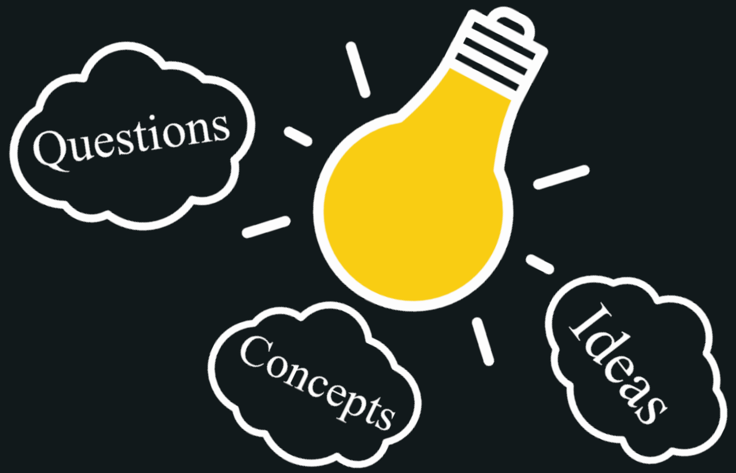


OOP's In C++

C++ Programming

VEDINESH

Why OOPs ?

1. Procedural Programming :- It is a list of instruction in a single block.

Suitable for small program .

```
#include<iostream>

Void main()
{
-----------------------------------------------

-----------------------------------------------

-----------------------------------------------

-----------------------------------------------

-----------------------------------------------

}
```

Why OOPs ?

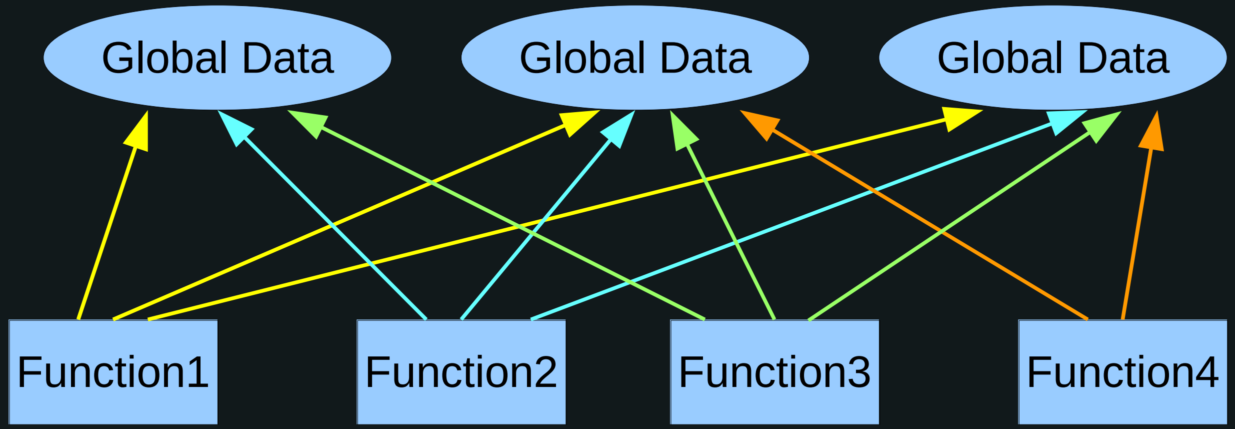
2. Modular Programming :- In this procedural program is divided into functions & each function has a clear purpose.

Suitable for large program (earlier).
(better then procedural programming)

```
#include<iostream>

Void main()
{

--------------------------------------------

          Function1( );

--------------------------------------------

          Function2( );

--------------------------------------------

}

Void Function1( )
{

 -------------------------------------------

}

Void Function2( )
{
 -------------------------------------------

}
```

Problems With Modular Programming

Data remains alive within module, so we need some data to global.



