Object Oriented Programming:

1. Concept of Object Oriented Programming –
2. Data hiding
3. Data encapsulation
4. Class and Object
5. Abstract class
6. Concrete class
7. Polymorphism (Implementation of polymorphism using(Function overloading as an example in C++)
8. Inheritance
9. Advantages of Object Oriented Programming over earlier programming methodologies

Basic Concept

Let's get an idea of what is OOPS. C++ & Java are the Object Oriented Programming language. The limitations of C language are evident when a software project is too large to be handled. In some circumstances, error occurs at some place and some patch up programs are added to rectify that error. These programs introduce an unexpected error at some other location and this process continues.
In 1980 Bjarne Stroustrup addressed this problem by adding several extensions to C language. The most important addition was the concept of CLASS. The addition made were mainly aimed at extending the language in such a way that it supports object-oriented programming.

Definition: "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."

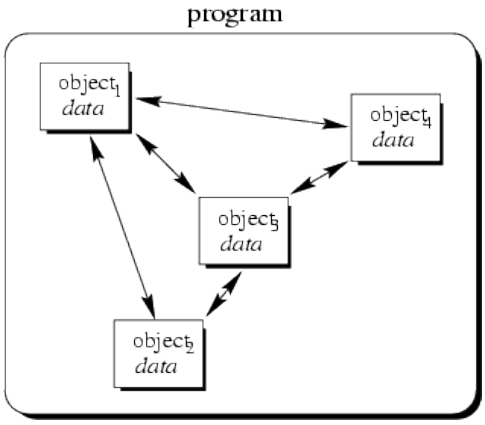
Thus, an object is considered to be partitioned area of computer memory that stores data and set of operations that can
access that data. Since the memory partitions are independent, the objects can be used in a variety of different programs
without modifications.

Characteristics of Oriented Programming

· Emphasis is on data rather than procedure.
· Programs are divided into what are known as objects.
· Data structures are designed such that they characterize the objects.
· Functions that operate on the data of an object are tied together in the data structure.
· Data is hidden and cannot be accessed by external functions.
· Objects may communicate with each other through functions.
· New data and functions can be easily added whenever necessary.
· Follows bottom-up approach in program design.

OOP: In general Terms

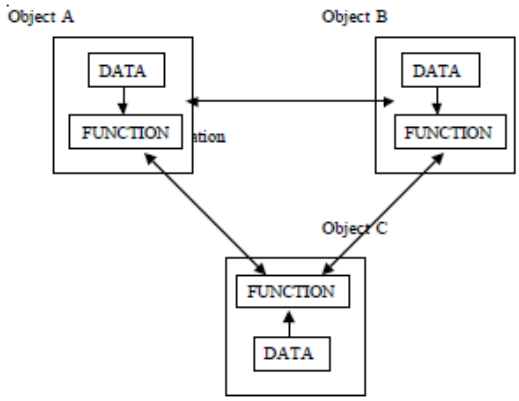
Object-orientation or object oriented programming (OOP) is introduced as a new programming concept which should help one in developing high quality software. Object-orientation is also introduced as a concept which makes developing of projects easier. Object oriented programming attempts to solve the problems with only one approach; dividing the problems in sub-modules and using different objects. Objects of the program interact by sending messages to each other. The drawing below illustrates this clearly -



Object Oriented Programming Pattern

Inspiring factor in the invention of object-oriented approach is to remove some of the flaws encountered in the procedural approach. OOP treats data as a critical element in the program development and does not allow it to flow freely around the system. It ties data more closely to the functions that operate on it, and protects it from accidental modification from outside functions. OOP allows decomposition of a problem into a number of entities called objects and then builds data and functions around these objects.

Following fig shows the organization of data and functions in object-oriented programs .The data of an object can be accessed only by the functions associated with that object. However, functions of one object can access the functions of other objects.



Organization of data and functions in OOP

<u>To understand the actual concept of object orientation and the OOP, we should first be acqiented with the basic concepts of OOP</u>

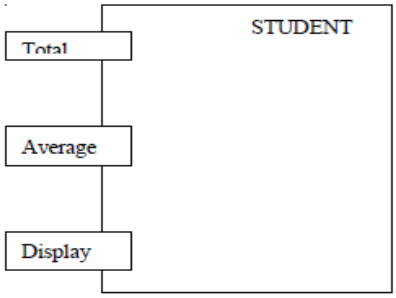
The basic terms that you ought to know about OOP concept are-

1. Objects
2. Classes
3. Inheritance
4. Polymorphism
5. Data abstraction
6. Data encapsulation
7. Dynamic binding
8. Message passing

- **Objects**

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. They may also represent user-defined data.
When a program is executed, the objects interact by sending messages to one another. For example, if "customer" and "account" are two objects in a program, then the customer object may send a message to the account object requesting for the bank balance. Each object contains data, and code to manipulate the data. Objects can interact without having to
know details of each other's data or code. It is sufficient to know the type of message accepted, and the type of response
returned by the objects. Although different authors represent them differently.

