18CSC202J

Object Oriented Design and Programming

Prepared by

Department of Data Science and Business Systems & Networking and Communications

Session 1

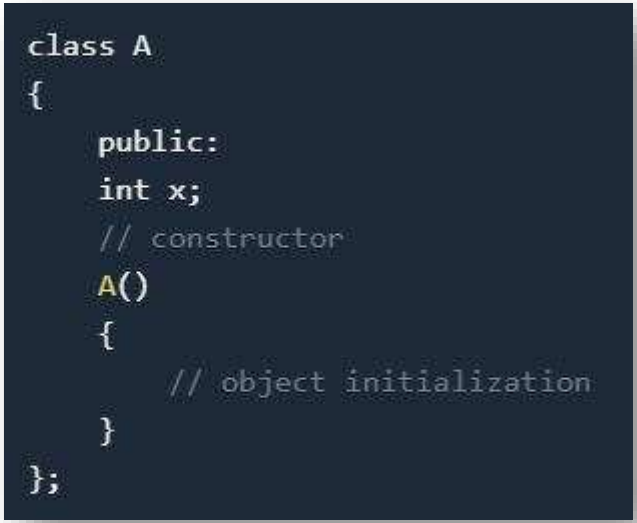
Topic : Types of Constructor

CONSTRUCTORS

- It is very common for some part of an object to require initialization before it can be used.
- Suppose you are working on 100's of objects and the default value of a particular data member is needed to be zero.
- Initializing all objects manually will be very tedious job.
- Instead, you can define a constructor function which initializes that data member to zero. Then all you have to do is declare object and constructor will initialize object automatically

CONSTRUCTORS

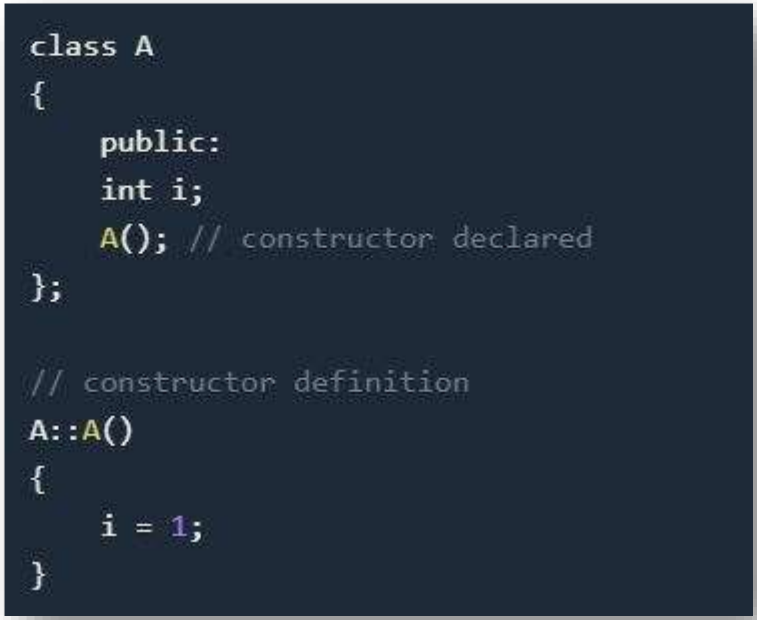
- While defining a constructor you must remember that the name of constructor will be same as the name of the class, and constructors will never have a return type.



```
class A
{
    public:
    int x;
    // constructor
    A()
    {
        // object initialization
    }
};
```

CONSTRUCTORS

- Constructors can be defined either inside the class definition or outside class definition using class name and scope resolution :: operator.



```
class A
{
    public:
    int i;
    A(); // constructor declared
};

// constructor definition
A::A()
{
    i = 1;
}
```

CONSTRUCTOR CHARACTERS

- They must be declared in the public scope.
- They are invoked automatically when the objects are created.
- They do not have return types, not even void and they cannot return values.
- They cannot be inherited, though a derived class can call the base class constructor.
- Like other C++ functions, Constructors can have default arguments.

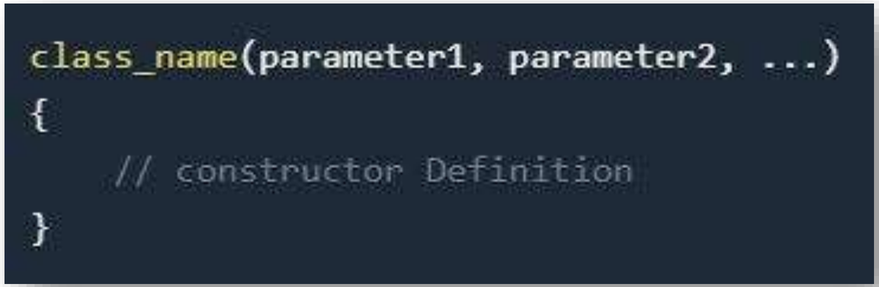
- Constructors cannot be virtual.
- We can not refer to their addresses.
- An object with a constructor (or destructor) can not be used as a member of a union.
- They make ‘implicit calls’ to the operators new and delete when memory allocation is required.

CONSTRUCTOR TYPES

- Constructors are of three types:
  - Default Constructor
  - Parameterized Constructor
  - Copy Constructor

DEFAULT CONSTRUCTOR

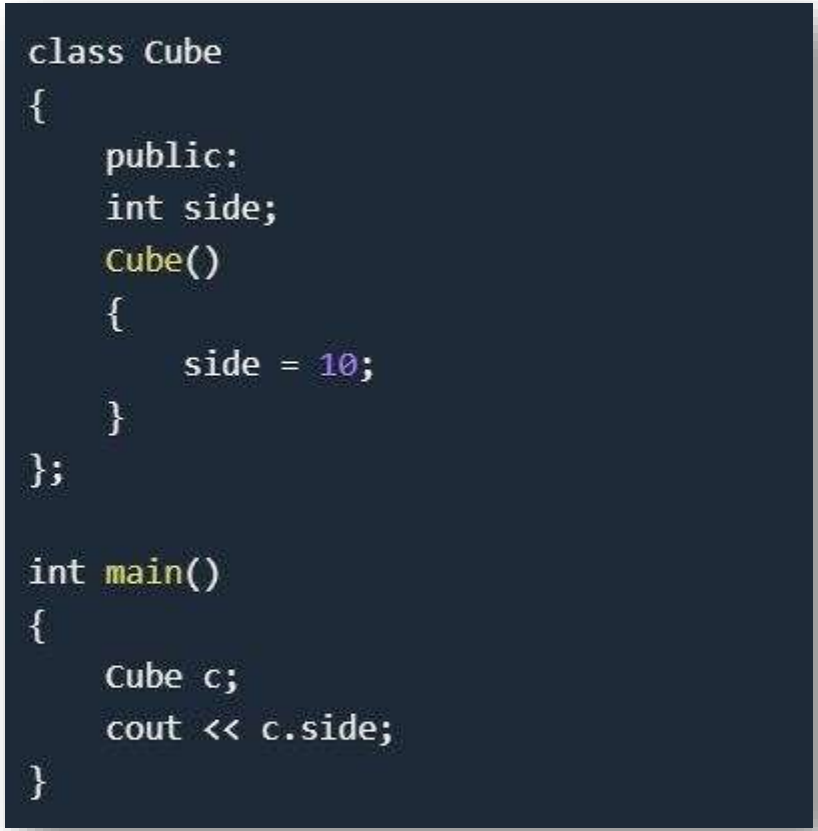
- Default constructor is the constructor which doesn't take any argument. It has no parameter.
  - Syntax :



```
class_name(parameter1, parameter2, ...)
{
    // constructor Definition
}
```

DEFAULT CONSTRUCTOR

– Example



```
class Cube
{
    public:
    int side;
    Cube()
    {
        side = 10;
    }
};

int main()
{
    Cube c;
    cout << c.side;
}
```

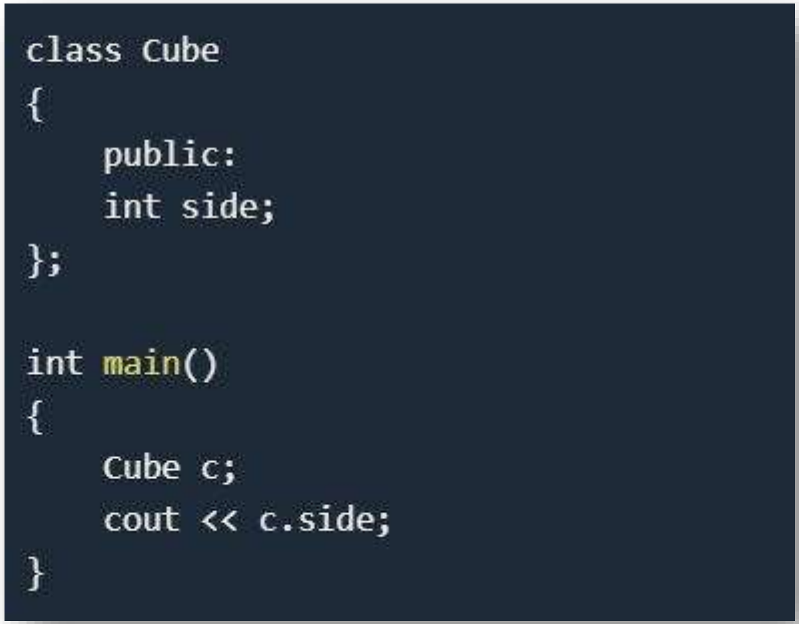
– Output : 10

DEFAULT CONSTRUCTOR

- As soon as the object is created the constructor is called which initializes its data members.

- A default constructor is so important for initialization of object members, that even if we do not define a constructor explicitly, the compiler will provide a default constructor implicitly.

