memory.
* Examples : Animal, Human being, Automobiles

We never actually write the code for an object: what you write is the classes that is used to make objects.

Classes increase the efficiency and power of the object by:

* Classifying objects
* Relating objects to one another
* Providing a mechanism to define and manage objects

Class Contains -

• Member Data

* Variables defined inside a class
* Normally Member data are hidden

• Member Methods

* Functions defined inside the class
* Member methods are public in nature and accessible from outside the class

Encapsulation

• A software development technique that consists of isolating a system function or a set of data and operations on those data within a module and providing precise specifications for the module -IEEE

• Principle of binding processes (Member Method) and data (Member Data) to form an integrated unit is the essence of encapsulation

• Implemented by using the concept of Class.

Encapsulation

* The wrapping up of data and functions into a single unit (called class) is known as encapsulation.
* The data is not accessible to the outside world and only those functions which are wrapped in the class can access it.
* Encapsulation containing and hiding information about an object, such as internal data structures and code.
* Encapsulation means hiding the internal details of an object, i.e. how an object does something.
* Encapsulation prevents clients from seeing it’s inside view, where the behavior of the abstraction is implemented.
* Encapsulation is a technique used to protect the information in an object from the other object.
* Hide the data for security such as making the variables as private and expose the property to access the private data which would be public.

Encapsulation is "a technique that separates the external aspects of an object form the internal implementation.

• A "capsule" or module, packages together functions and the related data, and protects its data from other modules or outside functions.

• Best example here can be: the difference between “Capsule” and “Tablet”. Here we can see that “Tablet” is exposed where as “Capsule” is within a package.

• The main objective of Object oriented programming is to hide the data item

• Accessibility of data item with a class is only possible through member methods.

• Abstraction & Encapsulation:

• Abstraction tells us what external face we should present to the world where as Encapsulation ensures that the implementation of the interface doesn’t leak out to the outside world.

• Encapsulation hides the details of how we implement the state behind the behavior which we present to the outside world.

Real world of Encapsulation

* Let’s take an example of Mobile phone and Mobile phone manufacturer.
* Suppose you are a Mobile phone manufacturer and you designed a Mobile phone design(class), now by using machinery we are manufacturing a Mobile phone (object) for selling, when we sell your mobile phone the user only know how to use mobile phone but not that how this mobile works.
* I also add an example like ATM machine. The user punch his card and get the money but he does not know what type of work is done inside the ATM machine.

Advantage:

* Prevents the data from unwanted access.
* Safe from outside interference and misuse.

Data Hiding

• Process of hiding the members from outside the class

• Implemented by the concept of “Private” access specifiers

• Can be accessed only by the member methods of that class where it is defined

• Data hiding is an important feature of OO programming which allows the data member to be kept safe

* This is also called as information hiding.

Inheritance

• Ability to compose new abstraction from existing one

• Implements the concept of Re-usability

• Classes are arranged in a tree like structure called a hierarchy

• Base class:

* The class providing the implementation

• Derived class:

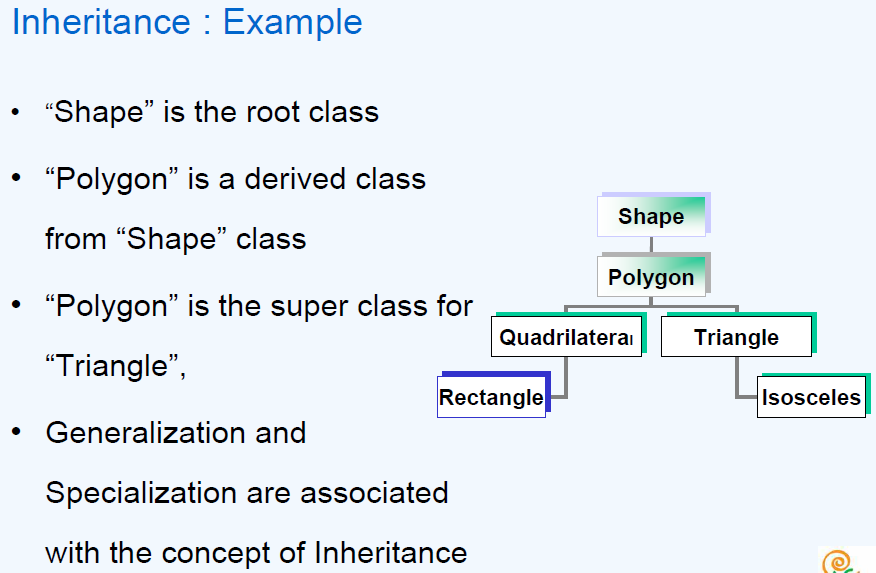
* The class inheriting the implementation

Inheritance also promotes reuse. You don't have to start from scratch when you write a new program. You can simply reuse an existing repertoire of classes that have behaviors similar to what you need in the new program.

