

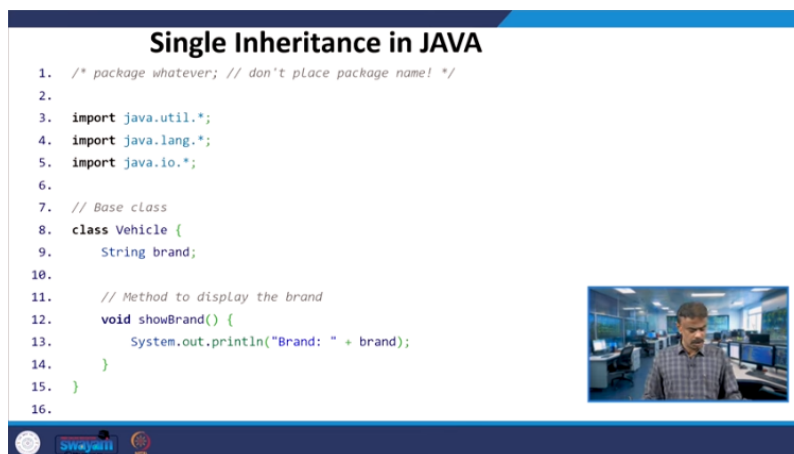
FUNDAMENTALS OF OBJECT ORIENTED PROGRAMMING

Lecture05

Lecture 5: Introduction to Paradigms of OOP

Welcome to lecture 5 Introduction to Object Oriented Programming. And in today's class, we are going to see inheritance in the example of Java and we are going to see polymorphism. So in the last class, we had seen the single inheritance in C++. And in this lecture, you can see the code in Java. All right.

So the same code. You have a class vehicle. So which is called as a base class. All right. So this is the base class.



```
1.  /* package whatever; // don't place package name! */
2.
3.  import java.util.*;
4.  import java.lang.*;
5.  import java.io.*;
6.
7.  // Base class
8.  class Vehicle {
9.      String brand;
10.
11.      // Method to display the brand
12.      void showBrand() {
13.          System.out.println("Brand: " + brand);
14.      }
15.  }
16.
```

And I have a data member called brand, right, whose data type is string. So like C++, we have written the member function. Here you call it as method. In Java, you call this as method, okay. So we have a method called show brand, right.

So return type is void. And you are printing, you have only one statement, system.out.println, so which will print So the brand will be printed in the double quote and then whatever the value or whatever the string of brand will be printed, right? So this is of base class in Java and if you look, there is no semicolon. Whereas in the case of C++, we had semicolon.


So this is the base class, right? Vehicle is your base class, right? So this we call it as because we use the notation A. So A is the base class. And we have a derived class called car, right. So, we have a derived class called car.

So, the car is extending the vehicle. So, that means vehicle is a base class. I will just draw the diagram. Vehicle is a base class, okay. So, your car, the derived class will inherit, right, the properties of vehicle, right.

Single Inheritance in JAVA

```
17. // Derived class inheriting from Vehicle (single inheritance)
18. class Car extends Vehicle {
19.     String model;
20.
21.     // Method to display the model
22.     void showModel() {
23.         System.out.println("Model: " + model);
24.     }
25. }
26.
```

Veh.




So, this is the You in fact call it as a superclass. In Java you call it as a superclass. Right. This is same as base class.

Single Inheritance in JAVA

```
17. // Derived class inheriting from Vehicle (single inheritance)
18. class Car extends Vehicle {
19.     String model;
20.
21.     // Method to display the model
22.     void showModel() {
23.         System.out.println("Model: " + model);
24.     }
25. }
26.
```

Vehicle Super class

Car



And a car is a subclass in Java. So what we studied in C++ it is a derived class. Right. So here you use. Class car extends, right?

So we use extends, extends vehicle, right? So like what we have done in the case of C++, you put class car and then public vehicle, something like that we have done. If you recall what we have done in the case of C++, syntax is slightly

different, right? So here we use the keyword called extends. So car extends vehicle, right?


