CAP444
OBJECT ORIENTED PROGRAMMING USING C++

Unit1

Created By:
Kumar Vishal
(SCA), LPU

Unit-1

Principles of OOP :

- basic concepts of object oriented programming,
- object oriented languages,
- classes and objects,
- access specifiers
- constructors: types of constructors
- multiple constructor in a class
- Destructors
- functions overloading
- friend function
- inheritance: types of inheritance

❖ object oriented languages

Procedural Programming and Object Oriented Programming:

Languages used in Procedural Programming:
FORTRAN, ALGOL, COBOL,
BASIC, Pascal and C.

Languages used in Object Oriented Programming:
Java, C++, C#, Python,
PHP, JavaScript, Ruby, Perl,
Objective-C, Dart, Swift, Scala.

OOPs Features

1. *Class*
2. *Objects*
3. *Encapsulation*
4. *Abstraction*
5. *Polymorphism*
6. *Inheritance*
7. *Dynamic Binding*
8. *Message Passing*

Class

Class is a collection of similar types of objects.

For example: Fruits is class of mango, apple , orange etc.

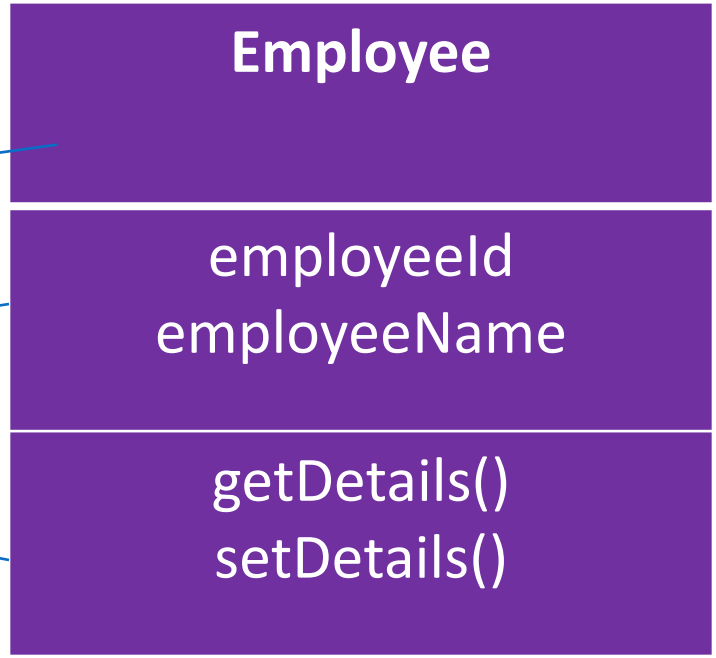
Fruits

Class

Class is a collection of data members and member functions.

Example:

```
class Employee{
//data members
// member functions
}
```



```
class student
{
public:
int regno;
void getStudentDetails();
}
```

In this code which option is correct?

A: regno is data member
B. getStudentDetails() is member function
C. above both options are correct

Object

Object is an instance of a Class.

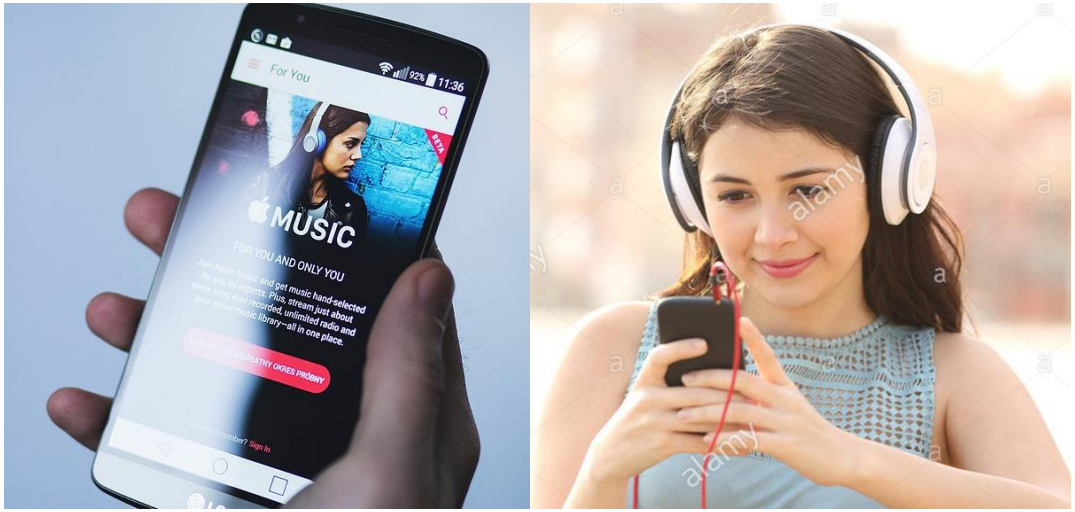
```
class Employee
{
        int employeeId;
        char employeeName[20];
        public:
        void getDetails(){}
        void setDetails(){}
};
int main()
{
Employee e1; // e1 is a object
return 0;
}
```

