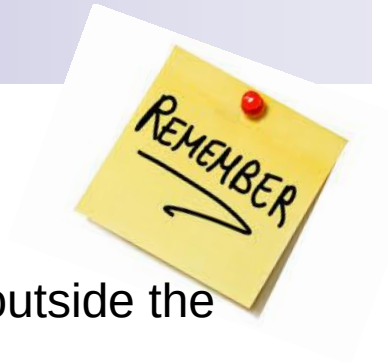
Java™ Education & Technology Services

Object Oriented programming Using

C++

Class



- **Visibility in class:**
  - All members are private by default.
  - Public: its access is available from anywhere inside or outside the class.
  - Private: the access availability within the class only.

  - Make the attributes members private. (90%)
  - Make the functions members public. (70%)

```
class Test() {
        int a;
    public:
        void setA( int x);
}
```

```
main() {
        Test obj;
        obj.setA( 10 );
        obj.a = 10;
}
```

Default Arguments in function calls

```
int sum ( int x, int y, int z, int m) {
int s;
s = x + y + z + m;
return s;
}
main () {
      sum (1, 2, 3, 4) ;          //…….1
      sum (1, 2, 3);              //……..2
      sum (1, 2);                 //……3
// compiler error function not found at lines 2 and 3
}
```

Default Arguments in function calls

```
int sum ( int x, int y, int z, int m = 0) {
int s;
s = x + y + z + m;
return s;
}
main () {
    sum (1, 2, 3, 4) ;          //…….1
    sum (1, 2, 3);              //……..2
    sum (1, 2);                 //……..3
    // compiler error function not found at line 3
}
```

Default Arguments in function calls

```
int sum ( int x, int y, int z =0, int m = 0) {
int s;
s = x + y + z + m;
return s;
}
main () {
      sum (1, 2, 3, 4) ;          //…….1
      sum (1, 2, 3);              //……..2
      sum (1, 2);                 //……..3
      sum (1, 2,     , 4);        //……..4
}
```

Default Arguments in function calls

- put less used argument at the right side of the function header.
- If one argument has default value , all of the next argument must have also default values.
- **Some Cases :**
  - int sum (int x = 0, int y) ;

  - int sum (int x, int y = 0) ;
  - int sum (int x) ;

Polymorphism [Overloading Functions]

- Function Overloading :
  - Many functions in one class with the same name but with different function signatures 

```
class Complex
{
     :
     :
     void setComplex(int r, int i){
          real=r;
          img=i;
     }



};
```

```
int main() {
     Complex c;

     c.setComplex(3,5);

     c.setComplex(5);
     return 0;
}
```

Polymorphism [Overloading Functions]

- Function Overloading :
  - Many functions in one class with the same name but with different function signatures

```
class Complex
{
    :
    :
    void setComplex(int r, int i=0){
        real=r;
        img=i;
    }

    void setComplex(int v){
        real=img=v;
    }
};
```

Ambiguity Error ☹

```
int main() {
    Complex c;

    c.setComplex(3,5);

    c.setComplex(5);
    return 0;
}
```

Constructor

– Constructor:

- if you want to give the object's attributes initial values through the object creation, use constructor which is a special function that:

1. Has the same name of the class
2. Doesn't return value [void is a value]
3. User can't call it
4. Call only when creating the objects.
5. Can be overloaded.

```
int main() {
    Complex c;
    c.print();
```

```
class Complex
{
    public:

    Complex()
    {
        real = 0 ;
        imag = 0 ;

    }
```

