Object oriented programming- C#

Application : set of programs. We need to design and understand program before developing an application.

Program : set of instructions.

Elements of program:

- Variables : identity given to memory location.
- Identity : unique. Programs classes and methods have unique identity.
- Methods : block of instructions with identity. Performs operations on data. Takes input data, perform operations and return the results.

C# is an object oriented programming language.
But it is not 100% object oriented as it has primitive data types too, which can be accessed without creating object for the data.

What is OOPS?
Concept of defining objects and establishing communication between them.
Principles of OOPS:

- Encapsulation : protecting the data within the class itself. Using getter setter methods. Variables are private and methods (get and set) are public.
  Abstraction : hides implementation and shows the functionlity.
- Polymorphism : ability to take many forms. Concept where object behaves differently in different situations
- Inheritance : defining the new class by using the members and methods of previous class and also can add extra functionality in the new class.

Building blocks:

- Class : blueprint. Contains variables and methods
- Object : Instance of class. Instance variables of class get memory inside the class
- Methods
- Variables

Variables

Store information of class(object).

Four types of variables:

- Static variables : can only be accessed by class. These variables are same for all the instances created from the particular class.
- Instance variables : accessed with the help of object. Can be different for differemt instances of the same class.
- Local variables : accessible only inside the method. Store processed information inside the method.

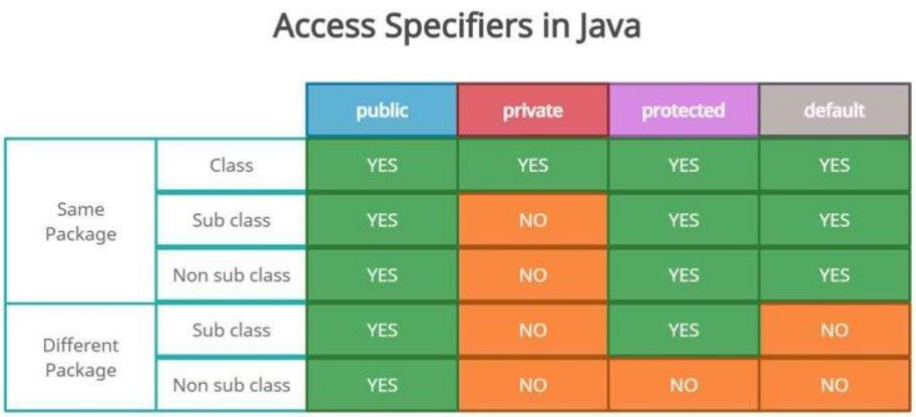
- Method Parameters : variables inside the method.

Static variables when defined with a data type, they have a default value.

- Int - 0
- Double – 0
- Char – blank
- Boolean – False
- String – null

Access Modifiers

- Private : can only be accessed in the defined class
- Public : can be accessed from anywhere in the project
- Protected : can be accessed only by the defined and the derived classes
- Internal : provides access in the same module
- Protected internal : provides access in the same module of the class defines and the derived classes.
- Default access modifier of variable in classes, struct and enum is Internal
- For member of a class it is private.(if we dont explicitly mention the access modifier)



