

UNIT-II: IMPERATIVE PARADIGM: DATA ABSTRACTION IN OBJECT ORIENTATION



Faculty In-charge

Aaysha Shaik

Assistant Professor (IT
Dept.)

email: aayshashaikh@sfit.ac.in

24-Aug-22



Ms.Aaysha Shaikh

OF UNIT-2

The material in this presentation belongs to St. Francis Institute of Technology and is solely for educational purposes.

OUTLINE

2.1 Grouping of data and operations

2.2 Encapsulation

✓ 2.3 Overloading and polymorphism

✓ 2.4 Inheritance

✓ 2.5 Initialization and finalization

2.6 Dynamic Binding



2

Ms.Aaysha Shaikh

The material in this presentation belongs to St. Francis Institute of Technology and is solely for educational purposes. Distribution and modifications of the content is prohibited.

2.3:-Overloading and Polymorphism

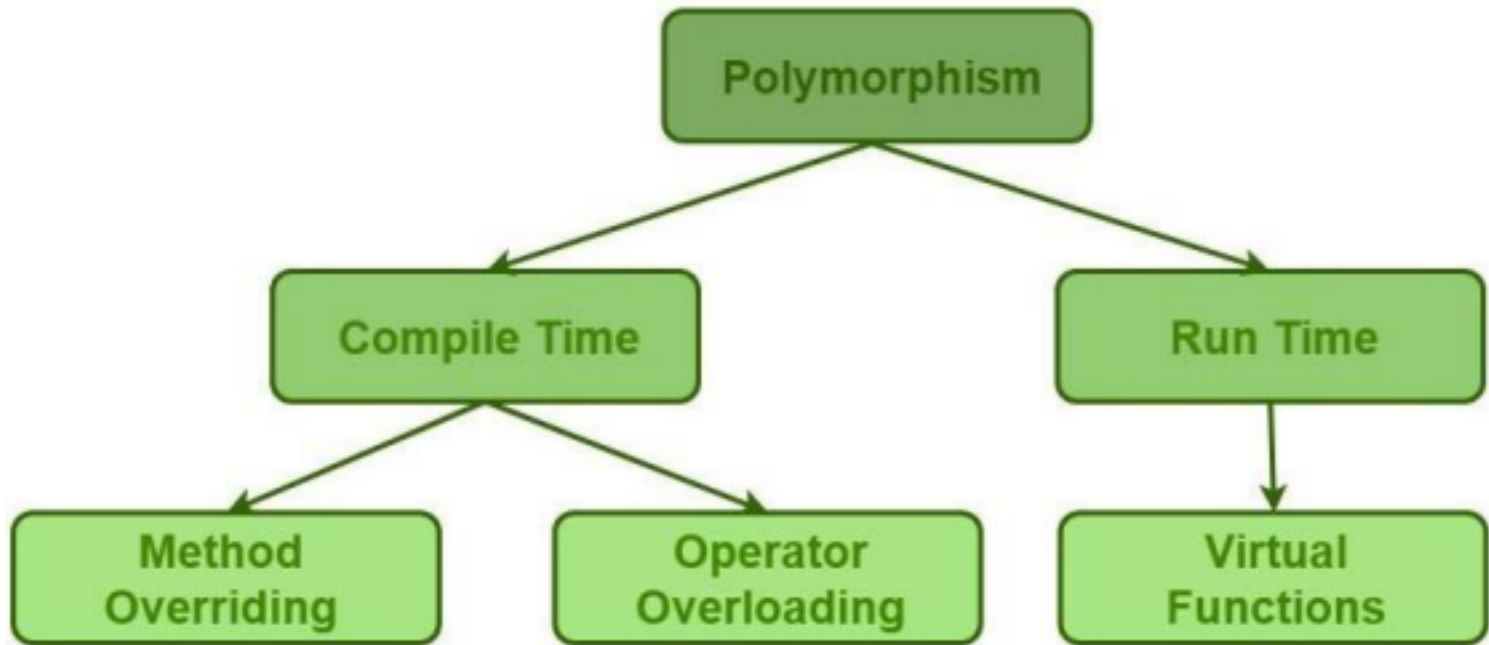


24-Aug-22 Ms. Aaysha Shaikh

The material in this presentation belongs to St. Francis Institute of Technology and is solely for educational purposes. Distribution and modifications of the content is prohibited.