Constructor

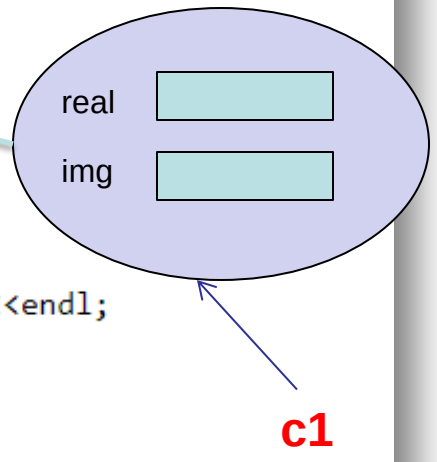
– Constructor:

```
class Complex
{
    public:

    Complex()
    {
        real = 0 ;
        imag = 0 ;
        cout<<"This is the default constructor"<<endl;
    }

    Complex(float n)
    {
        real = imag = n ;
        cout<<"This is the overloaded constructor, with one parameter"<<endl;
    }
    Complex(float r, float i)
    {
        real =r;
        imag = i ;
        cout<<"This is the overloaded constructor, with two parameter"<<endl;
    }
```

```
int main()
{
    Complex c1; // Complex c1();
    Complex c2(3);
    Complex c3(3,5);
```



Object Creation : 1) Allocate the Memory 2) call constructor method

Constructor

- Constructor:
  - If there is **No** constructor , a default one is made which take **No** parameters and has an **empty** body.

```
class Complex
{
    public:
```

```
int main()
{
    Complex c1; // Complex c1();
    Complex c2(3);
    Complex c3(3,5);
```

Constructor

- Constructor:
  - If a constructor is defined as a **private** , **No** objects can be created from the class by it.

```
class Complex
{
    Complex(float n)
    {
        real = imag = n ;
        cout<<"This is the overloaded constructor, with
    }
    public:

    Complex()
    {
        real = 0 ;
        imag = 0 ;
        cout<<"This is the default constructor"<<endl;
    }


    Complex(float r, float i)
    {
        real =r;
        imag = i ;
        cout<<"This is the overloaded constructor, with two parameter"<<endl;
    }

};
```

```
int main()
{
    Complex c1; // Complex c1();
    Complex c2(3);
    Complex c3(3,5);
```

Constructor

- Constructor:
  - If **No** default constructor made and there are others constructor it.

```
class Complex
{

    public:

    Complex(float r, float i)
    {
        real =r;
        imag = i ;
        cout<<"This is the overloaded constructor, with two parameter"<<endl;
    }

};
```

```
int main()
{
    Complex c1; // Cor        c1();
    Complex c2(3);
    Complex c3(3,5);
```

Destructor

- Destructor:
  - called when object destroyed. it is a special function that:

1. Has the same name of the class with ~
2. Doesn't return value [void is a value]
3. Doesn't take any parameters [can not be overloaded]
4. User can't call it

```
class Complex
{
    public:


    ~Complex()
    {

        cout<<"the object destructor"<<endl;
    }
```

```
int main()
{
    Complex c1;
    Complex c2(3);
    Complex c3(3,5);
```

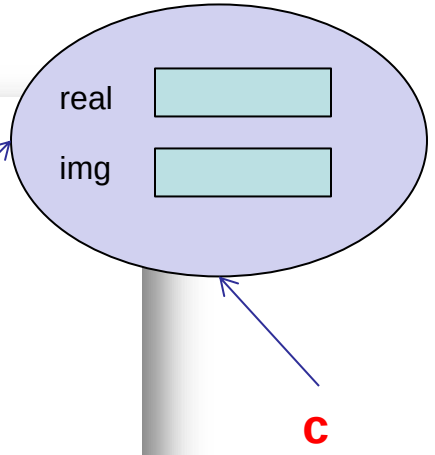
“this” Pointer

– “this” pointer:

- When a member function is called by an object, there is implicitly a pointer to the caller object send to the function. This pointer is called “this”.

```
class Complex
{
    public:

     :
     :
    void setComplex(int r, int i){ // 3 parameters this, r and i
        real=r; // this -> real=r;
        img=i;
    }
```



