Sr. Software Engineer (SSE)

ABSTRACT

This is one of the subject from my personal notes series named “Coding-With-Arqam” that I am developing from the start of my professional development career.

Subject

Object Oriented Programming (OOP)

OOP

--> General Points:

-> Static Member functions can never be virtual.

--> Interface:

-> An interface only contains declarations of method, properties, indexers, and events.

-> An interface can be implement implicitly or explicitly by a class or struct.

-> A class or struct which implements an interface, must use 'public' access modifier.

-> An interface cannot include private, protected, or internal members.

-> Interface looks like a class but it is not a class because it never implements the methods like class do.

-> A class can implement multiple interfaces.

-> An interface can have methods and variables just like the class but the methods declared in interface are by default abstract (only method signature, no body).

-> Also, the variables declared in an interface are public, static & final by default.

-> An interface is a programming structure/syntax that allows the computer to enforce certain properties on an object (class).

-> For example, say we have a car class and a scooter class and a truck class. Each of these three classes should have a start_engine() action.

-> Unlike a class, an interface never implements methods; instead, classes that implement the interface implement the methods defined by the interface.

-> A class can implement multiple intxerfaces.

-> Java uses Interface to implement multiple inheritance.

-> The class that implements interface must implement all the methods of that interface.

-> It compulsory to implement all methods of interface unless and until that class is an Abstract class.

-> Advantages: multiple inheritance, security.

-> A sub-class implements interface and extends abstract class.

--> Abstract Class:

-> A class that is declared abstract.

-> It may or may not include abstract methods.

-> The subclass usually provides implementations for all of the abstract methods in its parent class. However, if it does not, then the subclass must also be declared abstract.

-> You don't have to implement all methods of an abstract class. But you must implement all abstract methods of it.

-> A class can extend only one abstract class while a class can implement multiple interfaces.

-> In fact extending an abstract class has no difference then extending a normal class.

-> It's not like implementing interfaces.

-> A sub-class implements interface and extends abstract class.

-> The purpose of an abstract class is to define some common behavior that can be inherited by multiple subclasses, without implementing the entire class.

-> Abstract classes can have constructors.

-> Though we cannot create an object of an abstract class, when we create an object of a class which is concrete and subclass of the abstract class, the constructor of the abstract class is automatically invoked.

-> Abstract Class it cannot be instantiated but that does not mean an Abstract class cannot have a constructor. Each abstract class must have a concrete subclass which will implement the abstract methods of that abstract class.

-> Example:

-> Shape class (Abstract Class). a general class. no defination. no area, etc of a general shape. differen shapes will inherit from it.

-> Although we cannot create an object of it but, Abstract class have a construction.

-> Abstract class cannot be instantiated, but pointers and refrences of Abstract class type can be created.

-> Mainly use for UpCasting.

--> Abstract Function:

--> Upcasting (opposite of Downcasting):

-> using the Super class's reference or pointer to refer to a Sub class's object.

-> Or we can say that, the act of converting a Sub class's reference or pointer into its Super class's reference or pointer is called Upcasting.

```
-> Super* ptr;    // Super class pointer
        Sub obj;
        ptr = &obj;
```

--> Abstraction + Interface + Encapsulation:

-> Let suppose we have detail case study. To take the required attributes from the case study is called absraction.

-> To make a class of those things is known as encapsulation because we have encapsulated them in a class.

-> It is not necessary to implement all methods of abstract class. And we can use methods as it is and can also override them. Abstract methods contain body.

-> In interface, we have to implement all the methods and have to override all of them because interface contains only prototypes.

-> Abstraction Example:

-> "Div" in html (is a class) has some default attributes. We also override some of the attributes. so it is an abtract class example.

-> mat-slider in material angular. def min value is 0 and max is 100. Also known as "Partial Abstration".

-> Interface Example: "img" tag in html. we need to override "src" everytime because there is no default image set. Also known as "Full Abstration".

-> We can say interface is just a next step of abstraction (necessary to implement all the methods), otherwise it is almost same.

--> Abstraction VS Encapsulation:

-> OOP concepts that allow real objects to be implemented in Code and program.

