Object Oriented Programming Fundamentals

- **What is the main difference between a class and an object?**

- A class is a template for creating objects in program whereas the object is an instance of a class.
- A class is a logical entity while object is a physical entity.
- A class does not allocate memory space on the other hand object allocates memory space.
- You can declare class only once but you can create more than one object using a class.
- Classes can't be manipulated while objects can be manipulated.
- Classes doesn't have any values, whereas objects have its own values.
- You can create class using "class" keyword while hand you can create object using "new" keyword in Java.

- **What is Encapsulation? Explain with a used case**

Encapsulation is defined as wrapping up of data and information under a single unit. In Object Oriented Programming, Encapsulation is defined as binding together the data and the functions that manipulates them.

We can create a fully encapsulated class in Java by making all the data members of the class private. Now we can use setter and getter methods to set and get the data in it.

The Java Bean **class is the example of a fully encapsulated class.**

- **What is Polymorphism? Explain with a used case**

Polymorphism is the ability of an object to take on many forms. The most common use of polymorphism in OOP occurs when a parent class reference is used to refer to a child class object.

Let us look at an example.

```
public interface Vegetarian{}
public class Animal{}
public class Deer extends Animal implements Vegetarian{}
```

Now, the Deer class is considered to be polymorphic since this has multiple inheritance. Following are true for the above examples −

- A Deer IS-A Animal
- A Deer IS-A Vegetarian
- A Deer IS-A Deer
- A Deer IS-A Object

When we apply the reference variable facts to a Deer object reference, the following declarations are legal −

Example

```
Deer d = new Deer();
Animal a = d;
Vegetarian v = d;
Object o = d;
```

All the reference variables d, a, v, o refer to the same Deer object in the heap.

- **Explain Overriding & Overloading and its advantages**

Overloading occurs when two or more methods in one class have the same method name but different parameters.

Overriding occurs when two methods have the same method name and parameters. One of the methods is in the parent class, and the other is in the child class. Overriding allows a child class to provide the specific implementation of a method that is *already* present in its parent class.

Advantages

- The one main advantage of these overriding and overloading is time-saving.
- Save memory space.
- The readability of the code is increased.
- Here, for function overloading concept, we can use different same function names for different operations eliminating the use of different function names.
- Flexibility and maintainability of code become easier.
- In the case of overriding, the child class can have functions of parent class and can even have its own implementation of that function.
- The objects can be referenced and the functions of both the parent class and child class can be accessed by the child class.

Object Oriented Programming Fundamentals

- **What is Inheritance and different types of inheritance? Explain with a used case**

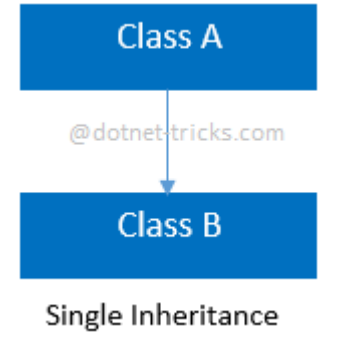
Inheritance is a mechanism in which one class acquires the property of another class. For example, a child inherits the traits of his/her parents. With inheritance, we can reuse the fields and methods of the existing class. Hence, inheritance facilitates Reusability and is an important concept of OOPs.

OOPs support the six different types of inheritance as given below :

- Single inheritance
- Multi-level inheritance
- Multiple inheritance
- Multipath inheritance
- Hierarchical Inheritance
- Hybrid Inheritance

1. Single inheritance

In this inheritance, a derived class is created from a single base class.
In the given example, Class A is the parent class and Class B is the child class since Class B inherits the features and behavior of the parent class A.



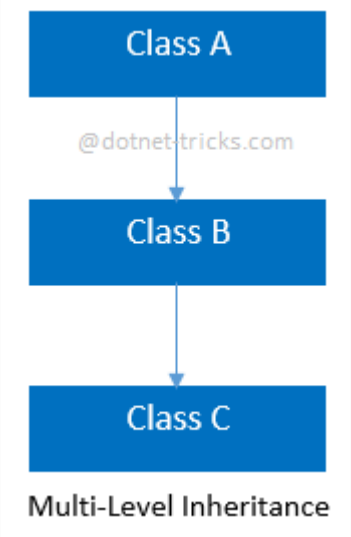
Syntax for Single Inheritance

```
//Base Class

class A

{

 public void fooA()

 {

 //TO DO:

 }

}
```

```
//Derived Class
class B : A
{
 public void fooB()
 {
 //TO DO:
 }
}
```

2. Multi-level inheritance

In this inheritance, a derived class is created from another derived class.
In the given example, class c inherits the properties and behavior of class B and class B inherits the properties and behavior of class B. So, here A is the parent class of B and class B is the parent class of C. So, here class C implicitly inherits the properties and behavior of class A along with Class B i.e there is a multilevel of inheritance.