DEFAULT CONSTRUCTOR


```
class Cube
{
    public:
    int side;
};

int main()
{
    Cube c;
    cout << c.side;
}
```

Output: 0 or any random value

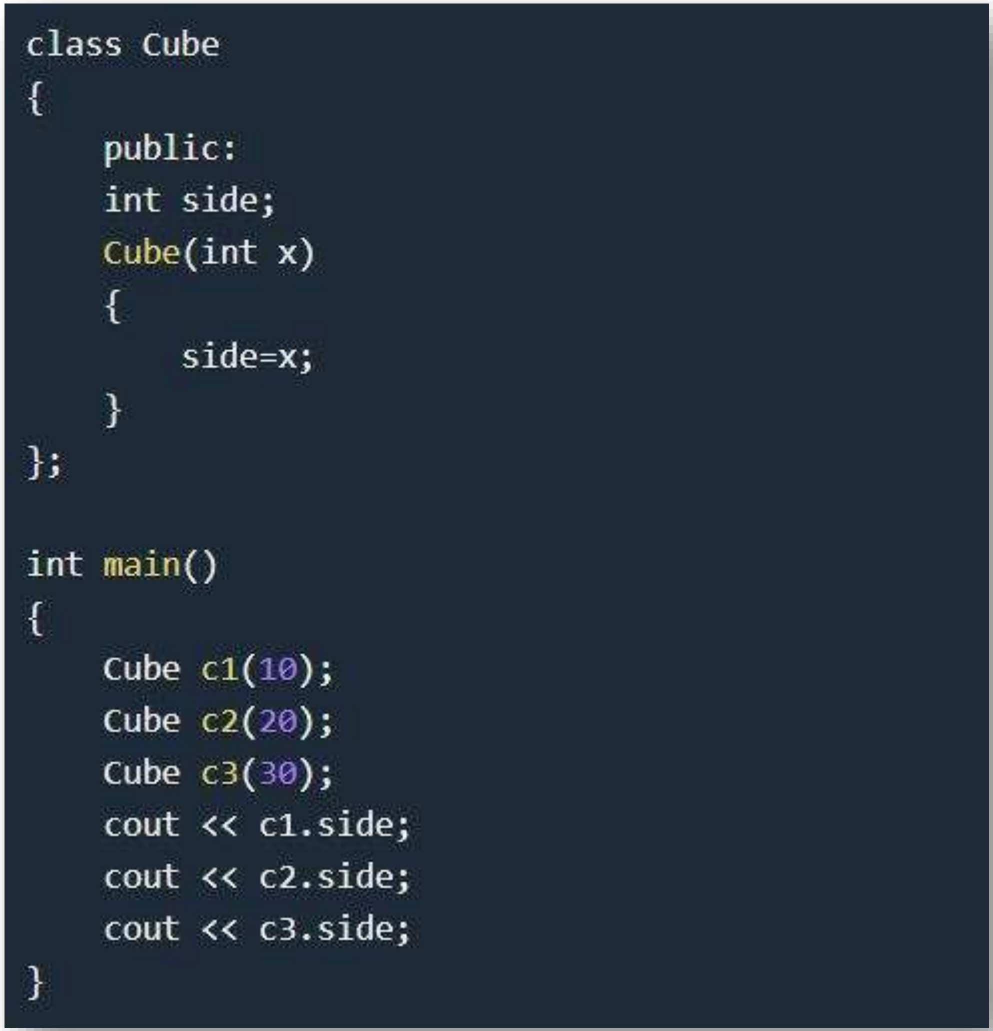
- In this case, default constructor provided by the compiler will be called which will initialize the object data members to default value, that will be 0 or any random integer value in this case.

PARAMETERIZED CONSTRUCTOR

- These are the constructors with parameter.

- Using this Constructor you can provide different values to data members of different objects, by passing the appropriate values as argument.

PARAMETERIZED CONSTRUCTOR

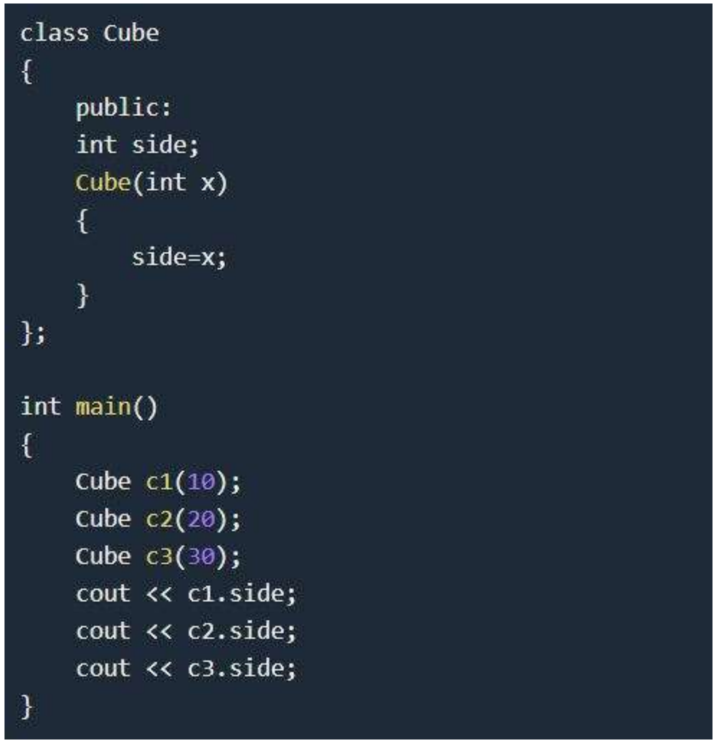


```
class Cube
{
    public:
    int side;
    Cube(int x)
    {
        side=x;
    }
};

int main()
{
    Cube c1(10);
    Cube c2(20);
    Cube c3(30);
    cout << c1.side;
    cout << c2.side;
    cout << c3.side;
}
```

- OUTPUT

```
10
20
30
```

PARAMETERIZED CONSTRUCTOR



```
class Cube
{
    public:
    int side;
    Cube(int x)
    {
        side=x;
    }
};

int main()
{
    Cube c1(10);
    Cube c2(20);
    Cube c3(30);
    cout << c1.side;
    cout << c2.side;
    cout << c3.side;
}
```

- By using parameterized constructor in above case, we have initialized 3 objects with user defined values. We can have any number of parameters in a constructor.

COPY CONSTRUCTOR

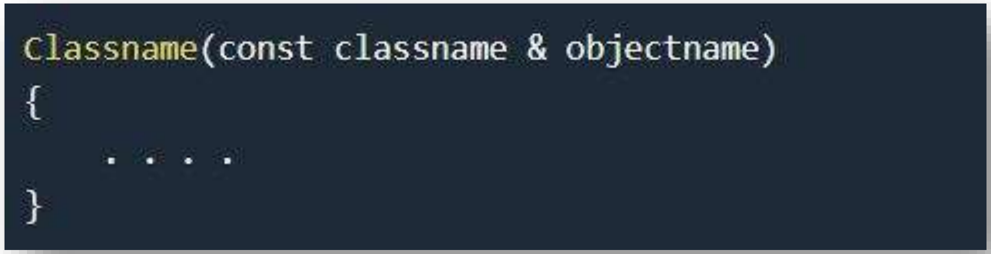
- These are special type of Constructors which takes an object as argument, and is used to copy values of data members of one object into other object.

COPY CONSTRUCTOR

- These are special type of Constructors which takes an object as argument, and is used to copy values of data members of one object into other object.

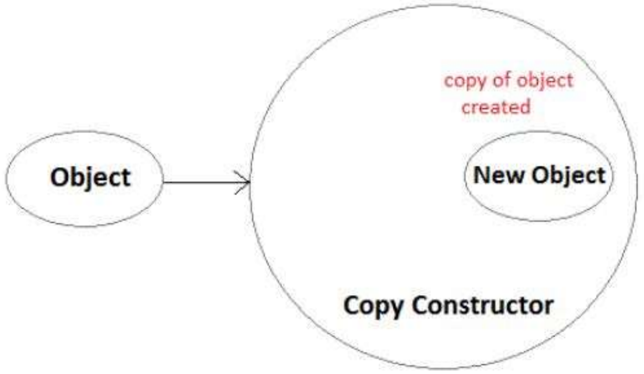
- It is usually of the form **X (X&)**, where X is the class name. The compiler provides a default Copy Constructor to all the classes.

COPY CONSTRUCTOR

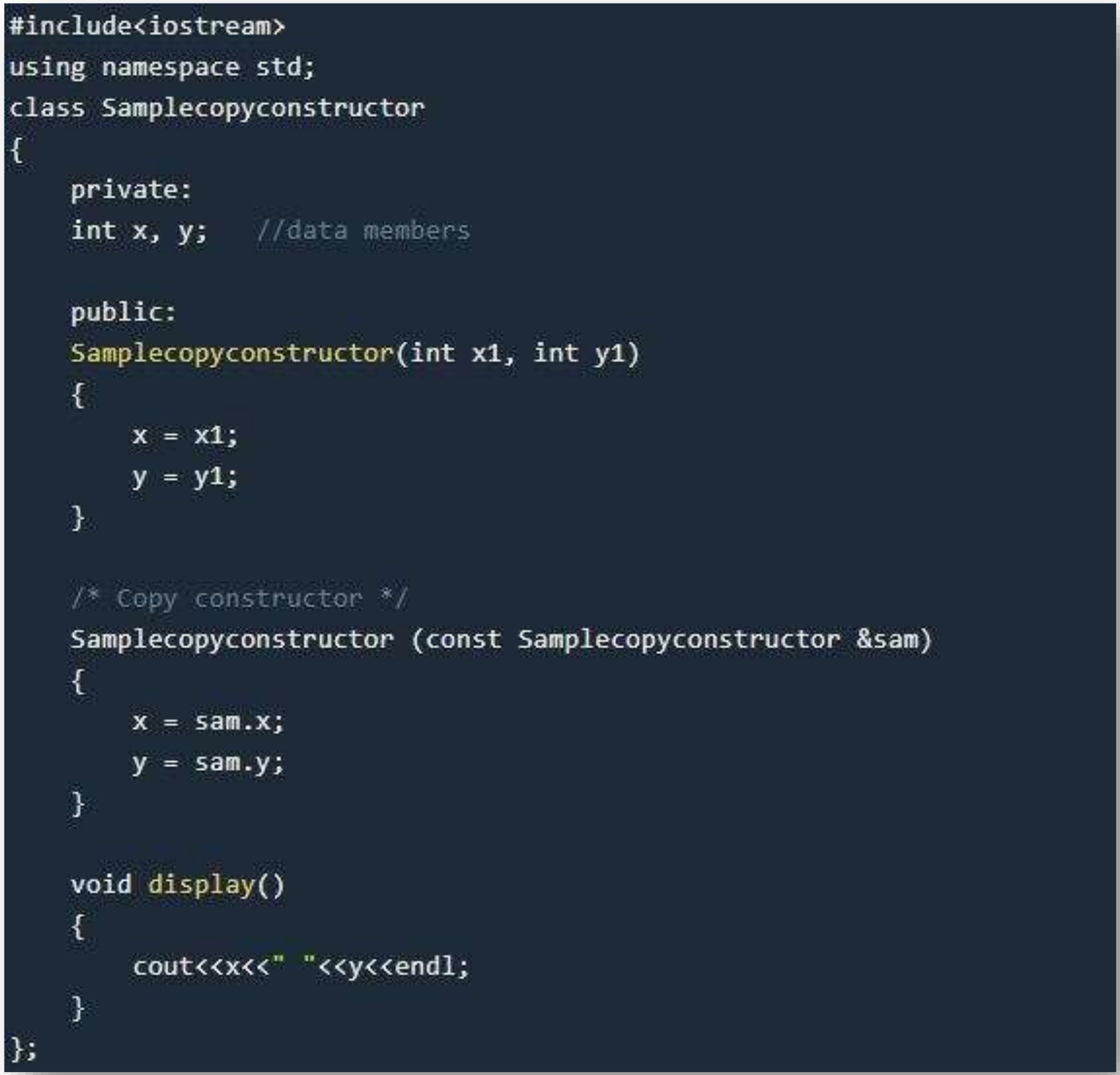


```
Classname(const classname & objectname)
{
    . . . .
}
```