Global Data

Global Data

Global Data

Function1

Function2

Function3

Function4

ATM application

withdrawal( )

check_balance( )

gen. PIN( )

mini stmt( )

Balance

PIN

name

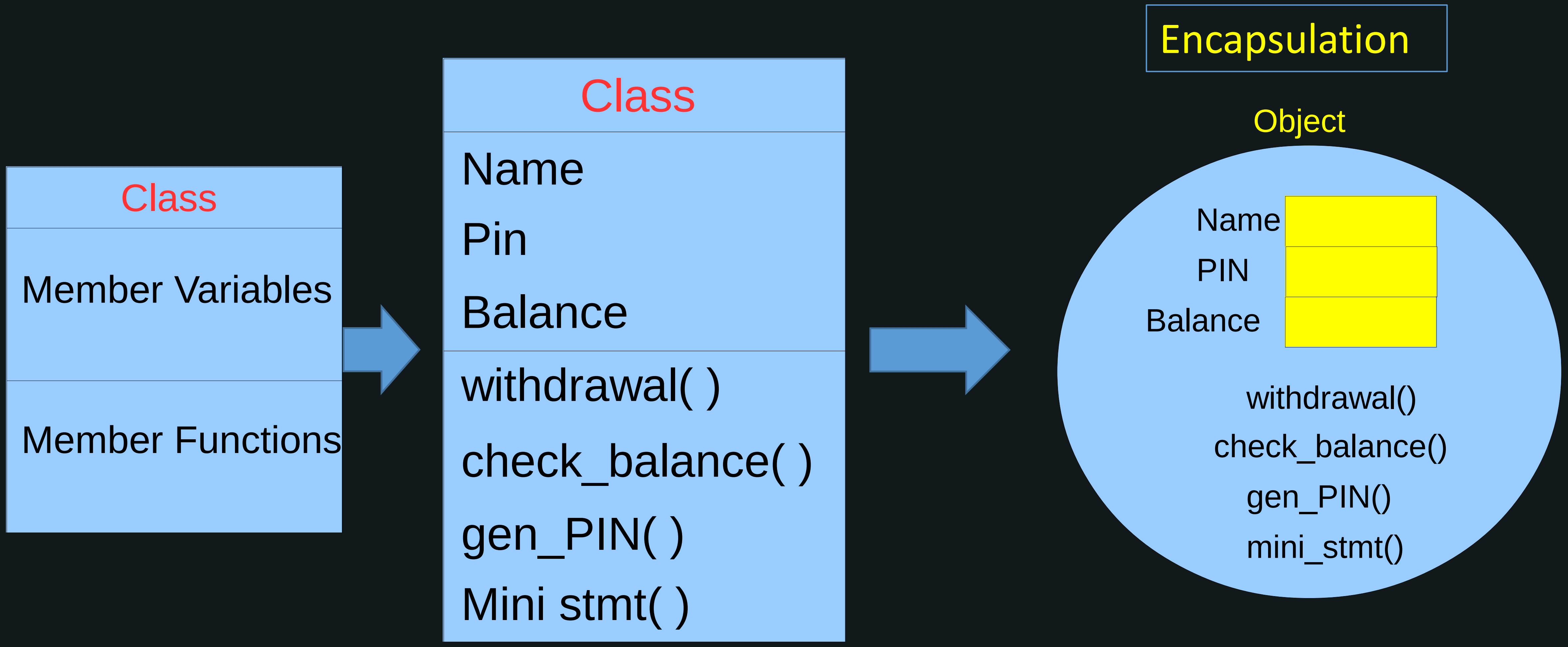
In Large project :-

> Difficult to conceptualise.

> Difficult to modify.