• Base class is also called Super Class and Derived class is called Child class.

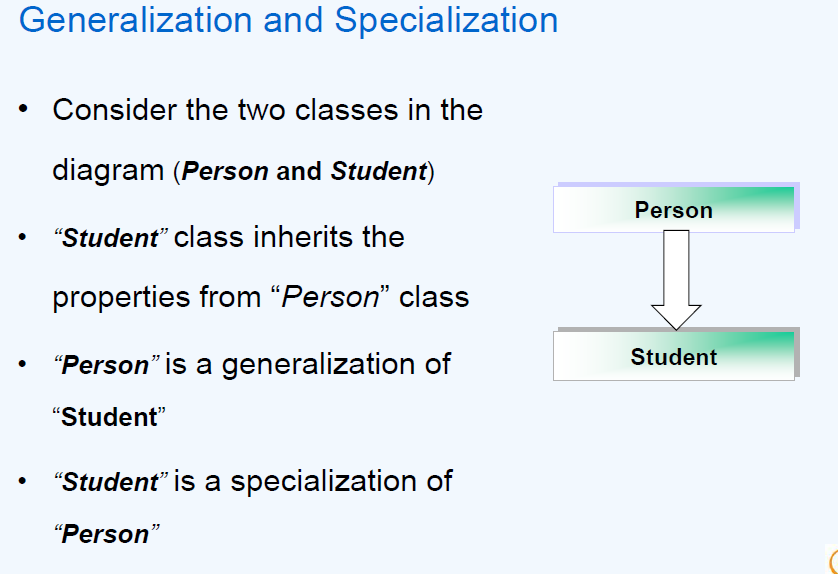
• Derived class can also override the inherited method

• By using the concept of inheritance, it is possible to create a new class from an existing one and add new features to it. Thus inheritance provides a mechanism for class level reusability.



• In a class hierarchy, every class has a superclass except for the class at the top of the hierarchy; this class is called the root class

• In our example, Shape is the root of the hierarchy. From the other end, the classes that don’t have any subclasses, like Rectangle and Isosceles are called leaf classes.

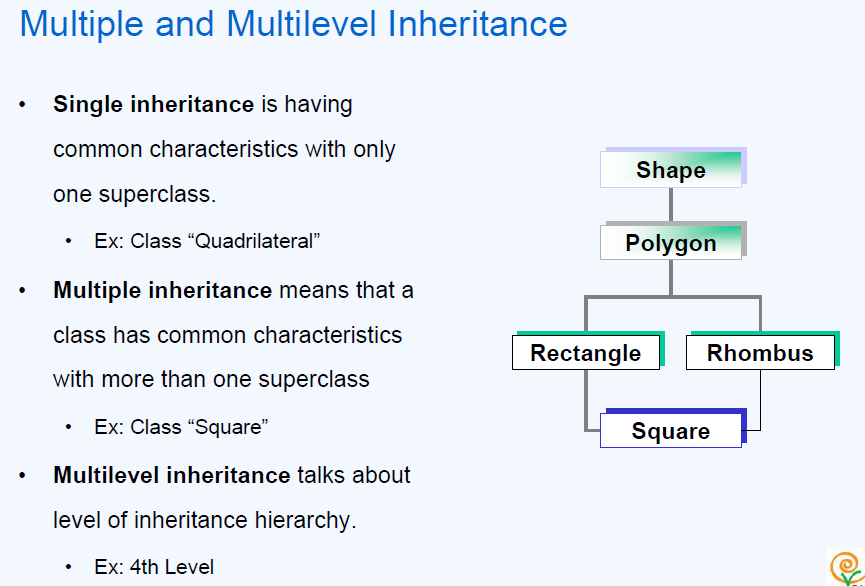


Specialization is the process as we go down the class hierarchy

* of maximizing the differences between members of an entity by identifying their distinguishing characteristics
* of defining subclasses that are more refined than superclass – its specialized

Generalization is the process as we go up in the class hierarchy

* of minimizing the differences between entities by identifying their common characteristics
* of defining Superclass which contains common characteristics of its subclasses – its generalized



* Single inheritance, where each class (except the root) has one and only one superclass.