- As it is used to create an object, hence it is called a constructor. And, it creates a new object, which is exact copy of the existing copy, hence it is called **copy constru**

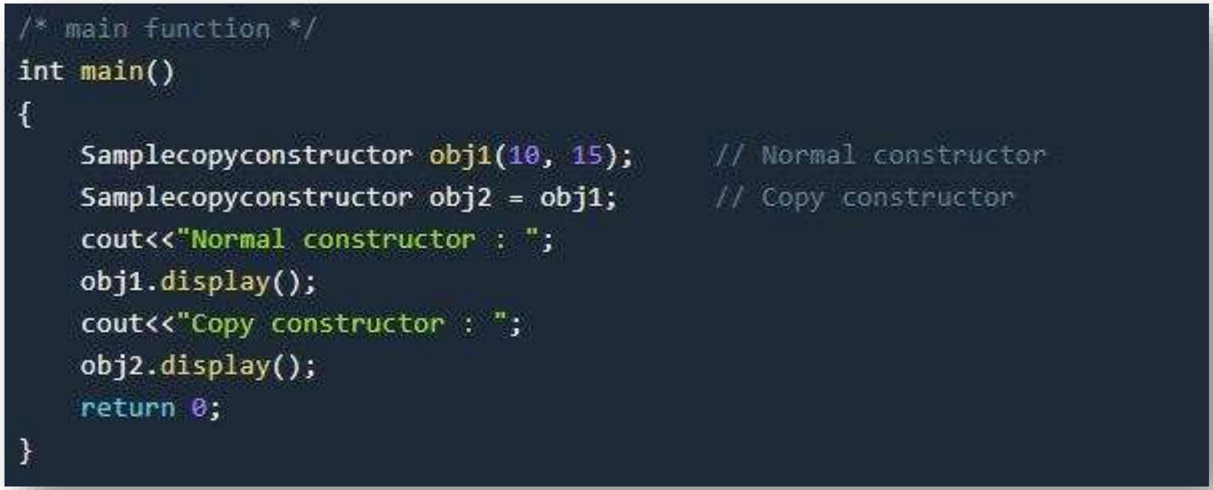


COPY CONSTRUCTOR



```
#include<iostream>
using namespace std;
class Samplecopyconstructor
{
    private:
    int x, y;   //data members

    public:
    Samplecopyconstructor(int x1, int y1)
    {
        x = x1;
        y = y1;
    }

    /* Copy constructor */
    Samplecopyconstructor (const Samplecopyconstructor &sam)
    {
        x = sam.x;
        y = sam.y;
    }

    void display()
    {
        cout<<x<<" "<<y<<endl;
    }
};
```

COPY CONSTRUCTOR



```
/* main function */
int main()
{
    Samplecopyconstructor obj1(10, 15);     // Normal constructor
    Samplecopyconstructor obj2 = obj1;      // Copy constructor
    cout<<"Normal constructor : ";
    obj1.display();
    cout<<"Copy constructor : ";
    obj2.display();
    return 0;
}
```

- Output :
  - Normal constructor : 10 15
  - Copy constructor : 10 15

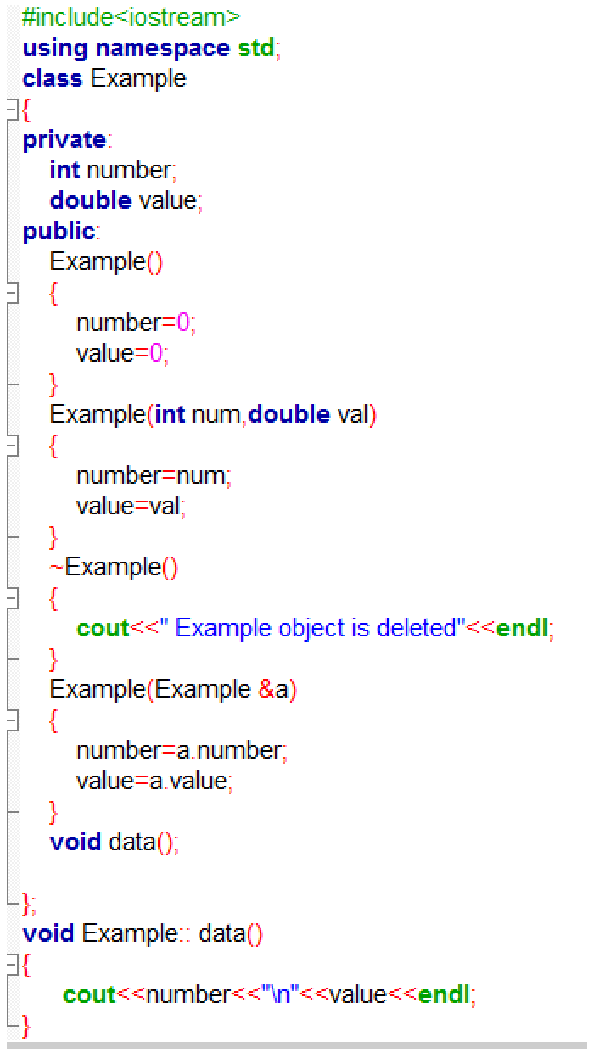
STATIC CONSTRUCTOR

- C++ doesn’t have static constructors but you can emulate them using a static instance of a nested class.

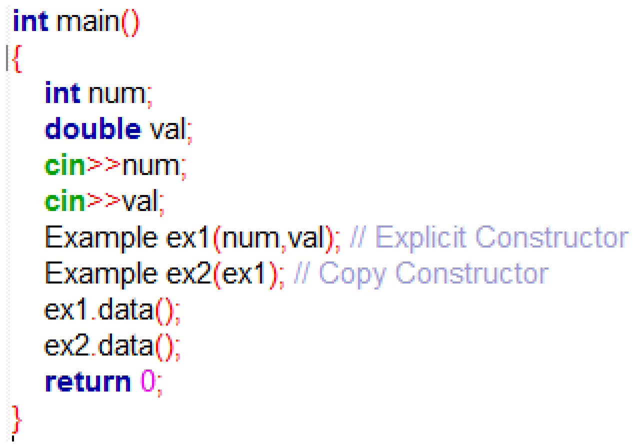
```
class has_static_constructor {
    friend class constructor;
    struct constructor {
        constructor() { /* do some constructing here … */ }
    };
    static constructor cons;
};
```

```
// C++ needs to define static members externally.
has_static_constructor::constructor has_static_constructor::cons;
```

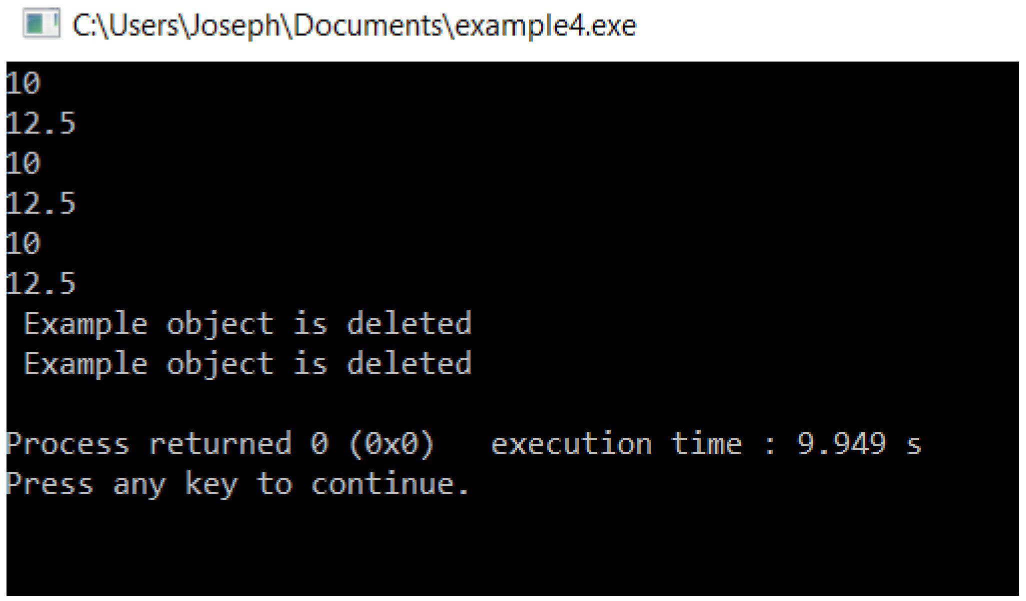
Try out program



```
#include<iostream>
using namespace std;
class Example
{
private:
    int number;
    double value;
public:
    Example()
    {
        number=0;
        value=0;
    }
    Example(int num,double val)
    {
        number=num;
        value=val;
    }
    ~Example()
    {
        cout<<" Example object is deleted"<<endl;
    }
    Example(Example &a)
    {
        number=a.number;
        value=a.value;
    }
    void data();

};
void Example:: data()
{
     cout<<number<<"\n"<<value<<endl;
}
```



```
int main()
{
    int num;
    double val;
    cin>>num;
    cin>>val;
    Example ex1(num,val); // Explicit Constructor
    Example ex2(ex1); // Copy Constructor
    ex1.data();
    ex2.data();
    return 0;
}
```



C:\Users\Joseph\Documents\example4.exe

```
10
12.5
10
12.5
10
12.5
 Example object is deleted
 Example object is deleted

Process returned 0 (0x0)   execution time : 9.949 s
Press any key to continue.
```

Questions

1. What is a copy constructor?
   a) A constructor that allows a user to move data from one object to another
   b) A constructor to initialize an object with the values of another object
   c) A constructor to check the whether to objects are equal or not
   d) A constructor to kill other copies of a given object.
2. What happens if a user forgets to define a constructor inside a class?
   a) Error occurs
   b) Segmentation fault
   c) Objects are not created properly
   d) Compiler provides a default constructor to avoid faults/errors.
3. How many parameters does a copy constructor require?
   a) 1
   b) 2
   c) 0
   d) 3

Session 2 & 3

Feature Polymorphism: Constructor overloading & Method overloading

Polymorphism

- The word polymorphism means having many forms.
- In simple words, we can define polymorphism as the ability of a message to be displayed in more than one form.
- Real life example of polymorphism: A person at the same time can have different characteristic. Like a man at the same time is a father, a husband, an employee.
- So the same person posses different behaviour in different situations. This is called polymorphism.