Problem :- Data And Functions are seperate

Object Oriented Approach



Encapsulation

Class

Member Variables

Member Functions

Class

Name

Pin

Balance

withdrawal( )

check_balance( )

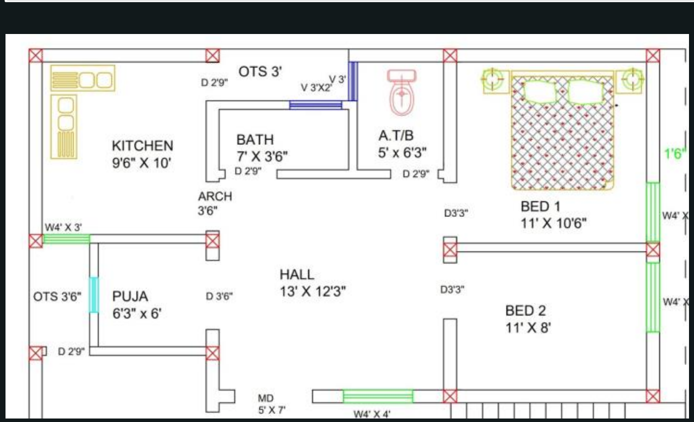
gen_PIN( )

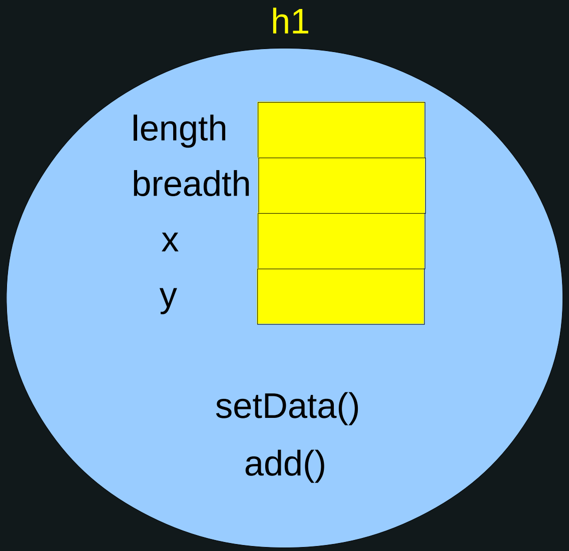
Mini stmt( )

Object

While creating object ---- > compiler will refer to class for (Memory Allocation)

OOP's Example

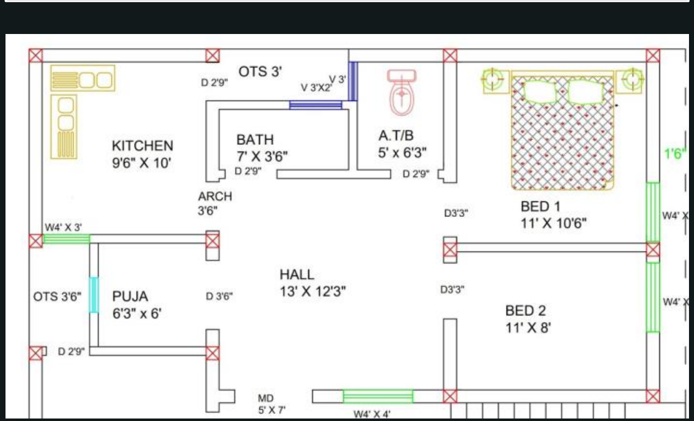


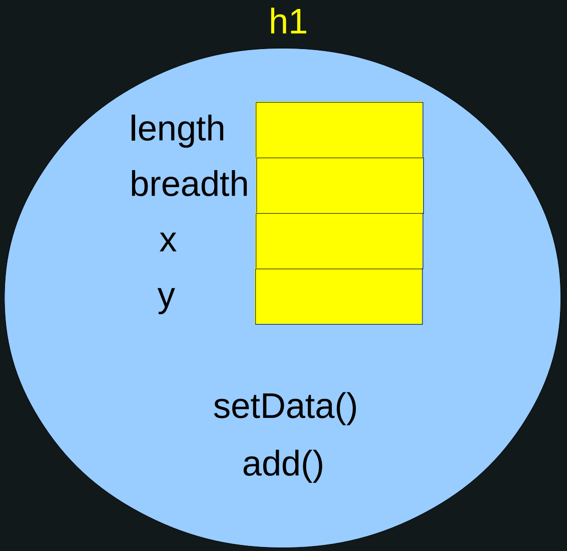


```
class house
{
  int length, breadth;                // member variable

  void setData(int x , int y)         // member function
   { length = x;   breadth = y; }

  void  area ()
   { cout << length * breadth ; }
}
```

```
void main ()
{

  house h1;          // memory allocated

  h1.setData( 500, 600 );
  h1.area();

 }
```

Defination :-





h1

length

breadth

x

y

setData()

add()

Class :- A class is the building block or blue print of the instance/object.