Encapsulation

- Wrapping up of data and information under a single unit.
- Binding together the data and the functions in a single unit
- Encapsulation also hides the data
- We can achieve encapsulation features by making data member as a private and use get and set accessor methods to access data members.

Abstraction

- Hiding background details and showing features or functionality only



You know how to drive the car but you don't know about the internal details this is the example of ….?

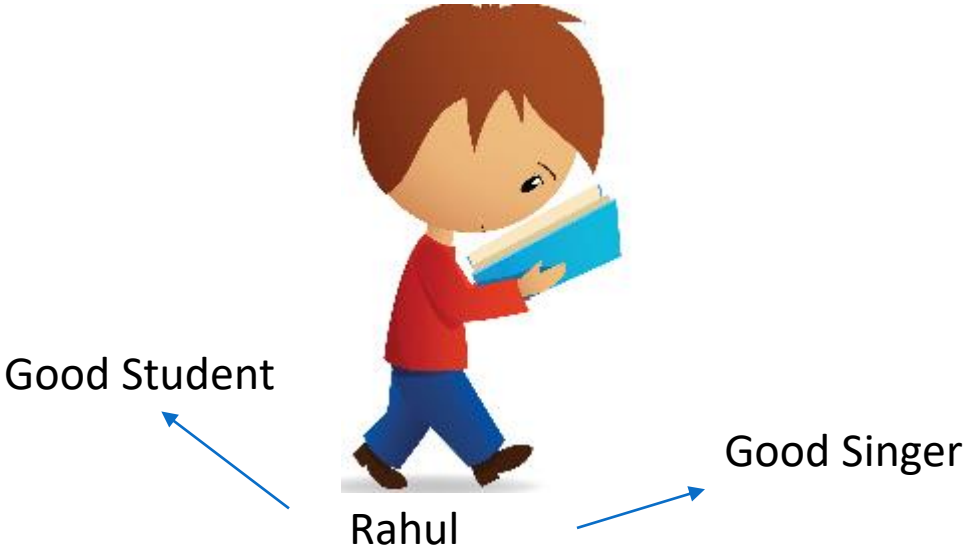
A. Encapsulation
B. Abstraction
C. None

How to achieve abstraction in C++ ?

- An abstraction can be achieved using classes.
- A class has the responsibility to determine which data member is to be visible outside and which is not.
- Access specifiers is used to follow this access mechanism within class

Polymorphism

- The ability to perform a task in more than one form.
- We use Function overloading and Function overriding to achieve polymorphism.



- Continued in next class_____

- basic concepts of object oriented programming

Variable and its types

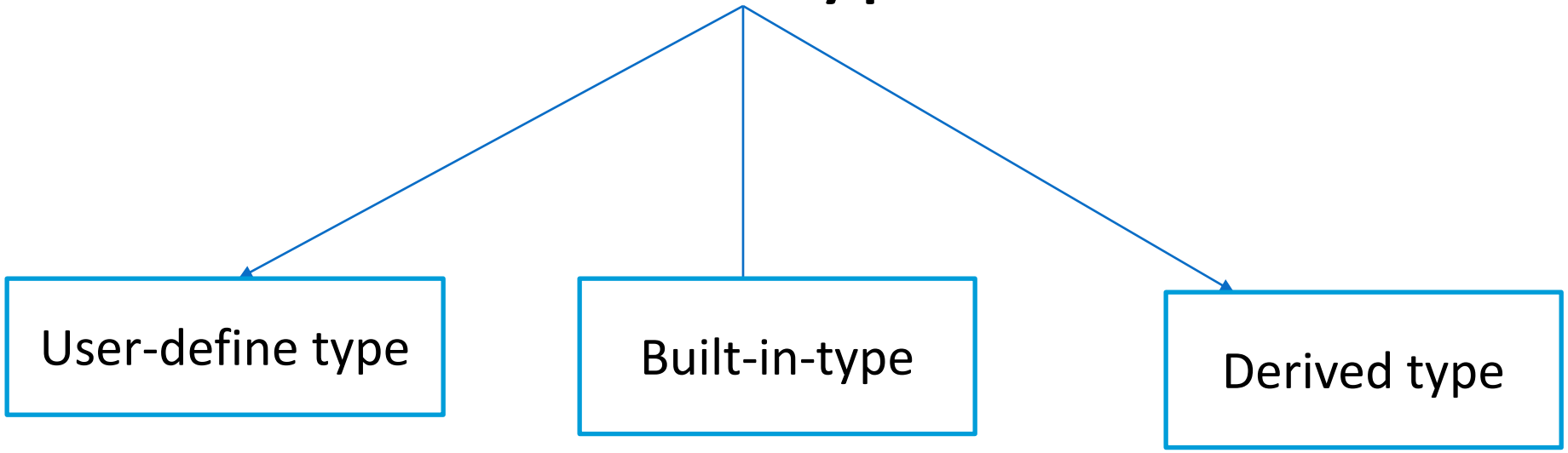
- A name of storage location which holds some value

