OOPS

**1. What is OOPS?**  
OOPS is abbreviated as Object Oriented Programming system in which programs are considered as a collection of objects. Each object is nothing but an instance of a class.

It is a problem solving technique to develop software systems. It is a technique to think real world in terms of objects. Object maps the software model to real world concept. These objects have responsibilities and provide services to application or other objects.

**2. Write basic concepts of OOPS?**  
Following are the concepts of OOPS and are as follows:.  
Abstraction.   
Encapsulation.   
Inheritance.   
Polymorphism.  
  
**3. What is a 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.  
  
**4. What is an object?**  
Object is termed as an instance of a class, and it has its own state, behavior and identity.  
  
**5. What is Encapsulation?**  
Encapsulation is an attribute of an object, and it contains all data which is hidden. That hidden data can be restricted to the members of that class.  
  
Levels are Public,Protected, Private, Internal and Protected Internal.

**6. What is Polymorphism?**  
Polymorphism is nothing butassigning behavior or value in a subclass to something that was already declared in the main class. Simply, polymorphism takes more than one form.  
  
**7. What is Inheritance?**  
Inheritance is a concept where one class shares the structure and behavior defined in another class. Ifinheritance applied on one class is called Single Inheritance, and if it depends on multiple classes, then it is called multiple Inheritance.

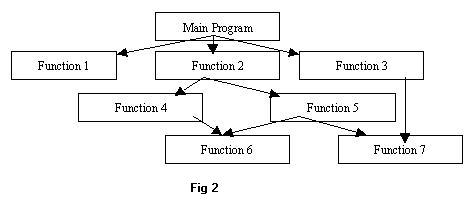
[**Object-Oriented Programming**](http://www.cpp-home.com/archives/206.html)

**Objected-oriented programming (OOP):**  in a new way of approaching the job of programming. Approaching to programming have changed dramatically since the invention of the computer, primarily to accommodate the increasing complexity of program. Assembly language was invented so that a program could deal with longer, increasingly complex programs using symbolic representations of the machine instructions. As programs continued to grow high level language were introduced that gave the programmer more tools with which to handle complexity. The first widespread language was FORTRAN. Although FORTRAN was a very impressive first step, it is hardly a language that encourages clear, easy-to–understandable program.

The 1960 gave birth to structured programming. This in the method encouraged by languages such as C and Pascal. The are of structured languages made it possible to write moderately complex programs fairly easily. However, even using structured programming methods a project becomes uncontrollable once it reaches a certain size i.e. once its complexity excess that which a program can mange.

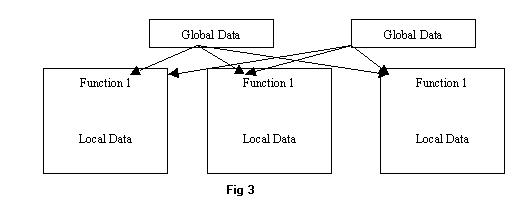
At each milestone in the development of programming methods were created to allow the programmer to deal with greater complexity. Each step of the way, the new approach tools the best elements of the previous method and moved forward. Today, many projects are near at longer works. To solve this problem, object oriented programming was invented.

Before going into details about object oriented programming have a look at procedure oriented programming. In the procedure-oriented approach, the problem is viewed as a sequence of things to be done, such as reading, calculating and printing. A number of functions are written to accomplish these tasks. The primary focus is on functions. A typical program structure for procedural programming is shown in Fig 2. The technique of hierarchical decomposition has been used to specify the tasks to be completed in order to solve a problem.



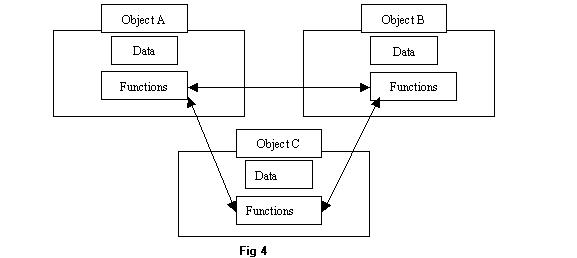
**Procedure-oriented programming :** basically consists of writing a list of instructions for the computer to follow, and organizing these instructions into groups known as functions. We normally use a flowchart to organize these actions and represent the flow of control from one action to another.