Polymorphism

- Polymorphism is considered as one of the important features of Object Oriented Programming.
- Polymorphism allows us to perform a single action in different ways. In other words, polymorphism allows you to define one interface and have multiple implementations.
- The word “poly” means many and “morphs” means forms, So it means many forms.

Polymorphism

- Overloading
  - Constructor Overloading
  - Method Overloading
  - Operator Overloading
- Overriding
  - Method Overriding

Constructor Overloading

- In C++, We can have more than one constructor in a class with same name, as long as each has a different list of arguments. This concept is known as Constructor Overloading

- Overloaded constructors essentially have the same name (name of the class) and different number of arguments.

- A constructor is called depending upon the number and type of arguments passed.

- While creating the object, arguments must be passed to let compiler know, which constructor needs to be called.

Constructor Overloading

- In C++, We can have more than one constructor in a class with same name, as long as each has a different list of arguments. This concept is known as Constructor Overloading
- Example

```
class construct{
 public:
  float area;
  // Constructor with no parameters
  construct() {
     area = 0;
  }
  // Constructor with two parameters
  construct(int a, int b) {
     area = a * b;
  }
  void disp() {
     cout<< area<< endl;
  }
};
```

```
int main() {
   construct o;
   construct o2( 10, 20);

   o.disp();
   o2.disp();
   return 1;
}
```

Output:

```
     0
     200
```

Constructor Overloading

```
// C++ program to demonstrate constructor overl
oading
#include <iostream>
using namespace std;
class Person { // create person class
private:
     int age; // data member
public:
   // 1. Constructor with no arguments
Person()
   {

age = 20; // when object is created the age will be
 20
   }
   // 2. Constructor with an argument
Person(int a)
{ // when parameterized Constructor is called with
 a value the  age passed will be initialized
        age = a;
   }
                   Int getAge()
{ // getter to return the age
        return age;
    }
};


int main()
{
   Person person1, person2(45);
// called the object of person class in diffent
way

cout<< "Person1 Age = " << person1.getAge()
 <<endl;
cout<< "Person2 Age = " << person2.getAge(
) <<endl;
return 0;
}
```

MCQ Questions

1. Which among the following best describes constructor overloading?

   a) Defining one constructor in each class of a program
   b) Defining more than one constructor in single class
   c) Defining more than one constructor in single class with different signature
   d) Defining destructor with each constructor

Answer: c

Explanation: If more than one constructors are defined in a class with same signature, then that results in error. The signatures must be different. So that the constructors can be differentiated.

MCQ Questions

2. Can constructors be overloaded in derived class?

a) Yes, always
b) Yes, if derived class has no constructor
c) No, programmer can't do it
d) No, never

Answer: d
Explanation: The constructor must be having the same name as that of a class. Hence a constructor of one class can't even be defined in another class. Since the constructors can't be defined in derived class, it can't be overloaded too, in derived class.

MCQ Questions

3. Does constructor overloading include different return types for constructors to be overloaded?

a) Yes, if return types are different, signature becomes different
b) Yes, because return types can differentiate two functions
c) No, return type can't differentiate two functions
d) No, constructors doesn't have any return type

Answer: d

Explanation: The constructors doesn't have any return type. When we can't have return type of a constructor, overloading based on the return type is not possible. Hence only parameters can be different.

MCQ Questions

4. Why do we use constructor overloading?

a) To use different types of constructors
b) Because it’s a feature provided
c) To initialize the object in different ways
d) To differentiate one constructor from another

Answer: c
Explanation: The constructors are overloaded to initialize the objects of a class in different ways. This allows us to initialize the object with either default values or used given values. If data members are not initialized then program may give unexpected results.

MCQ Questions

5. Which constructor will be called from the object created in the code below?

```
class A
{       int i;
        A()
        {
                i=0; cout&lt;&lt;i;
        }
        A(int x=0)
        {
                i=x;  cout&lt;&lt;I;
        }
};
A obj1;
```

a) Default constructor
b) Parameterized constructor
c) Compile time error
d) Run time error

ANSWER : C

Explanation: When a default constructor is defined and another constructor with 1 default value argument is defined, creating object without parameter will create ambiguity for the compiler. The compiler won't be able to decide which constructor should be called, hence compile time error.

Method Overloading

- Method overloading is a feature in C++ that allows creation of several methods with the same name but with different parameters.
- For example, print(), print(int), and print("Hello") are overloaded methods.
- While calling print() , no arguments are passed to the function
- When calling print(int) and print("Hello") , an integer and a string arguments are passed to the called function.
- Allows one function to perform different tasks

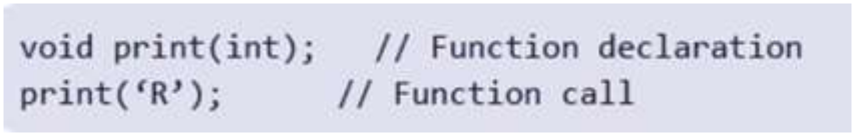
Types of Polymorphism

- Basically, there are two types of polymorphism:
  - Compile time (or static) polymorphism
  - Run-time (or dynamic) polymorphism.

- Static polymorphism -> Method overloading - calls a function using the best match technique or overload resolution.

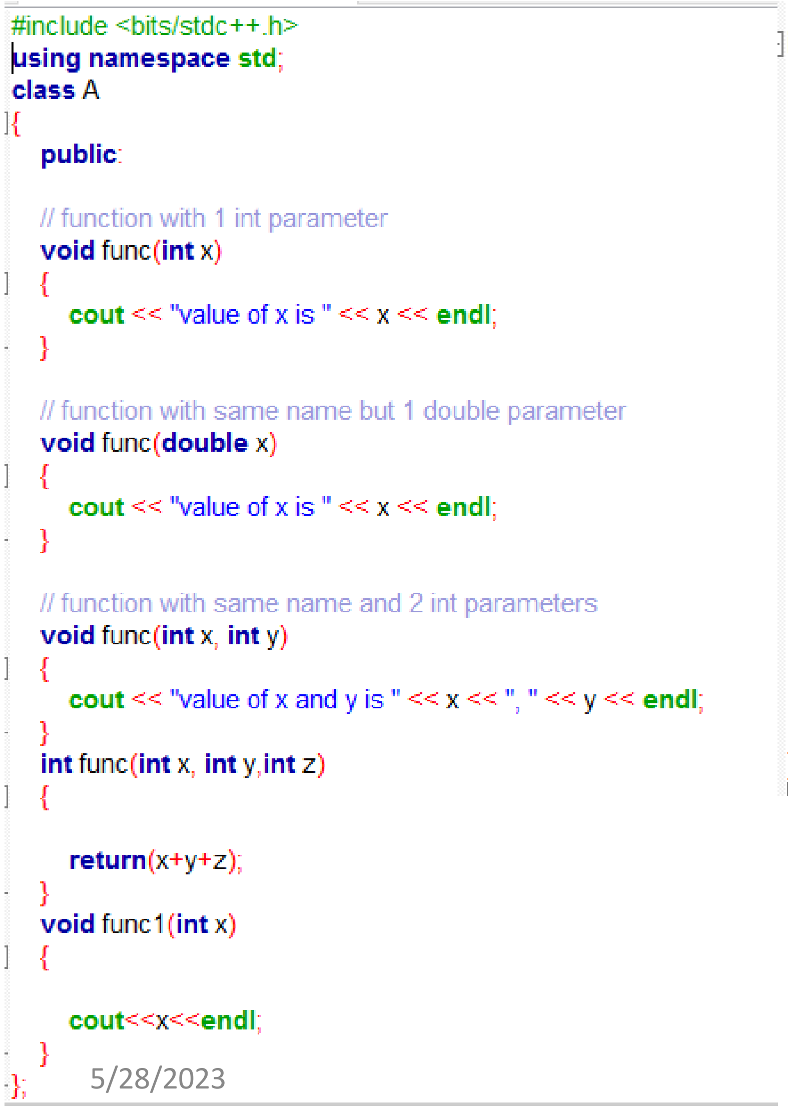
Matching Function Calls With Overloaded Methods

- When an overloaded function is called, one of the following cases occurs:
- Case 1: A direct match is found, and there is no confusion in calling the appropriate overloaded function.
- Case 2: If a match is not found, a linker error will be generated. However, if a direct match is not found, then, at first, the compiler will try to find a match through the type conversion or type casting.
- Case 3: If an ambiguous match is found, that is, when the arguments match more than one overloaded function, a compiler error will be generated. This usually happens because all standard conversions are treated equal.

