Core Java Important Notes

What is OOPs?

- It stands for Object-Oriented Programming.
- It is based on objects
- It follows Bottom-up programming approach.
- It is based on real world.
- It provides data hiding so it is very secure.
- It provides reusability feature.

What is a class?

A class is a collection of objects. Classes don't consume any space in the memory.

It is a user defined data type that act as a template for creating objects
of the identical type.

A large number of objects can be created using the same class. Therefore, Class is
considered as the blueprint for the object.

What is an object?

An object is a real world entity which have properties and functionalities.
Object is also called an instance of class. Objects take some space in memory.

For eg .
Fruit is **class** and its **object** s are mango ,apple , banana
Furniture is **class** and its **objects** are table , chair , desk

What is the difference between a class and an object?

Class	Object
1. It is a collection of objects.	It is an instance of a class.
2. It doesn't take up space in memory.	It takes space in memory.
3. Class does not exist physically	Object exist physically.
4. Classes are declared just once	Objects can be declared as and when required

What is the difference between a class and a structure?

Class	Structure
1.Class is a collection of objects.	Structure is a collection of variables of different data types under a single unit
2. Class is used to combine data and methods together.	Structure is used to grouping data.
3. Class's objects are created on the heap memory.	Structure's objects are created on the stack memory.
4. A class can inherit another class.	A structure can't inherit another structure.
5. A class has all members private by default	A structure has all members public by default
6. Classes are ideal **for larger** or complex objects	Structures are ideal **for small** and isolated model objects