POLYMORPHISM

- Important concept of object oriented programming
- Polymorphism , a Greek term means the ability to take more than one form
- The behaviour depends upon the types of data used in operation Eg. *A person possess different behaviour- a father, husband, employee*



24-Aug-22

Ms. Aaysha Shaikh

The material in this presentation belongs to St. Francis Institute of Technology and is solely for educational purposes. Distribution and modifications of the content is prohibited.

Compile Time Polymorphism	Run Time Polymorphism
The function to be invoked is known at the compile time.	The function to be invoked is known at run time.
It is also known as <i>overloading</i> , <i>early binding</i> and <i>static binding</i> .	It is also known as <i>overriding</i> , <i>Dynamic binding</i> and <i>late binding</i> .

More than one method is having the same name but with the different number of parameters or the type of the parameters.	More than one method is having the same name, number of parameters and the type of the parameters.
It is achieved by function overloading and operator overloading.	It is achieved by virtual functions and pointers.
It provides fast execution as it is known at the compile time.	It provides slow execution as it is known at the run time.
It is less flexible as mainly all the things execute at the compile time.	It is more flexible as all the things execute at the run time.



24-Aug-22

Ms. Aaysha Shaikh

The material in this presentation belongs to St. Francis Institute of Technology and is solely for educational purposes. Distribution and modifications of the content is prohibited.

POLYMORPHISM-Function/Method 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**.
- In C++, (Rules for function overloading)

1) Function declarations that differ only in the return type is not

```
9  #include<iostream>
10 int foo() {
11     return 10;
12 }
13
14 char foo() {
15     return 'a';
16 }
17
18 int main()
19 {
20     char x = foo();
21     getchar();
22     return 0;
23 }
24
```

input

Compilation failed due to following error(s).

```
main.cpp: In function 'char foo()':
main.cpp:14:6: error: ambiguating new declaration of 'char foo()'
char foo() {
^~~~~~
```

allowed.



24-Aug-22

Ms.

Aaysha Shaikh

The material in this presentation belongs to St. Francis Institute of Technology and is solely for educational purposes. Distribution and modifications of the content is prohibited.

2) Member function declarations with the same name and the name parameter-type-list cannot be overloaded if any of them is a static member function declaration. For example, following program fails in compilation.


```

8  ****
9  #include<iostream>
10 class Test {
11     static void fun(int i) {}
12     void fun(int i) {}
13 };
14
15 int main()
16 {
17     Test t;
18     getchar();
19     return 0;
20 }
21
22
23

```

input

Compilation failed due to following error(s).

```

main.cpp:12:6: error: 'void Test::fun(int)' cannot be overloaded
    void fun(int i) {}
        ^~~~
main.cpp:11:13: error: with 'static void Test::fun(int)'

```



24-Aug-22

Ms. Aaysha Shaikh

The material in this presentation belongs to St. Francis Institute of Technology and is solely for educational purposes. Distribution and modifications of the content is prohibited.

3) Parameter declarations that differ only in a pointer * versus an array

[] are equivalent.