The class car extends vehicle. That means the car can use the data member and methods right assume that if it is publicly available that can be used right. So here you have a data member call model whose return type is whose data type is string okay and then you have one more method all right. So you have a one more in the sense we have already discussed one method in superclass

Single Inheritance in JAVA

```
17. // Derived class inheriting from Vehicle (single inheritance)
18. class Car extends Vehicle {
19.     String model;
20.
21.     // Method to display the model
22.     void showModel() {
23.         System.out.println("Model: " + model);
24.     }
25. }
26.
```

Vehicle Super class (Base class)

Car Subclass (Derived class)



In the subclass, you have another method called showModel. So, it is a void method. So, you have system.out.println model. So, the model will be printed. That is also string.

So, now we will see in the main program, it will be interesting. In the main program, so let us consider myCar. So, main. So, you have a void main, almost like in C++ syntax. So, you have myCar, which is an object.


So, this is the object you are instantiating in the class car. So, car is a subclass. You go back, car is a subclass. So, under subclass, I am creating an object. So, the new operator, here you have a new operator.


Single Inheritance in JAVA

```

27. public class Main {
28.     public static void main(String[] args) {
29.         // Create an object of the derived class
30.         Car myCar = new Car();
31.
32.         // Set values for the inherited and new properties
33.         myCar.brand = "Toyota"; // Inherited from Vehicle class
34.         myCar.model = "Corolla"; // Specific to Car class
35.
36.         // Access methods from both the base and derived classes
37.         myCar.showBrand(); // Method from base class
38.         myCar.showModel(); // Method from derived class
39.     }
40. }

```

 stdout
 Brand: Toyota
 Model: Corolla



So, suppose I want to declare an array in C++, int A of 10. Let us assume in C++.

So, what I will do, I will use a dynamic array. So dynamic array you might have used a new operator. I am just recalling the procedural oriented programming.

You might have done this. Int star A. A equal to new int of 10. So this is array. And it is called a dynamic array. Of 10 elements.

So you might have used a new operator. In a similar way when I am creating an object. So, Java all are dynamic. Alright. So, when you want to create a dynamic object, you use the new operator.

Right. And then this is we are going to study this is constructor. Right. So, this is a constructor. So, dynamically the memory space will be allocated for my car.


Right. So, that is a meaning. And then you can see. Right. So, my car dot brand.

Single Inheritance in JAVA


```

27. public class Main {
28.     public static void main(String[] args) {
29.         // Create an object of the derived class
30.         Car myCar = new Car();
31.
32.         // Set values for the inherited and new properties
33.         myCar.brand = "Toyota"; // Inherited from Vehicle class
34.         myCar.model = "Corolla"; // Specific to Car class
35.
36.         // Access methods from both the base and derived classes
37.         myCar.showBrand(); // Method from base class
38.         myCar.showModel(); // Method from derived class
39.     }
40. }

```

 stdout
 Brand: Toyota
 Model: Corolla

int * a;
 a = new int[10];



Right. So, it can access. Right. My car can access brand. Brand is available in


Your superclass. Right. Superclass A. And I call this as B. Because we have used the notation. Right. I call this as B. Right.

Single Inheritance in JAVA

```
17. // Derived class inheriting from Vehicle (single inheritance)
18. class Car extends Vehicle {
19.     String model;
20.
21.     // Method to display the model
22.     void showModel() {
23.         System.out.println("Model: " + model);
24.     }
25. }
26.
```

Vehicle Super class (Base class)

Car Subclass (Derived class)



So you can find brand in. Superclass or base class. Right. So here your object. What you have created in B. Right.

That can access. Right. The object that you have created in main. So that can access both brand and model. So this is available in superclass and this is available in, I mean to say brand is available in superclass A and model is available in subclass B. So now my car, you have created under car, yes B. So car is B. My car, the object you have created under B. So now this can access any because you are using the inheritance property.

So now when your object is invoking the method, show brand. Right. So show brand is available in the base class. Right. So this is possible and this will print whatever be the brand.