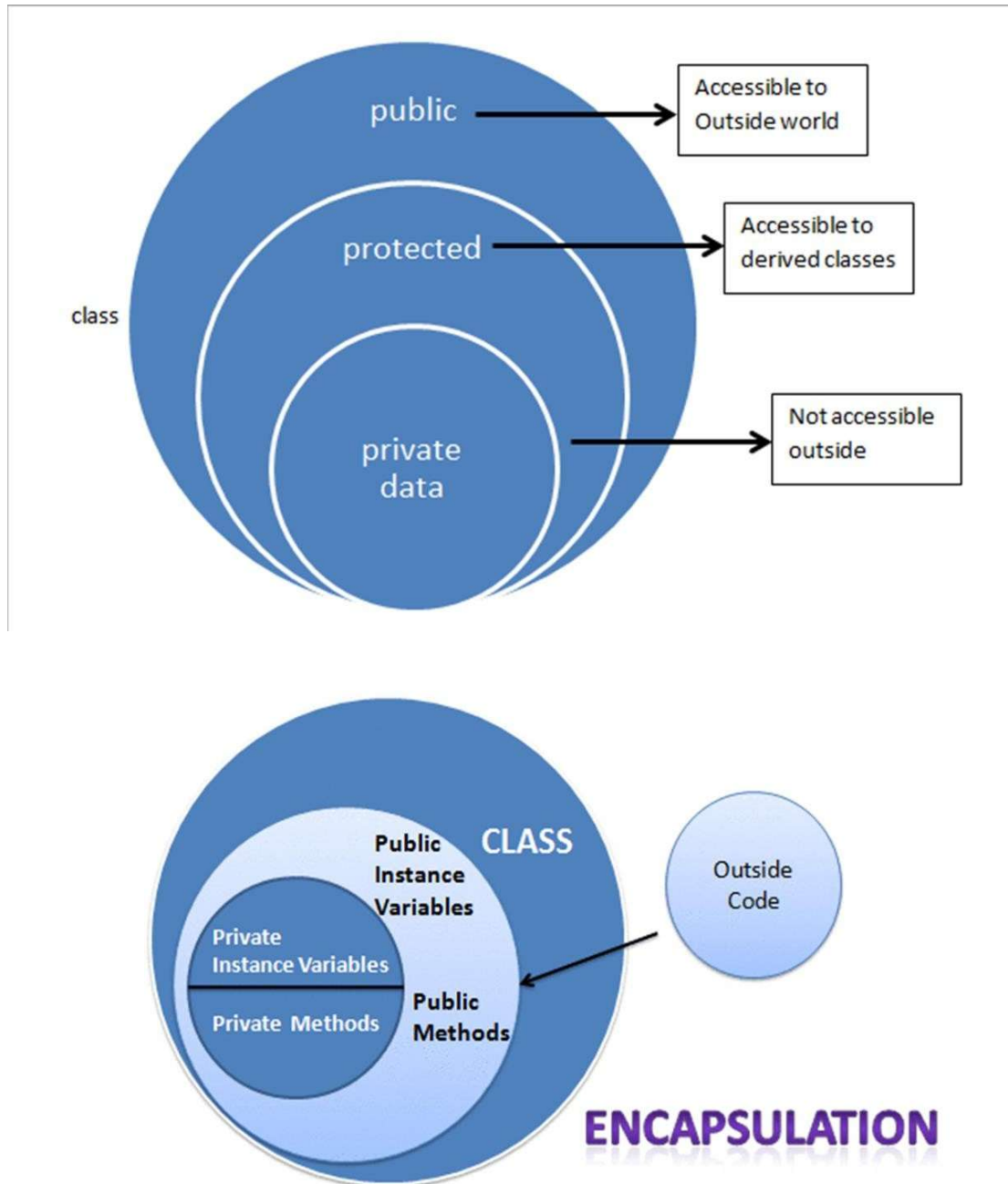## Following are the basic features of OOPs -
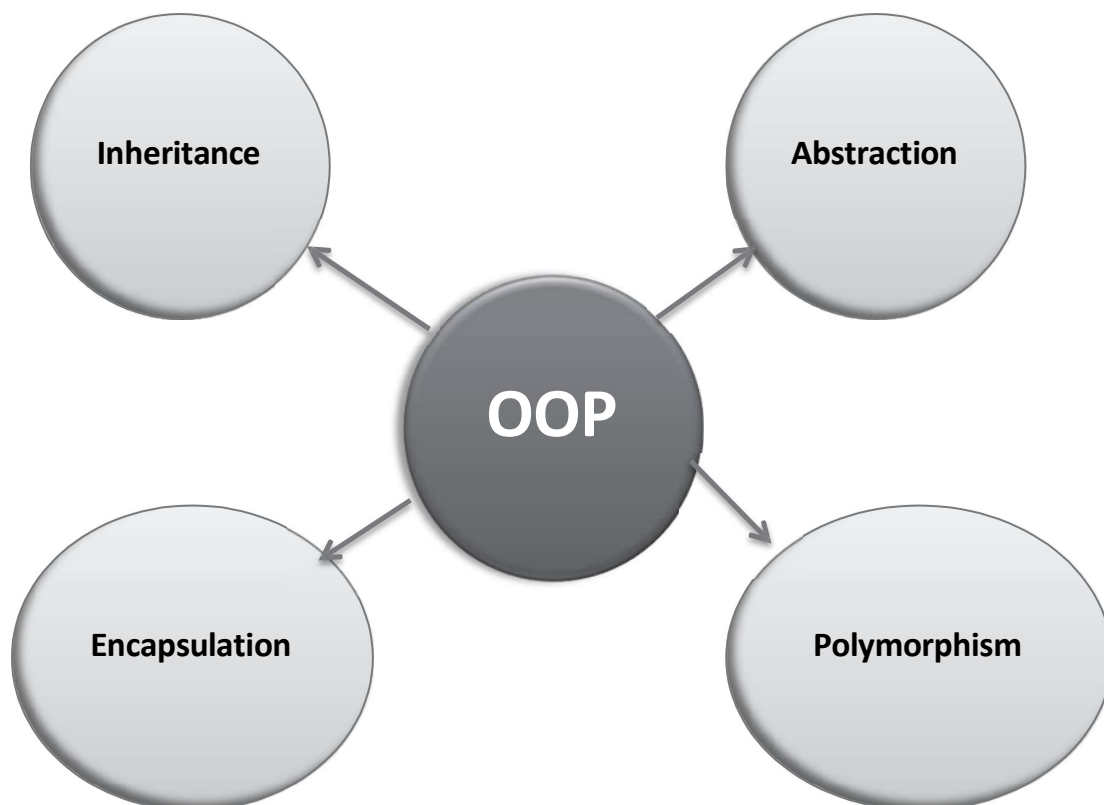
Encapsulation

The main advantage of encapsulation is that data is hidden and protected from randomly access by outside non-member methods of a class.