That is, the array declaration is adjusted to become a pointer declaration. Only the second and subsequent array dimensions are significant in parameter types. For example, following two function declarations are equivalent.

```
int fun(int *ptr);  
int fun(int ptr[]); // redeclaration of fun(int *ptr)
```



4) Two parameter declarations that differ only in their default arguments are equivalent. For example, following program fails in compilation with error “*redefinition of `int f(int, int)`*”

t f(int, int)'



24-Aug-22

Ms. Aaysha Shaikh

```

8  ****
9  // C++ program for function overloading
10 #include <bits/stdc++.h>
11 using namespace std;
12 class Geeks
13 {
14     public:
15     // function with 1 int parameter
16     void func(int x)
17     {
18         cout << "value of x is " << x << endl;
19     }
20     // function with same name but 1 double parameter
21     void func(double x)
22     {
23         cout << "value of x is " << x << endl;
24     }
25     // function with same name and 2 int parameters
26     void func(int x, int y)
27     {
28         cout << "value of x and y is " << x << ", " << y << endl;
29     }
30 };
31

```

24-Aug-22

Ms. Aaysha Shaikh

The material in this presentation belongs to St. Francis Institute of Technology and is solely for educational purposes. Distribution and modifications of the content is prohibited.



In the above example, a single function named *func* acts differently in three different situations which is the property of polymorphism.

24-Aug-22

Ms. Aaysha Shaikh



24-Aug-22



24-Aug-22 Ms. Aaysha Shaikh

The material in this presentation belongs to St. Francis Institute of Technology and is solely for educational purposes. Distribution and modifications of the content is prohibited.

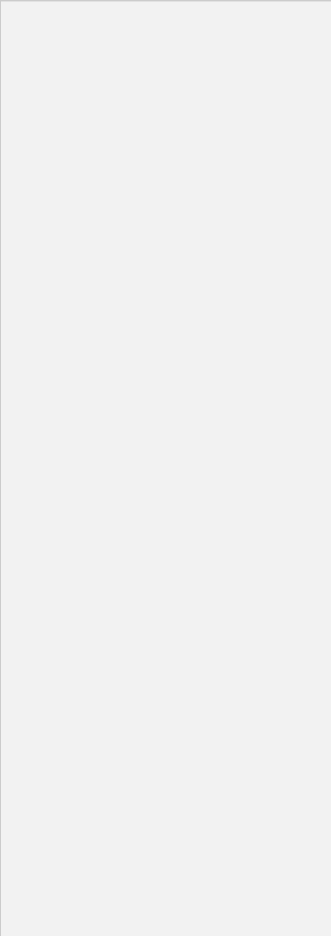
POLYMORPHISM-Operator Overloading

- In C++, we can make operators to work for user defined classes. ▪ This means C++ has the ability to provide the operators with a special meaning for a data type.

- For example, we can overload an operator ‘+’ in a class like String so that we can concatenate two strings by just using +.

24-Aug-22





Ms. Aaysha
Shaikh

The material in this presentation belongs to St. Francis Institute of Technology and is solely for educational purposes. Distribution and modifications of the content is prohibited.



✓ Operator functions are same as normal functions.

Operators that cannot be overloaded-

➤ . (dot)

➤ :: (Scope resolution operator) ➤ ?:

(Conditional operator) ➤ sizeof

✓ The only differences are, name of an operator function is always operator

keyword followed by symbol of operator

✓ Operator functions are called when the

corresponding operator is used. 24-Aug-22 Ms.Aaysha Shaikh

Early binding and Late binding in C++

Binding refers to the process of converting identifiers (such as variable and performance names) into addresses. Binding is done for each variable and functions. **For functions, it means that matching the call with the right function definition by the compiler.** It takes place either at compile time or at runtime.



24-Aug-22 Ms. Aaysha Shaikh

Early Binding (compile-time time polymorphism) As the name indicates, compiler (or linker) directly associate an address to the function call. It replaces the call with a machine language instruction that tells the mainframe to leap to the address of the function.