• Multiple inheritance, where a class inherits from more than one superclass.

• Multilevel inheritance, where class hierarchy goes for more than one level. The level of hierarchy in the example shown in the slide above is 4.

Advantages of Inheritance

• Software reusability

• Design and code sharing

• Software components

• Rapid prototyping

Abstract Class

• A class that contains one or more abstract methods (methods which are not complete in terms of implementation)

• Other classes can extend them and make them concrete by implementing the abstract methods

Abstract Class:

At times some of the base classes in our hierarchies do not represent anything concrete enough to instantiate an object in their own rights. Such a class only exists as a ‘holder’ for the shared (inherited) attributes and methods of derived classes and is known as an ‘abstract’ class, because it does not represent a concrete type of object.

Example of Abstract Class:

When we try to think of designing a class for animal, we know all animals exhibit attributes called color and speed. But when we try to initialize these two attributes in methods, then we don't know with what value to initialize them. If it is not a specialized class, it is very difficult to initialize these attributes. In C++ we can define an Abstract class as follows:

class animal

{

Private:

int speed;

char color[20];

public:

virtual void init\_speed()=0;

virtual void init\_color()=0;

}; // Class definition ends here

Abstract Class (contd..)

• Instantiation of an abstract class is not possible

Polymorphism

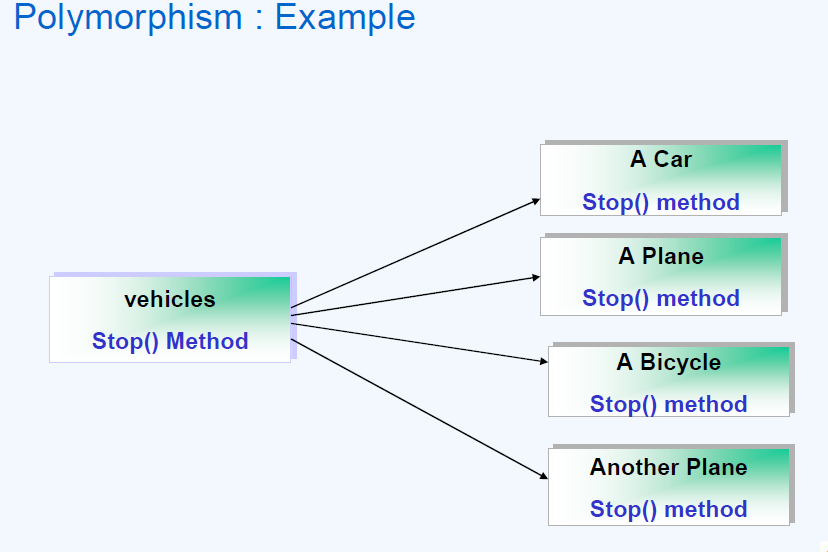
• Is the ability of different objects to respond in their own unique way to the same message?

• Implemented by overloading and overriding

• Possible because of Encapsulation and Inheritance

• Example of a class hierarchy where vehicles is the base class and there are some classes which are derived from vehicles. (Car, Plane, Bicycle)

• Each of the classes need a functionality called “Stop”



Explanation of the above Example:

* All vehicles offered an operation “stop”.
* Implementation of the “stop” method will be different for different classes.
* At runtime user will invoke a method “stop”.
* The effect is that upon receiving the message the object knows to which class it belongs to, and thus executes the “stop” method for its own class.

• Message is the same called “stop” but which method is going to get invoked depends upon which type of class the object is sending the message.

• Message is the same but the execution might vary depending upon the derived class implementation.

Polymorphism

• Different Object Oriented Languages use different concepts of implementing polymorphism

* Function Overloading
* Operator Overloading
* Function Overriding

Polymorphism

* Polymorphism means one name having multiple forms.
* It allows us to have more than one function with the same in a program.
* An operation may exhibit different behaviors in different instances.
* The behavior depends upon the types of data used in the operation.

Example

* Consider the operation of addition. For two numbers, the operation will generate a sum. If the operands are strings, the operation would produce a third string by concatenation.

Function/Method Overloading

• Ability to define two or more methods with the same names and different signatures

• Signature means number of arguments, type of arguments and sequence of arguments

• Ex:

1. void disp(int ix);

2. void disp(int ix,int iy);

3. void disp(int x,char cy);

4. void disp(char cx,int iy);