Encapsulation is the process of **binding** data and methods in a **single unit.**

In encapsulation, data(variables) are declared as private and methods are declared as public.

What are access specifiers ?

It allows us to restrict the scope or visibility of a package, class, constructor, methods, variables, or other data members.

There are three types of most **common access specifiers,** which are following.
• Private
• Public
• Protected

Public Modifiers :

means that class, variable or method is accessible throughout from within or outside the class, within or outside the package, etc.

It provides **highest level** of accessibility.

Private Modifiers :

means that class, variable or method is not accessible from within or outside the class, within or
 outside the package, etc.

Private field or method can't be inherited to sub class.

This provides **lowest level** of accessibility.

Protected Modifiers :

means that class, variable or method is accessible from classes in the same package, sub-classes in
 the same package, subclasses in other packages but not accessible from classes in other packages.

Access Modifiers	Accessible by classes in the same package	Accessible by classes in other packages	Accessible by subclasses in the same package	Accessible by subclasses in other packages
Private	NO	No	No	No
Public	Yes	Yes	Yes	Yes
Protected	Yes	NO	Yes	Yes

Allows to hide unnecessary data from the user. This reduces program complexity efforts.

it displays only the necessary information to the user and hides all the internal background details.

If we talk about data abstraction in programming language, the code implementation is hidden from the user and only the necessary functionality is shown or provided to the user.

In other words , it deals with the outside view of an object (Interface).

Eg.
-All are performing operations on the ATM machine like cash withdrawal etc.
but we can't know internal details about ATM

-phone call we don't know the internal processing

We can achieve data abstraction by using
 1. Abstract class
2. Interface

What is an abstract class?
Abstract class is that class which contains abstract method.

Abstract methods are those methods which have only declaration not the implementation.

An abstract class is declared with abstract keyword.

An abstract class can also contain non-abstract methods.

Inheritance is the procedure in which one class inherits the attributes and methods of another class.
In other words It is a mechanism of acquiring properties or behaviors of existing class to a new class

The Base Class, also known as the Parent Class is a class, from which other classes are derived.

**The Derived Class**, also known as Child Class, is a class that is created from an existing class

There are four types of inheritance in OOP:
- Single Level Inheritance
- Hierarchical Inheritance
- Multi-Level Inheritance
- Multiple Inheritance
- Hybrid inheritance

Single Level Inheritance

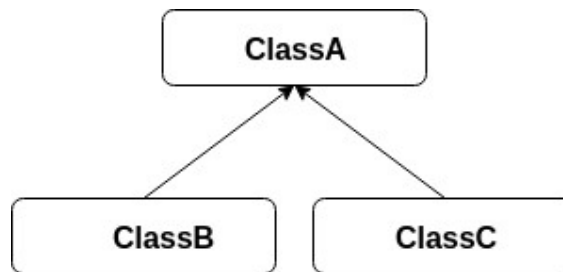When a class inherits properties and behaviour of only one class.
In other words, in single inheritance there is only one base class and only one sub class.

ClassA

↑
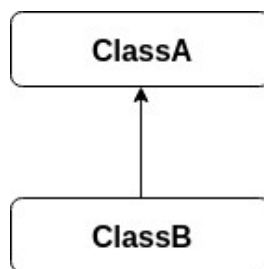
ClassB

Single Inheritance
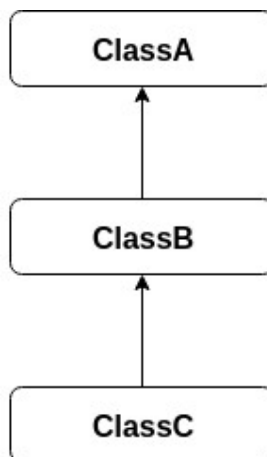
Hierarchical Inheritance

When more than class inherit properties and behaviour of only one class
In Hierarchical Inheritance there are only one parent and many child class