While we concentrate on the development of functions, very little attention is given to the data that are being used be various functions. What happens to data? How they are affected by the functions that work on them?



In multi-function program, many important data items are places as global so that they may be accessed by all the functions. Each function may have its own local data. Fig 3 shows the relationship of data and functions in a procedure

**Object-oriented programming paradigm :**The major motivating factor in the invention of object-oriented approach is to salvage 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 us to decompose a problem into a number of entities called objects and then builds data and functions around these entities. The organization of data and functions in object-oriented programs is shown here.



The data of an object can access the functions of other objects.

However, functions of one object can access the functions of other objects.

 Some of the striking features of object-oriented programming are:

 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.

Object-oriented programming is the most recent concept among programming paradigms is means different things to different people. It is therefore important to have a working definite object-oriented programming before we proceed further. Our definition of object-oriented programming is as follows “Object-oriented programming is an approach that provides a way of molding 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.”

That is, an object is considered to be a 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 are used in a variety of different programs without modifications.

**What is object oriented programming?**

Object oriented programming (OOP) have taken the best ideas of structured programming and combined them with several powerful new concept that encourage you the approach the task of programming in a new way. In general when programming in an object oriented fashion you break down a problem into subgroup of related parts that take into account both code and data related to each group. Also, you organize this subgroup into a hierarchical for all intents and purpose, an object in a variable of area-defined type. It may seem strong at first to think of an object, which lines both code and data on variable. However, in object oriented programming, this is precisely the case. When you define an object, you are implicitly creating a new data type.

**Basic concepts of object-oriented programming**

‘Object-Oriented’ remains a term, which is interpreted differently by different people. It is therefore necessary to understand some of the concepts used extensively in object-oriented programming. We shall discuss in this section the following general concepts:

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

**1.Objects**

Objects are the basic run-time entities in an object-oriented system. They may represent a person, i. e. a bank account, a table of data or any item that the program must handle. They may also resent user-defined data such as vectors, time and lists. Programming problem is analyzed in terms of objects and the nature of communication between them. Program objects should be chosen such that they match closely with the real-world objects. As pointed out earlier, objects take up space in the memory and have an associated address like a record in Pascal, or a structure in C.

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,

**2.Classes**

We just mentioned that objects contain data and code to manipulate that data. The entire set of data and code of an object can be made a user-defined data type with the help of a class. In fact, objects are variables of 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 created. A class is thus a collection of objects of similar type. For example, 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. For example, 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.

**3.Data abstraction**

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*‘.

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 al1 the essential properties of the objects that are to be created. Since the classes use the concept of data abstraction, they are known as Abstract Data Types (ADT).

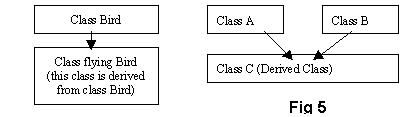
**6.Encapsulation**

Encapsulation in the mechanism that binds together code and data and that leaps both safe from outside interference or misure. It also allows the creation of an object. More simply, an object in a logical entity that encapsulate both data and the code that manipulators that data.

Within an object, some of the code and/ or data may be private to the objected and in accessible to anything outside the object. In this way and object provides a significant level of protection against some other unrelated part of the program accidentally modifying or incorrectly using the private parts of the object.

**4.Inheritance**

Inheritance is the process by which objects of one class acquire the properties of objects of another class. It supports the concept of hierarchical classification. For example, the bird robin is a part the class flying bird which is again a part of the class bird. As illustrated in Fig. 5, the principle behind this sort of division is that each derived class shares common characteristics with the class from which it is derived.



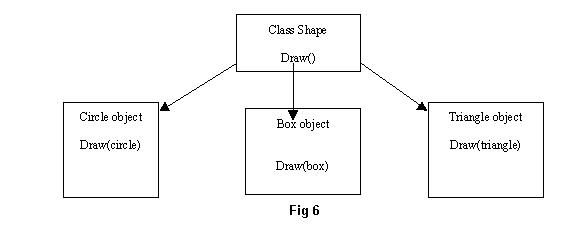
In OOP, the concept of inheritance provides the idea of reusability. This means that we can add additional features to an existing class without modifying it. This is possible by deriving a new class from the existing one. The new class will have the combined features of both the classes. The real appeal 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.

