Sai Inheritance III

Press Esc to exit full screen

Constructors and Destructors

- Two Important Questions!!!
  - When are base class and derived class constructors are called?
    - Base class, or derived class or both can contain constructors and destructors
    - When an object of the derived class comes into existence,
      - It is important to understand the order of execution of these functions
  - How can we pass the parameters to the base class constructors?

6:38 / 17:01

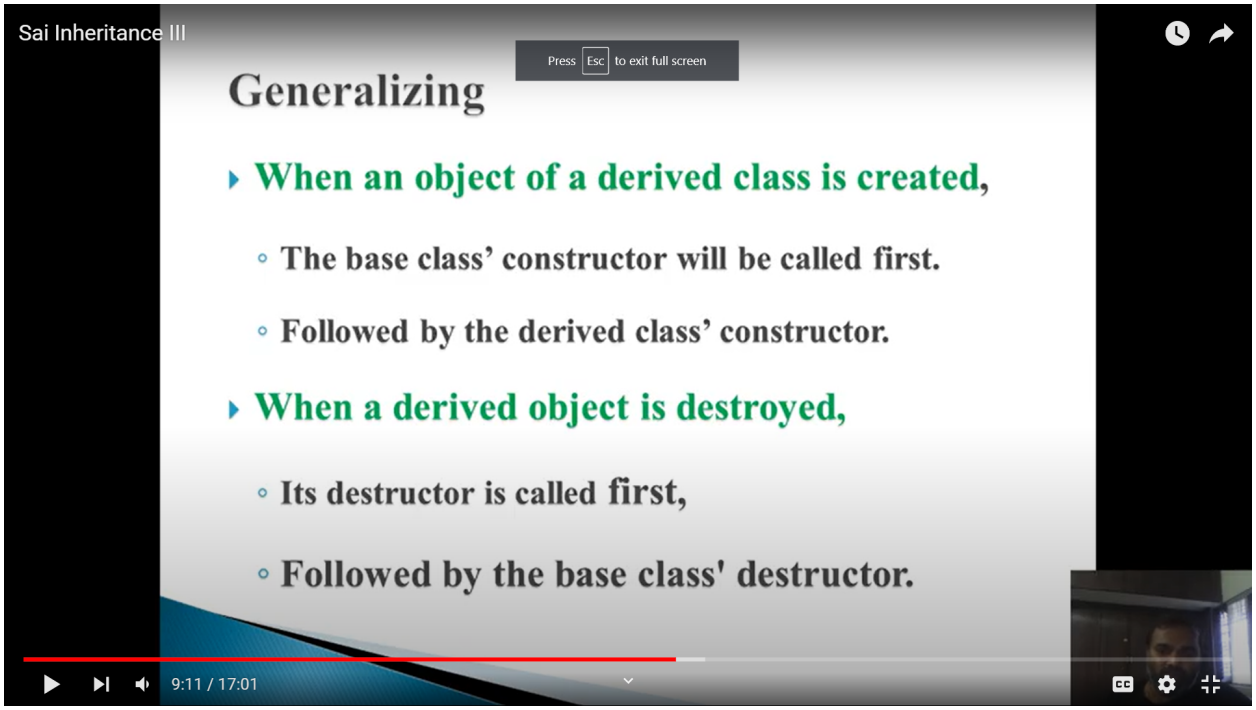