By default early binding happens in C++. Late binding (discussed below) is achieved with the help of [virtual keyword](#))

```

// CPP Program to illustrate early binding. // Any normal function call (without virtual) // is binded early. Here we have taken base // and derived class example so that readers // can easily compare and see difference in // outputs.
#include<iostream>
using namespace std;

class Base
{
public:
    void show() { cout<<" In Base \n"; }
};

class Derived: public Base
{
public:
    void show() { cout<<"In Derived \n"; }
};

int main(void)
{
    Base *bp ;
    Derived obj1; bp=&obj1;

    // The function call decided at // compile time (compiler sees type // of pointer and calls base class // function.
    bp->show();

    return 0;
}

```

Output:
In Base

24-Aug-22 Ms. Aaysha Shaikh

Late Binding : (Run time polymorphism) In this, the compiler adds code that identifies the kind of object at runtime then matches the call with the right function definition. This can be achieved by declaring a [virtual function](#).

```

// CPP Program to illustrate late binding
#include<iostream>
using namespace std;

class Base
{
public:
virtual void show() { cout<<" In
Base \n"; } };

class Derived: public Base
{
public:
void show() { cout<<"In Derived

```

```

\n"; } };

```

```

int main(void)
{
Base *bp = new Derived;
bp->show(); // RUN-TIME
POLYMORPHISM return 0;
}

```

Output:

In Derived

Virtual functions- [Used for Runtime Polymorphism]

- A virtual function is a member function which is declared within a base class and is re-defined(Overridden) by a derived class.
- Virtual functions in C++ allow a derived class to supply the function

body . ▪ Suppose base class shape has a virtual function draw.

- If derived class Ellipse also has a function draw, then the body in the derived class is used, overriding the body in the base class
- They are mainly used to achieve [Runtime polymorphism](#)
- Functions are declared with a **virtual** keyword in base class.
- The resolving of function call is done at Run-time.
- Virtual functions cannot be static and also cannot be a friend function of another Class
- Virtual functions should be accessed using pointer or reference of base class type to achieve run time polymorphism.



Ms. Aaysha Shaikh

The material in this presentation belongs to St. Francis Institute of Technology and is solely for educational purposes. Distribution and modifications of the content is prohibited.



24-Aug-22 Ms. Aaysha Shaikh

The material in this presentation belongs to St. Francis Institute of Technology and is solely for educational purposes. Distribution and modifications of the content is prohibited.

✓ Initially, create a pointer of

type base class

✓ Initialize it with the address of the derived class object.

When we create an object of the derived class, the compiler creates a pointer as a data member of the class containing the address of VTABLE of the derived class.



St. Francis Institute of Technology If this is implemented



```

#include <iostream>
using namespace std;
class Shape {
protected: int width, height;
public: Shape( int a = 0, int b = 0)
        { width = a; height = b; }
virtual int area()
        { cout << "Parent class area : " <<
          width * height << endl; return
            width * height; }
};

class Rectangle: public Shape {

        int area ()
        {
        cout << "Triangle class area : " <<
          (width * height)/2 << endl;
          return (width * height / 2); }
height); }

};

class Triangle: public Shape {

        public: Triangle( int a = 0, int b = 0):Shape(a, b) { }

```

```

}; // Main function for the
program int main() {
        Shape *shape;
        Rectangle rec(10,7);
        Triangle tri(10,5); // store the
        address of Rectangle
        shape = &rec; // call rectangle area.
public: Rectangle( int a = 0, int b =
0):Shape(a, b) { }
int area ()
{ cout << "Rectangle class area : " <<
width * height << endl; return (width *
shape->area()); // store the address of
Triangle
shape = &tri; // call triangle area.
shape->area();
return 0;
}

```

Rectangle class area :70

24-Aug-22

Ms. Aaysha
Shaikh

```
#include <iostream>
using namespace std;
class Shape {
protected: int width, height;
public: Shape( int a = 0, int b = 0){
    width = a; height = b; }
int area() {
    cout << "Parent class area :" << width
    * height << endl; return width *
    height; }
};
class Rectangle: public Shape {
public: Rectangle( int a = 0, int b =
0):Shape(a, b) { }
int area () {
    cout << "Rectangle class area :" <<
    width * height << endl; return (width
    * height); }
};
```

