**Inheritance**

Inheritance is the process by which one object can acquire the properties of another. An object can inherit a general set of properties to which it can add those features that are specific only to itself. Inheritance is important because it allows an object to support the concept of *hierarchical classification.* Most information is made manageable by hierarchical classification.

The child class inherits all those qualities associated with the parent and adds to them its own defining characteristics.

Encapsulation

Encapsulation is the mechanism that binds together code and the data it manipulates, and keeps them both safe from outside. In an object-oriented language, code and data can be combined in such a way that a self-contained

‘black box’ is created. When code and data are linked together in this fashion , an *object* is created:

**OBJECT**

Data

Methods: code

Within an object, code, data, or both may be *private* to that object or *public*. Private code or data is known to and accessible only by another part of the object (i.e. cannot be accessed by a piece of the program that exists outside the object. Public code or data can be accessed by other parts of the program even though it is defined within an object.

Public parts of an object are used to provide a controlled interface to the private elements of the object.

Polymorphism

Polymorphism is the quality that allows one name to be used for two or more related but technically different purposes. Polymorphism allows one name to specify a general class of actions. Within a general class of actions, the specific action to be applied is determined by the type of data. For example, in C, the absolute value action requires three distinct function names: **abs( )** for integer, **labs( )** for long integer, and **fabs( )** for floating-point value. However in C++, each function can be called by the same name, such as **abs( ).**  The type of data used to call the function determines which specific version of the function is actually executed. In C++ it is possible to use one function name for many different purposes. This type of polymorphism is called *function overloading*. Polymorphism can also be applied to operators. In that case it is called *operator overloading.* More generally the concept of polymorphism is characterized by the idea ‘one interface, multiple methods’. The key point to remember about polymorphism is that it allows you to handle greater complexity by allowing the creation of standard interfaces to related activities.

·         **Behavior**: - Every software component has some specific tasks to be done. So, the behavior of a software component is the set of actions it can perform in the program after it gets request. The complete description of all the behavior for a component is also called prototype. For example, in the Interactive Intelligent Kitchen Helper, the ‘Greeter’ component is responsible to display informative initial message, offer user choice of options, pass control to other related components etc.

·         State:- The state of a component represents all the information held within it at a given point of time. So, state is a property of an instance. For example, in the Interactive Intelligent Kitchen Helper, the ‘Recipe’ component’s state includes the ingredients and preparation instructions.

 Instance: - A representative of the collection is called an instance of the class. So, an instance is an individual representative of a class. All the instances can perform the same actions but use different values. For example, all florist can perform the same actions such as take order, sells flower, dispatch flower etc. but they can use different flowers in each order.

Responsibility

Responsibility means behavior (method) to response the message and it is one of the fundamental concept in object-oriented programming. In the example let us consider that Chris made request to florist Fred to send flower bookie to his friend Robin, we see that Chris’s request indicates only the desired outcome i.e. to send flower to Robin and florist Fred is free to use any technique to achieve that desire of Chris. So, there is greater independence between objects in solving the complex problem.

**Data Hiding :-**

Data hiding is a software development technique specifically used in object-oriented programming (OOP) to hide internal object details (data members). Data hiding ensures exclusive data access to class members and protects object integrity by preventing unintended or intended changes.  
  
Data hiding also reduces system complexity for increased robustness by limiting interdependencies between software components. Data hiding is also known as data encapsulation or information hiding. Data hiding was introduced as part of the OOP methodology, in which a program is separated into objects with specific data and functions. This technique enhances a programmer’s ability to create classes with unique data sets and functions, avoiding unnecessary penetration from other program classes.

Because software architecture techniques rarely differ, there are few data hiding contradictions. Data hiding only hides class data components, whereas data encapsulation hides class data parts and private methods.

**Methods and Functions : -**

 A **method** in object-oriented programming is a procedure associated with a class. A method defines the behavior of the objects that are created from the class. Another way to say this is that a method is an action that an object is able to perform. The association between method and class is called binding. Consider the example of an object of the type 'person,' created using the person class. Methods associated with this class could consist of things like walking and driving. Methods are sometimes confused with functions, but they are distinct.

A **function** is a combination of instructions that are combined to achieve some result. A function typically requires some input (called arguments) and returns some results. For example, consider the example of driving a car. To determine the mileage, you need to perform a calculation using the distance driven and the amount of fuel used. You could write a function to do this calculation. The arguments going into the function would be distance and fuel consumption, and the result would be mileage. Anytime you want to determine the mileage, you simply call the function to perform the calculation.