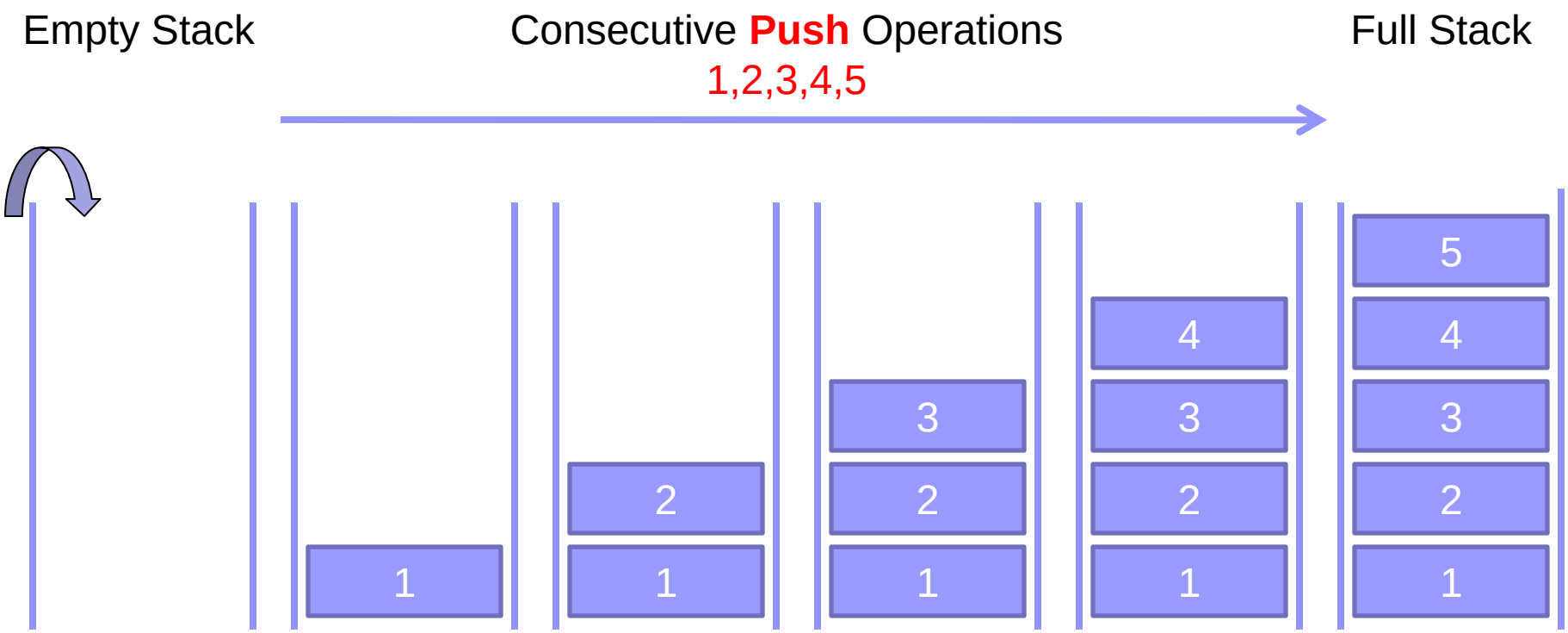

```
int main() {
    Complex c;

    c.setComplex(3,5); // this is send which is = &c
```