Sai Inheritance III

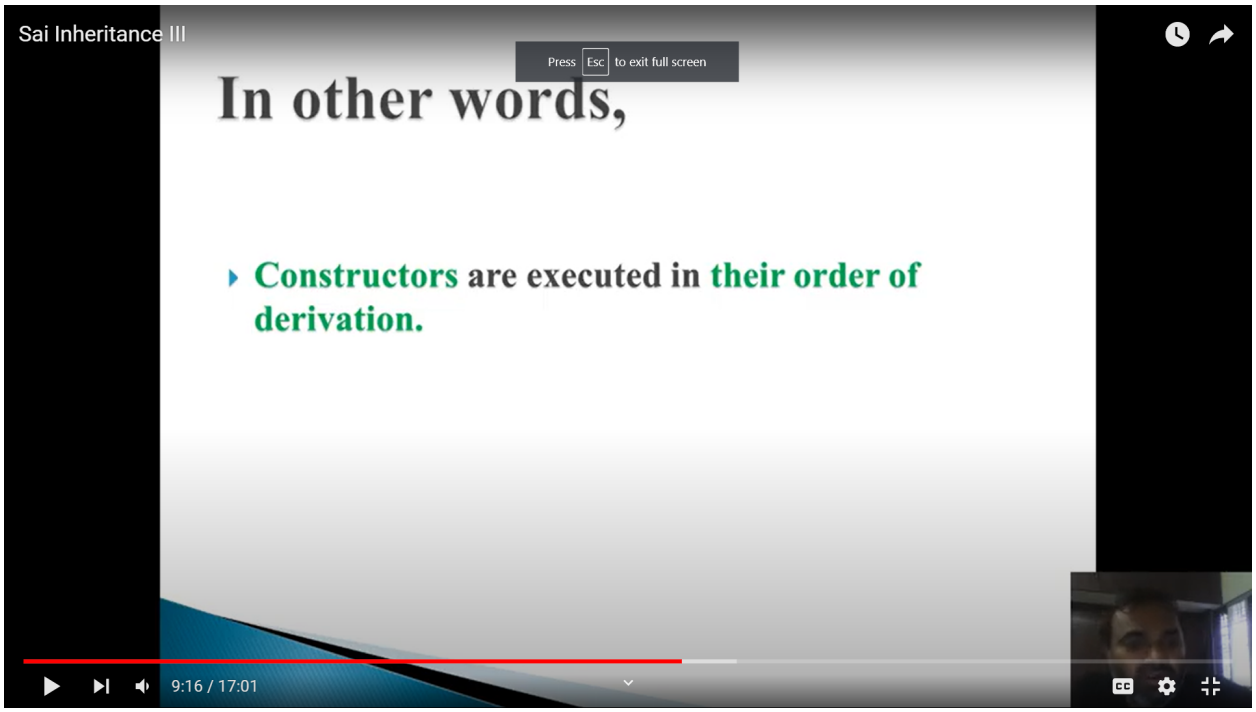
Press Esc to exit full screen

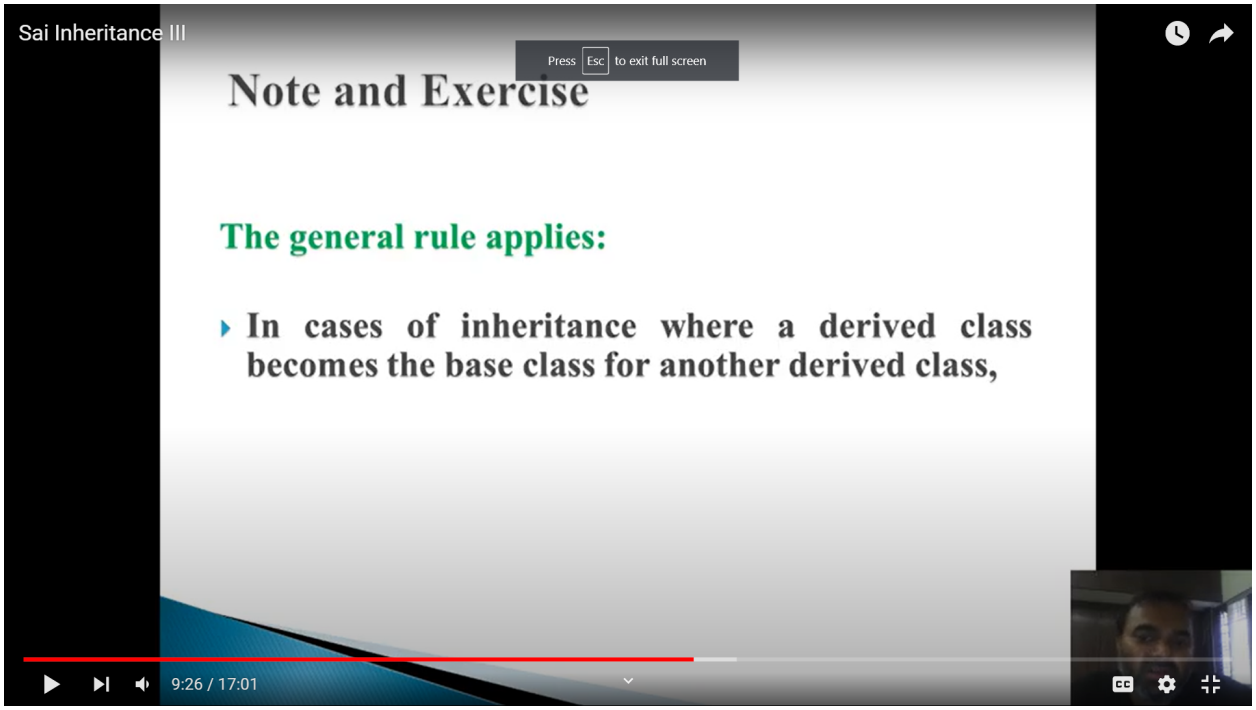
```
#include <iostream>
using namespace std;

class base {
public:
  base() { cout << "Constructing base\n"; }
  ~base() { cout << "Destructing base\n"; }
};

class derived: public base {
public:
  derived() { cout << "Constructing derived\n"; }
  ~derived() { cout << "Destructing derived\n"; }
};

int main()
{
  derived ob;

  // do nothing but construct and destruct ob

  return 0;
}
```