Syntax:

Var_type var_name;// declaration

Var_type var_name=value;//initialization

Data types



- Class
- Structure
- Union
- Enumeration

- Integer
- Character
- Boolean
- Floating Point
- Double
- Void
- Wide Character

- Function
- Array
- Pointer
- Reference

Data type modifiers are:

- Signed
- Unsigned
- Short
- Long

Data type	Size(in byte)	Range
char	1 =8 bits (2^8)	-127 to 127 or 0 to 255
unsigned char	1	0 to 255
signed char	1	-127 to 127
int	4=32 bits (2^{32})	-2,147,483,648 to 2,147,483,647
short int	2	-32,768 to 32,767
unsigned short int	2	0 to 65,535
unsigned int	4	0 to 4,294,967,295
float	4	
double	8	
long double	12	

Memory representation

1 Byte= 8 Bits

128	64	32	16	8	4	2	1
0	1	0	0	0	0	0	1

A(65)=01000001

char ch=65;
Or
char ch='A'

Char is occupying 1 Byte memory

Operators

- Arithmetic operators
- Assignment operators
- Comparison operators
- Logical operators
- Bitwise operators

Arithmetic operators

Operator	Name	Example
+	Addition	x + y
-	Subtraction	x - y
*	Multiplication	x * y
/	Division	x / y
%	Modulus	x % y
++	Increment	++x
--	Decrement	--x

Assignment Operators

Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
&=	x &= 3	x = x & 3
|=	x |= 3	x = x | 3
^=	x ^= 3	x = x ^ 3
>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3

Comparison operators

Operator	Name	Example
==	Equal to	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

Logical operators

Operator	Name	Description	Example
&&	Logical and	Returns true if both statements are true	x < 5 && x < 10
||	Logical or	Returns true if one of the statements is true	x < 5 || x < 4
!	Logical not	Reverse the result, returns false if the result is true	!(x < 5 && x < 10)

Write a program for shopping mall, In shopping mall, there is a mobile shop, in this mobile shop you are getting offer if you will purchase mobile phone with power bank you will get 10% discount , but if you purchase any mobile or power bank you will get only 5 % discount. Display this offer to the user screen whenever user is selecting the option.

Select option :

1. Mobile

Select option:

1. Only mobile
2. Only power bank
3. mobile with power bank
4. not any

Bitwise operators

Operator	Description
&	AND Operator
|	OR Operator
^	XOR Operator
~	Ones Complement Operator
<<	Left Shift Operator
>>	Right Shift Operator

Real life example for bitwise operator

AND Operator (&)

If both side bit is on result will be On

a	b	a & b
0	0	0
0	1	0
1	0	0
1	1	1

Steps to solve:-

- **a = 12 (find binary form:1100)**
- **b = 25 (find binary form:11001)**

How to find Binary:

64	32	16	8	4	2	1	
		0	1	1	0	0	12
		1	1	0	0	1	25
			1	0	0	0	8

a & b=

01100 (12)

11001 (25)

01000 (8) Ans.

What will be output?

```
#include <iostream>

using namespace std;

int main()
{
    int a=20;
    int b=25;
    cout<<(a&b);
return 0;
}
```

A. 15
B. 16
C. 20

OR Operator (|)

If any side bit is on result will be On

a	b	a | b
0	0	0
0	1	1
1	0	1
1	1	1

Steps to solve:-

- **a = 12 (find binary form:1100)**
- **b = 25 (find binary form:11001)**

How to find Binary:

64	32	16	8	4	2	1	
		0	1	1	0	0	12
		1	1	0	0	1	25
		1	1	1	0	1	29

a | b=

01100 (12)

11001 (25)

11101 (29) Ans.