**7.Polymorphism**

Polymorphism is another important OOP concept. Polymorphism means the ability to take more than one form. For example, an operation may exhibit different behavior in different instances. The behavior depends upon the types of data, used in the operation. For example, consider the operation of addition. For two numbers, the operation will generate a sum. If the operands are strings, then the operation would produce a third string by concatenation.

The picture below illustrates at 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.



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 through specific actions associated with each operation may differ. Polymorphism is extensively used in implementing inheritance.

Object oriented programming languages support polymorphism, which in characterized by the phase “on interface multiple method”. In simple terms, polymorphism in an attribute that allows one interface to be used with a general class of actions. Polymorphism helps in reducing complexity by allowing the same interface to specify a general class of action. It is compiler’s job to select the “specify action” on it applies to each situation. The programmers don’t need to make this selection manually operator, overloading, function, overloading and overlooking example of polymorphism structure. Finally you translate these subgroups self-contained units called object.

In a multi-function program, many important data items are placed as global so that they may be accessed by all the functions. Each function-may have its own local data. Global data are more vulnerable to an inadvertent change by a function. In a large program it is very difficult to identify what data is used by which function. In case we need to revise an external data structure, we should also revise all functions that access the data. This provides an opportunity for bugs to creep in.

**5.Dynamic Binding**

Binding refers to the linking of a procedure call to the code to be executed in response to the call. Dynamic 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 polymorphic reference depends on the dynamic type of that reference.

Consider the procedure “draw” in Fig. 6. 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 called.

**8.Message Communication**

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.  
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.

*Example:*  
Objects have a life cycle. They can be created and destroyed. Communication with an object is feasible as long as it is alive.

**Benefits of OOP**

OOP offers several benefits to both the program designer and the user. Object-orientation contribution 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:

1. Through inheritance, we can eliminate redundant code and extend the use of existing classes.  
2. 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.  
3. The principle of data hiding helps the programmer to build secure programs that cannot be invaded by code in other parts of the program.  
4. It is possible to have multiple instances of an object to co-exist without any interference.  
5. It is possible to map objects in the problem domain to those objects in the program.  
6. It is easy to partition the work in a project based on objects.  
7. The data-centered design approach enables us to capture more details of a model in implementable form.  
8. Object-oriented systems can be easily upgraded from small to large systems.  
9. Message passing techniques for communication between objects makes the interface descriptions with external systems much simpler.  
10. Software complexity can be easily managed.

While it is possible to incorporate all these features in an object-oriented system, their importance their importance depends on the type of the project and the preference of the programmer. There are a number of issues that need to be tackled to get some of the benefits stated above. For instance, object libraries must be available for reuse. The technology is still developing and current products may be superseded quickly. Strict controls and protocols need to be developed if reuse is not to be compromised.

Developing software that is easy to use makes it hard to build. It is hoped that the object-oriented programming tools would help manage this problem.

**Object-oriented languages**

Object-oriented programming is not the right of any particular language. Like structured programming, OOP concepts can be implemented using languages such as C and Pascal. However, programming becomes clumsy and may generate confusion when the programs grow large. A language that is specially designed to support the OOP concepts makes it easier to implement them.

The languages should support several of the OOP concepts to claim that they are object-oriented. Depending upon the features they support, they can be classified into the following two categories:

1. Object-based programming languages  
2. Object-oriented programming languages

Object-based programming is the style of programming that primarily supports encapsulation and object identity.

Major features that are required for object-based programming are:

 Data encapsulation

 Data hiding and access mechanisms

 Automatic initialization and clear-up of objects

 Operator overloading

Languages that support programming with objects are said to be object-based programming languages. They do not support inheritance and dynamic binding.

Ada is a typical object-based programming language.

Object-oriented programming incorporates all of object-based programming features along with two additional features, namely, inheritance and dynamic binding.

**Application of OOP**

The Promising areas for application of oops include:

1. Real-time systems  
2. Simulation and modeling  
3. Object-oriented databases  
4. Hypertext, hypermedia and expertext  
5. AI and expert systems  
6. Neural networks and parallel programming  
7. Decision support and office automation systems  
8. CIM/CAM/CAD systems

**Object Oriented Programming(OOP) Features**

* Object oriented programming (OOP) is a programming model where programs are organized around object and data rather than action and logic.
* OOP allows decomposition of a problem into a number of entities called Object and then builds data and function around these objects.
* The program is divided into a number of small units called Object. The data and function are build around these objects.
* The data of the objects can be accessed only by the functions associated with that object.
* The functions of one object can access the functions of other object.