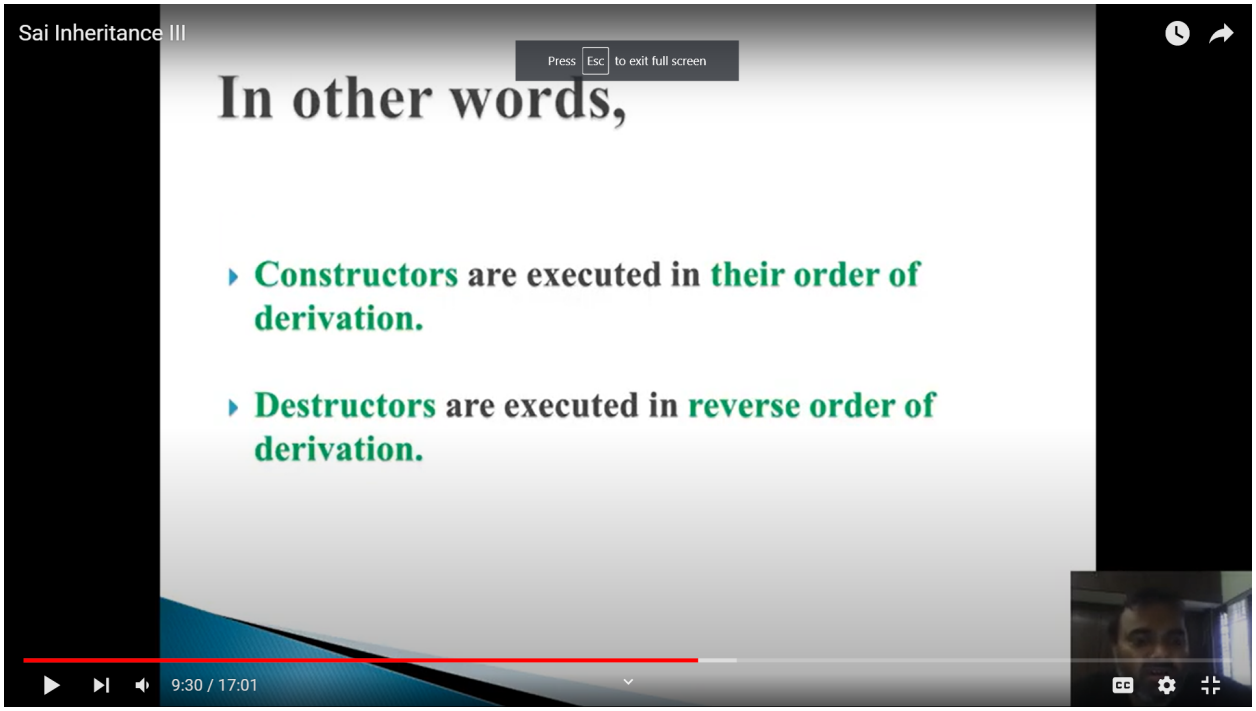
```
Constructing base
Constructing derived
Destructing derived
Destructing base
```