Hierarchical Inheritance

Multi-Level Inheritance
In this type of inheritance, a derived class is created from another derived class
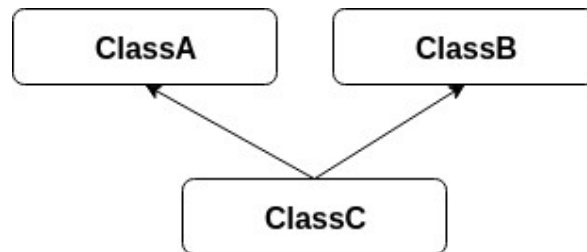

Multilevel Inheritance

Multiple Inheritance

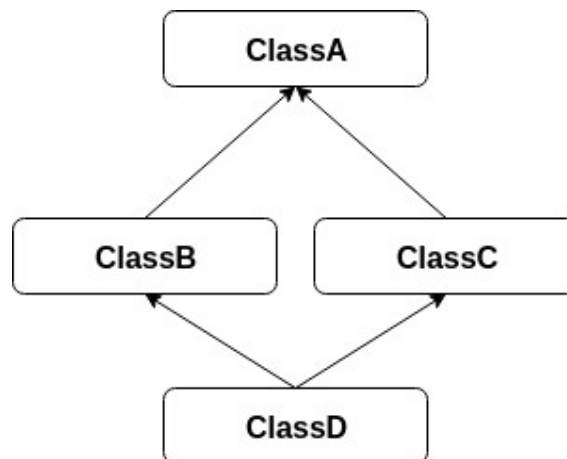When a class inherits the properties and the behaviour of more than one class

Java, C#, most of high level language don't support Multiple Inheritance

Multiple Inheritance

Hybrid Inheritance

Hybrid inheritance is a of inheritance is a combination of more than

one type of inheritance.

Hybrid Inheritance

## Why Java or C# don't support multiple inheritance?

because of following reasons –

Ambiguity Around The Diamond Problem Multiple inheritance does complicate the design and creates problem during casting, constructor chaining etc.

Polymorphism is the ability of an object to take on **many forms.**
we can define polymorphism as the ability of a message to be displayed in more than one form.

SHAPE
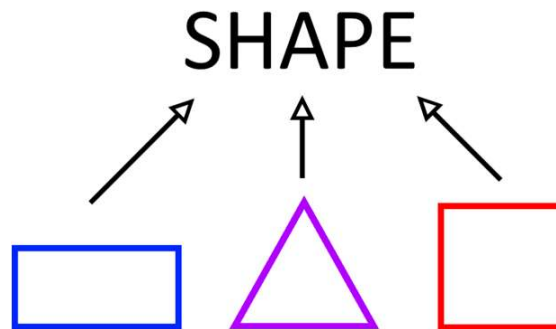
A real-life example of polymorphism, a man at the same
 time is a father, a husband, an employee.

Another good real time example of polymorphism is water. Water is a liquid at normal temperature, but it can be changed to solid when it frozen, or same water changes to a gas when it is heated at its boiling point .Thus, same water exhibiting different roles is polymorphism.
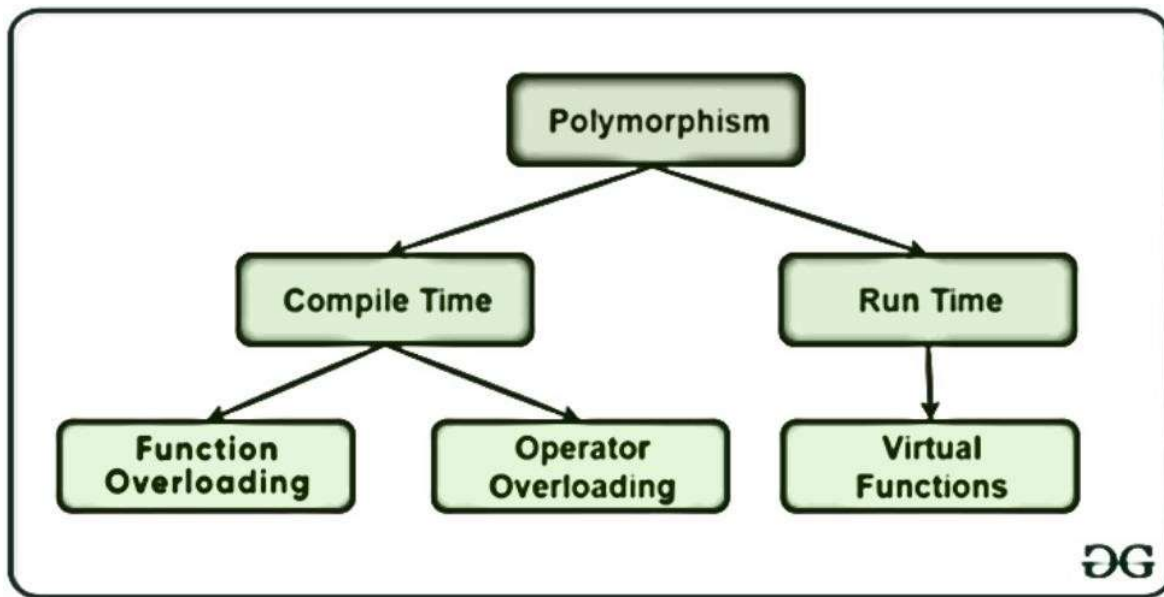
Polymorphism