Syntax for Multi-level Inheritance

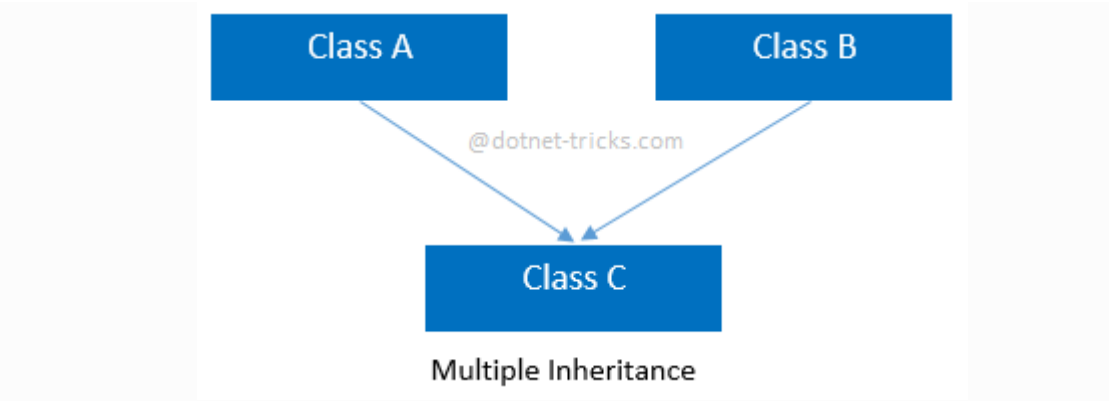
```
//Base Class
class A
{
 public void fooA()
 {
 //TO DO:
```

```
 }
}


//Derived Class
class B : A
{
 public void fooB()
 {
 //TO DO:
 }
}


//Derived Class
class C : B
{
 public void fooC()
 {
 //TO DO:
 }
}
```

3. Multiple inheritance

In this inheritance, a derived class is created from more than one base class. This inheritance is not supported by .NET Languages like C#, F# etc. and Java Language.
In the given example, class c inherits the properties and behavior of class B and class A at same level. So, here A and Class B both are the parent classes for Class C.

Object Oriented Programming Fundamentals



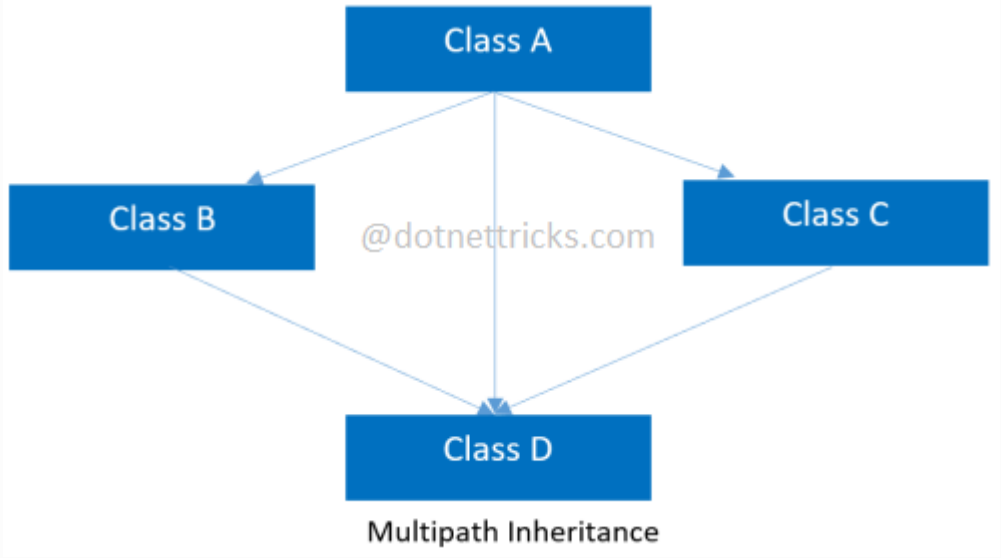
Syntax for Multiple Inheritance

```
//Base Class
class A
{
 public void fooA()
 {
 //TO DO:
 }
}

//Base Class
class B
{
 public void fooB()
 {
 //TO DO:
 }
}

//Derived Class
class C : A, B
{
```

```
public void fooC()

{

//TO DO:

}

}
```

