FullStack.Cafe - Kill Your Tech Interview

Q1: What is *Inheritance*? ☆

Topics: OOP

Answer:

Inheritance allows us to define a class in terms of another class, which makes it easier to create and maintain an application. This also provides an opportunity to reuse the code functionality and speeds up implementation time.

When creating a class, instead of writing completely new data members and member functions, the programmer can designate that the new class should inherit the members of an existing class. This existing class is called the base class, and the new class is referred to as the derived class.

The idea of inheritance implements the IS-A relationship. For example, mammal IS A animal, dog IS-A mammal hence dog IS-A animal as well, and so on.

Q2: What is *Object-Oriented Programming* (OOP)? ☆

Topics: OOP

Answer:

OOP is a technique to develop logical modules, such as classes that contain properties, methods, fields, and events. An object is created in the program to represent a class. Therefore, an object encapsulates all the features, such as data and behavior that are associated to a class. OOP allows developers to develop modular programs and assemble them as software. Objects are used to access data and behaviors of different software modules, such as classes, namespaces, and sharable assemblies. .NET Framework supports only OOP languages, such as Visual Basic .NET, Visual C#, and Visual C++.

Q3: What is *Encapsulation*? ☆☆

Topics: OOP

Answer:

Encapsulation is defined *as the process of enclosing one or more items within a physical or logical package*. Encapsulation, in object oriented programming methodology, prevents access to implementation details.

Q4: What is *Polymorphism*? ☆☆

Topics: OOP

Answer:

The word polymorphism means having many forms. In object-oriented programming paradigm, polymorphism is often expressed as *one interface, multiple functions*.

Q5: What is a `class`? ☆☆

Topics: OOP

Answer:

A class describes all the attributes of objects, as well as the methods that implement the behavior of member objects. It is a comprehensive data type, which represents a blue print of objects. It is a template of object.

A class can be defined as the primary building block of OOP. It also serves as a template that describes the properties, state, and behaviors common to a particular group of objects.

A class contains data and behavior of an entity. For example, the aircraft class can contain data, such as model number, category, and color and behavior, such as duration of flight, speed, and number of passengers. A class inherits the data members and behaviors of other classes by extending from them.

Q6: What is an `object`? ☆☆

Topics: OOP

Answer:

Objeects are instance of classes. It is a basic unit of a system. An object is an entity that has attributes, behavior, and identity. Attributes and behavior of an object are defined by the class definition.

Q7: What is the relationship between a `class` and an `object`? ☆☆

Topics: OOP

Answer:

A class acts as a blue-print that defines the properties, states, and behaviors that are common to a number of objects. An object is an instance of the class. For example, you have a class called *Vehicle* and *Car* is the object of that class. You can create any number of objects for the class named *Vehicle*, such as *Van*, *Truck*, and *Auto*.

The *new* operator is used to create an object of a class. When an object of a class is instantiated, the system allocates memory for every data member that is present in the class.

Q8: Explain the basic features of OOPs ☆☆

Topics: OOP

Answer:

The following are the four basic features of OOP:

- **Abstraction** - Refers to the process of exposing only the relevant and essential data to the users without showing unnecessary information.
- **Polymorphism** - Allows you to use an entity in multiple forms.
- **Encapsulation** - Prevents the data from unwanted access by binding of code and data in a single unit called object.
- **Inheritance** - Promotes the reusability of code and eliminates the use of redundant code. It is the property through which a child class obtains all the features defined in its parent class. When a class inherits the

common properties of another class, the class inheriting the properties is called a derived class and the class that allows inheritance of its common properties is called a base class.

Q9: What is the difference between a `class` and a `structure`? ☆☆

Topics: OOP C#

Answer:

Class:

- A class is a *reference type*
- While instantiating a class, CLR allocates memory for its instance in *heap*
- Classes support inheritance
- Variables of a class can be assigned as `null`
- Class can contain constructor/destructor

Structure:

- A structure is a *value type*
- In structure, memory is allocated on *stack*
- Structures do not support inheritance
- Structure members cannot have `null` values
- Structure does not require constructor/destructor and members can be initialiazed automatically

Q10: Why is the `virtual` keyword used in code? ☆☆

Topics: OOP

Answer:

The `virtual` keyword is used while defining a class to specify that the methods and the properties of that class can be overridden in derived classes.

Q11: Explain the concept of *Constructor* ☆☆

Topics: OOP

Answer:

Constructor is a special method of a class, which is called automatically when the instance of a class is created. It is created with the same name as the class and initializes all class members, whenever you access the class. The main features of a constructor are as follows:

- Constructors do not have any return type.
- Constructors can be overloaded.
- It is not mandatory to declare a constructor; it is invoked automatically by .NET Framework.