Many   Forms
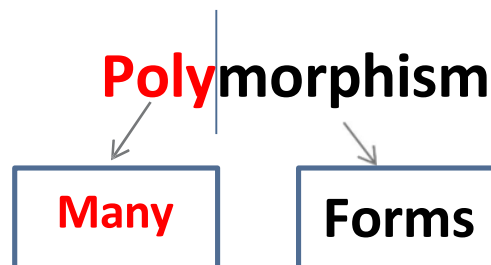
polymorphism is mainly divided into two types:

• **Compile time Polymorphism (CTP)**

It is also called **static** polymorphism or **early binding.**

• **Runtime Polymorphism (RTP)**

It is also called **dynamic** polymorphism or **late binding.**

Types of Polymorphism:

1. Compile time polymorphism:
This type of polymorphism is achieved by function overloading or operator overloading.

Function overloading:
When there are multiple functions with same name but different parameters then these functions are said to be overloaded. Functions can be overloaded by change in number of arguments or/and change in type of arguments
multiple methods of same names performs different tasks within the same class.

2.Runtime polymorphism:
Runtime polymorphism refers to the process when a call to an overridden process is resolved at the run time.
This type of polymorphism is achieved by Function Overriding.

Function Overriding:
on the other hand, occurs when a derived class has a definition for one of the member functions of the base class.
methods having same name which can have different functionalities.
That base function is said to be overridden.

Overriding

```java
class Dog{
    public void bark(){
        System.out.println("woof ");
    }
}
class Hound extends Dog{
    public void sniff(){
        System.out.println("sniff ");
    }

    public void bark(){
        System.out.println("bowl");
    }
}
```

Same Method Name,
Same parameter

Overloading

```java
class Dog{
    public void bark(){
        System.out.println("woof ");
    }

    //overloading method
    public void bark(int num){
        for(int i=0; i<num; i++)
            System.out.println("woof ");
    }
}
```

Same Method Name,
Different Parameter

Different between Abstract Classes & Interfaces

Features	Abstract Class	Interface
Multiple Inheritance	A class can inherit only one abstract class	A class can inherit multiple interfaces
Default Implementation	Provide signature ,partial and full implementation of its methods ,variables and other members	Provide only the signature of its methods ,variables ,properties and other member
Access Modifier	It allows to assign access modifier to its members	No access modifier can be assigned .All the members are treated as public
Core VS Peripheral	It defines the core identity of the class and there is used for objects of same type	It identify the peripheral identity of the class .it means human and vehicle can inherit from IMovable interface
Homogeneity	If various implementation of same nature which requires shared code that represent same status or behaviour , then use Abstract class	If various implementation of different nature and requires the member with same signature ,then use interface
Performance	It is faster to access the implemented class member	It takes time to find the members of the corresponding class
Extensibility (Versioning)	If any changes made to the abstract class ,not necessarily We need to change all the implementation classes	If any changes made to the interfaces , changes should be made in all the implemented classes
Field and Constants	Fields and constants can be defined	No fields and constants can be defined

What is static function?