4. Multipath inheritance

In this inheritance, a derived class is created from another derived classes and the same base class of another derived classes. This inheritance is not supported by .NET Languages like C#, F# etc.
In the given example, class D inherits the properties and behavior of class C and class B as well as Class A. Both class C and class B inherits the Class A. So, Class A is the parent for Class B and Class C as well as Class D. So it's making it Multipath inheritance.



Syntax for Multipath Inheritance

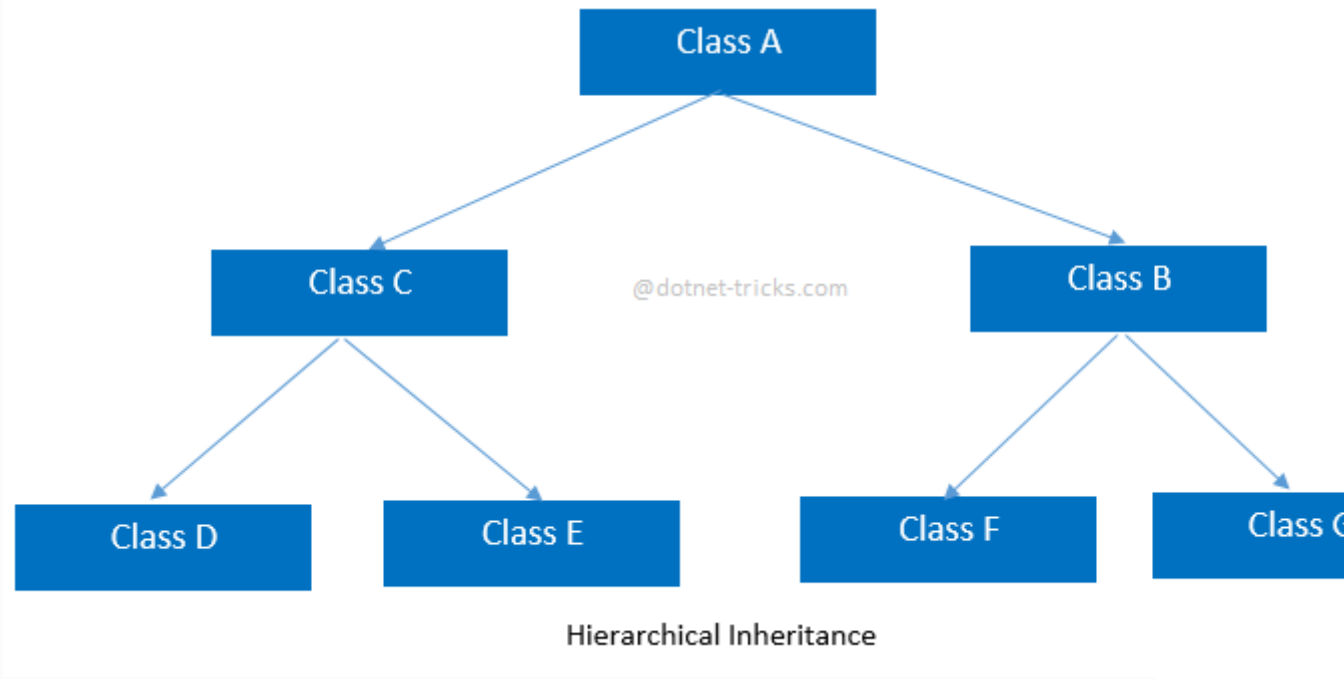
```
//Base Class

class A

{

public void fooA()

{

//TO DO:

}

}
```

```
//Derived Class
class B : A
{
 public void fooB()
 {
 //TO DO:
 }
}

//Derived Class
class C : A
{
 public void fooC()
 {
 //TO DO:
 }
}

//Derived Class
class D : B, A, C
{
 public void fooD()
 {
 //TO DO:
 }
}
```

5. Hierarchical Inheritance

In this inheritance, more than one derived classes are created from a single base class and futher child classes act as parent classes for more than one child classes.