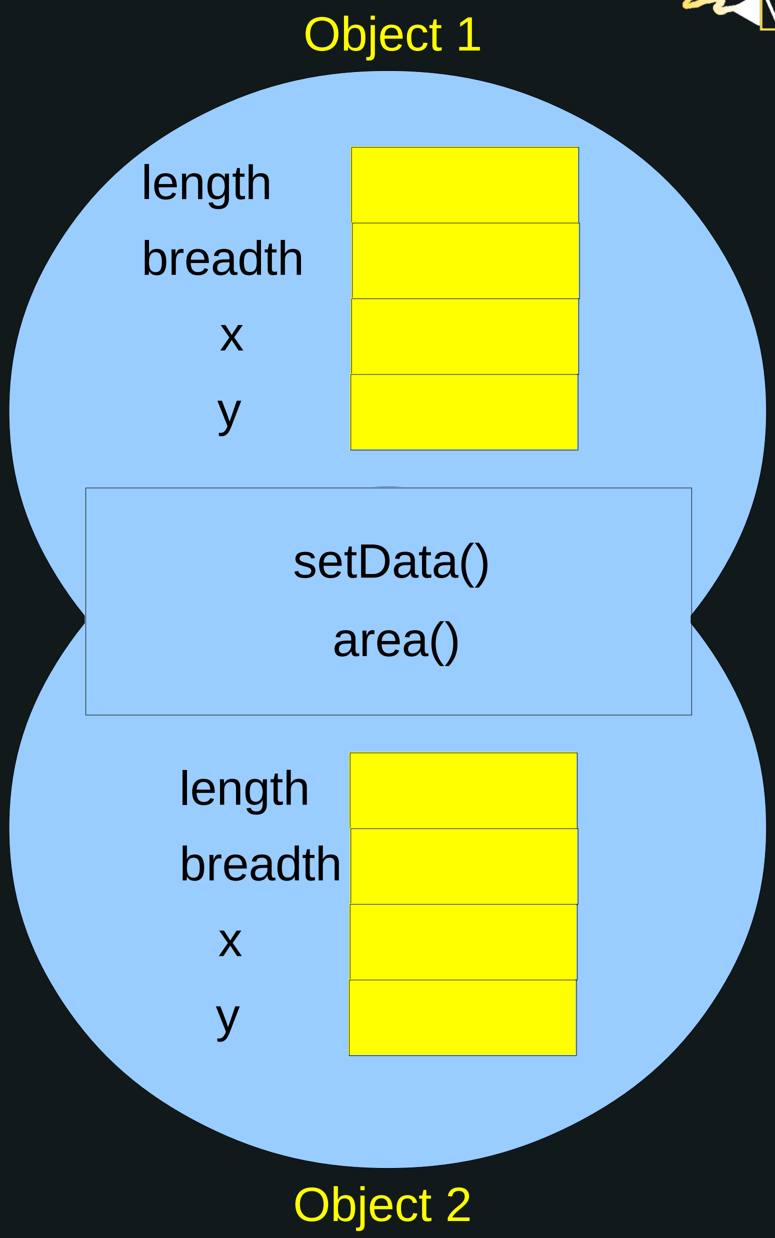
Class is user defined datatype, which holds its own member variables and member functions, that can be accessed and used by creating an instance of the class.

Object is an instance of class, when created memory is allocated to member variables and member functions .