-> Encapsulation binds data and functionalities into one component while limiting access to certain components has been established.

-> Works on the design level => Works on the application level

-> Hide unnecessary data and withdrawing relevant data => Hiding the code and the data together from the outside world or misuse.

-> It highlights what is the work of an object instead of how the object works => It focuses on the inner details of how the object works.

-> Focuses on outside viewing, for example shifting the car => Focuses on internal working or inner viewing, for example, the production of the car.

-> Abstraction is supported in Java with the interface and the abstract class => Supported using e.g. public, private and secure access modification systems.

-> Hiding implementation with the help of an interface and an abstract class. => Encapsulation is hiding the data with the help of getters and setters.

--> Interitance:

-> Public:

-> One of the ways of implementing the has-a relationship.

-> With private inheritance, public and protected member of the base class become private members of the derived class.

-> Public => Public

-> Protected => Protected

-> Private => Private

-> Protected:

-> Public => Protected

-> Protected => Protected

-> Private => Not Accessible

-> Private:

-> Public => Not Accessible

-> Protected => Not Accessible

-> Private => Not Accessible

--> Composition VS Aggregation VS Association:

-> A relationship between two objects is referred to as an association.

-> An association is known as composition when one object owns another.

-> An association is known as aggregation when one object uses another object.

-> Child cannot exist independent of the parent in composition like House has Rooms.

-> Child can exist independent of the parent in aggregation like ClassRoom has Students.

--> Overloaded Functions:

-> By changing number of Arguments.

-> By having different types of argument.

-> By changing' sequence of parameters.

-> Example:

-> int sum(int, int) , double sum(int, int) // Both are not overloaded.

-> func (int i, double j) , func (double i, int j) // Both are overloaded.

-> Example:

-> demoFunction(float i, double j), demoFunction(double i, float j)

-> Parameter (3, 3.0) // first fucntion will be called

-> Parameter (3.0, 3) // second fucntion will be called

-> Parameter (3.0, 3.0) // Now, Error will occour.

--> Concrete class:

-> A concrete class is a class that has an implementation for all of its methods. They cannot have any unimplemented methods.

--> Interface VS Abstract Class:

-> We cannot keep variable and objects in interface but we can do in abstract class.

-> There are some methods which are not needed to redefine. we can do it in abstract class by declaring it as non abstract method.

-> We cannot create objects of both.

-> It is mandatory to implement all the methods of interface and abstract class.

--> Final Keyword:

-> Final Variables: To create constant variable

-> Final Method: Prevent method overriding

-> Final Class: Prevent inheritance

--> Virtual-function VS Pure-virtual-fucntion:

-> Having definition in the base class => on=ly decaration in the base class.

-> If derived class does not implement the virtual function, no compile error ocuurs.

-> If derived class does not implement the Pure virtual function, no compile error ocuurs but child class will become abstract like base class.

-> Pure Virtual functions can be given a small definition in the Abstract class, which you want all the derived classes to have.

->

--> Functions that are never Inherited:

-> Constructors, Destructors, assignment operator.

OOP with JS

--> General Points:

-> When we need to use parent class variable, we will use "this" key word in child class.

-> When we need to use parent class method, we will use "super" key word in child class.

-> When we need to use self class method, we will use "this" key word in child class.

-> We cannot call function without "this" or "super". like this.func(); // self func call super.func();//parent func call

-> Statis variable is related to the class not the object.

-> When we need to access the static variable in child class, we will use parent class name despite "this" keyword.

-> We can access static variable outside any class using class name.

-> We can access simple variable outside any class using object name.

-> We can update a value of a static variable but not of a const variable.

-> We can access static variable through out the file and these variable will reside in the memory till application life.

-> We can measure count of the objects of a class by declaring a static variable and increment it in constructor and decrement it in destructor.

-> A static member function can be called even if no objects of the class exist and the static functions are accessed using only the class name and the scope resolution operator(::)

-> We cannot override static method.

-> Overloading = dynamic binding

-> Overriding = static binding

-> There is no overloading in javascript.

->

--> Access Modifiers:

-> default is public.

-> use "#" for private decaration

Reference Links

- *https://www.studytonight.com/cpp/upcasting.php*