7:56 / 17:01


Sai Inheritance III
Press Esc to exit full screen
Generalizing
When an object of a derived class is created,
The base class' constructor will be called first.
Followed by the derived class' constructor.
When a derived object is destroyed,
Its destructor is called first,
Followed by the base class' destructor.
9:11 / 17:01


Sai Inheritance III
Press Esc to exit full screen
In other words,
Constructors are executed in their order of derivation.
9:16 / 17:01


Sai Inheritance III
Press Esc to exit full screen
Note and Exercise
The general rule applies:
In cases of inheritance where a derived class becomes the base class for another derived class,
9:26 / 17:01


Sai Inheritance III
Press Esc to exit full screen
In other words,
Constructors are executed in their order of derivation.
Destructors are executed in reverse order of derivation.
9:30 / 17:01

Sai Inheritance III

Press Esc to exit full screen

```
#include <iostream>
using namespace std;

class base {
public:
  base() { cout << "Constructing base\n"; }
  ~base() { cout << "Destructing base\n"; }
};

class derived1 : public base {
public:
  derived1() { cout << "Constructing derived1\n"; }
  ~derived1() { cout << "Destructing derived1\n"; }
};

class derived2: public derived1 {
public:
  derived2() { cout << "Constructing derived2\n"; }
  ~derived2() { cout << "Destructing derived2\n"; }
};

int main()
{
  derived2 ob;

  return 0;
}
```

```
Constructing base
Constructing derived1
Constructing derived2
Destructing derived2
Destructing derived1
Destructing base
```

10:10 / 17:01

Sai Inheritance III

Press Esc to exit full screen

- **The same rule applies,**
  - **When multiple base classes are involved in creating a derived class**

10:25 / 17:01

Sai Inheritance III

Press Esc to exit full screen

```
#include <iostream>
using namespace std;

class base1 {
public:
  base1() { cout << "Constructing base1\n"; }
  ~base1() { cout << "Destructing base1\n"; }
};

class base2 {
public:
  base2() { cout << "Constructing base2\n"; }
  ~base2() { cout << "Destructing base2\n"; }
};
```

```
class derived: public base1, public base2 {
public:
  derived() { cout << "Constructing derived\n"; }
  ~derived() { cout << "Destructing derived\n"; }
};

int main()
{
  derived ob;

  return 0;
}
```

```
Constructing base1
Constructing base2
Constructing derived
Destructing derived
Destructing base2
Destructing base1
```

10:45 / 17:01

Sai Inheritance III

Press Esc to exit full screen

How to pass parameters to Base-Class Constructors

```
derived-constructor(arg-list) : base1(arg-list),
                                base2(arg-list),
                                // ...
                                baseN(arg-list)
{
  // body of derived constructor
}
```

12:50 / 17:01

Sai Inheritance III

Press Esc to exit full screen

```
class base
{
  protected:
  int i;
  public:
  base(int x)
  {
    i=x;
    cout << "Constructing base\n";

  }
  ~base()
  {
    cout << "Destructing base\n";

  }
};
```

```
class derived: public base
{
  int j;
  public:
  derived(int x, int y): base(y)
  {
    j=x;
    cout << "Constructing derived\n";

  }
  ~derived()
  {
    cout << "Destructing derived\n";

  }
  void show()
  {
    cout << i << " " << j << "\n";

  }
};
```

```
int main()
{
  derived ob(3, 4);
  ob.show();
  return 0;
}
```

13:59 / 17:01

Sai Inheritance III

Press Esc to exit full screen

```
class base1
{
  protected:
  int i;
  public:
  base1(int x)
  {
    i=x;
    cout << "Constructing base1\n";

  }
  ~base1()
  {
    cout << "Destructing base1\n";

  }
};
```

```
class base2
{
  protected:
  int k;
  public:
  base2(int x)
  {
    k=x;
    cout << "Constructing base2\n";
  }
  ~base2()
  {
    cout << "Destructing base2\n";
  }
};
```

