OOPS ASSIGNMENT

GTBIT

HARPREET SINGH

CSE-3

00776802719

Q1. What is inheritance ? Give its various types with some examples.

Ans1. Inheritance is a mechanism of acquiring the features and behaviours of a class by another class. The class whose members are inherited is called the base class, and the class that inherits those members is called the derived class. Inheritance implements the IS-A relationship.

For example, mammal IS-A animal, dog IS-A mammal; Hence dog IS-A animal as well.

Different Types of Inheritance

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

1. Single inheritance
2. Multi-level inheritance
3. Multiple inheritance
4. Multipath inheritance
5. Hierarchical Inheritance
6. 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.

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.

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:
 }
}
```

Q2. What is operator overloading? Write code to overload == operator. Take suitable example to explain.

Ans2. Operator overloading is a technique by which operators used in a programming language are implemented in user-defined types with customized logic that is based on the types of arguments passed.

Operator overloading facilitates the specification of user-defined implementation for operations wherein one or both operands are of user-defined class or structure type. This helps user-defined types to behave much like the fundamental primitive data types. Operator overloading is helpful in cases where the operators used for certain types provide semantics related to the domain context and syntactic support as found in the programming language. It is used for syntactical convenience, readability and maintainability.

Code:-

```
#include<iostream>
using namespace std;
class Time
{
   int hr, min, sec;
   public:
   Time()
```

```
    {
        hr=0, min=0; sec=0;
    }

    Time(int h, int m, int s)
    {
        hr=h, min=m; sec=s;
    }

    friend bool operator==(Time &t1, Time &t2);
};

bool operator== (Time &t1, Time &t2)
{
    return ( t1.hr == t2.hr && t1.min == t2.min && t1.sec == t2.sec );
}

int main()
{
    Time t1(3,15,45);
    Time t2(3,15,45);
    if(t1 == t2)
    {
        cout << "\nBoth the time values are equal\n";
    }
    else
    {
        cout << "\nBoth the time values are not equal\n";
    }
```

}

Output:-