24-Aug-22

```
class Triangle: public Shape {
public: Triangle( int a = 0, int b = 0):Shape(a, b)
{ }
int area () {
    cout << "Triangle class area :" <<
    (width * height)/2 << endl;
    return (width * height / 2);
}
}; // Main function for the program
int main() {
    Shape *shape; Rectangle rec(10,7); Triangle
    tri(10,5); // store the address of Rectangle
    shape = &rec; // call rectangle area.
    shape->area(); // store the address of Triangle
```

```

shape = &tri; // call triangle area.
shape->area();
return 0; }

```

Parent class area :70
 Parent class area :50

Ms. Aaysha
Shaikh

how to call super class constructor in child class in C++

If you want to call a superclass constructor with an argument, you must use the subclass's constructor initialization list. Unlike Java, C++ supports multiple inheritance (for better or worse), so the base class must be referred to by name, rather than "super()".

```

class SuperClass
{
    public: SuperClass(int f)
    {
        // do something with f }
};

```

```

class SubClass : public SuperClass
{
    public: SubClass(int fo, int b) :
        SuperClass(fo) // Call the superclass
        constructor in the subclass'
        initialization list.
        { // do something with bar }
};

```

how to call super class constructor in child class in java

To explicitly call the superclass constructor from the subclass constructor, we use `super()`. It's a special form of the `super` keyword. `super()` can be used only inside the subclass constructor and must be the first statement.

```

class Animal {
    // default or no-arg constructor of class Animal
    Animal() {
        System.out.println("I am an animal"); }
}

class Dog extends Animal
{
    // default or no-arg constructor of class Dog
    Dog()
    {
        // calling default constructor of the superclass
        super();
    }
}

```

```
        System.out.println("I am a dog"); }  
    }  
class Main  
{ public static void main(String[] args)  
    { Dog dog1 = new Dog();
```

```
    }  
} OUTPUT  
I am an animal
```

24-Aug-22

Ms. Aaysha Shaikh I am a dog

The material in this presentation belongs to St. Francis Institute of Technology and is solely for educational purposes. Distribution and modifications of the content is prohibited.

POLYMORPHISM-Runtime Polymorphism

This type of polymorphism is achieved by Function Overriding.

Creating virtual
functions

Inheriting the virtual
function

Runtime

24-Aug-22 Ms. Aaysha Shaikh

The material in this presentation belongs to St. Francis Institute of Technology and is solely for educational purposes. Distribution and modifications of the content is prohibited.

2.4:-Inheritance

INHERITANCE

- Inheritance is a relationship between two or more classes where derived class inherits the properties of existing base classes
- Base class: It is the class whose properties are inherited by another class. It is also called as Super class or Parent class
- Derived class: It is the class that inherit properties from the base class(es). It is also called sub class or child class

It is useful for code reusability: reuse attributes and methods of

an existing class when you
new class.



create a



24-Aug-22

Ms. Aaysha Shaikh

The material in this presentation belongs to St. Francis Institute of Technology and is solely for educational purposes. Distribution and modifications of the content is prohibited.

NEED FOR INHERITANCE



- There is duplication of same code 3 times

is increases the chances of error and data redundancy.

avoid this type of situation, inheritance is used

24-Aug-22 Ms. Aaysha Shaikh

The material in this presentation belongs to St. Francis Institute of Technology and is solely for educational purposes. Distribution and modifications of the content is prohibited.

Syntax for creating sub-class

```
class Sub-class_name: visibility-mode base_class_name{
```

```
//bodyof subclass};
```

 ■ Subclass_name is the name of the sub class

- Visibility-mode is the mode in which you want to inherit this subclass **[public, private or protected]**
- Base_class_name is the name of the base class from which you want to inherit the

subclass

- The visibility mode is optional
- The default visibility mode is **private**.

