UNIT-3 (THEORY SOLUTION)

1. **When base class and derived class have the same function name what happens when derived class object calls the function?[PU 2017 fall]**

When the base class and derived class have the same function name, the derived class function overrides the function that is inherited from base class, which is known as function overriding. Now, when derived class object calls that overridden function, the derived class member function is accessed.

For example:

```
#include<iostream>
using namespace std;
class Base
{
public:
void display()
{
cout<<"This is base class"<<endl;
}
};
class Derived:public Base
{
public:
void display()
{
cout<<"This is derived class"<<endl;
}
};
int main()
{
Derived d;
d.display();
return 0;
}
```

Output:

```
This is derived class
```

In this example, the class derived inherits the public member function display() .When we redefine this function in derived class then the inherited members from base are overridden.so when derived class object calls the function display() it actually refers the function in derived class but not the inherited member function in base class.

2. **How inheritance support reusability features of OOP? Explain with example. [PU:2010 spring]**

Using the concept of inheritance the data member and member function of classes can be reused. When once a class has been written and tested, it can be adapted by another programmer to suit their requirements. This is basically done by creating new classes, reusing the properties of the existing ones. This mechanism of deriving a new class from an old one is called inheritance.

A derived class includes all features of the base class and then it can add additional features specific to derived class.

Example:

```
#include<iostream>
using namespace std;
class Sum
{
protected:
int a,b;
public:
void getdata()
{
cout<<"Enter two numbers"<<endl;
cin>>a>>b;
}
void display()
{
cout<<"a="<<a<<endl;
cout<<"b="<<b<<endl;
}
};
class Result:public Sum
{
private:
int total;
public:
void disp_result()
{
total=a+b;
cout<<"sum="<<total<<endl;
}
};
```

```
int main()
{
Result r;
r.getdata();
r.display();
r.disp_result();
return 0;
}
```

Here, In above example **Sum** is base class and **Result** is derived class. The data member **named a and b** and member function display () of base class are inherited and they are used by using the object of derived class named **Result.** Hence, we can see that inheritance provides reusability.

3.**How are arguments are sent to base constructors in multiple inheritance? Who is responsible for it.[PU:2013 spring]**

In Multiple Inheritance arguments are sent to base class in the order in which they appear in the declaration of the derived class**.**
For example:
Class gamma: public beta, public alpha
{

}
Order of execution
beta(); base constructor (first)
alpha(); base constructor(second)
gamma();derived (last)
The constructor of derived class is responsible to supply values to the base class constructor.

Program:

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

class alpha
{
int x;
public:
alpha(int a)
{
x=a;
cout<<"Alpha is initialized"<<endl;
}
```

```
void showa()
{
cout<<"x="<<x<<endl;
}
};
class beta
{
int y;
public:
beta(int b)
{
y=b;
cout<<"Beta is initialized"<<endl;
}
void showb()
{
cout<<"y="<<y<<endl;
}
};
class gamma:public beta,public alpha
{
int z;
public:
gamma(int a,int b,int c):alpha(a),beta(b)
{
z=c;
cout<<"Gamma is initialized"<<endl;
}
void showg()
{
cout<<"z="<<z<<endl;
}
};

int main()
{
gamma g(5,10,15);
g.showa();
g.showb();
g.showg();
return 0;
}
```

Output:

```
Beta is initialized
Alpha is initialized
Gamma is initialized
x=5
y=10
z=15
```

Here, **beta** is initialized first although it appears second in the derived constructor *as it has been* declared first in the derived class header line
class gamma: public beta, public alpha
{ }
If we change the order *to*
class gamma: public alpha, public beta
{ }
then alpha will be initialized first

4.Class 'Y' has been derived from class 'X' .The class 'Y' does not contain any data members of its own. Does the class 'Y' require constructors? If yes why.[PU:2013 spring]

When Class 'Y' has been derived from class 'X' and class 'Y' does not contain any data members of its own then,

- It is mandatory have constructor in derived class 'Y', whether it has data members to be initialized or not, if there is constructor with arguments(parameterized constructor) in base class 'X'.
- It not necessary to have constructor in derived class 'Y' if base class 'X' does not contain a constructor with arguments (parameterized constructor).

Derived class constructor is used to provide the values to the base class constructor.

Example