OOP has the following important features:

All the programming languages supporting object oriented programming will be supporting these three main concepts:

1. Encapsulation
2. Inheritance
3. Polymorphism

### Abstraction

Abstraction is "To represent the essential feature without representing the background details."

Abstraction lets you focus on what the object does instead of how it does it.

Abstraction provides you a generalized view of your classes or object by providing relevant information.

Abstraction is the process of hiding the working style of an object, and showing the information of an object in an understandable manner.

#### Real World Example of Abstraction

Suppose you have an object Mobile Phone.

Suppose you have 3 mobile phones as follows:

* Nokia 1400 (Features:- Calling, SMS)
* Nokia 2700 (Features:- Calling, SMS, FM Radio, MP3, Camera)
* Black Berry (Features:-Calling, SMS, FM Radio, MP3, Camera, Video Recording, Reading E-mails)

Abstract information (Necessary and Common Information) for the object "Mobile Phone" is to make a call to any number and can send SMS." so that, for mobile phone object you will have an abstract class like the following:

abstract class MobilePhone

{

public void Calling();

public void SendSMS();

}

public class Nokia1400 : MobilePhone

{

}

public class Nokia2700 : MobilePhone

{

public void FMRadio();

public void MP3();

public void Camera();

}

public class BlackBerry : MobilePhone

{

public void FMRadio();

public void MP3();

public void Camera();

public void Recording();

public void ReadAndSendEmails();

}

Abstraction means putting all the variables and methods in a class which is necessary.

For example: Abstract class and abstract method.

Abstraction is the common thing.

##### Examples

* If somebody in your college tells you to fill an application form, you will fill your details like name, address, data of birth, which semester, percentage you got, etc.
* If some doctor gives you an application to fill the details, you will fill the details like name, address, date of birth, blood group, height and weight.

See in the above examples what is the common thing?

Age, name, address so you can create the class which consists of the common thing that is called abstract class.

That class is not complete and it can be inherited by another class.

### Encapsulation

Wrapping up data member and method together into a single unit (i.e. Class) is called Encapsulation.

Encapsulation is like enclosing in a capsule. That is enclosing the related operations and data related to an object into that object.

Encapsulation is like your bag in which you can keep your pen, book, etc. It means this is the property of encapsulating members and functions.

class Bag

{

book;

pen;

ReadBook();

}

Encapsulation means hiding the internal details of an object, i.e., how an object does something.

Encapsulation prevents clients from seeing its 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.

So, when you access the property, you can validate the data and set it.

##### Example

class Demo

{

private int \_mark;

public int Mark

{

get { return \_mark; }

set { if (\_mark > 0) \_mark = value; else \_mark = 0; }

}

}

#### Real World Example of Encapsulation

Let's take an example of Mobile Phone and Mobile Phone Manufacturer.

Suppose you are a Mobile Phone Manufacturer and you designed and developed a Mobile Phone design(class), now by using machinery you are manufacturing a Mobile Phone(object) for selling, when you sell your Mobile Phone, the user only learns how to use the Mobile Phone but not how this Mobile Phone works.

This means that you are creating the class with function and by making object (capsule) of it, you are making availability of the functionality of your class by that object and without the interference in the original class.

##### Example 2

TV operation

It is encapsulated with cover and we can operate with remote and no need to open TV and change the channel.

Here everything is in private except remote so that anyone can access not to operate and change the things in TV.

### Inheritance

When a class acquire the property of another class, it is known as inheritance.

Inheritance is the process of object reusability.

For example: A child acquires the property of parents.

public class ParentClass

{

public ParentClass()

{

Console.WriteLine("Parent Constructor.");

}

public void print()

{

Console.WriteLine("I'm a Parent Class.");

}

}

public class ChildClass : ParentClass

{

public ChildClass()

{

Console.WriteLine("Child Constructor.");

}

public static void Main()

{

ChildClass child = new ChildClass();

child.print();

}

}

#### Output

Parent Constructor.

Child Constructor.

I'm a Parent Class.