24-Aug-22 Ms. Aaysha Shaikh

The material in this presentation belongs to St. Francis Institute of Technology and is solely for educational purposes. Distribution and modifications of the content is prohibited.

INHERITANCE-Visibility Modes

(i) When a base class is privately inherited by the derived class

- Public members of the base class become ‘private members’ of the derived class
- Public members of the base class can only be accessed by the member functions of the derived class.

(ii) When the base class is publicly inherited

- Public members of the base class become ‘public members’ of the derived class
- Hence they are accessible to the objects of the derived class

In both cases the private members are not inherited and therefore, the private members of the base class will never become the members of its derived class.

C++ provides third visibility modifier, **protected**, which serves a limited purpose. A member declared as protected is accessible by the member function within its class and any class immediately derived from it. It cannot be accessed by the functions outside these two classes

and is solely for educational purposes. Distribution and modifications of the content is prohibited.

BaseClassB

Private: int a

Public:

int b

get_ab(),get_a(),

show_a()

Public Inheritance



DerivedClassD

```
Public:  
    int b  
    get_ab(),get_a(), show_a()
```

Members of D

24-Aug-22 Ms. Aaysha Shaikh

The material in this presentation belongs to St. Francis Institute of Technology and is solely for educational purposes. Distribution and modifications of the content is prohibited.



Derivedclassaccessing
baseclassdataand
methods

Conclusions-

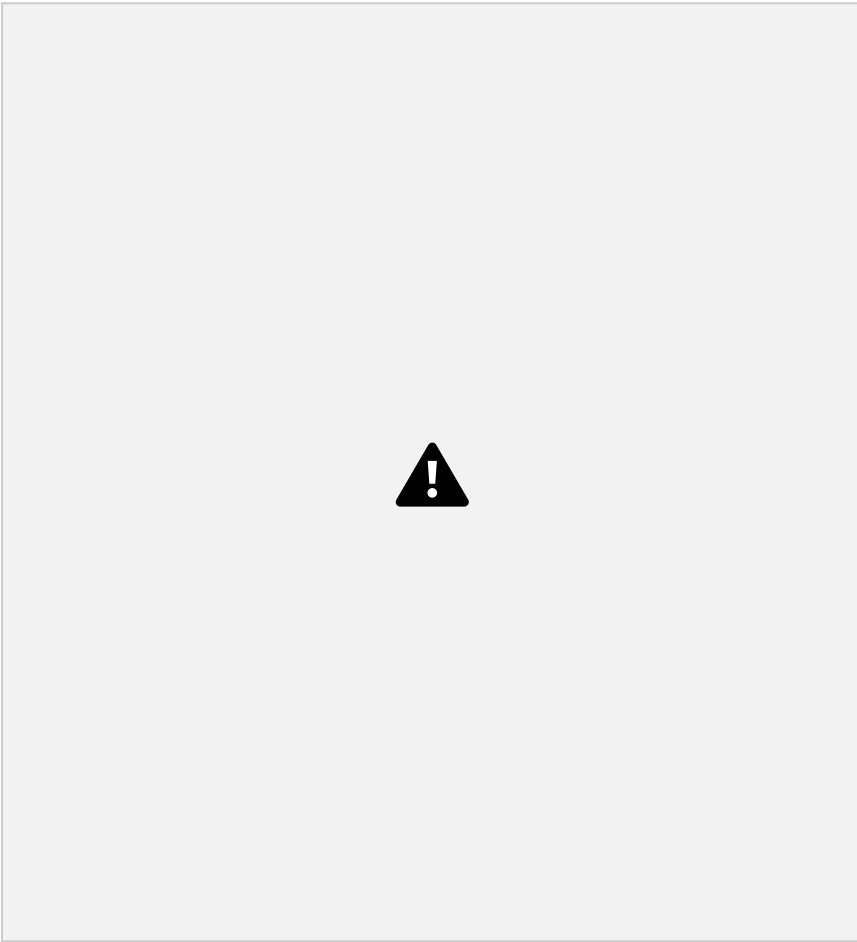
- The class D is a public derivation of the base class B.
Therefore,
- D inherits all the public members of the B and inherits their visibility.
- A public member of the base class B is also a public member of the derived class D.
- The private members of B cannot be inherited by D.
- Note that although the data member 'a' is private in B and cannot be inherited, objects of D are able to access it through an inherited member function of B.