In the given example, class A has two childs class B and class D. Further, class B and class C both are having two childs - class D and E; class F and G respectively.



Syntax for Hierarchical Inheritance

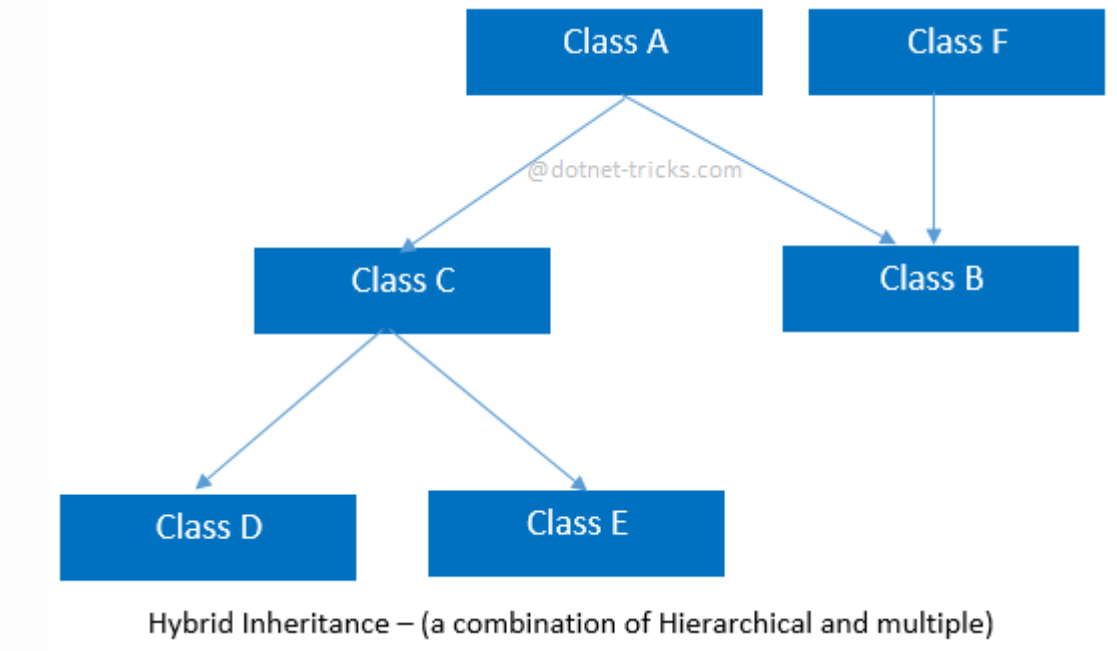
```
//Base Class

class A

{

 public void fooA()

 {

 //TO DO:

 }

}


//Derived Class

class B : A

{

 public void fooB()

 {

 //TO DO:

 }
```

```
}

//Derived Class
class C : A
{
 public void fooC()
 {
 //TO DO:
 }
}

//Derived Class
class D : C
{
 public void fooD()
 {
 //TO DO:
 }
}

//Derived Class
class E : C
{
 public void fooE()
 {
 //TO DO:
 }
}
```

```
//Derived Class
class F : B
{
 public void fooF()
 {
 //TO DO:
 }
}


//Derived Class
class G :B
{
 public void fooG()
 {
 //TO DO:
 }
}
```

6. Hybrid inheritance

This is combination of more than one inheritance. Hence, it may be a combination of Multilevel and Multiple inheritance or Hierarchical and Multilevel inheritance or Hierarchical and Multipath inheritance or Hierarchical, Multilevel and Multiple inheritance.
Since .NET Languages like C#, F# etc. does not support multiple and multipath inheritance. Hence hybrid inheritance with a combination of multiple or multipath inheritances is not supported by .NET Languages.