Static functions are those functions that can be called without creating an object of the class. That means, Static methods do not use any instance variables of any object of the class they are defined in.

Static methods can not be overridden. They **are stored in heap** space of the memory.

What are virtual functions?

Virtual function is a function or method used to override the behavior of the function in an inherited class with the same signature to achieve the polymorphism.

Virtual function defined in the base class and overridden in the inherited class.

The Virtual function **cannot be private**, as the private functions cannot be overridden. It is used to achieve runtime polymorphism.

What are pure virtual functions?

A pure virtual function is that function which have no definition. That means a virtual function that doesn't need implementation is called pure virtual function.

A pure virtual function have not definitions but we must override that function in the derived class, otherwise the derived class will also become abstract class.

What is Constructor?

Constructor is a special type of member function which is used to initialize an object.

It is similar as functions but it's name should be same as its class name and must have no explicit return type.

It is called when an object of the class is created. At the time of calling constructor, memory for the object is allocated in the memory.

We use constructor to assign values to the class variables at the time of object creation.

What are the types of Constructor?

Constructor have following types –

• Default constructor

• Parameterized constructor

• Copy constructor

• Static constructor

• Private constructor

What is default constructor?

A constructor with 0 parameters is known as default constructor.

What is private constructor?

if a constructor is declared private, we cannot create an object of the class.

What is copy constructor?

A copy constructor is that constructor which use existing object to create a new object.

It copy variables from another object of the same class to create a new object.

What is static constructor?

A static constructor is automatically called when the first instance is generated,

or any static member is referenced.

The static constructor is explicitly declared by using a static keyword

What is destructor?

Destructor is a type of member function which is used to destroy an object.

 It is called automatically when the object goes out of scope or is explicitly destroyed by a call to delete.

It destroy the objects when they are no longer in use.

A destructor has the same name as the class, preceded by a tilde (~).

Shallow Copy and Deep Copy

Shallow Copy and Deep Copy play important role in copying the objects in Prototype Design Pattern.

Shallow copy

In the case of Shallow copy, it will create the new object from the existing object and then copying the **value type** fields of the current object to the new object.

But in the case of **reference type**, it will only copy the reference, not the referred object itself.

Therefore the original and clone refer to the same object in the case of reference type. In order to understand this better, please have a look at the following diagram.
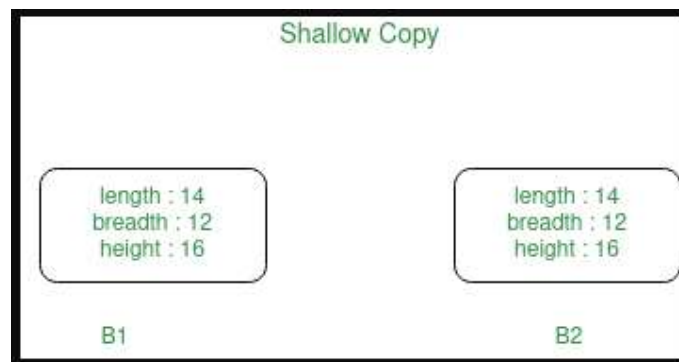
Example: Shallow Copy

```csharp
public class Employee
{
    public string Name { get; set; }
    public string Department { get; set; }
    public Address EmpAddress { get; set; }

    public Employee GetClone()
    {
        return (Employee)this.MemberwiseClone();
    }
}

public class Address
{
    public string address { get; set; }
}
```
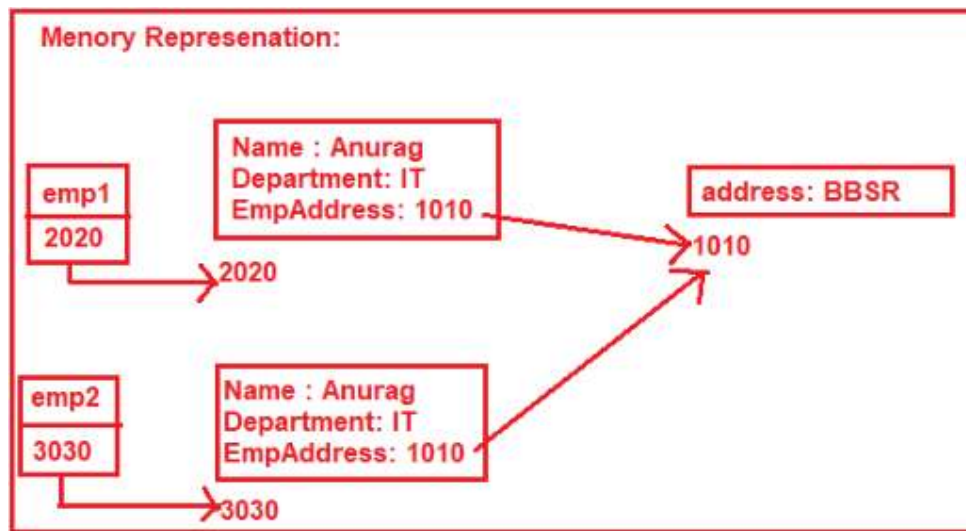
```csharp
static void Main(string[] args)
{
    Employee emp1 = new Employee();
    emp1.Name = "Anurag";
    emp1.Department = "IT";
    emp1.EmpAddress = new Address() { address = "BBSR" };

    Employee emp2 = emp1.GetClone();
    emp2.Name = "Pranaya";
    emp2.EmpAddress.address = "Mumbai";
}
```