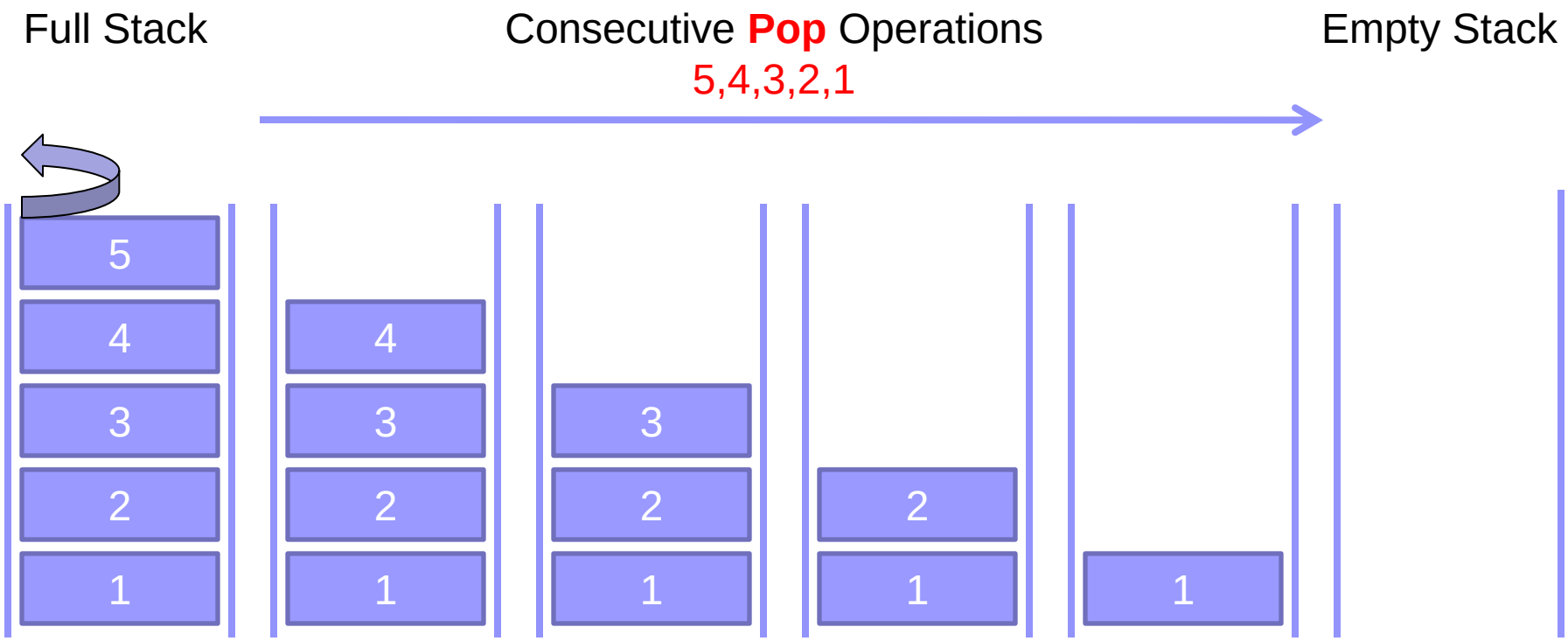
Stack Example

- Create A class which describe stack of integers
- Handle stack Push and Pop functions
  - LIFO- Last In First Out