Fig. Shows two notations that are popularly used in objectoriented analysis and design.



Classes
Objects contain data, and code to manipulate that data. The entire set of data and code of an object can be made a userdefined 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. Each object is associated with the data of type class with which they are created.
Definition
A class is a collection of objects of similar type.
For e.g., mango, apple and orange are members of the class fruit. Classes are user-defined data types and behave like the
built-in types of a programming language. The syntax used to create an object is no different than the syntax used to create an integer object in C. If fruit has been defined as a class, then the
statement
Fruit mango;
will create an object mango belonging to the class fruit.

Data Abstraction and Encapsulation
The wrapping up of data and functions into a single unit (called class) is known as encapsulation. Data encapsulation is the most striking feature of a class. The data is not accessible to the outside world, and only those functions, which are wrapped in the class, can access it. These functions provide the interface between the object's data and the program. This insulation of the data from direct access by the program is called data hiding or information hiding.

Abstraction refers to the act of representing essential features without including the background details or explanations.
Classes use the concept of abstraction and are defined as a list of abstract attributes such as size, weight and cost, and
functions to operate on these attributes. They encapsulate all the essential properties of the objects that are to be created. The attributes are sometimes called data members because they hold information. The functions that operate on these data are sometimes called methods or member functions.

Since the classes use the concept of data abstraction, they are known as Abstract Data Types (ADT)

Inheritance

Inheritance is the process by which objects of one class acquire the properties of objects of another class. For example, the bird 'robin' is a part of the class "flying bird' which is again a part of the class 'bird'. The prin-ciple behind this sort of division is that each derived class shares common characteristics with the class from which it is derived as illustrated in Fig.

Concept of inheritance provides the idea of reusability. This means that we can add additional features to an existing class without modifying it. This is possible by deriving a new class from the existing one. The new class will have the combined features of both the classes. The real appeal and power of the inheritance mechanism is that it allows the programmer to reuse a class that is almost, but not exactly, what he wants, and to tailor the class in such a way that it does not introduce any undesirable side-effects into the rest of the classes.

Note that each sub-class defines only those features that are unique to it. Without the use of classification, each class would have to explicitly include all of its features.

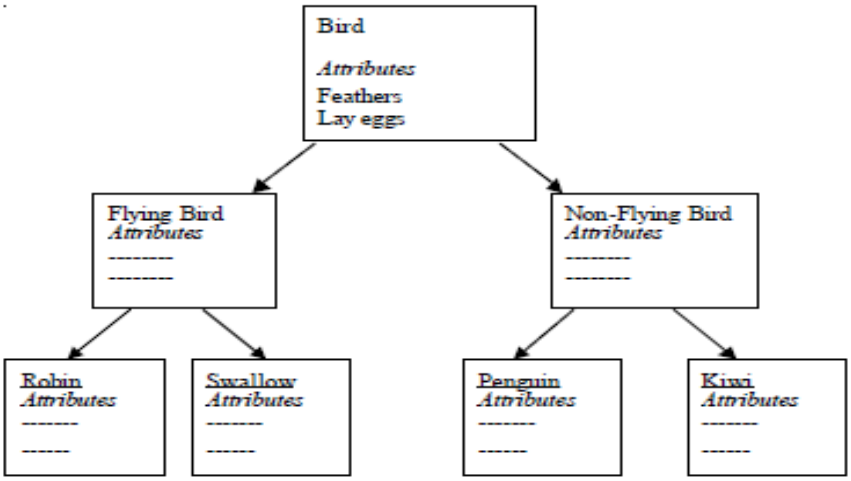


Fig. Property inheritance

Polymorphism

Polymorphism, means the ability to take more than one form. An operation may exhibit different behaviors in different
instances. The behavior depends upon the types of data used in the operation.

For e.g., consider the operation of addition of two numbers, the operation will generate a sum. If the operands are strings, then the operation would produce a third string by concatenation.

The process of making an operator to exhibit different behaviors in different instances is known as operator overloading.