Client Code

Menory Represenation:

As shown in the above diagram, first we create an object i.e. emp1, and then initialize the object with some values. Then we create the second object i.e. emp2 using the GetClone method. As shown in the memory representation, **the value type** fields (Name and Department) are copied and stored in a different memory location while **the reference type** field i.e. EmpAddress is still pointing to the same old memory location. That means now, both the object i.e. emp1 and emp2 is now referring to the same Address object. So, if we do any changes to the employee address then it will affect each other.

Deep Copy

In the case of deep copy, it will create the new object from the existing object and then copying the fields of the current object to the newly created object. If the field is a value type, then a bit-by-bit copy of the field will be performed. If the field is a reference type, then a new copy of the referred object is created.
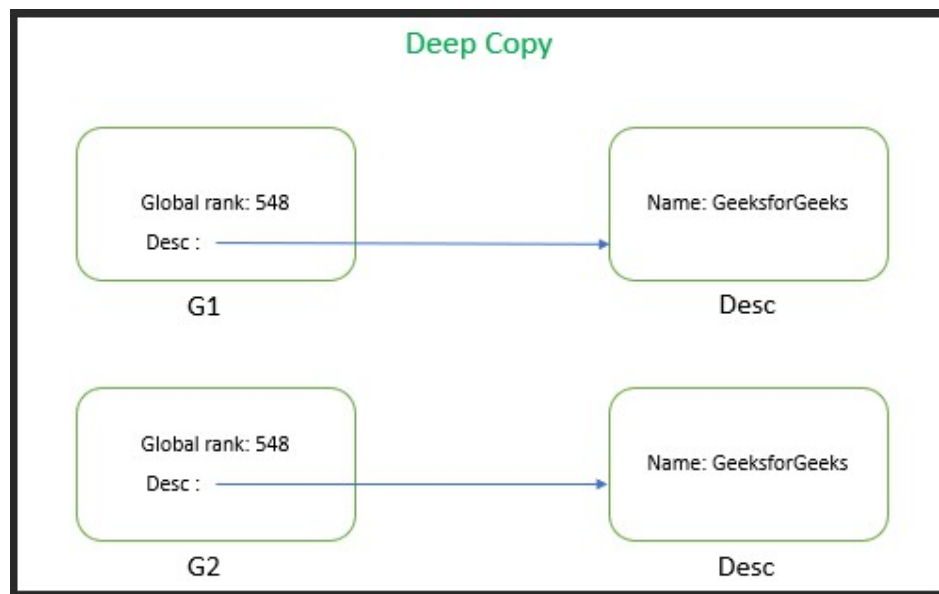
```csharp
public class Employee
{
    public string Name { get; set; }
    public string Department { get; set; }
    public Address EmpAddress { get; set; }

    public Employee GetClone()
    {

        Employee employee = (Employee)this.MemberwiseClone();
        employee.EmpAddress = EmpAddress.GetClone();
        return employee;
    }
}

public class Address
{
    public string address { get; set; }
    public Address GetClone()
    {
        return (Address)this.MemberwiseClone();
    }
}
```
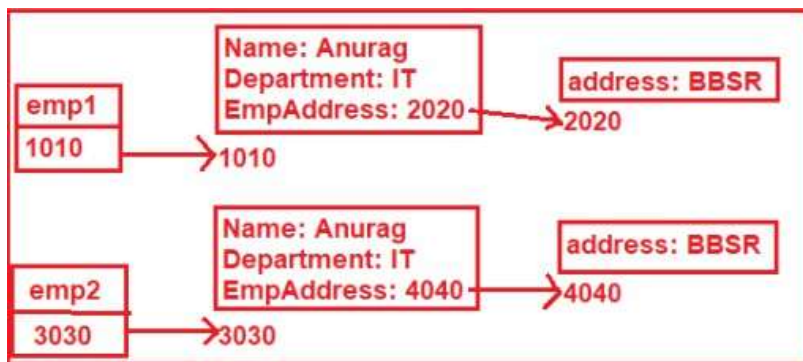
```csharp
static void Main(string[] args)
{
    Employee emp1 = new Employee();
    emp1.Name = "Anurag";
    emp1.Department = "IT";
    emp1.EmpAddress = new Address()
    {
        address = "BBSR"
    };

    Employee emp2 = emp1.GetClone();
    emp2.Name = "Pranaya";
    emp2.EmpAddress.address = "Mumbai";
}
```