What will be output?

```
#include <iostream>

using namespace std;

int main()
{
   int a=20;
   int b=15;
   cout<<(a|b);
return 0;
}
```

A. 31
B. 32
C. 22
D. 32

XOR Operator (^)

If both side bit is opposite result will be On

a	b	a | b
0	0	0
0	1	1
1	0	1
1	1	0

Steps to solve:-

- **a = 12 (find binary form:1100)**
- **b = 25 (find binary form:11001)**

How to find Binary:

64	32	16	8	4	2	1	
		0	1	1	0	0	12
		1	1	0	0	1	25
		1	0	1	0	1	21

a ^ b=

01100 (12)

11001 (25)

10101 (21) Ans.

Left Shift Operator(<<)

a=10 (1010)

a<<1

1010.0

10100(20) Ans.

a<<2

1010.00

101000(40) Ans.

Right Shift Operator(>>)

a=10 (1010)

a>>1

1010.

101(5) Ans.

a<<2

1010.

10(2) Ans.

What will be output?

```
#include <iostream>
using namespace std;
int main()
{
  int a=15;
  cout<<(a>>1);
return 0;
}
```

Options:

A. 5

B. 6

C. 7

D. 8

What will be output?

```
#include <iostream>
using namespace std;
int main()
{
int a=10;
cout<<(a<<2);
return 0;
}
```

A. 10

B. 20

C. 40

What will be output?

```
#include <iostream>
using namespace std;
int main()
{
int a=40;
cout<<(a>>2);
return 0;
}
```

A. 10

B. 20

C. 30

insertion operator(<<):

The cout is used in conjunction with stream insertion operator (<<) to display the output on a console

extraction operator (>>):

The cin is used in conjunction with stream extraction operator (>>) to read the input from a console.

Control structure

- Conditional structure: if and else
- Selective structure: switch case
- Iteration structures (loops): while, do while, for
- Jump statements: break, continue, goto

Conditional structure

```
if(condition)
{
//statement
}
else
{
//statement
}
```

❖classes and objects

Real Life Example class and object:


BP-6M
NOKIA
micro SD 16GB
M2
8GB

Class

Class is a collection of similar types of objects.

For example: Fruits is class of mango, apple , orange etc.

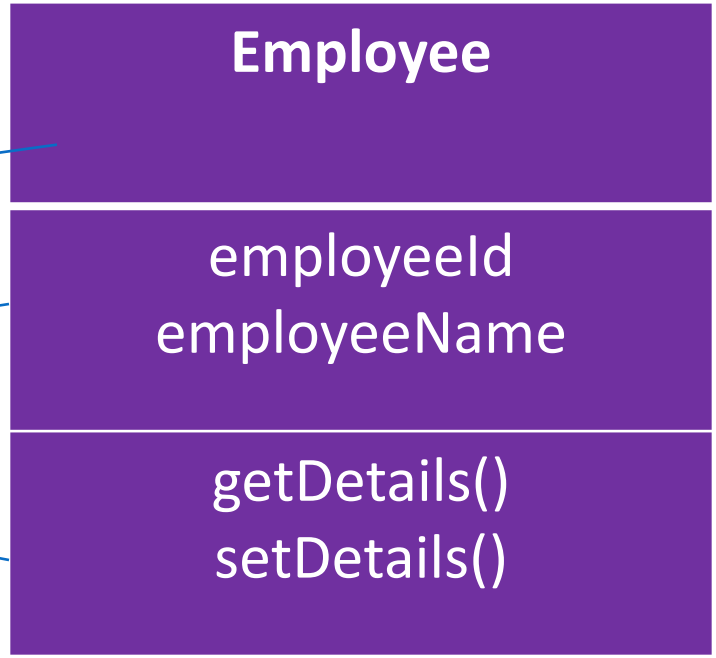
Fruits

Class

Class is a collection of data members and member functions.

Example:

```
class Employee{
//data members
// member functions
}
```



Object

Object is an instance of a Class.

```
class Employee
{
        int employeeId;
        char employeeName[20];
        public:
        void getDetails(){}
        void setDetails(){}
};
int main()
{
Employee e1; // e1 is a object
return 0;
}
```

In this code which option is correct?

```
class student
{
public:
int regno;
void getStudentDetails();
}
```

A: regno is data member

B. getStudentDetails() is member function

C. above both options are correct

❖ access specifier

- private
- public
- protected
- By default, all members of a class are private if you don't specify an access specifier.