Figure below illustrates that a single function name can be used to handle different number and different types of arguments. This is something similar to a particular word having several different meanings depending on the context. Using a single **function name to perform different types of tasks is known as function overloading.**

Polymorphism plays an important role in allowing objects having different internal structures to share the same external
interface. This means that a general class of operations may be accessed in the same manner even though specific actions
associated with each operation may differ. Polymorphism is extensively used in implementing inheritance.

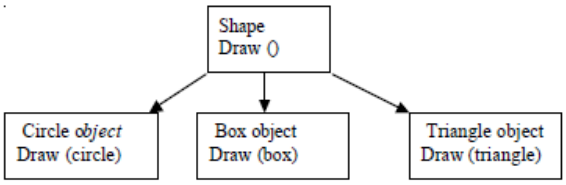


Fig. Polymorphism

Dynamic Binding

Binding refers to the linking of a procedure call to the code to be executed in response to the call. Dynamic binding (also
known as late binding) means that the code associated with a given procedure call is not known until the time of the call at run-time. It is associated with polymorphism and inheritance.
A function call associated with a polymorphism reference depends on the dynamic type of that reference.
E.g. Consider the procedure "draw" in Fig. By inheritance, every object will have this procedure. Its algorithm is, however, unique to each object and so the draw procedure will be redefined in each class that defines the object. At run-time, the code matching the object under current reference will be caned.

Message Passing

An object-oriented program 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 define objects and their behavior,
2. Creating objects from class definitions, and
3. Establishing communication among objects.

Objects communicate with one another by sending and receiving information much the same way as people pass messages to one another. The concept of message passing makes it easier to talk about building systems that directly model or simulate their real-world counterparts. A message for an object is a request for execution of a procedure, and therefore will invoke a function (procedure) in the receiving object that generates the desired result. Message passing involves specifying the name of the object, the name of the function (message) and the information to be sent.
Objects have a life cycle. They can be created and destroyed. Communication with an object is feasible as long as it is a'1ive.

Benefits of oop

OOP offers several benefits to both the program designer and the user. Object-orientation contributes to the solution of
many problems associated with the development and quality of software products. The new technology promises greater
programmer productivity, better quality of software and lesser maintenance cost. The principal advantages are:
Through inheritance, we can eliminate redundant code and extend the use of existing classes.
· We can build programs from the standard working modules that communicate with one another, rather than having to start writing the code from scratch. This leads to saving of development time and higher productivity.
· The principle of data hiding helps the programmer to build secure programs that cannot be invaded by code in other
parts of the program.
· It is possible to have multiple instances of an object to coexist without any interference.
· It is possible to map objects in the problem domain to those in the program.
· It is easy to partition the work in a, project based on objects.
· The data-centered design approach enables us to capture more details of a model in implementable form.
· Object-oriented systems can be easily upgraded from small to large systems.
· Message passing techniques for communication between objects makes the interface descriptions with external systems much simpler.

· Software complexity can be easily managed.

Applications of Oop

Applications of OOP are beginning to gain importance in many areas. The most popular application of object-oriented
programming, up to now, has been in the area of user interface design such as windows. Hundreds of windowing systems have been developed, using the OOP techniques.

Real-business systems are often much more complex and contain many more objects with complicated attributes and methods. OOP is useful in these types of applications because it can simplify a complex problem.

The promising areas for application of OOP include:

· Real-time systems
· Simulation and modeling
· Object-oriented databases
· Hypertext, hypermedia and expertext
· AI and expert systems
· Neural networks and Parallel programming Decision support and office automation systems
· CIM/CAM/CAD systems

Object-oriented technology is certainly going to change the way the software engineers think, analyze, design and implement future systems.

An **abstract class** is one which defines an interface, but it may not provide implementations for all its member functions. Generally an abstract class is used as the base class from which other classes are
derived. The derived class provides implementations for the member functions that are not implemented in the base class. Abstract classes act as expressions of general concepts from which more specific classes can be derived. You cannot create an object of an abstract class type

A derived class that implements all the missing functionality is called a **concrete class.** A concrete class, however, is a class for which entities (instances) may be created

Note- In C++, an abstract class is defined as a class having at least one pure virtual method, i.e., an abstract method, which may or may not possess an implementation.

Example

```
class AbstractClass {
public:
  virtual void AbstractMemberFunction() = 0; //pure virtual function makes
this class Abstract class
  virtual void NonAbstractMemberFunction1(); //virtual function

  void NonAbstractMemberFunction2();
};
```