****

School of System and Technology

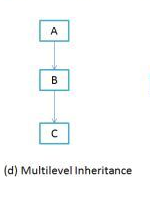
Department of Computer Science

Program: BS (CS)

Lab Manual 9

**Multilevel Inheritance:**

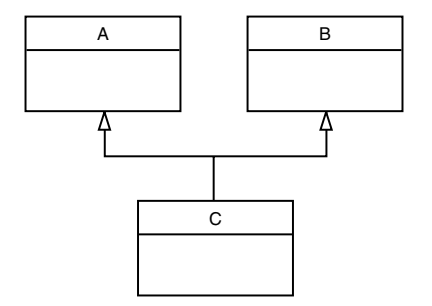
**It** refers to a mechanism in OO technology where one can inherit from a derived class, thereby making this derived class the base class for the new class. As you can see in below flow diagram C is subclass or child class of B and B is a child class of A.



class A  
{ };  
class B : public A  
{ };  
class C : public B  
{ };

**Multiple Inheritance:**

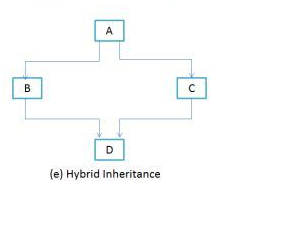
A class can be derived from more than one base class. This is called multiple inheritance.  
Figure shows how this looks when a class C is derived from base classes A and B.



The syntax for multiple inheritance is similar to that for single inheritance. In the situation  
shown in Figure above the relationship is expressed like this:  
class A // base class A  
{  
};  
class B // base class B  
{  
};  
class C : public A, public B // C is derived from A and B  
{  
};  
The base classes from which C is derived are listed following the colon in C’s specification;  
they are separated by commas.

### **Hybrid Inheritance**

In simple terms you can say that Hybrid inheritance is a combination of **more than one type of** inheritance. A typical flow diagram would look like below. A hybrid inheritance can be achieved in the java in a same way as multiple inheritance can be!! Using interfaces. yes you heard it right.



**UML Diagram of Employee Inheritance**

**Sample Code**

|  |
| --- |
| #include<iostream>  #include<string>  using namespace std;  class Person  {  private:  int a;  protected:  string name;  int age;  public:  Person(string na = "None", int ag = 15) :name(na), age(ag)  {  cout << "Person Constructor called" << endl;  }  void setPerson(string na, int ag)  {  name = na;  age = ag;  }  void disPerson()  {  cout << "Name: " << name << "\n Age: " << age<<endl;  }  };  class Faculty: virtual public Person  {  protected:  string des;  int salary;  public:  Faculty(string d = "Lecture", int sal = 10000) : des(d), salary(sal)  {  cout << "Faculty Constructor called" << endl;  }  void setFaculty(string d, int s)  {  des = d;  salary = s;  }  void disFaculty()  {  //disPerson();  cout << "Desgination: " << des << "\n Salary: " << salary<<endl;  }  };  class Student :virtual public Person  {  protected:  string degree;  float cgpa;  public:  Student(string deg = "CS", float cgp = 0.0) :degree(deg), cgpa(cgp)  {  cout << "Student constructor called" << endl;  }  void setStudent(string deg, float cgp)  {  degree = deg;  cgpa = cgp;  }  void disStudent()  {  cout << "Degree: " << degree << "\nCGPA: " << cgpa;  }  };  class Researcher :public Student, public Faculty  {  protected:  int publication;  public:  Researcher(int pub = 0) : publication(pub)  {  cout << "Researcher Constructor called" << endl;  }  void setReseacher( int s)  {  publication = s;  }  void disResearher()  {  //disPerson();  disFaculty();  disStudent();  cout << "No of Publication: " << publication <<endl;  }  };  int main()  {  Researcher r1;  r1.setReseacher(2);  r1.setPerson("Aizaz", 25);  r1.setFaculty("Lecture", 2500);  r1.setStudent("PhD", 3.5);  r1.disResearher();    cout << sizeof(r1);  system("pause");  return 0;  } |

**Lab Task**

|  |
| --- |
| **Task 1:**  **Implement the given UML, you have to identify the appropriate attributes of each class and you have to write the getter setter methods, default parameterized constructor, set method and display method of each class**    **Task 2:**  **Implement the given UML, you have to identify the appropriate attributes of each class and you have to write the getter setter methods, default parameterized constructor, set method and display method of each class**  **Person** |