Constructor Chaining

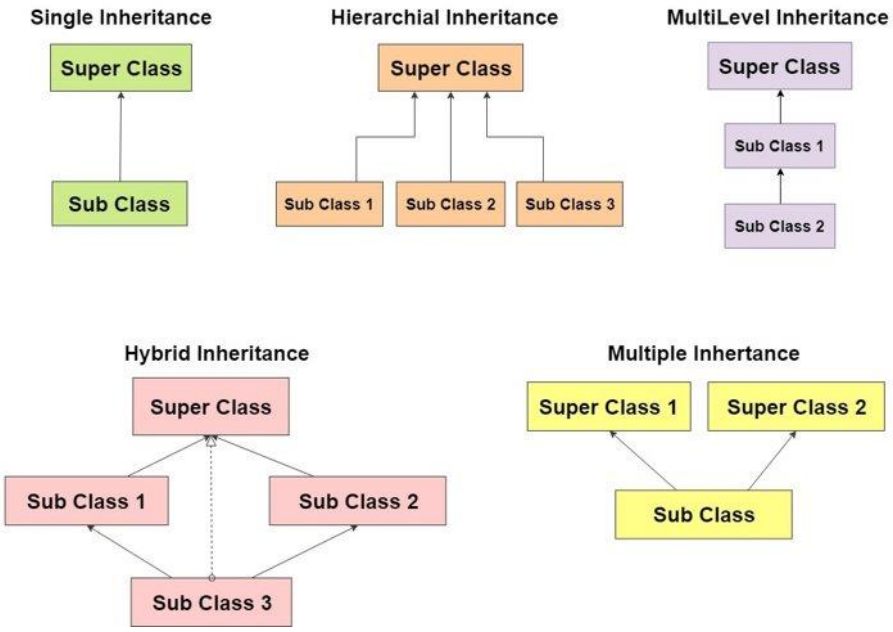
```
public class Test
{
        public Test() : this(10) {
                Console.WriteLine("Zero args"); }
        public Test(int x) : this(10, 20) {
                Console.WriteLine("Args"); }
        public Test(int x, int y) {
                Console.WriteLine("Two args"); }
```

```
}

 public class Access {
public static void Main() { Test obj = new Test();
 }
 }
```

Output:
Two Args
Args
Zero Args

Inheritance



Multiple Inheritance can be implemented by Interfaces

This keyword
refers to the current instance of the class.
It is often used to differentiate between instance variables and parameters or to invoke other instance members within the same class
invoke other constructors in the same class (constructor chaining).

Base Keyword
refers to the base class (parent class) from within a derived class.
to access members of the base class, including methods, properties, and fields.
often used when there is a name conflict between a member in the derived class and a member in the base class.

Object Upcasting
Can store address of child class into a parent class reference variable.
Parent ad=new Child();
(OR)
Child ad=new Child();
Parent p=ad;

Object Downcasting
Collecting child object type from the parent reference
Parent p=new Child();
Child c=(Child)p

Abstract Classes
Cant be instantiated
Serves as blueprint for other classes
Contain both concrete as well as abstract methods.
Concrete methods: method with body
Abstract methods: method without body
Abstract methods are meant to be implemented in the sub classes.
Concrete methods used in abstract classes, can be overridden by sub classes or used as it is in the sub classes.
Can contain properties and fields like other classes.
Derived / sub classes of the abstract classes must contain all the implementation of the methods in abstract class.

Interfaces
A contract that has a set of methods, properties that a class must implement.
Multiple inheritance can be implemented with the help of ineterfaces.
Interface methods are public abstract by default.

Data Types

Define what and how data should be used

C# is a type safe language

Type safe language: variables and expressions must be used in a manner consistent with their declared types, and type-related errors are caught at compile-time rather than at runtime. Makes programs more robust and helps in maintaining code readability and maintainability as errors will be caught at the runtime.

In C#, there are two types of data types are used:

1. Value Types : contains only an object with the value.

int a=50

2. Reference types : contains a reference to an object. Another variable may contain a reference to the same object.

Object obj=new Object();

obj.val=50;

Predefined data types: Size of these data types are predefined.

string X = "\"Hello \nHow are you\""; \ -> “ , \n->new line

There are three types:

1) Value Types:
   Integral Types:
   Signed : sbyte,short,int,long
   Unsigned : byte, ushort,uint,ulong
   - byte: 8-bit unsigned integer (0 to 2^8-1)
   - sbyte: 8-bit signed integer (-2^7 to 2^7-1
   - short: 16-bit signed integer (-2^15 to 2^15-1)
   - ushort: 16-bit unsigned integer (0 to 2^16-1)
   - int: 32-bit signed integer (-2^31 to 2^31-1)
   - uint: 32-bit unsigned integer (0 to 2^32-1)
   - long: 64-bit signed integer (-2^63 to 2^63-1)
   - ulong: 64-bit unsigned integer (0 to 2^64-1)

   Floating-Point Types:
   - float: 32-bit single-precision floating-point (a real number represented using 32 bits of memory, with 24 bits dedicated to the significant digits)
   - double: 64-bit double-precision floating-point (a real numnber represented using 64 bits of memory, with 53 bits dediacted to significant digits allowing a wide range of precission. The larger the number of bits allocated for the significant digits, the higher the precision of the floating-point representation.)
   - decimal: 128-bit decimal floating-point (useful for financial calculations) (a real number represented using 128 bits of memory, with a base-10 representation for precise decimal calculations.)