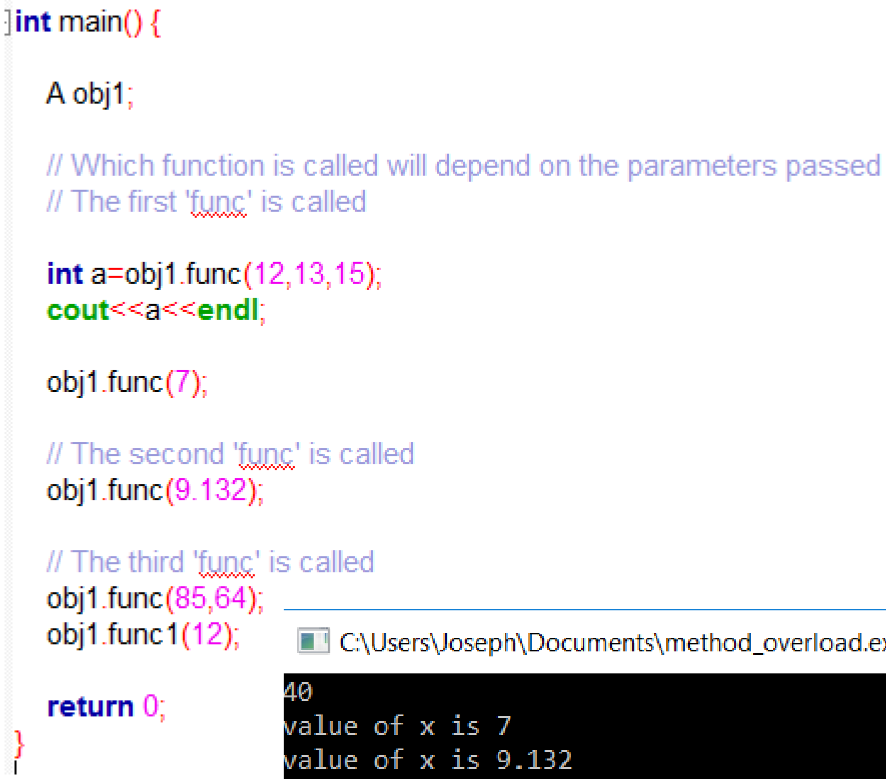


```
void print(int);    // Function declaration
print('R');       // Function call
```

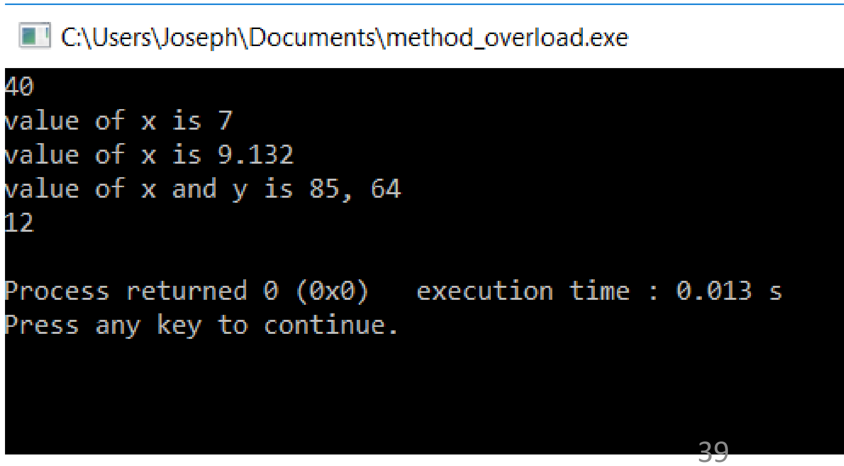
Try out Program



```
#include <bits/stdc++.h>
using namespace std;
class A
{
    public:

    // function with 1 int parameter
    void func(int x)
    {
        cout << "value of x is " << x << endl;
    }

    // function with same name but 1 double parameter
    void func(double x)
    {
        cout << "value of x is " << x << endl;
    }

    // function with same name and 2 int parameters
    void func(int x, int y)
    {
        cout << "value of x and y is " << x << ", " << y << endl;
    }
    int func(int x, int y,int z)
    {

        return(x+y+z);
    }
    void func1(int x)
    {

        cout<<x<<endl;
    }
};
```



```
int main() {

    A obj1;

    // Which function is called will depend on the parameters passed
    // The first 'func' is called

    int a=obj1.func(12,13,15);
    cout<<a<<endl;

    obj1.func(7);

    // The second 'func' is called
    obj1.func(9.132);

    // The third 'func' is called
    obj1.func(85,64);
    obj1.func1(12);

    return 0;
}
```



C:\Users\Joseph\Documents\method_overload.exe

```
40
value of x is 7
value of x is 9.132
value of x and y is 85, 64
12

Process returned 0 (0x0)   execution time : 0.013 s
Press any key to continue.
```

Questions

1. Which of the following permits function overloading on c++?
   a) type
   b) number of arguments
   c) type & number of arguments
   d) number of objects
2. Overloaded functions are ________________
   a) Very long functions that can hardly run
   b) One function containing another one or more functions inside it
   c) Two or more functions with the same name but different number of parameters or type
   d) Very long functions
3. What should be passed in parameters when function does not require any parameters?
   a) void
   b) blank space
   c) both void & blank space
   d) tab space

Session 6, 7 & 8
Operator Overloading & Types

INTRODUCTION

- The utility of operators such as +, =, *, /, >, <, and so on is predefined in any programming language.
- Programmers can use them directly on built-in data types to write their programs.
- However, these operators do not work for user-defined types such as objects.
- Therefore, C++ allows programmers to redefine the meaning of operators when they operate on class objects.
  This feature is called *operator overloading*

INTRODUCTION

Operator Overloading:

- **Operator** – It is a symbol that indicates an operation. Arithmetic operators are + (add two numbers), - (subtract two numbers), * (Multiply two numbers), / (Divide between two numbers).
- At now, we will take an Addition '+' Sign, its use of '+' sign is

5+5=10

2.5+2.5=5

INTRODUCTION

- **Operator Overloading** means multiple functions or multiple jobs. In operator overloading the '+' sign use to add the two objects.
- One of C++'s great features is its extensibility, Operator Overloading is major functionality related to extensibility.
- In C++, most of operators can be overloaded so that they can perform special operations relative to the classes you create.

INTRODUCTION

- **For Example,** ‘+’ operator can be overloaded to perform an operation of string concatenation along with its pre-defined job of adding two numeric values.
- When an operator is overloaded, none of its original meaning will be lost.
- After overloading the appropriate operators, you can use C++’s built in data types.

TYPES OF OPERATOR

- **Unary Operator**

  - Operators attached to a single operand.

  (-a, +a, --a, ++a, a--, a++)

- **Binary Operator**

  - Operators attached to two operand.

  (a-b, a+b, a*b, a/b, a%b, a>b, a<b)

Syntax

```
return-type class-name:: operator op(arg-list)
{
function body
}
```

EXPLANATION

- return type – It is the type of value returned by the specified operation.
- op - It is the operator being overloaded. It may be unary or binary operator. It is preceded by the keyword operator.
- operator op - It is the function name, Where operator is a keyword.

UNARY OPERATOR OVERLOADING

Introduction

- One of the exciting features of C++
- Works only on the single variable
- It can be overloaded two ways
  1. Static member function
  2. Friend function
- -,+,++,-- those are unary operator which we can overloaded.

Using a member function to Overload Unary Operator

```
#include<iostream.h>
class Number

{   private:
        int x;
    public:
        Number()
        {   x = 0;     };
        Number(int n)  //parameterized constructor
        {   x = n;     }
        void operator -() // operator overloaded function
        {   x = -x; }
        void show_data()
        {   cout<<"\n x = "<<x;       }
};
main()
{   Number N(7);   // create object
    N.show_data();
    -N;            // invoke operator overloaded function
    N.show_data();
}
```

OUTPUT

```
x = 7
x = -7
```

UNARY OPERATOR OVERLOADING

Example Program:

Write a program which will convert an positive values in an object to negative value.

<u>Code:</u>

```
#include <iostream.h>
class demo
{

int x,y,z;
public:
void getdata (int a, int b,int c)
{
x=a;
y=b;
z=c;
}
```

Contd...,

```
void display();
void operator –();
};
void demo::display()
{
cout<<“x=“<<x<<“\ny=“<<y<<“\nz=“<<z<<endl;
}
void demo::operator –()
{
x=-x;
y=-y;
z=-z;
}
int main()
{
demo obj1;
```

CONTD...,

```
obj1.getdata(10,20,30);
obj1.display();
-obj1;
obj1.display();
return 0;
}
```

Output:

```
x=10
y=20
z=30

x=-10
y=-20
z=-30
```

BINARY OPERATOR OVERLOADING IN C++

BINARY OPERATOR

INTRODUCTION

- In Binary operator overloading function, there should be one argument to be passed.
- It is overloading of an operator operating on two operands.

BINARY OPERATOR OVERLOADING

```
#include<iostream>  class
multiply
{
int first,second;
public:
void getdata(int a,int b)
{
first=a;  second=b;
}
```

Contd...,

```
void display()
{
cout<<“first=“<<first<<“second=“<<secon<<endl;
}
multiply operator *(multiply c);
};
void multiply::operator *(multiply c)
{
multiply temp;
temp.first=first*c.first;
temp.second=second*c.second;
return temp;
}
```

Contd...

```
int main()
{
multiply obj1,obj2,obj3;
obj1.getdata(15,20);
obj2.getdata(3,45);
obj3=obj1*obj2;
obj3.display();
return 0;
}
```

Output:

```
45
900
```

MCQ Questions

I. **In case of operator overloading, operator function must be ______ .**

1. Static member functions

2. Non- static member functions

3. Friend Functions

a. Only 2

b. Only 1, 3

c. Only 2 , 3

d. All 1 , 2, 3

MCQ Questions

In case of operator overloading, operator function must be ______ .

1. Static member functions

2. Non- static member functions

3. Friend Functions

a. Only 2

b. Only 1, 3

c. Only 2 , 3

d. All 1 , 2, 3

MCQ Questions

II. Using friend operator function, following perfect set of operators may not be overloaded.

a. = , ( ) , [ ] , ->

b. <<, = = , [ ] , >>

c. ?, = , ( ) , ++

d. +,-,--,++

MCQ Questions

II. Using friend operator function, following perfect set of operators may not be overloaded.

a. = , ( ) , [ ] , ->

b. <<, = = , [ ] , >>

c. ?, = , ( ) , ++

d. +,-,--,++

MCQ Questions

III. When overloading unary operators using Friend function,it requires_____ argument/s.

a. Zero

b. One

c. Two

d. None of these.

MCQ Questions

III. When overloading unary operators using Friend function,it requires_____ argument/s.

a. Zero

b. One

c. Two

d. None of these.

MCQ Questions

IV. In case of binary operator overloading with member function, which of following statement should be taken into consideration?

a. Right hand operand must be object.

b. Left hand operand must be object.

c. Both the operands must be objects.

d. All of these should be considered.