Client Code

Name: Anurag
Department: IT
EmpAddress: 2020 → **address: BBSR**
→ **2020**

emp1
1010 → **1010**

Memory Represenatation

Name: Anurag
Department: IT
EmpAddress: 4040 → **address: BBSR**
→ **4040**

emp2
3030 → **3030**

As shown in the above image, the Name and Department properties are **value types** so it creates a copy of that and stores it in a different location. The EmpAddress is a **Reference type** property and in Deep Copy there is a clone of the reference type field which also will be stored in a different location. So, the point that you need to keep in mind is, in the case of Deep Copy the field type does not matter whether it is a value type or reference type. It always makes a copy of the whole data and stores it in a different memory location.

In C++ we can pass arguments into a function in different ways. These different ways are

- Call by Value
- Call by Reference
- Call by Address

Sometimes the call by address is referred to as call by reference, but they are different in C++. In call by address, we use pointer variables to send the exact memory address, but in call by reference we pass the reference variable (alias of that variable). This feature is not present in C, there we have to pass the pointer to get that effect. In this section we will see what are the advantages of call by reference over call by value, and where to use them

Call by Value

In call by value, the actual value that is passed as argument is not changed after performing some operation on it. When call by value is used, it creates a copy of that variable into the stack section in memory. When the value is changed, it changes the value of that copy, the actual value remains the same.

Example Code

```cpp
#include<iostream>
using namespace std;

void my_function(int x) {
    x = 50;
    cout << "Value of x from my_function: " << x << endl;
}

main() {
    int x = 10;
    my_function(x);
    cout << "Value of x from main function: " << x;
}
```

Output

Value of x from my_function: 50

Value of x from main function: 10

Call by Reference

In call by reference the actual value that is passed as argument is changed after performing some operation on it. When call by reference is used, it creates a copy of the reference of that variable into the stack section in memory. Is uses a reference to get the value. So when the value is changed using the reference it changes the value of the actual variable.

```cpp
#include<iostream>
using namespace std;

void my_function(int &x) {
  x = 50;
  cout << "Value of x from my_function: " << x << endl;
}

main() {
  int x = 10;
  my_function(x);
  cout << "Value of x from main function: " << x;
}
```

Output

Value of x from my_function: 50
Value of x from main function: 50

Where to use Call by reference?

- The call by reference is mainly used when we want to change the value of the passed argument into the invoker function.
- One function can return only one value. When we need more than one value from a function, we can pass them as an output argument in this manner