   Character Types:
   - char: 16-bit Unicode character

   Boolean Type:
   - bool: Represents true or false values

   Reference Types:
   - Object Type:
     object: The ultimate base class for all types in C#.
   - String Type: Represents a sequence of characters.

   User-Defined Types:
   - Struct Type:

struct: Represents lightweight, stack-based structures.

- Enum Type:
  enum: Represents a set of named integral constants.

Void Keyword

Return type
To indicate that method doesnot return any value.
Method performs an action but doesnot return any value.

```
public void DisplayMessage()
 {
        Console.WriteLine("Hello, world!");
}
```

1byte= 8 bits
Int – 32 bit (4bytes)
Int16 – 16 bits (2 bytes)
Int64 – 64 bits (8 bytes)

Boxing and Unboxing

Involve the conversion between value and reference types.

Boxing:

- Converting value type to reference type
- When a value type is boxed, new object is created on the top of the heap
- The value of the value type is copied to the allocated object.
- This allows value types to be treated as objects
  Ex : int a=10;
  Object intobj=a;

Unboxing:

- Reverse process of boxing
- Converts the previously converted reference type into the respective value type.
  Ex: Object intobj=10;
  int a=(int)intobj;

Object

Reference type
Can store any value but actual type is known at runtime.
Generally preferred when we dont know the data type.

Can use explicit casting to access the original type members
Ex : Object o;
    o = 10;

Var

Implicitly typed local variable.
The compiler identifies the type of the variable based on the initialization expression.
The type is determined at compile time, and once the type is inferred, it cannot be changed.
var is most used in cases where the type is obvious from the right-hand side of the assignment.

```
Var a = 10;
Var b = "string value";
Var c = new Product(); // object of product type
Var d;    //Error, as the value is not assigned at time of declaration of variable
```

Dynamic

A type introduced in C# 4.0 that allows you to work with objects whose type is not known until runtime.
The type is resolved at runtime, and method invocations and member accesses are resolved dynamically.
While dynamic provides flexibility, it sacrifices compile-time type checking.
Object unboxing is not required.

```
dynamic myDynamic = 42;
Console.WriteLine(myDynamic.GetType());
myDynamic = "Hello, World!";
Console.WriteLine(myDynamic.GetType());
```

Git and Github

Version control system : tracks changes in code.
Git is popular, free,open source, fast and scalable
Features of git:

- To track the history of the project
  Ex : while building a website, if we think to add extra features but that extra features have some issues, so we want to implement in the later version but the changes made in this version should be preserved and the new change which we left for next version should be build on top of the changes made in this version.
- To collobarate with other people so that it doesnt create any conflicts when two or more people work on the same module.

Github : website where developers store and manage the code.

We upload project in the form of folder. These folders are called as reposiotory.(repos)

Clone: git clone <link>
Status : git status

Change or modify file (untracked file) ----> add file (staged file) ----> commitg file (unchanged)

Add (stage)
To add new or changed files into working directory in the staging area
Git add <file name>
Git add . (to add all the files from the folder

Commit
To record the change
Git commit –m 'message'

Push
To add files from the local computer to the repository
Local->repo
Git push origin <branch name>

Init
To create new git repository
Git init
Git remote add origin <link>
Git remote –v (to verify origin)
Git branch (to check branch)
Git branch –m rename_branch_name (to rename branch name)
Git push origin rename_branch_name

Branches
Git branch (to check in what branch we are)
Git branch –m new_name
Git checkout another_branch (to go from one branch to another)
Git checkout –b new_branch (to move into new branch)
Git branch –d branch_name (to delete branch)

Merge
1)Git diff branch_name (to compare commits,branches and everything that is changed in the branch)
2)Git merge main (add the compared branch to main
Another method is to use pull request

Pull

Used to fetch and download content from the remote repository and immediately update the local repository to match the content with the remote repositpory

Git pull origin main