Hybrid Inheritance – (a combination of Hierarchical and multiple)

Syntax for Hybrid Inheritance

```
//Base Class
class A
{
 public void fooA()
 {
 //TO DO:
 }
}

//Base Class
class F
{
 public void fooF()
 {
 //TO DO:
 }
}
```

```
//Derived Class
class B : A, F
{
 public void fooB()
 {
 //TO DO:
 }
}

//Derived Class
class C : A
{
 public void fooC()
 {
 //TO DO:
 }
}

//Derived Class
class D : C
{
 public void fooD()
 {
 //TO DO:
 }
}
```

```
//Derived Class
class E : C
{
 public void fooE()
 {
 //TO DO:
 }
}
```

- **What is an abstract class?**

In C++, if a class has at least one pure virtual function, then the class becomes abstract. Unlike C++, in Java, a separate keyword *abstract* is used to make a class abstract.

```
// An example abstract class in Java
abstract class Shape {
    int color;

    // An abstract function (like a pure virtual function in C++)
    abstract void draw();
}
```

- **What is an interface and how multiple inheritance is achieved with this**

An **interface in Java** is a blueprint of a class. It has static constants and abstract methods.

The interface in Java is *a mechanism to achieve abstraction*. There can be only abstract methods in the Java interface, not method body. It is used to achieve abstraction and multiple inheritance in Java.

Example

Live Demo

```
interface AnimalEat {
   void eat();
}
interface AnimalTravel {
   void travel();
}
class Animal implements AnimalEat, AnimalTravel {
   public void eat() {
      System.out.println("Animal is eating");
   }
   public void travel() {
      System.out.println("Animal is travelling");
   }
}
public class Demo {
   public static void main(String args[]) {
      Animal a = new Animal();
      a.eat();
      a.travel();
   }
}
```

Output

```
Animal is eating
Animal is travelling
```

- **What are the access modifiers?**

The access modifiers in Java specifies the accessibility or scope of a field, method, constructor, or class. We can change the access level of fields, constructors, methods, and class by applying the access modifier on it.

Object Oriented Programming Fundamentals

There are four types of Java access modifiers:

- **Private**: The access level of a private modifier is only within the class. It cannot be accessed from outside the class.
- **Default**: The access level of a default modifier is only within the package. It cannot be accessed from outside the package. If you do not specify any access level, it will be the default.
- **Protected**: The access level of a protected modifier is within the package and outside the package through child class. If you do not make the child class, it cannot be accessed from outside the package.
- **Public**: The access level of a public modifier is everywhere. It can be accessed from within the class, outside the class, within the package and outside the package.

➢ What are the various types of constructors?

A constructor is a block of codes similar to the method. It is called when an instance of the class is created. At the time of calling constructor, memory for the object is allocated in the memory.

It is a special type of method which is used to initialize the object.

Every time an object is created using the new() keyword, at least one constructor is called.

There are two types of constructors in Java:

- Default constructor (no-arg constructor)
- Parameterized constructor

➢ What is 'this' pointer?

The **this** pointer is an implicit parameter to all member functions. Therefore, inside a member function, this may be used to refer to the invoking object.

Friend functions do not have a **this** pointer, because friends are not members of a class. Only member functions have a **this** pointer.

➢ What is static and dynamic Binding?

- In Java static binding refers to the execution of a program where type of object is determined/known at compile time i.e when compiler executes the code it know the type of object or class to which object belongs.While in case of dynamic binding the type of object is determined at runtime.

Also static binding uses type of class to bind while dynamic binding uses type of object as the resolution happens only at runtime because object only created during runtime due to which dynamic binding becomes slower than in case of static binding.

As private,final and static modifiers binds to the class level so methods and variables uses static binding and bonded by compiler while the other methods are bonded during runtime based upon runtime object.

In general we can say that overloaded methods are bonded using static binding while overridden methods are bonded using dynamic binding.

- **How many instances can be created for an abstract class and why?**

We can't create instance directly using new keyword, but you can call its constructor not to create its instance but to provide initialization to its member variables such that those variables can be accessible and used in its derived classes.