MCQ Questions

IV. In case of binary operator overloading with member function, which of following statement should be taken into consideration?

a. Right hand operand must be object.

b. Left hand operand must be object.

c. Both the operands must be objects.

d. All of these should be considered.

MCQ Questions

V.Which is the correct statement anout operator overloading in C++?

A. Only arithmetic operators can be overloaded
B. Only non-arithmetic operators can be overloaded
C. Precedence of operators are changed after overlaoding
D. Associativity and precedence of operators does not change

MCQ Questions

V. Which is the correct statement anout operator overloading in C++?

A. Only arithmetic operators can be overloaded
B. Only non-arithmetic operators can be overloaded
C. Precedence of operators are changed after overlaoding
D. **Associativity and precedence of operators does not change**

Session 11

UML Sequence Diagram

Interaction Diagram

- Interaction diagrams are used to observe the dynamic behavior of a system.
- Interaction diagram visualizes the communication and sequence of message passing in the system.
- Interaction diagram represents the structural aspects of various objects in the system.
- Interaction diagram represents the ordered sequence of interactions within a system.
- Interaction diagram provides the means of visualizing the real time data via UML.

- This interactive behavior is represented in UML by two diagrams known as
  - Sequence diagram
  - Collaboration diagram.
- Sequence diagram emphasizes on time sequence of messages from one object to another.
- Collaboration diagram emphasizes on the structural organization of the objects that send and receive messages.

How to Draw an Interaction Diagram?

- The purpose of interaction diagrams is to capture the dynamic aspect of a system.

- So to capture the dynamic aspect, we need to understand what a dynamic aspect is and how it is visualized. Dynamic aspect can be defined as the snapshot of the running system at a particular moment.

- Following things are to be identified clearly before drawing the interaction diagram
  - Objects taking part in the interaction.
  - Message flows among the objects.
  - The sequence in which the messages are flowing.
  - Object organization.

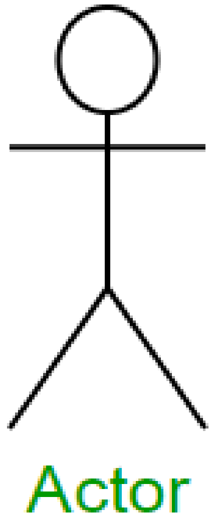
Sequence Diagram

- A sequence diagram simply depicts interaction between objects in a sequential order i.e. the order in which these interactions take place.

Sequence Diagram Notations

Actors :

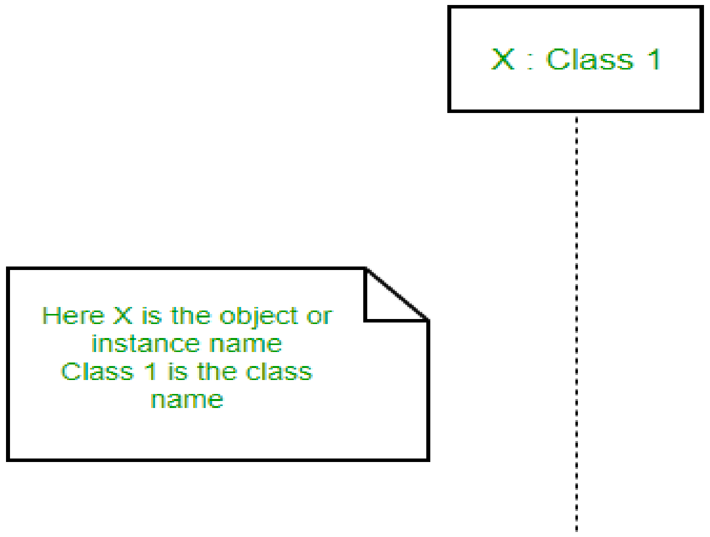
An actor in a UML diagram represents a type of role where it interacts with the system and its objects.



2.Lifelines :

- A lifeline is a named element which depicts an individual participant in a sequence diagram. So basically each instance in a sequence diagram is represented by a lifeline.
- Lifeline elements are located at the top in a sequence diagram.
- lifeline follows the following format :

Instance Name : Class Name


X : Class 1
Here X is the object or instance name Class 1 is the class name

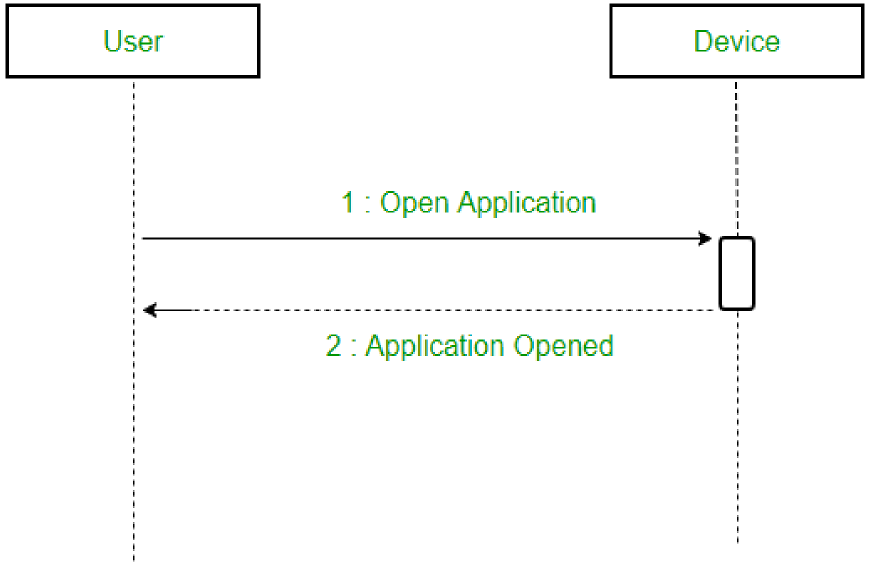
3.Messages :

- Communication between objects is depicted using messages. The messages appear in a sequential order on the lifeline.

- We represent messages using arrows. Lifelines and messages form the core of a sequence diagram.

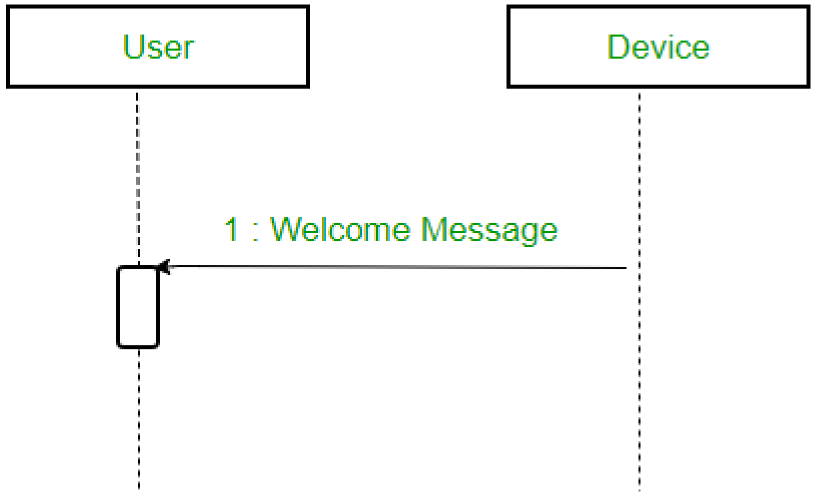
Synchronous messages

- A synchronous message waits for a reply before the interaction can move forward.
- The sender waits until the receiver has completed the processing of the message.
- The caller continues only when it knows that the receiver has processed the previous message i.e. it receives a reply message.
- A large number of calls in object oriented programming are synchronous. We use a solid arrow head to represent a synchronous message.



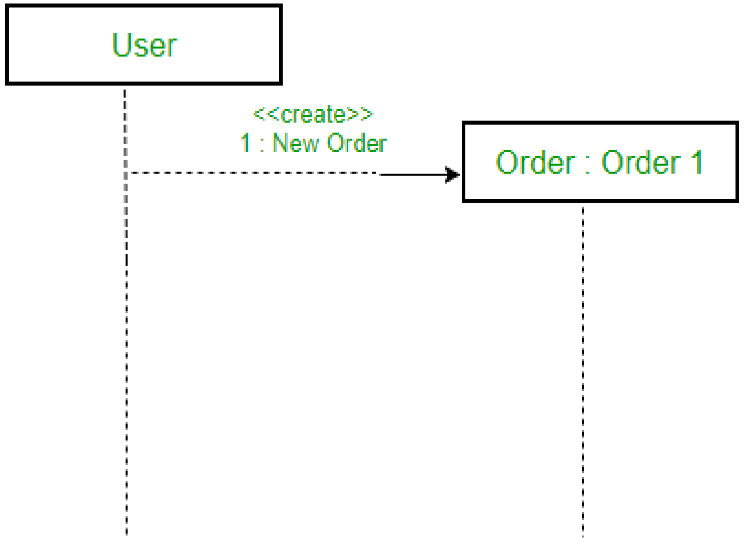
Asynchronous Messages

- An asynchronous message does not wait for a reply from the receiver.
- The interaction moves forward irrespective of the receiver processing the previous message or not.
- We use a lined arrow head to represent an asynchronous message.



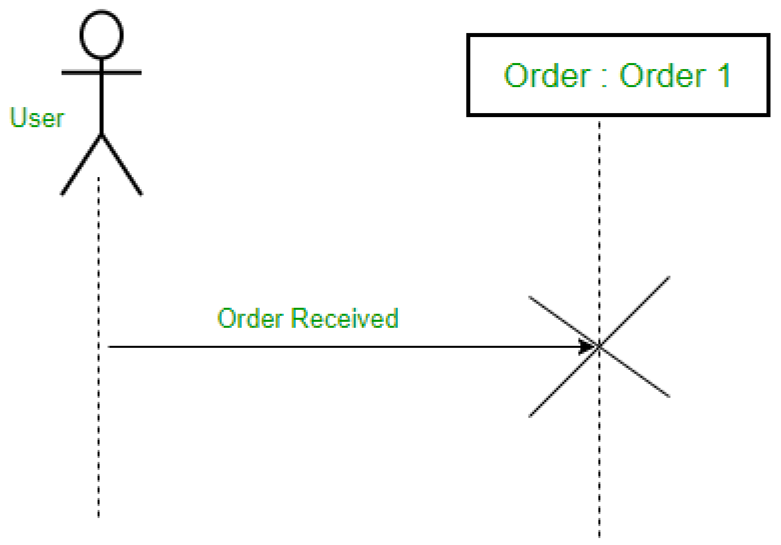
Create message

- We use a Create message to instantiate a new object in the sequence diagram.
- It is represented with a dotted arrow and create word labeled on it to specify that it is the create Message symbol.
  For example :
  - The creation of a new order on a e-commerce website would require a new object of Order class to be created.



