1. Encapsulation

2. Abstraction

3. Inheritance

4. Polymorphism.

These concepts are the four main gods of OOP world and in software term, they are called four main Object Oriented Programming (OOP) Concepts.

4.16. What is Inheritance?

The ability of a new class to be created, from an existing class by extending it, is called *inheritance*.

### 4.10. What is Abstraction and Generalization?

Abstraction is an emphasis on the idea, qualities and properties rather than the particulars (a suppression of detail). The importance of abstraction is derived from its ability to hide irrelevant details from the a user.

Abstraction reduces complexity by hiding irrelevant detail,

### 4.17. What is Polymorphism?

Polymorphisms is a generic term that means 'many shapes'. More precisely Polymorphisms means the ability to request that the same operations be performed by a wide range of different types of things.

In OOP polymorphisms is achieved by using many different techniques named method overloading, operator overloading, and method overriding,

Allows you to invoke derived class methods through a base class reference during runtime.

### 4.7. What is Encapsulation (or Information Hiding)?

The encapsulation is the inclusion-within a program object-of all the resources needed for the object to function, basically, the methods and the data. In OOP the encapsulation is mainly achieved by creating classes, the classes expose public methods and properties.

NOTE: What is a class - A class is kind of a container or capsule or a cell, which encapsulate a set of methods, attribute and properties to provide its indented functionalities to other classes.

==================================================================================

### 4.12. What is an Interface?

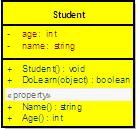
In summary the Interface separates the implementation and defines the structure, and this concept is very useful in cases where you need the implementation to be interchangeable.

### 4.4. What is an Object?

An object can be considered a "thing" that can perform a set of related activities. The set of activities that the object performs defines the object's behavior. For example, the Hand (object) can grip something, or a Student(object) can give their name or address.

In pure OOP terms an object is an instance of a class.

### 4.5. What is a Class?



A class is simply a representation of a type of object. It is the blueprint, or plan, or template, that describes the details of an object. A class is the blueprint from which the individual objects are created. Class is composed of three things: a name, attributes, and operations.

### 4.11. What is an Abstract class?

Abstract classes, which declared with the abstract keyword, **cannot be instantiated**. It can only be used as a super-class for other classes that extend the abstract class. Abstract class is the concept and implementation gets completed when it is being realized by a subclass. In addition to this a class can inherit only from one abstract class (but a class may implement many interfaces) and and must override all its methods/properties that are declared to be abstract and may override virtual methods/ properties.

4.13. What is the difference between a Class and an Interface?

In .NET/ C#, a *class* can be defined to implement an *interface* and also it supports multiple implementations.

A *class* and an *interface* are two different types (conceptually). Theoretically a *class* emphasis the idea of encapsulation, while an *interface* emphasis the idea of abstraction (by suppressing the details of the implementation). The two poses a clear separation from one to another.

4.14. What is the difference between an Interface and an Abstract class?

There are quite a big difference between an *interface* and an *abstract class*, even though both look similar.

* Interface definition begins with a keyword interface so it is of type interface
* Abstract classes are declared with the abstract keyword so it is of type class
* Interface has no implementation, but they have to be implemented.
* Abstract class’s methods can have their own default implementations and they may be extended. The Abstract class’s methods could run independant of the inherting class.
* Interfaces can only have method declaration (implicitly public and abstract) and properties (implicitly public static)
* Abstract class’s methods can’t have implementation only when declared abstract.
* Interface can inherit more than one interfaces
* Abstract class can implement more than one interfaces, but can inherit only one class
* Abstract class must override all abstract method and may override virtual methods
* Interface can be used when the implementation is changing
* Abstract class can be used to provide some default behavior for a base class.
* Interface makes implementation interchangeable
* Interface increase security by hiding the implementation
* Abstract class can be used when implementing framework
* Abstract classes are an excellent way to create planned inheritance hierarchies and also to use as non-leaf classes in class hierarchies.

Abstract classes let you define some behaviors; they force your subclasses to provide others. For example, if you have an application framework, an abstract class can be used to provide the default implementation of the services and all mandatory modules such as event logging and message handling etc. This approach allows the developers to develop the application within the guided help provided by the framework.

### 4.16. What is Inheritance?

The ability of a new class to be created, from an existing class by extending it, is called inheritance.

### 4.18. What is Method Overloading?

Method overloading is the ability to define several methods all with the same name by providing unique method signatures.

### 4.19. What is Operator Overloading?

The operator overloading (less commonly known as ad-hoc polymorphisms) is a specific case of polymorphisms in which some or all of operators like +, - or == are treated as polymorphic functions and as such have different behaviors depending on the types of its arguments.

### 4.20. What is Method Overriding?

Method overriding is a language feature that allows a subclass to override a specific implementation of a method that is already provided by one of its super-classes.

### 4.15. What are Implicit and Explicit Interface Implementations?

As mentioned before .Net support multiple implementations, the concept of implicit and explicit implementation provide safe way to implement methods of multiple interfaces by hiding, exposing or preserving identities of each of interface methods, even when the method signatures are the same.

 If a class inherits from 2 or more interfaces and if the interfaces happen to have the same method names, the class doesn't know which interface method is being implemented if you use implicit interface implementation. This is one of the scenarios when you would explicitly implement an interface.

When implementing and interface explicitly we CANNOT use access modifiers on the implementing method.

Implicit implementation gives you an is-a type conversion, explicit implementation won't be accessible unless the object is explicitly type cast to that interface type.

4.28. What is SOA?

A service-oriented architecture is essentially a collection of services. These services communicate with each other. The communication can involve either simple data passing or it could involve two or more services coordinating some activity. Some means of connecting services to each other is needed.

.NET Framework introduced SOA by means of web services.