Key Note

```
class house
{
  int length, breadth;                  // member variable

  void setData(int x , int y)           // member function
   { length= x;   breadth = y; }
  void  area()
   { cout << length*breadth; }
}
```

```
void main ()
{
house h1, h2;           // memory allocated
  h1.setData( 5, 6 );
  h1.area( );
------------------------------------------------
 h2.setData( 7, 1 );
 h2.area( );
 }
```



Object 1

length

breadth

x

y

setData()

area()

length

breadth

x

y

Object 2

Access specifiers

```
class house
vate:
nt length, b                                  ber variable
blic:
oid setData                                   ber function
 length= x;
oid  area ()
 cout << le
```

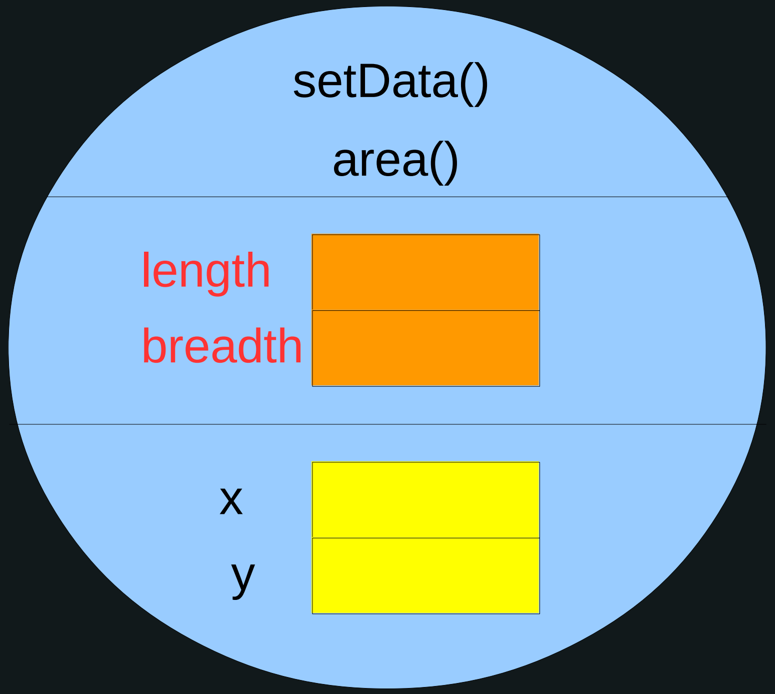
Class 1	Class 2	other
private : int x	private : Can't access	private : Can't access
protected : int y	protected y = 10;	protected Can't access
public : int z	public z = 20;	public z = 20;

Data Hiding with Access specifiers

```
class house
{
private:
   int length, breadth;              // member variable
 public:
   void setData(int x , int y)     // member function
    { length= x;   breadth= y; }
   void  area ()
    { cout << length*breadth; }
}
```