If we derive a sub class from a Private base class.

Then both public member and protected members of the base class will become Private in derived class.

In private derivation, the public members of the base class become private members of the derived class. Therefore the objects of D cannot have a direct access to the public member function of B





24-Aug-22 Ms. Aaysha Shaikh

?? How can we access these functions?

functions can be used inside mul() and display()

```
void mul()
{
    get_ab();
    c=b*get_a();
}
void display()
{
    show_a();
    cout<<"
    b<<b<<endl;
    cout<<"c="<<c<<endl;
}
```

```
void mul( )  
{  
    get_ab( );  
    c=b*get_a();  
}  
Void display()  
{  
    show_a();  
    cout<<" b="<<b<<endl;  
    cout<<" c="<<c<<endl;  
}
```

Public Inheritance – When deriving a class from a **public** base class,

- **Public** members of the base class become **public** members of the derived class. ▪

Protected members of the base class become **protected** members of the derived class.

- A base class's **private** members are never accessible directly from a derived class, but can be accessed through calls to the **public** and **protected** members of the base class.

Protected Inheritance – When deriving from a **protected** base

- **Public** and **protected** members of the base class become **protected** members of the derived class.

Private Inheritance – When deriving from a **private** base class, **public** and **protected** members of the base class become **private** members of the derived class.

Types of Heritage



Base

Derived

TYPES OF INHERITANCE





24-Aug-22 Ms. Aaysha Shaikh

The material in this presentation belongs to St. Francis Institute of Technology and is solely for educational purposes. Distribution and modifications of the content is prohibited.

Example: Assume that the test results of a batch of students are stored in three different classes. Class student stores the roll number, class test stores the marks of two subjects and class result contains the total marks obtained in the test.



24-Aug-22 Ms. Aaysha Shaikh

The material in this presentation belongs to St. Francis Institute of Technology and is solely for educational purposes. Distribution and modifications of the content is prohibited.







A class can inherit the attribute of two or more classes. This is known as multiple inheritance. It allows us to inherit the features of several existing classes as a starting point for defining new classes.

It is like a child inheriting the physical features of one parent and intelligence of another.

The syntax for multiple base classes is as follows

Class D: visibilityA, visibility B, visibility C

{..... }

Where visibility maybe either private or public





24-Aug-22 Ms. Aaysha Shaikh





24-Aug-22 Ms. Aaysha Shaikh

The material in this presentation belongs to St. Francis Institute of Technology and is solely for educational purposes. Distribution and modifications of the content is prohibited.





24-Aug-22 Ms. Aaysha Shaikh

The material in this presentation belongs to St. Francis Institute of Technology and is solely for educational purposes. Distribution and modifications of the content is prohibited.





24-Aug-22 Ms. Aaysha Shaikh

The material in this presentation belongs to St. Francis Institute of Technology and is solely for educational purposes. Distribution and modifications of the content is prohibited.





24-Aug-22 Ms. Aaysha Shaikh

The material in this presentation belongs to St. Francis Institute of Technology and is solely for educational purposes. Distribution and modifications of the content is prohibited.



24-Aug-22 Ms. Aaysha Shaikh

2.5:-Initialization and Finalization (through self study)

24-Aug-22 Ms. Aaysha Shaikh

2.6:-Dynamic Method Binding (Through Q&A)