14:41 / 17:01

Sai Inheritance III

Press Esc to exit full screen

```
class derived: public base1, public base2
{
  public:

  derived(int x, int y): base1(x), base2(y)
  {
    cout << "Constructing derived\n";
  }
  ~derived()
  {
    cout << "Destructing derived\n";
  }
  void show()
  {
    cout << i << " " << k << "\n";
  }
};
```

```
int main()
{
  derived ob(3, 4);
  ob.show();
  return 0;
}
```

15:41 / 17:01

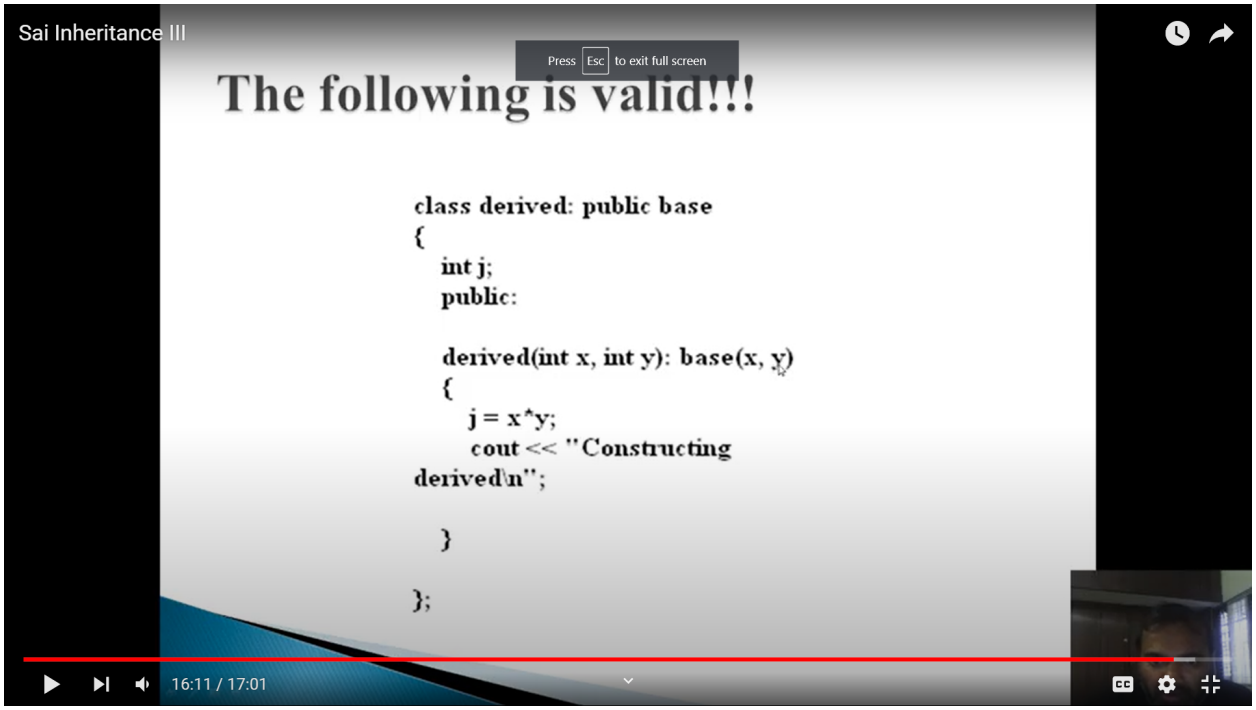
Sai Inheritance III

Press Esc to exit full screen

Derived Constructor can use!!!

- Even if one or more parameters are passed along to a base class,
- A derived class' constructor is free
  - To make use of any and all parameters that it is declared as taking.

15:56 / 17:01


Sai Inheritance III
Press Esc to exit full screen
The following is valid!!!
class derived: public base
{
int j;
public:
derived(int x, int y): base(x, y)
{
j = x*y;
cout << "Constructing
derived\n";
}
};
16:11 / 17:01