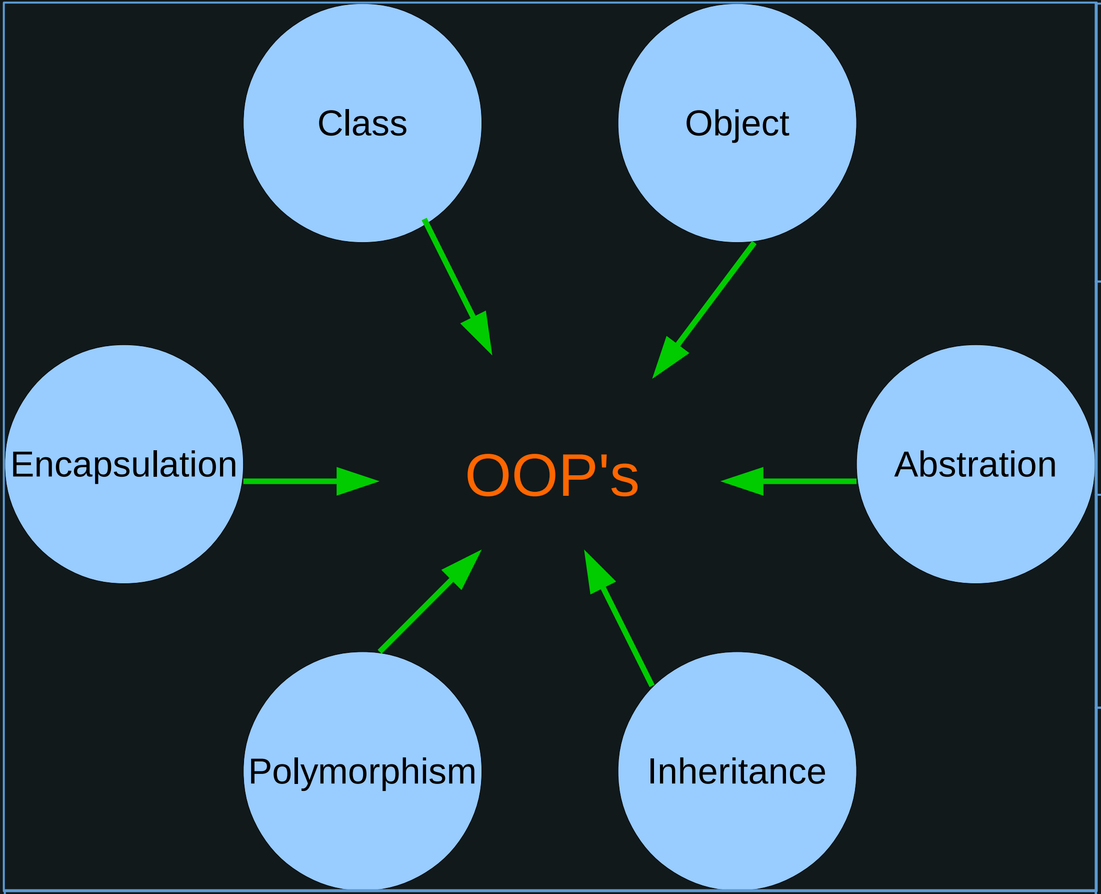
```
void main ()
{
    house h1, h2;        // memory allocated
    h1.setData( 5, 6 );
    h1.add( );
 }
```

```
h1.a = 3;              ( incorrect )
h1.setData(5,6)   (correct )
```



Object

Characteristics Of OOP's



-> Abstraction:- means hiding complicated things from the user.

-> Class is a blueprint and Object is instance of class.

-> Class is a user-defined data type, which holds its own data members and member functions.

-> Helps in code reusability.

-> Encapsulation wraping up variables and methods in class.

-> It help in data hiding.

-> Polymorphism means having many forms

-> In class method may behave differently, depending on the inputs. function overloading

-> Inheritance means property of a child class to inherit characteristic of parent class.

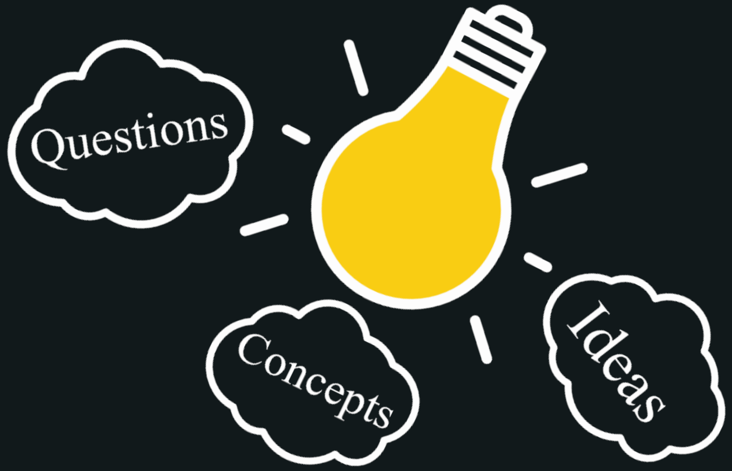
like :-

Dog, Cat , Cow Class Inherit from Animal Class.