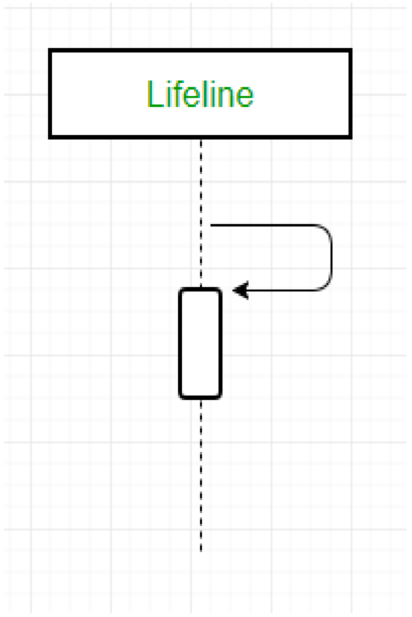
Delete Message

- We use a Delete Message to delete an object.
- It destroys the occurrence of the object in the system.
- It is represented by an arrow terminating with a x.
  For example – In the scenario below when the order is received by the user, the object of order class can be destroyed



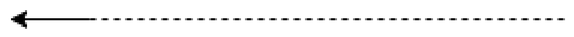
Self Message

- A message an object sends to itself, usually shown as a U shaped arrow pointing back to itself.



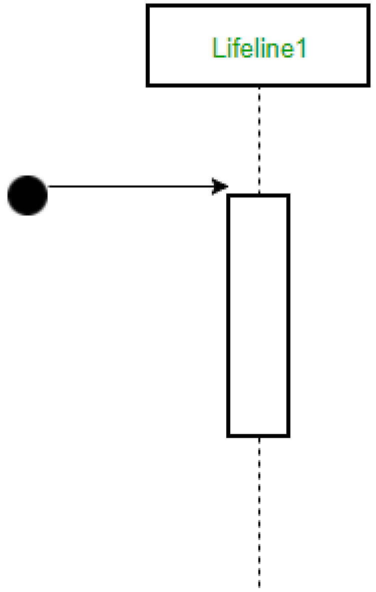
Reply Message

- Reply messages are used to show the message being sent from the receiver to the sender.
- We represent a return/reply message using an open arrowhead with a dotted line.
- The interaction moves forward only when a reply message is sent by the receiver.



Found Message

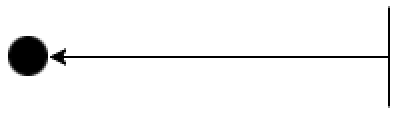
- A Found message is used to represent a scenario where an unknown source sends the message.
- It is represented using an arrow directed towards a lifeline from an end point.



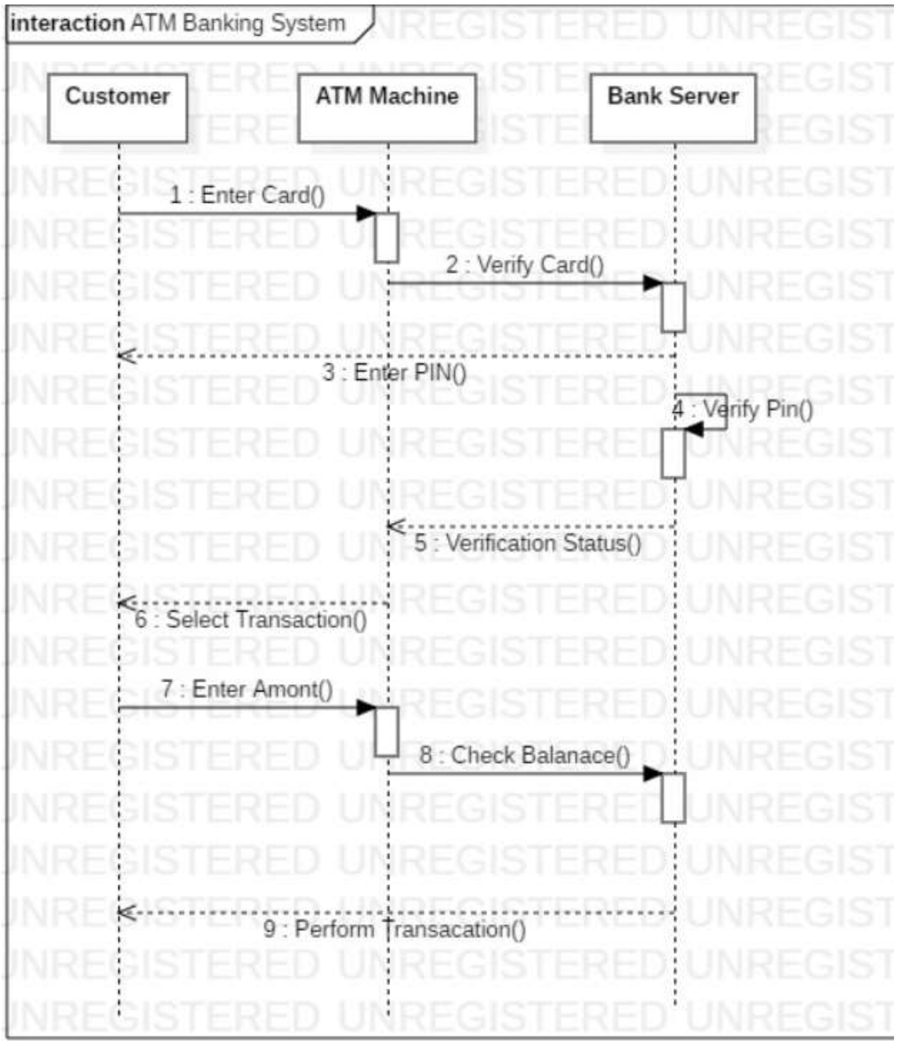
Lost Message

- A Lost message is used to represent a scenario where the recipient is not known to the system.
- It is represented using an arrow directed towards an end point from a lifeline.

For example:



Example – Sequence Diagram



Questions

1. What does a message mean?
   a) **It Passes all communications from one object to another and are represented by message arrows in sequence diagrams.**
   b) The message goes from the sending object's lifeline to the receiving object's lifeline.
   c) It is a rectangle containing an identifier with a dashed line extending below the rectangle.
   d) List of all attributes.
2. What is a lifeline?
   a) It is a frame consisting of a rectangle with a pentagon in its upper left-hand corner
   b) It is a rectangle containing an identifier with a dashed line extending below the rectangle
   c) It is a name compartment; the interaction is represented inside the rectangle
   d) Emergency situation in real world approach.

Session 12

UML Collaboration Diagram

Collaboration diagram

➢**COLLABORATION DIAGRAM** depicts the relationships and interactions among software objects. They are used to understand the object architecture within a system rather than the flow of a message as in a sequence diagram. They are also known as “Communication Diagrams.”

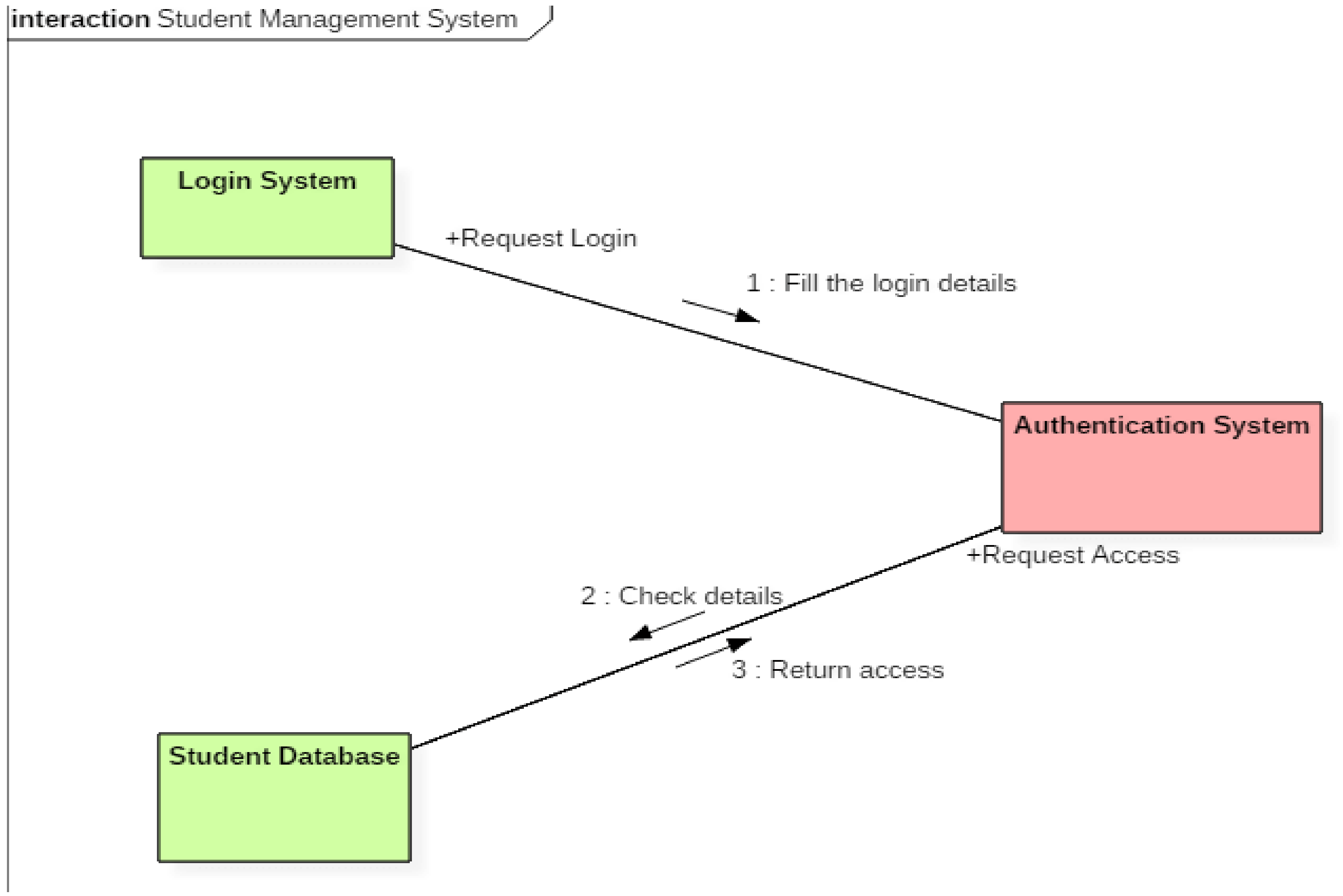
In the collaboration diagram, the method call sequence is indicated by some numbering technique. The number indicates how the methods are called one after another.