```
#include<iostream>
using namespace std;
class X
{
private:
int a;
public:
X (int m)
{
a=m;
}
```

```
void display()
{
cout<<"a="<<a<<endl;
}
};
class Y:public X
{
public:
Y(int b):X(b)
{ }
};
int main()
{
Y obj(5);
obj.display();
return 0;
}
```

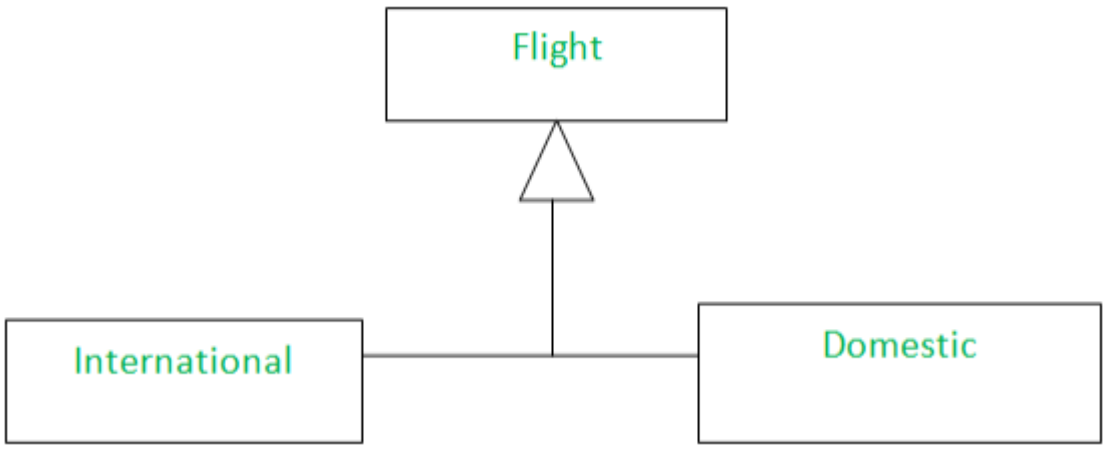
Here, in above example, class 'Y' does not have any data members but there is a parameterized constructor in class 'X' so, there is necessary to have constructor in class 'Y'.

5. **Write a short notes on:**

Generalization[PU:2013 spring]

Generalization is the process of taking out common properties and functionalities from two or more classes and combining them together into another class which acts as the parent class of those classes or what we may say the generalized class of those specialized classes. All the subclasses are a type of superclass. So we can say that subclass "is-A" superclass. Therefore Generalization is termed as "is-A relationship".

Here is an example of generalization:



In this figure, we see that there are two types of flights so we made a flight class which will contain common properties and then we has an international and domestic class which are an extension of flight class and will have properties of flight as well as their own. Here flight is the parent/superclass and the other two are child/subclass. International Flight “is-a” flight as well as the domestic flight.

6. **Explain how composition provide reusability.**

In composition one class contains object of another class as its member data, which means members of one class is available in another class. Composition exhibits “has-a” relationship. For example, Company **has an** employee.
So, properties of one class can be used by another class independently.

Let us consider an example

```
#include<iostream>
using namespace std;
class Employee
{
int eid;
float salary;
public:
void getdata()
{
cout<<"Enter id and salary of employee"<<endl;
cin>>eid>>salary;
}
void display()
{
cout<<"Emp ID:"<<eid<<endl;
cout<<"Salary:"<<salary<<endl;
}
};
class Company
{
char cname[20];
char department[20];
Employee e;
public:
void getdata()
{
e.getdata();
cout<<"Enter company name and Deparment:"<<endl;
cin>>cname>>department;
}
```

```
void display()
{
e.display();
cout<<"Company name:"<<cname<<endl;
cout<<"Department:"<<department<<endl;
}
};

int main()
{
Company c;
c.getdata();
c.display();
return 0;
}
```

In above example class company contains the object of another class employee. As we know "company **has a** employee" sounds logical .so there exits has a relationship between company and employee. Class company contains the object of class employee and members of class employee are used in class company which provides reusability.