Mini Project (ATM)

Write a program showing ATM functionalities using OOP's

1. Check Balance
2. Cash WithDraw
3. User Details
4. Update Mobile No.

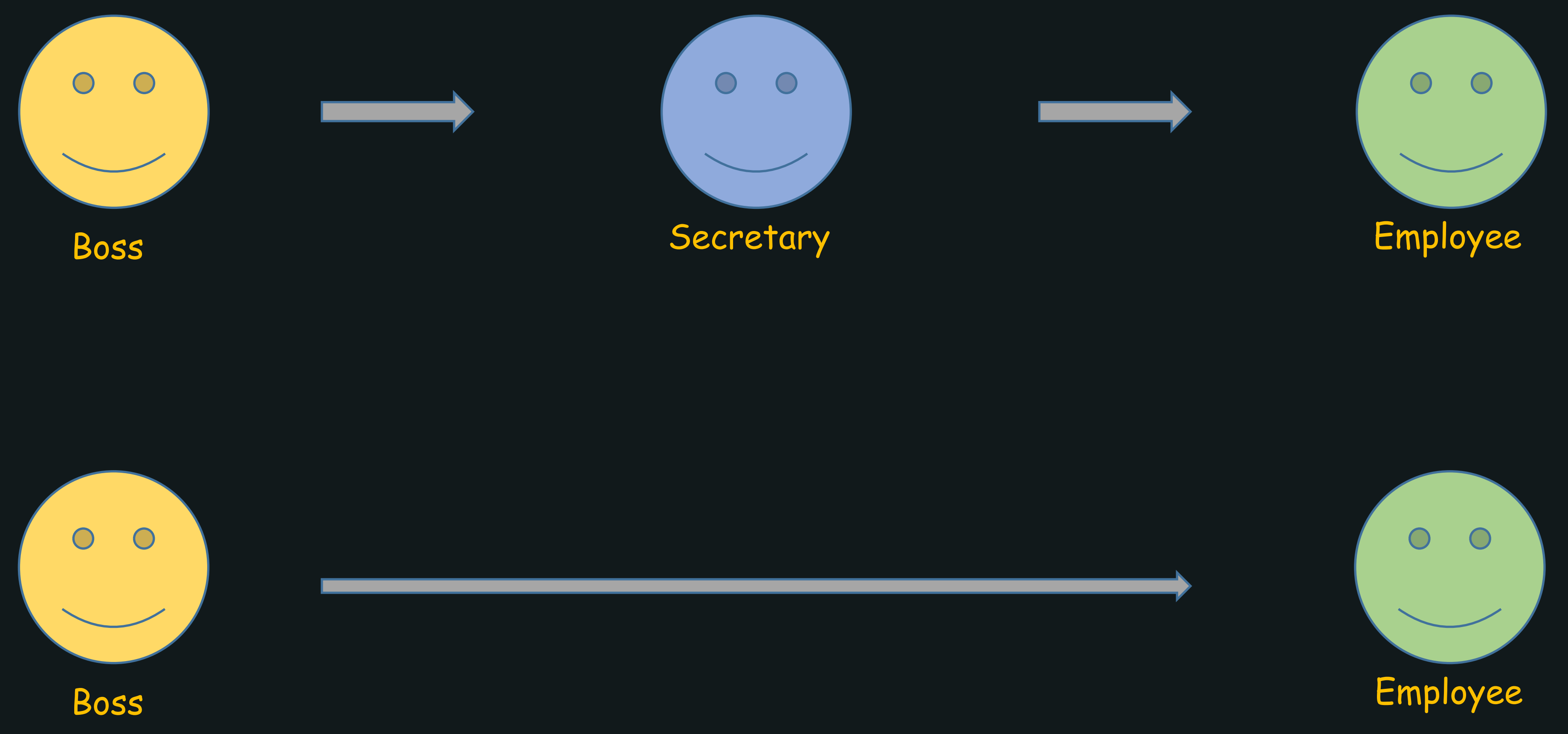


Constructors

C++ Programming

VEDINESH

Constructors



Constructors

```
#include<iostream>
 class A
{ private:
   int age;
   public:
   ---------------------------------------------------
   void setData( int x = 0 )
   { age = x; }
   ---------------------------------------------------
   int getData( )
   { return age ; }
  ---------------------------------------------------
}
```

```
void main ( )
{
  A a_obj ;
  a_obj.setData ( 28 );
  cout << a_obj.getData ( ) ;
}
```

```
#include<iostream>
 class A
{ private:
   int age;
   public:
   ---------------------------------------------------
   A ( int x  )          // constructor
   { age = x;  }       // same name as class & don't return
    ---------------------------------------------------
    int getData( )
   { return age ; }
}
```

```
void main ( )
{
  A a_obj ( 28 );
  cout << a_obj.getData ( ) ;
}
```

Constructors

Why :-

> Programmer may forget to initialize data members in object after creating it.

> When there are many objects, then it would be tedious job.

> Initialize & Allocate memory to Data Members .

Rules :-

> Same Name As Class Name.

> No Return Type.

Constructor Types

> Non - Parametrized Constructor. or > Default Constructor.

> Parametrized Constructor.

> Copy Constructor.

Non-Parametrized Constructor

> Constructor that does not take any argument.

```
#include<iostream>
 class A
{ private:
   int age;
   public:
   ----------------------------------------------------
   A (  )                   // Non Parametrized constructor
   { age = 0; }       // same name as class & don't return anything
   ----------------------------------------------------
     int getData( )
   { return age ; }
}
```

```
void main ( )
{
  A a_obj ;
  cout << a_obj.getData ( ) ;
}
```

Parametrized Constructor

> Constructor that take some argument.

```
#include<iostream>
 class A
{ private:
   int age;
   public:
   -----------------------------------------------------
   A ( int x )                    // Parametrized constructor