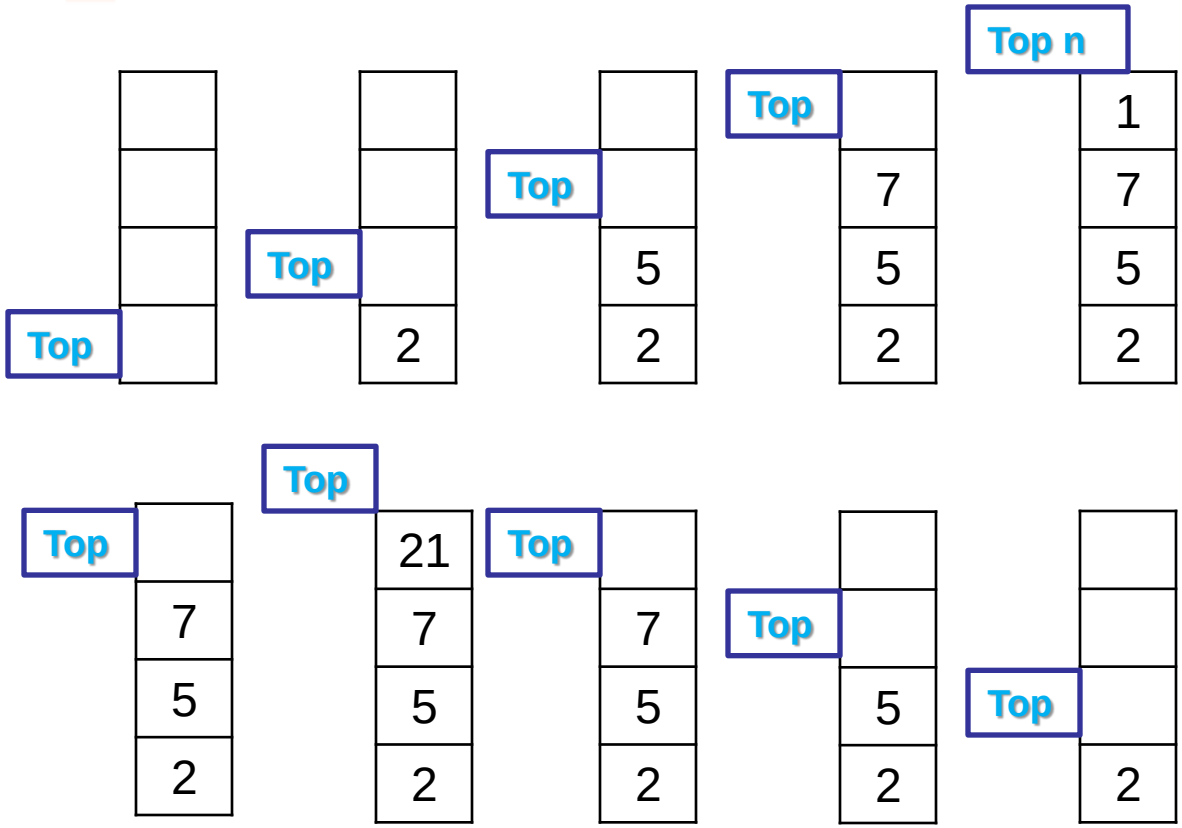
Stack Example



Stack Example



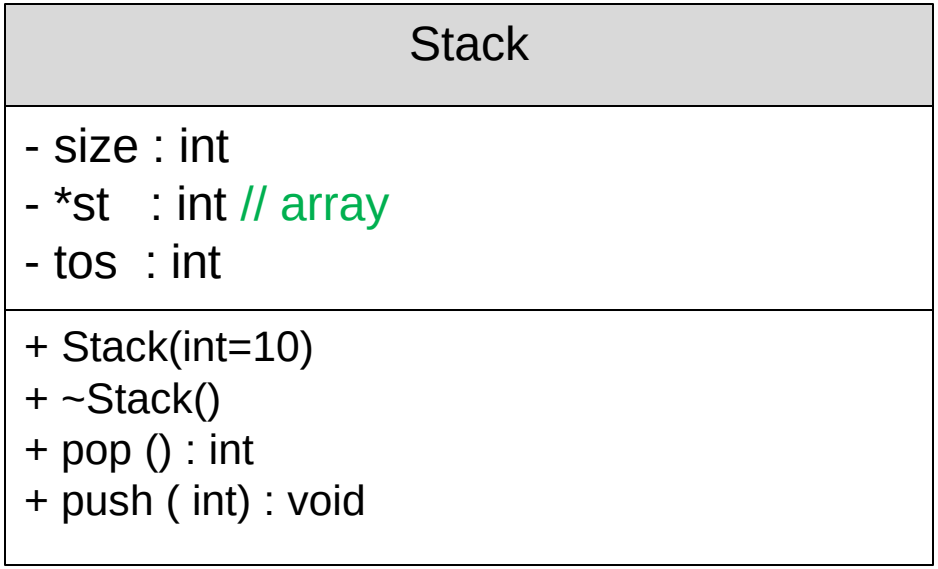
Stack Example



- Push (2)
- Push (5)
- Push (7)
- Push (1)
- Push (3) // stack is full 3 not added
- Pop()→ 1
- Push (21)
- Pop()→21
- Pop()→7
- Pop()→5
- Pop()→2
- Pop()→ // stack is empty

Stack Example

- UML



Stack Example

```
class Stack
{
  private:
    int tos ;
    int size;
    int *st;

  public:

    Stack(int n=10)
    {
        tos = 0;
        size = n;
        st = new int[size];
        cout<<"This is constructor of stack object with size "<<n<<endl;
    }

    ~Stack()
    {
        delete[] ptr;
        cout<<"This is the destructor " <<endl;
    }

    void push(int);
    int pop();
};
```

Stack Example

```
void Stack::push(int n)
{
    if (tos==size)
    {
        cout <<"Stack is full"<<endl;
    }
    else
    {
        st[top] = n;
        tos++;
    }
}
```

```
int Stack::pop()
{
    int retVal;

    if (tos==0)
    {
        cout <<"Stack is Empty"<<endl;
        retVal = -1 ;
    }
    else
    {
        tos--;
        retVal = st[top];
    }
    return retVal;
}
```