Specifiers	within same class	in derived class	outside the class
Private	Yes	No	No
Protected	Yes	Yes	No
Public	Yes	Yes	Yes

Select correct option

```
class Employee
{
        int employeeId;
        char employeeName[20];
        public:
        void getDetails(){}
        void setDetails(){}
};
int main()
{
Employee e1;
e1. employeeId=1234
Cout<< e1. employeeId;
return 0;
}
```

A. 1234
B. 0
C. Compilation error
D. Run time error

How to achieve encapsulation features in C++?

- Define private data members in a class and
- Define public set and get accessors functions which can be used to access these private data members.

Why Encapsulation?

Increased security of data

❖constructors: types of constructors

Constructors: types of constructors

- A special method which is used to initialize the object
- It is automatically called when an object of a class is created.
- it has the same name as the class name.
- it is always public
- it does not have any return type

Types of constructor:

- Default constructor
- Parameterized constructor
- Copy constructor

this keyword

- this is a keyword that refers to the current instance of the class
- It is used to pass current object as a parameter to another method.

Copy Constructor

Copy Constructor is a type of constructor which is used to create a copy of an existing object of a class.

Syntax:

class-_name**(**class_name **& object_name)**

{

}

To call this: class_name obj1(arguments);

class_name obj2 = obj1;

❖ Destructor

- Destructor is a special member function which destructs or deletes an object.
- A destructor is called automatically when object goes out of scope.
- Destructors have same name as the class preceded by a tilde (~)
- There can only one destructor in a class
- When a class contains a pointer to memory allocated in class, we should write a destructor to release memory

Which option is correct for defining the destructor?

Option1:

```
~mobile()
      {
 cout<<"destructor called"<<endl;
      }
```

Option2:

```
 ~mobile( string str)
      {
 cout<<"destructor called"<<endl;
      }
```

A. Option1 is correct
B. Option2 is correct
C. Both option is correct

❖ Functions overloading

- same function name but different parameters.
- same function name with different signature
- example of polymorphism(**compile time)**
- overloaded functions should be there in same scope.

What will be output?

```
#include <iostream>
   using namespace std;
   void print(int i)
   {
      cout << i;
   }
   void print(double  f)
   {
      cout << f;
   }
   int main(void)
   {
      print(5);
      print(500.263);
      return 0;
   }
```

A) 5500.263
B) 500.2635
C) 500.263

❖friend function

Friend function

- It can access all private and protected member of a class
- It can be access without object of the class
- It can define out side of the class scope

Rule:

Prototypes of friend function must be declare inside the class

It can be declared either in the private or the public part.

Simple example : friend function

```
#include <iostream>
using namespace std;
class A
{

private:
   int x;
public:
   A()
   {
      x=10;
   }
private:
   friend void newfriend(A &a);
};
void newfriend(A &a)
{
   a.x=20;
   cout<<a.x;
}
int main()
{
A a1;
newfriend(a1);
return 0;
}
```

How many member functions are there in this class except constructors and destructors?

```
class Box
{
        int capacity;
  public:
        void print();
        friend void show();
        bool compare();
        friend bool lost();
};
```

a) 1
b) 2
c) 3
d) 4

- inheritance: types of inheritance

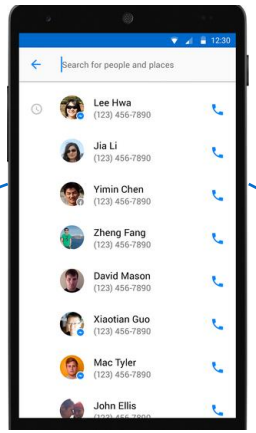
Real Life Examples Inheritance

father

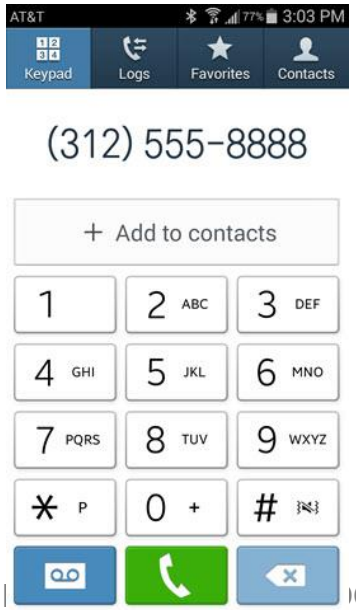
mother

child

More examples of inheritance:



Contact List

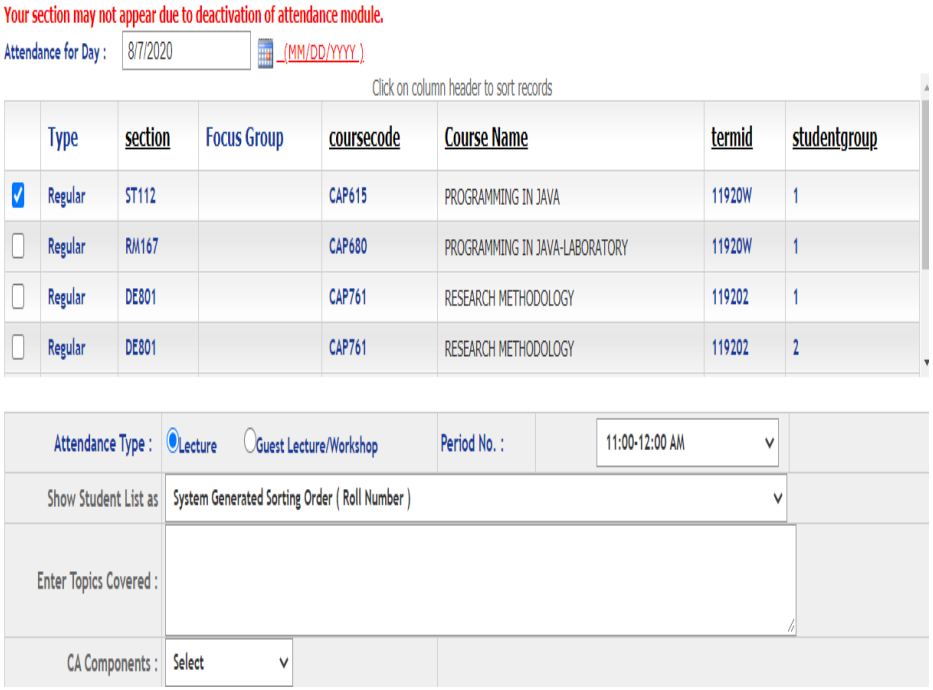


More examples of inheritance:

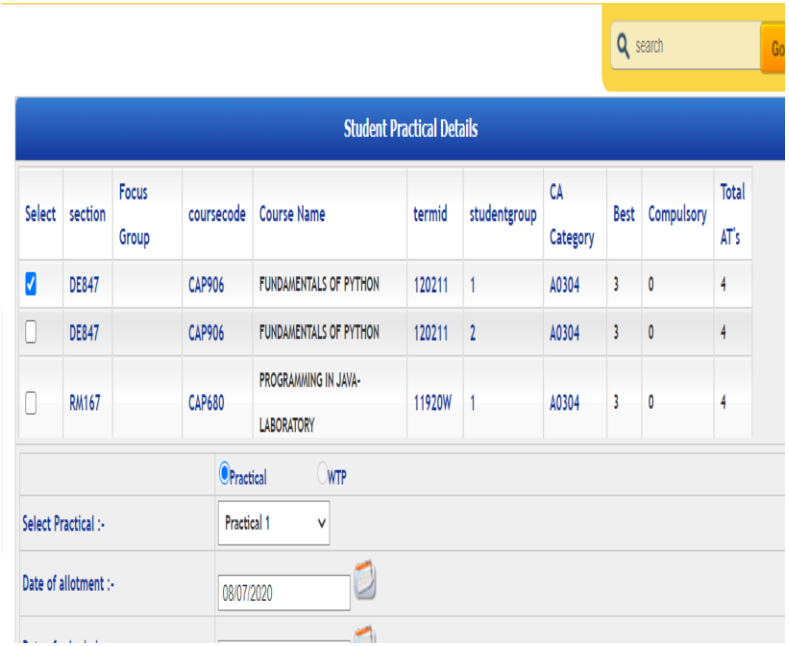
Students

RegistrationNumber	Stream
11601434	CAP - MCA
11604082	CAP - MCA
11616750	CAP - MCA
11907267	CAP - MCA
11908953	CAP - MCA
11909448	CAP - MCA
11909861	CAP - MCA
11910431	CAP - MCA
11910924	CAP - MCA
11911831	CAP - MCA
11814205	CAP - MCA
11915821	CAP - MCA
11915858	CAP - MCA
11916426	CAP - MCA
11900044	CAP - MCA
11901413	CAP - MCA
11901220	CAP - MCA