   { age = x; }       // same name as class & don't return anything
   -----------------------------------------------------
     int getData( )
   { return age ; }
}
```

```
void main ( )
{
  A a_obj ( 28 );
  cout << a_obj.getData ( ) ;
}
```

Copy Constructor

> Copy Constructor are used for creating new Object from existing object.

Copy Constructor

```
void main ( )
{
  A a_obj1 ( 28 );                    // Parametrized Constructor

 A a_obj2 ( a_obj1 ) ;              // Copy Constructor

  cout << a_obj2.getData ( ) ;
}
```

```
#include<iostream>
 class A
{ private:
   int age;
   public:
   ---------------------------------------------------
   A ( int x )                    // Parametrized constructor
   { age = x; }
   ---------------------------------------------------
  A ( A &a_obj1 )              // Copy constructor

   {  age = a_obj1.age;  }
  ----------------------------------------------------
     int getData( )
   { return age ; }
}
```

Overloaded Constructor

```
#include<iostream>
 class A
{ private:
   int age;
   public:
-------------------------------------------------------------------
  A (  )                         // Non Parametrized constructor
  { age = 0 }
   ------------------------------------------------------------------
   A ( int x )                   // Parametrized constructor
   { age = x; }
   ------------------------------------------------------------------
  A ( A &a_obj1 )                // Copy constructor

   {  age = a_obj1.age;  }
  ------------------------------------------------------------------
     int getData( )
   { return age ; }
}
```

```
#include<iostream>
 class A
{ private:
   int age;
   public:
   ------------------------------------------------------------------
   A ( int x = 0 )              // Parametrized constructor
   { age = x; }
   ------------------------------------------------------------------
  A ( A &a_obj1 )               // Copy constructor

   {  age = a_obj1.age;  }
  ------------------------------------------------------------------
     int getData( )
   { return age ; }
}
```

Program

Write a program, take Phone details as input and store them in object & use Constructors.

Phone Details :-

1. Name
2. RAM
3. Processor
4. Batter