Benefits of Collaboration Diagram

It is also called as a communication diagram.
It emphasizes the structural aspects of an interaction diagram - how lifeline connects.
Its syntax is similar to that of sequence diagram except that lifeline don't have tails.
Messages passed over sequencing is indicated by numbering each message hierarchically.
Compared to the sequence diagram communication diagram is semantically weak.
Object diagrams are special case of communication diagram.
It allows you to focus on the elements rather than focusing on the message flow as described in the sequence diagram.
Sequence diagrams can be easily converted into a collaboration diagram as collaboration diagrams are not very expressive.

Example

Following diagram represents the sequencing over student management system:



Example

The above collaboration diagram represents a student information management system. The flow of communication in the above diagram is given by,

A student requests a login through the login system.

An authentication mechanism of software checks the request.

If a student entry exists in the database, then the access is allowed;

otherwise, an error is returned.

MCQ’s

1. A collaboration diagram shows

a. Structural Aspect

b. Behavioral Aspect

c. Environmental Aspect

d. Both A and B

e**. Both B and C**

2. which diagram is used to show interactions between messages are classified as?

a.activity

b.state chart

c.collaboration

d.object lifeline

Session 13

Inheritance & Types

Inheritance

01 Definition

Inheritance is the capability of one class to acquire properties and characteristics from another class. The class whose properties are inherited by other class is called the **Parent** or **Base** or **Super** class. And, the class which inherits

02 Synta
x

```
class Subclass_name : access_mode Superclass_name
```

0 Type
3 s

Single Inheritance, Multiple Inheritance, Hierarchical Inheritance, Multilevel Inheritance, and Hybrid Inheritance (also known as Virtual Inheritance)

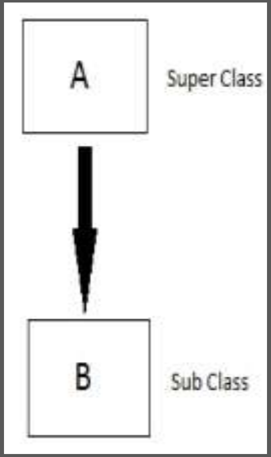
<u>Note</u> : All members of a class except Private, are inherited

04 Advantages

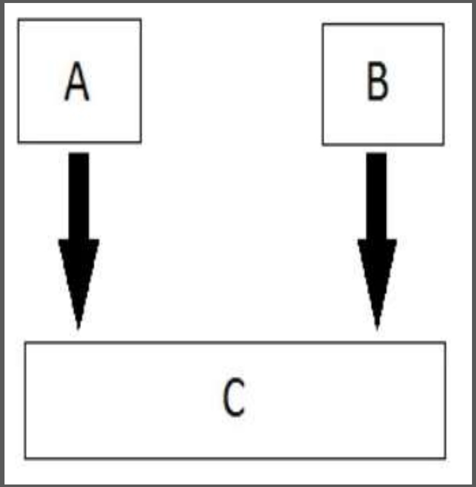
1. Code Reusability
2. Method Overriding (Hence, Runtime Polymorphism.)
3. Use of Virtual Keyword

Inheritance Types

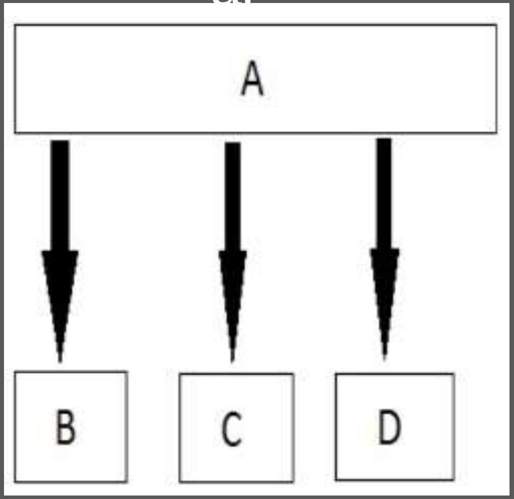
01 Single



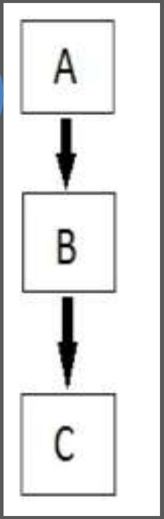
02 Multiple



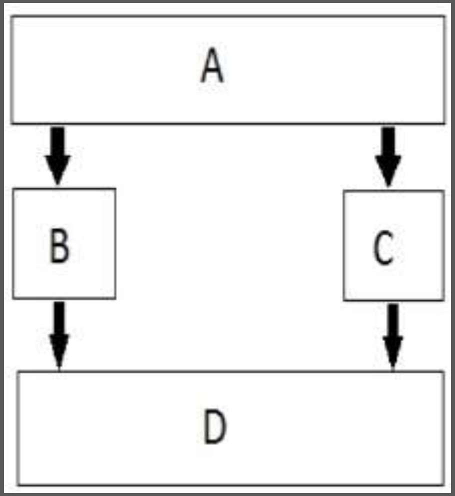
03 Hierarchic al



04 Multilevel



05 Hybrid



Modes of Inheritance

01 Public

If we derive a sub class from a public base class. Then the public member of the base class will become public in the derived class and protected members of the base class will become protected in derived class

02 Protected

If we derive a sub class from a Protected base class. Then both public member and protected members of the base class will become protected in derived class.

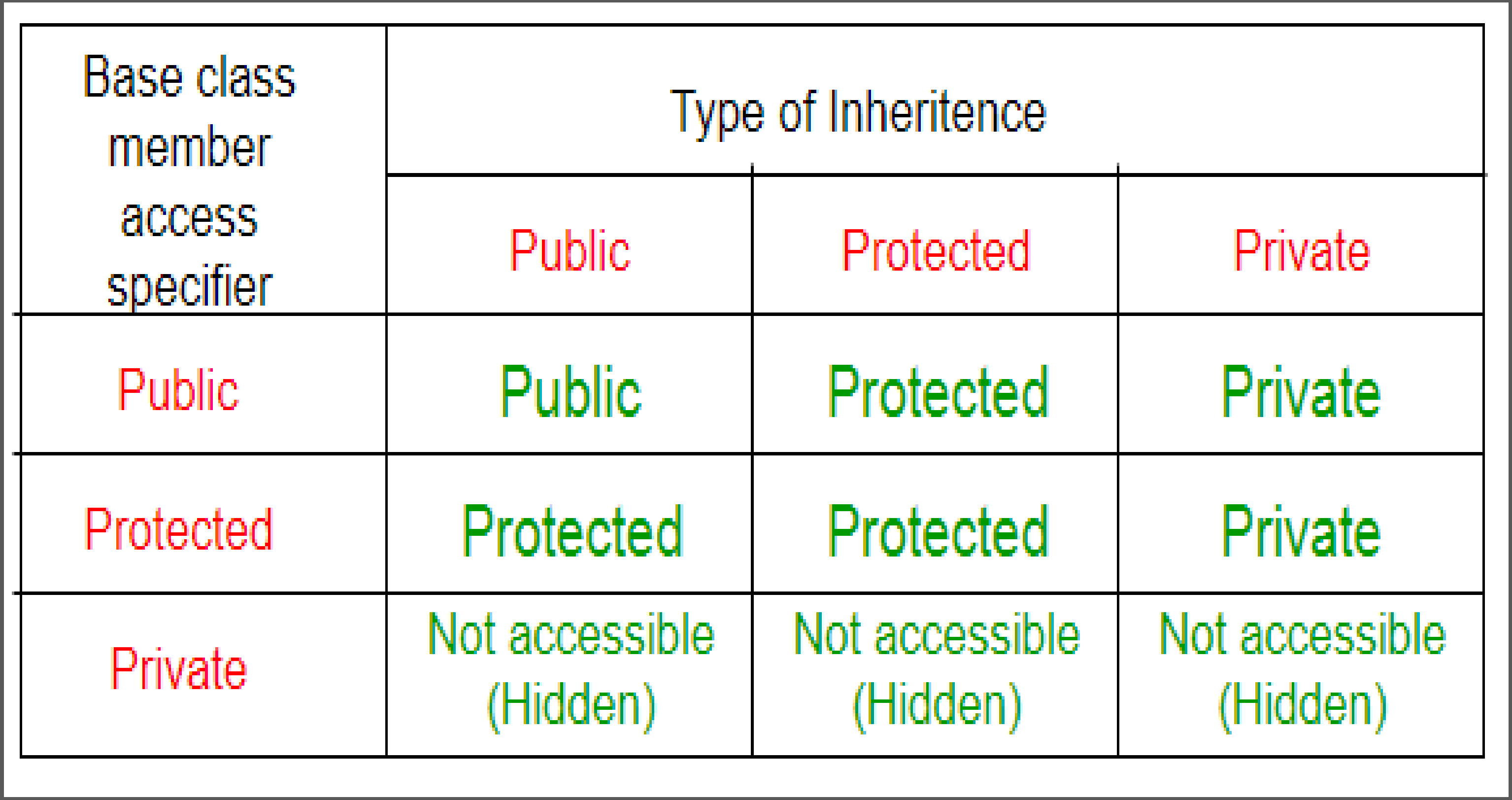
03 private

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.

Note:

The private members in the base class cannot be directly accessed in the derived class, while protected members can be directly accessed.

Inheritance Access Matrix



Base class member access specifier	Type of Inheritence		
	Public	Protected	Private
Public	Public	Protected	Private
Protected	Protected	Protected	Private
Private	Not accessible (Hidden)	Not accessible (Hidden)	Not accessible (Hidden)

Thank you