Stack Example

```
int main()
{
    Stack s1(2) ;

    s1.push(5);
    s1.push(14);
    s1.push(20) ;
    cout<<s1.pop() ;

    Stack s2() ;

    s2.push(3);
```

- Push (2)
- Push (5)
- Push (7)
- Push (1)
- Push (3) // stack is full 3 not added
- Pop()→ 1
- Push (21)
- Pop()→21
- Pop()→7
- Pop()→5
- Pop()→2
- Pop()→ // stack is empty

Write a main for that

Static Members

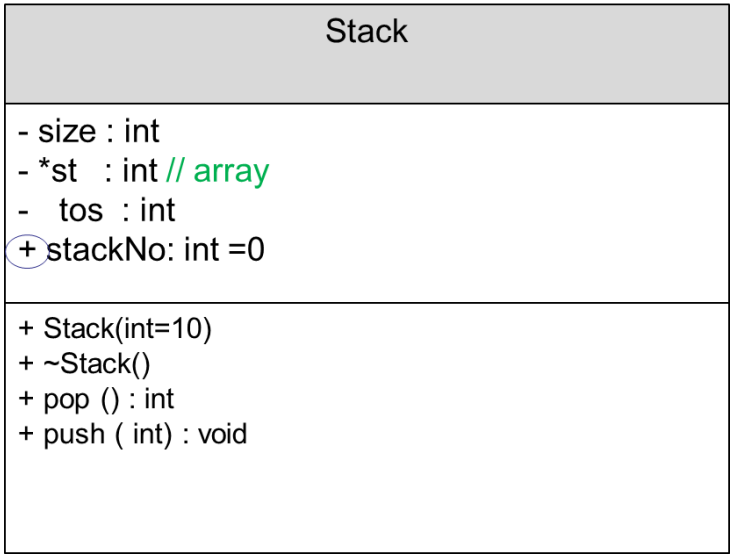
- Static Members:
  - Static Attributes [class attributes]:

1. shared attributes among all objects from this class.
2. its lifetime is the application lifetime
3. can be accessed from class name or any object [if it public]
4. has constant value.
5. must be initialized
6. can be private or public.
7. can be used in all class functions

```
cout << Stack :: stackNo ;

Stack s1;

s1.stackNo;
```



Stack Example

- use static variable to count the created stack objects

```
class Stack
{
  private:
    :
    :
    static int counter ;

  public:

    Stack(int n=10)
    {
        :
        counter++;
        cout << "this is stack object No." << counter;
    }

    ~Stack()
    {
        :
        counter--;
        cout << "will destroy stack object No." << counter
    }

   :
};
int Stack::counter = 0 ;
```

```
cout << Stack :: counter;

Stack s1;

s1.counter;
```

Static Members

- Static Members:
  - Static Functions[class Functions]:

1. deal only with static attributes.
2. “this” is not send to it
3. call by the class name or any object

Stack Example

```
class Stack
{
  private:
    :
    :
    static int counter ;

  public:

    Stack(int n=10)
    {
        :
        counter++;
        cout << "this is stack object No." << counter;
    }

    ~Stack()
    {
        :
        counter--;
        cout << "will destroy stack object No." << counter
    }
    static int getCounter()
    {
        return counter ;
    }

   :
};
int Stack::counter = 0 ;
```

```
int main()
{
    cout<< Stack::getCounter();

    Stack s1(2) ;
    cout<< s1.getCounter();
    :
    :

    Stack s2() ;
    cout<< Stack::getCounter();
    :
    :
return 0;
}
```

Lab Exercise

Lab Exercise

- **1st Assignment :**

1. **Complete Complex Class:**
    1. Constructors
    2. Destructor
    3. setComplex functions.
2. **Stack Class:**
    1. Constructors
    2. Destructor
    3. Static members
    4. Push & Pop
3. **B+ Stack Class with simple High lighted Menu:**
    1. Take stack size from user first
    2. Show & handle these menu options

Push

Pop

PrintStack