Q12: Can you *inherit* `private` members of a class? ☆☆

Topics: OOP

Answer:

No, you cannot inherit `private` members of a class because `private` members are accessible only to that class and not outside that class.

Q13: What is the difference between *procedural* and *object-oriented* programming? ☆☆

Topics: OOP

Answer:

Procedural programming is based upon the modular approach in which the larger programs are broken into procedures. Each procedure is a set of instructions that are executed one after another. On the other hand, OOP is based upon objects. An object consists of various elements, such as methods and variables.

Access modifiers are not used in procedural programming, which implies that the entire data can be accessed freely anywhere in the program. In OOP, you can specify the scope of a particular data by using access modifiers - *public*, *private*, *internal*, *protected*, and *protected* internal.

Q14: What is the difference between *Interface* and *Abstract Class*? ☆☆☆

Topics: C# OOP

Answer:

There are some differences between **Abstract Class** and **Interface** which are listed below:

- interfaces can have no state or implementation
- a class that implements an interface must provide an implementation of all the methods of that interface
- abstract classes may contain state (data members) and/or implementation (methods)
- abstract classes can be inherited without implementing the abstract methods (though such a derived class is abstract itself)
- interfaces may be multiple-inherited, abstract classes may not (this is probably the key concrete reason for interfaces to exist separately from abtract classes - they permit an implementation of multiple inheritance that removes many of the problems of general MI).

Consider using abstract classes if :

1. You want to share code among several closely related classes.
2. You expect that classes that extend your abstract class have many common methods or fields, or require access modifiers other than public (such as protected and private).
3. You want to declare non-static or non-final fields.

Consider using interfaces if :

1. You expect that unrelated classes would implement your interface. For example, many unrelated objects can implement `Serializable` interface.
2. You want to specify the behaviour of a particular data type, but not concerned about who implements its behaviour.
3. You want to take advantage of multiple inheritances of type.

Q15: What is the difference between *Virtual* method and *Abstract* method? ☆☆☆

Topics: C# OOP

Answer:

- A **Virtual method** must always have a default implementation. However, it can be overridden in the derived class, though not mandatory. It can be overridden using *override* keyword.
- An **Abstract method** does not have an implementation. It resides in the abstract class. It is mandatory that the derived class implements the abstract method. An *override* keyword is not necessary here though it can be used.

Q16: How is method `overriding` different from method `overloading`? ☆☆☆

Topics: OOP

Answer:

- **Overriding** involves the creation of two or more methods with the same name and same signature in different classes (one of them should be parent class and other should be child).
- **Overloading** is a concept of using a method at different places with same name and different signatures within the same class.

Q17: How can you prevent a class from *overriding* in C#? ☆☆☆

Topics: OOP C#

Answer:

You can prevent a class from overriding in C# by using the `sealed` keyword.

Q18: What are `abstract` classes? What are the distinct characteristics of an `abstract` class? ☆☆☆

Topics: OOP

Answer:

An **abstract** class is a class that **cannot be instantiated** and is always used as a base class. The following are the characteristics of an abstract class:

- You cannot instantiate an abstract class directly. This implies that you cannot create an object of the abstract class; it must be inherited.
- You can have abstract as well as non-abstract members in an abstract class.
- You must declare at least one abstract method in the abstract class.
- An abstract class is always public
- An abstract class is declared using the *abstract* keyword.

The basic purpose of an abstract class is to provide a common definition of the base class that multiple derived classes can share.

Q19: What is *Polymorphism*, what is it for, and how is it used? ☆☆☆

Topics: OOP

Answer:

Polymorphism describes a pattern in object-oriented programming in which classes have **different functionality** while sharing a **common interface**.

The beauty of polymorphism is that the code working with the different classes does not need to know which class it uses since they're all used the same way. A real-world analogy for the polymorphism is a button. Everyone knows how to use a button: you simply apply pressure to it. What a button "does," however, depends on what it is connected to and the context in which it is used — but the result does not affect how it is used. If your boss tells you to press a button, you already have all the information needed to perform the task.

Q20: When should I use a `struct` instead of a `class`? ☆☆☆

Topics: OOP

Answer:

Do not define a structure unless the type has all of the following characteristics:

- It logically represents a single value, similar to primitive types (integer, double, and so on).
- It has an instance size smaller than 16 bytes.
- It is immutable.
- It will not have to be boxed frequently.