### Polymorphism

Polymorphism means one name many forms.

One function behaves in different forms.

In other words, "Many forms of a single object is called Polymorphism."

#### Real World Example of Polymorphism

##### Example 1

A Teacher behaves with his student.

A Teacher behaves with his/her seniors.

Here teacher is an object but attitude is different in different situations.

##### Example 2

Person behaves like a SON in the house, at the same time that person behaves like an EMPLOYEE in office.

##### Example 3

Your mobile phone, one name but many forms:

* As phone
* As camera
* As mp3 player
* As radio

### Polymorphism in .NET

Difference between abstraction and encapsulation is as given in the table below:

| **Abstraction** | **Encapsulation** |
| --- | --- |
| 1. Abstraction solves the problem in the design level. | 1. Encapsulation solves the problem in the implementation level. |
| 2. Abstraction is used for hiding the unwanted data and giving relevant data. | 2. Encapsulation means hiding the code and data into a single unit to protect the data from outside world. |
| 3. Abstraction lets you focus on what the object does instead of how it does it. | 3. Encapsulation means hiding the internal details or mechanics of how an object does something. |
| 4. Abstraction- Outer layout, used in terms of design.  For example: Outer Look of a Mobile Phone, like it has a display screen and keypad buttons to dial a number. | 4. Encapsulation - Inner layout, used in terms of implementation.  For example: Inner Implementation detail of a Mobile Phone, how keypad button and Display Screen are connect with each other using circuits. |

The easier way to understand Abstraction and encapsulation is as follows:

#### Real World Example

Take an example of mobile phone:

You have a mobile phone, you can dial a number using keypad buttons. Even you don't know how these are working internally. This is called abstraction. You have the only information that is needed to dial a number. But not its internal working of mobile.

But how the mobile phone works internally and how keypad buttons are connected with internal circuit is called encapsulation.

**Summary**

* "Encapsulation is accomplished by using Class - keeping data and methods that accesses that data into a single unit"
* "Abstraction is accomplished by using Interface - just giving the abstract information about what it can do without specifying the background details"
* "Information/Data hiding is accomplished by using modifiers - by keeping the instance variables private or protected."

**Difference between struct and class**

The struct is value type in C# and it inherits from System.ValueType

The class is reference type in C# and it inherits from the System.Object Type

The struct value will be stored on the stack memory.

The class object is stored on the heap memory. The object will be under garbage collection and automatically removed when there is no reference to the created objects.

The struct use the array type and its good to use for read only and light weight object.

The class uses the collection object type and it can perform all the operations and designed for complex data type storage.

The struct can't be base type to the classes and also to the other structure.

The class can inherit another class, interface and it can be base class to another class.

The struct can only inherit the interfaces

The class can inherit the interfaces, abstract classes.

The struct can have only constructor.

The class can have the constructor and destructor.

The struct can instantiated without using the new keyword.

The new keyword should be used to create the object for the class

The struct can't have the default constructor

The class will have the default constructor

The struct is by default sealed class hence it will not allow to inherit. It can't use the abstract, sealed, base keyword.

The class can be declared as abstract, sealed class

The struct can't use the protected or protected internal modifier.

The class can use all the access modifiers.

The struct can't initialize at the time of declaration.

The class can have the initializes fields.

1. Classes are Reference types and Structures are Values types.

When I say Classes are reference types, basically they will contain the address of an instance variables. For example:

Class MyClass

{

Public Int DataMember; //By default, accessibility of class data members

//will be private. So I am making it as Public which

//can be accessed out side of the class.

}

In main method, I can create an instance of this class using new operator that allocates memory for this class and stores the base address of that into MyClass type variable.

Static Public void Main (string [] arg)

{

MyClass \_myClassObject1 =new MyClass ();

\_ myClassObject1.DataMember=10;

MyClass \_myClassObject2 =\_myClassObject1;

\_ myClassObject2.DataMember=20;

}

In the above program, “MyClass \_myClassObject2 =\_myClassObject1” instruction indicates that both variables of type MyClass myClassObject1 and myClassObject2 will point to the same memory location. It basically assigns the same memory location into another variable of same type.

So if any changes that we make in any one of the objects type MyClass will have an effect on another since both are pointing to the same memory location.