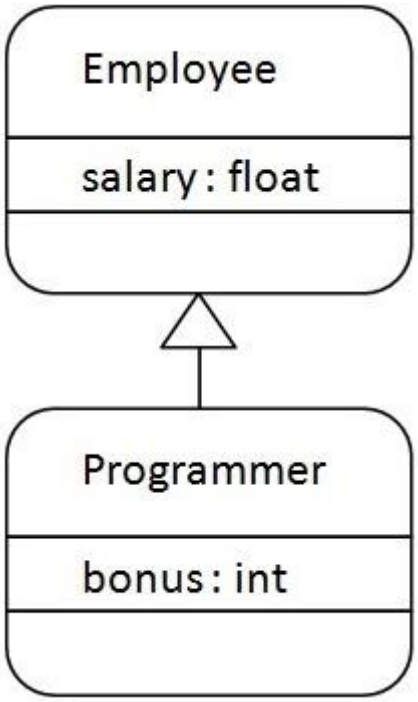
- **Which OOPS concept is used as a reuse mechanism and explain with a use case**

Inheritance is the OOPS concept that can be used as reuse mechanism

---

Java Inheritance Example

Object Oriented Programming Fundamentals



As displayed in the above figure, Programmer is the subclass and Employee is the superclass. The relationship between the two classes is **Programmer IS-A Employee**. It means that Programmer is a type of Employee.

```
1.  class Employee{
2.   float salary=40000;
3.  }
4.  class Programmer extends Employee{
5.   int bonus=10000;
6.   public static void main(String args[]){
7.     Programmer p=new Programmer();
8.     System.out.println("Programmer salary is:"+p.salary);
9.     System.out.println("Bonus of Programmer is:"+p.bonus);
10. }
11. }
```

Test it Now

```
Programmer salary is:40000.0
Bonus of programmer is:10000
```

In the above example, Programmer object can access the field of own class as well as of Employee class i.e. code reusability.

Object Oriented Programming Fundamentals

- **Please identify one practical scenario for each pillar of OOPs.**

The Four Principles of Object-Oriented-Programming (OOP):

Encapsulation

Encapsulation is accomplished when each object maintains a private state, inside a class. Other objects can not access this state directly, instead, they can only invoke a list of public functions. The object manages its own state via these functions and no other class can alter it unless explicitly allowed. In order to communicate with the object, you will need to utilize the methods provided. One way I like to think of encapsulation is by using the example of people and their dogs. If we want to apply encapsulation, we do so by encapsulating all “dog” logic into a Dog class. The “state” of the dog is in the private variables playful, hungry and energy and each of these variables has their respective fields.

There is also a private method: bark(). The dog class can call this whenever it wants, and the other classes can not tell the dog when to bark. There are also public methods such as sleep(), play() and eat() that are available to other classes. Each of these functions modifies the internal state of the Dog class and may invoke bark(), when this happens the private state and public methods are bonded.

Abstraction

Abstraction is an extension of encapsulation. It is the process of selecting data from a larger pool to show only the relevant details to the object. Suppose you want to create a dating application and you are asked to collect all the information about your users. You might receive the following data from your user: Full name, address, phone number, favorite food, favorite movie, hobbies, tax information, social security number, credit score. This amount of data is great however not all of it is required to create a dating profile. You only need to select the information that is pertinent to your dating application from that pool. Data like Full name, favorite food, favorite movie, and hobbies make sense for a dating application. The process of fetching/removing/selecting the user information from a larger pool is referred to as Abstraction. One of the advantages of Abstraction is being able to apply the same information you used for the dating application to other applications with little or no modification.

Inheritance

Inheritance is the ability of one object to acquire some/all properties of another object. For example, a child inherits the traits of his/her parents. With inheritance, reusability is a major advantage. You can reuse the fields and methods of the existing class. In Java, there are various types of inheritances: single, multiple, multilevel, hierarchical, and hybrid. For example, Apple is a fruit so assume that we have a class Fruit and a subclass of it named Apple. Our Apple acquires the properties of the Fruit class. Other classifications could be grape, pear, and mango, etc. Fruit defines a class of foods that are mature ovaries of a plant, fleshy, contains a large seed within or numerous tiny seeds. Apple the sub-class acquires these properties from Fruit and has some unique properties, which are different from other sub-classes of Fruit such as red, round, depression at the top.

Object Oriented Programming Fundamentals

Polymorphism

Polymorphism gives us a way to use a class exactly like its parent so there is no confusion with mixing types. This being said, each child sub-class keeps its own functions/methods as they are. If we had a superclass called Mammal that has a method called mammalSound(). The sub-classes of Mammals could be Dogs, whales, elephants, and horses. Each of these would have their own iteration of a mammal sound (dog-barks, whale-clicks).