Students in Attendance Module

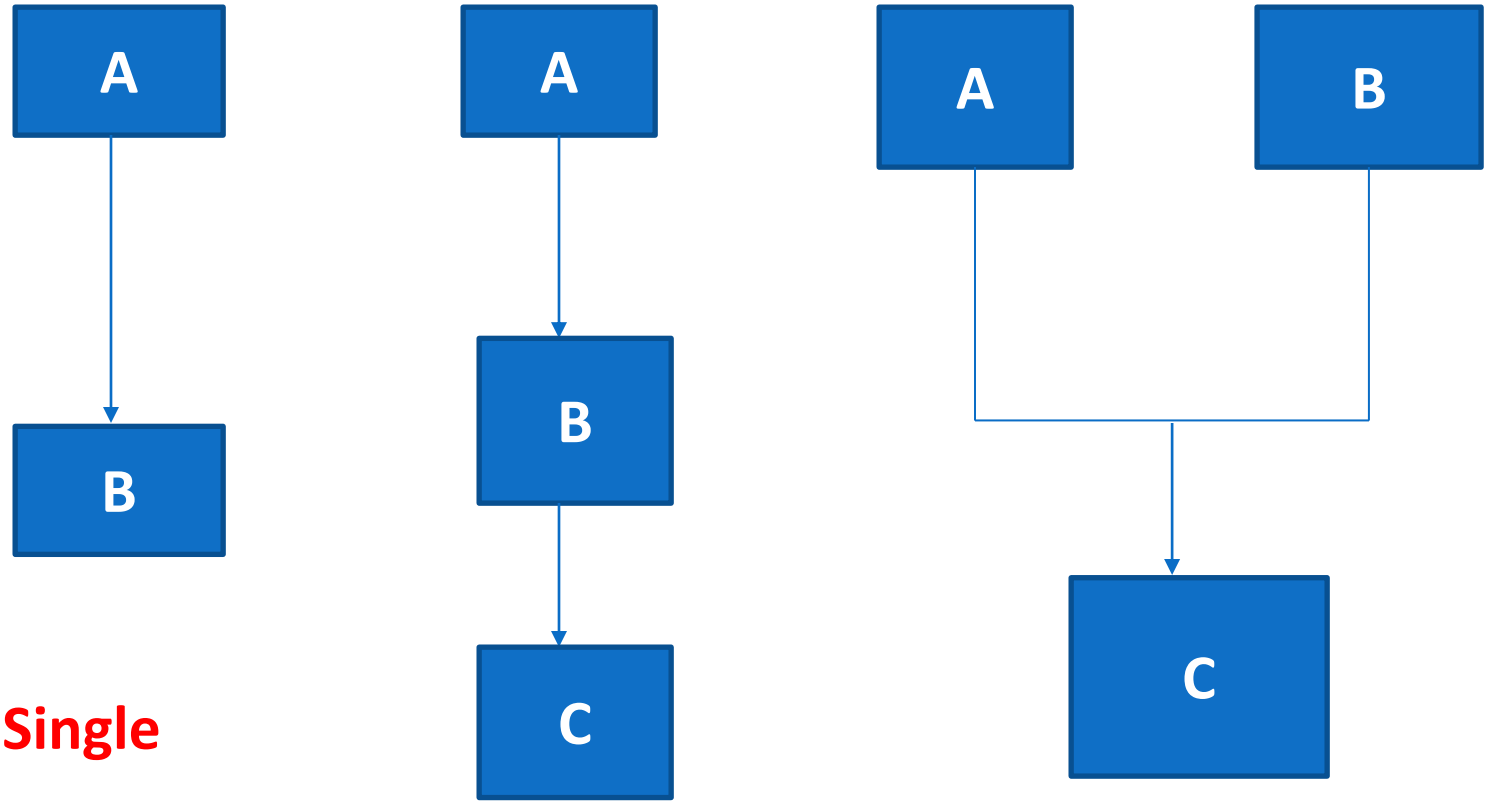


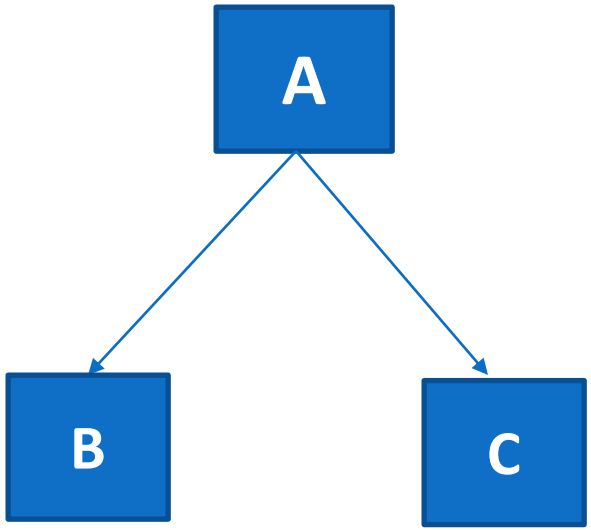
Same Students in CA Module



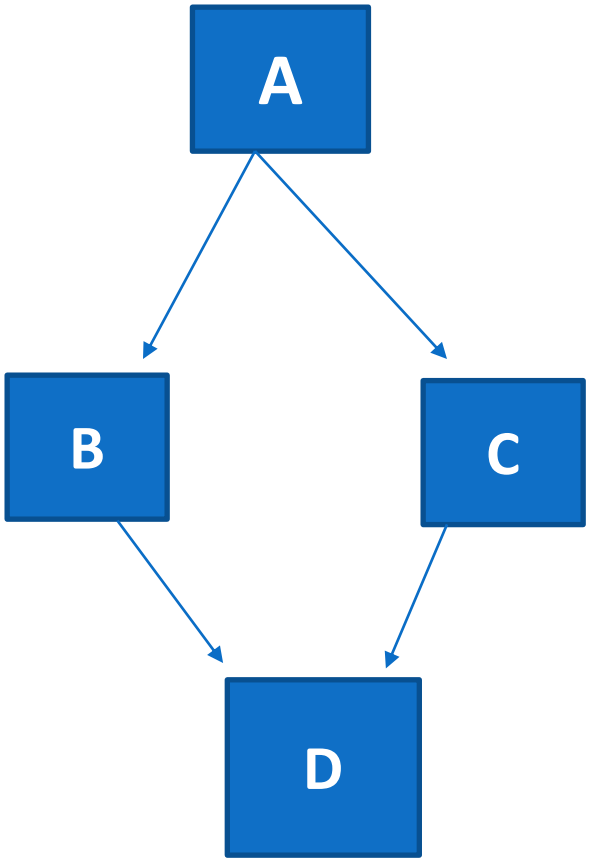
Inheritance: types of inheritance

- One class can hire properties from other class
- Advantages: Reusability





Hierarchical Inheritance



Hybrid Inheritance

Which among the following best defines single level inheritance?

A) A class inheriting a derived class

B) A class inheriting a base class

C) A class inheriting a nested class

D) A class which gets inherited by 2 classes

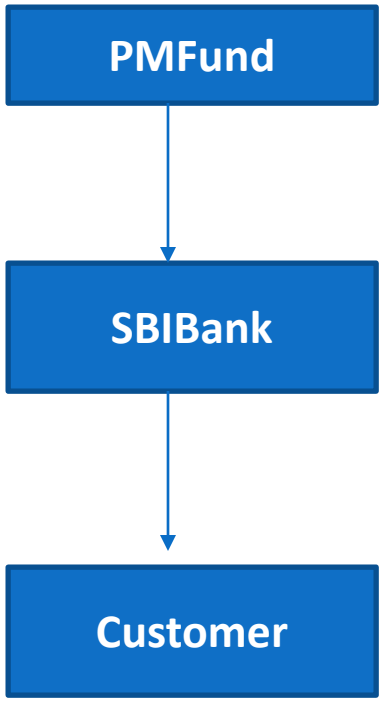
Single inheritance:

Syntax:

```
class derive_class_name : access_mode
base_class_name
{
  //body of derive_class
};
```

Problem Statement:

A person having account in SBI. His balance in bank account is 25000. He is also getting 1000 Rs. from PMFund every month. Create one application in which you have display person's total amount for every month how much you are getting. Use the concept of multilevel inheritance.



FundAmount=1250000

FundAmount=1000
BalanceAmount=25000

Total_amount=
FundAmount+BalanceAmount

Multiple Inheritance:

```
class derive_class_name : access_mode
base_class1, access_mode base_class2, ....
{
  //body of derive_class
};
```

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)

```
class base_class
{
//base class members (x, y)
};
class derive_class : access_Specifier base_class
{
//base class members (x, y)
//derive class members (a,b)
};
```

Which among the following is correct for a hierarchical inheritance?

a) Two base classes can be used to be derived into one single class

b) Two or more classes can be derived into one class

c) One base class can be derived into other two derived classes or more

d) One base class can be derived into only 2 classes

Any Query?

Unit-1 End