“\_ myClassObject1.DataMember=10” at this line both the object’s data members will contain the value of 10. \_ myClassObject2.DataMember=20 at this line both the object’s data member will contains the value of 20. Eventually, we are accessing datamembers of an object through pointers.

Unlike classes, structures are value types. For example:

Structure MyStructure

{

Public Int DataMember; //By default, accessibility of Structure data

//members will be private. So I am making it as

//Public which can be accessed out side of the structure.

}

Static Public void Main (string [] arg)

{

MyStructure \_myStructObject1 =new MyStructure ();

\_ myStructObject1.DataMember=10;

MyStructure \_ myStructObject2 =\_ myStructObject1;

\_ myStructObject2.DataMember=20;

}

In the above program, instantiating the object of MyStructure type using new operator and storing address into \_myStructObject variable of type MyStructure and assigning value 10 to data member of the structure using “\_ myStructObject1.DataMember=10”. In the next line, I am declaring another variable\_myStructObject2 of type MyStructure and assigning \_myStructObject1 into that. Here .NET C# compiler creates another copy of \_myStructureObject1 object and assigns that memory location into MyStructure variable \_myStructObject2.

So whatever change we make on \_myStructObject1 will never have an effect on another variable \_myStructObject2 of type MyStructrue. So that’s why we are saying Structures are value types.

So the immediate Base class for class is Object and immediate Base class for Structure is ValueType which inherits from Object.

1. Classes will support an Inheritance whereas Structures won’t.

How are we saying that? And what is the reason behind that? The answer is Classes.

It can be abstract, sealed, static, and partial and can’t be Private, Protected and protected internal.

1. Structures can’t have modifiers like abstract, sealed, and static whereas classes can have.

When to Use Structure and Class?

In general, classes can be used when you have more complex behavior or data. And if you think that these behaviour or data to be modified after creating an instance of class, then classes are absolute methods.

Structures can be used for small data structures. If developer feels that data members of structure cannot to be modified after creating structure, then having structure will suit.

**EXE vs DLL**

The terms EXE and DLL are very common in programming. When coding, you can either export your final project to either a DLL or an EXE. The term EXE is a shortened version of the word executable as it identifies the file as a program. On the other hand, DLL stands for Dynamic Link Library, which commonly contains functions and procedures that can be used by other programs.

In the basest application package, you would find at least a single EXE file that may or may not be accompanied with one or more DLL files. An EXE file contains the entry point or the part in the code where the operating system is supposed to begin the execution of the application. DLL files do not have this entry point and cannot be executed on their own.

The most major advantage of DLL files is in its reusability. A DLL file can be used in other applications as long as the coder knows the names and parameters of the functions and procedures in the DLL file. Because of this capability, DLL files are ideal for distributing device drivers. The DLL would facilitate the communication between the hardware and the application that wishes to use it. The application would not need to know the intricacies of accessing the hardware just as long as it is capable of calling the functions on the DLL.

Launching an EXE would mean creating a process for it to run on and a memory space. This is necessary in order for the program to run properly. Since a DLL is not launched by itself and is called by another application, it does not have its own memory space and process. It simply shares the process and memory space of the application that is calling it. Because of this, a DLL might have limited access to resources as it might be taken up by the application itself or by other DLLs.

Summary:

1.EXE is an extension used for executable files while DLL is the extension for a dynamic link library.

2.An EXE file can be run independently while a DLL is used by other applications.

3.An EXE file defines an entry point while a DLL does not.

4.A DLL file can be reused by other applications while an EXE cannot.

5.A DLL would share the same process and memory space of the calling application while an EXE creates its separate process and memory space.

.Exe  
1.These are outbound file.  
2.Only one .exe file exists per application.  
3. .Exe cannot be shared with other applications.  
4.exe is a executable file.  
  
.dll  
1.These are inbund file .  
2.Many .dll files may exists in one application.  
3. .dll can be shared with other applications.  
4. dll is a Dynamic Link Library

DLL : (Dynamic Link Library)  
  
1. DLLs are not directly executable. They are separate files containing functions that can be called by programs and other DLLs to perform computations and functions.   
2. A DLL can have only one instance.  
  
EXE : (Executable)  
  
1. An EXE is a program that can be executed( Ex. Windows Program)  
2. An Application can have multiple instances of itself running on the system simultaneously.

**Virtual and Override**  
In C#, if you like to override the parent class method then you must mark the parent method by keyword “**Virtual**” and method in derived class which intended to override the parent method should be marked by keyword “**override**”

**Note:**

1. If parent method is marked by keyword “Virtual” but **child is not marked** by keyword “override”, then program will compile but the parent method will not be overrided.
2. If Child method is marked by keyword “override” **but parent method is not marked by keyword “virtual”** then program will not compile. It will give following error :  
   ‘OOPS\_Concept.TestB.display()’: cannot override inherited member ‘OOPS\_Concept.TestA.display()’ **because it is not marked virtual, abstract, or override.**

**Method Hiding – Keyword new:**  
As discussed earlier, second point in output of first program was “Compiler generated warning message”.  
Because c# also support Method Hiding. To mark method as “hiding” use keyword “new” in derived class .  
Keyword “new” can be used with keyword “virtual” also.  
Quick Note :  
If keyword “override” is used in derive class then its override the parent method.  
If Keyword “new” is used in derive class then derive method hided by parent method. As shown in below program:

When a message can be processed in different ways is called polymorphism. Polymorphism means many forms.

Polymorphism is one of the fundamental concepts of OOP.

**Polymorphism provides following features:**

* It allows you to invoke methods of derived class through base class reference during runtime.
* It has the ability for classes to provide different implementations of methods that are called through the same name.

**Polymorphism is of two types:**

1. Compile time polymorphism/Overloading
2. Runtime polymorphism/Overriding

**Compile Time Polymorphism**

Compile time polymorphism is method and operators overloading. It is also called early binding.

In method overloading method performs the different task at the different input parameters.

**Runtime Time Polymorphism**

Runtime time polymorphism is done using inheritance and virtual functions. Method overriding is called runtime polymorphism. It is also called late binding.

When **overriding** a method, you change the behavior of the method for the derived class. **Overloading** a method simply involves having another method with the same prototype.

**Caution:** Don't confused method overloading with method overriding, they are different, unrelated concepts. But they sound similar.

Method overloading has nothing to do with inheritance or virtual methods.

**Following are examples of methods having different overloads:**

void area(int side);

void area(int l, int b);

void area(float radius);

**Practical example of Method Overloading (Compile Time Polymorphism)**

using System;

namespace method\_overloading

{

class Program

{

public class Print

{

public void display(string name)

{

Console.WriteLine("Your name is : " + name);

}

public void display(int age, float marks)

{

Console.WriteLine("Your age is : " + age);

Console.WriteLine("Your marks are :" + marks);

} }

static void Main(string[] args)

{

Print obj = new Print();

obj.display("George");

obj.display(34, 76.50f);

Console.ReadLine();

} } }

**Note:** In the code if you observe display method is called two times. Display method will work according to the number of parameters and type of parameters.

**When and why to use method overloading**

Use method overloading in situation where you want a class to be able to do something, but there is more than one possibility for what information is supplied to the method that carries out the task.

You should consider overloading a method when you for some reason need a couple of methods that take different parameters, but conceptually do the same thing.

**Method Overloading showing many forms.**

using System;

namespace method\_overloading\_polymorphism

{

class Program

{

public class Shape

{

public void Area(float r)

{

float a = (float)3.14 \* r;

// here we have used funtion overload with 1 parameter.

Console.WriteLine("Area of a circle: {0}",a);

}

public void Area(float l, float b)

{

float x = (float)l\* b;

// here we have used funtion overload with 2 parameters.

Console.WriteLine("Area of a rectangle: {0}",x);

}

public void Area(float a, float b, float c)

{

float s = (float)(a\*b\*c)/2;

// here we have used funtion overload with 3 parameters.

Console.WriteLine("Area of a circle: {0}", s);

}

}

static void Main(string[] args)

{

Shape ob = new Shape();

ob.Area(2.0f);

ob.Area(20.0f,30.0f);

ob.Area(2.0f,3.0f,4.0f);

Console.ReadLine();

} } }

**Things to keep in mind while method overloading**

If you use overload for method, there are couple of restrictions that the compiler imposes.

The rule is that overloads must be different in their signature, which means the name and the number and type of parameters.

There is no limit to how many overload of a method you can have. You simply declare them in a class, just as if they were different methods that happened to have the same name.