Single Inheritance in JAVA


```
27. public class Main {
28.     public static void main(String[] args) {
29.         // Create an object of the derived class
30.         Car myCar = new Car();
31.
32.         // Set values for the inherited and new properties
33.         myCar.brand = "Toyota"; // Inherited from Vehicle class
34.         myCar.model = "Corolla"; // Specific to Car class
35.
36.         // Access methods from both the base and derived classes
37.         myCar.showBrand(); // Method from base class
38.         myCar.showModel(); // Method from derived class
39.     }
40. }
```

std out

Brand: Toyota

Model: Corolla

int a;
a = new int[10];



Right. Toyota and my car dot show model. Right. So which will print Corolla. So that is what you are getting as the output.

Right. So this program we can see in Eclipse also. Right. So if I open Eclipse. Right.



```
1 // Base class
2 class Vehicle {
3     String brand;
4
5     // Method to display the brand
6     void showBrand() {
7         System.out.println("Brand: " + brand);
8     }
9 }
10
11 // Derived class inheriting from Vehicle (single inheritance)
12 class Car extends Vehicle {
13     String model;
14
15     // Method to display the model
16     void showModel() {
17         System.out.println("Model: " + model);
18     }
19 }
20
21 public class Main2 {
22     public static void main(String[] args) {
23         // Create an object of the derived class
24         Car myCar = new Car();
25
26         myCar.showBrand();
27         myCar.showModel();
28     }
29 }
```

Brand: Toyota
Model: Corolla

So here you go. The same program. If I run here, you can see you are getting the output. right, Toyota and Corolla, the same program. Whatever I have explained, the same program, if you run, you will get the output Toyota and Corolla.


So, we have seen only single inheritance and rest all other four, right, we will anyway study in detail in the coming weeks, right. So, rest four, we will study slightly later. So, now next concept is polymorphism, right. So, we know poly, the meaning of poly is many, right. right and in the sense forms so many forms so suppose I want to use a function right member function I will go back to C++ so in the member function suppose you are using I in fact gave an example in the introductory class suppose you are using max function can I use the same function name twice or multiple times yes you can do that right so which is called as polymorphism right so

I can use the single entity which is a member function in several forms multiple forms one I can use maximum of two numbers another I use maximum of three numbers or one I use maximum of two integers another maximum of two doubles right or a maximum of two strings or maximum of two character array or maximum of two characters right so you can use the same function name right multiple times. So, that is called polymorphism. So, here in the case of C++ and Java object oriented programming just now we studied inheritance. Suppose you

use let us say a same function name right in base class as well as derived class or super class as well as sub class right.

Polymorphism

- Poly means 'many,' and morph means 'forms.'
- Polymorphism refers to a single entity taking on multiple forms.
- In C++ and Java, polymorphism allows the properties of one class to be used in different classes in various ways.




Logo: Sreyash

So, which one will be called? So, that we are going to see. So in that case we are dividing this into compile time polymorphism and runtime polymorphism. So what do you mean by compile time polymorphism? So the compile time polymorphism in C++ or even in Java, so this can be achieved when we talk about the concept called function overloading.

So what do you mean by overloading? So suppose That is what I said. In one maximum function I used to find maximum of two numbers. Another one I used to find maximum of three numbers.

Compile-time Polymorphism

- This type of polymorphism in C++ can be achieved by function Overloading.
- Overloading by changing number of arguments or type of arguments is called compile-time polymorphism.



Logo: Sreyash

That means in one case I am using two arguments. And another case I am using three arguments. Right. So it is based on changing the number of arguments. That is the meaning.

Or I said another example. In one case I used integers. Maximum of two integers. Another case I can find maximum of two characters. So that means I am changing the type of arguments.

Alright. Either number of arguments or type of arguments you have to be very careful. You cannot use the same type of arguments and then you cannot say the return type is different. Alright. So the function will look for example you are writing float.

You write return type and then you try to write let us say max 1 of let us say int a comma int b. Right. So this is one function. So another function you are saying Your int is a return type and you are using the same name int a comma int b. So now if you look the arguments are same. Right now if you look the arguments are same but return type is different.

You can claim it may work. So unfortunately this won't work. This will throw an error because when you are calling max one in the main. Right. Max 1 of let us say you are passing 7 comma 8.

So which one will be called? Right. The compiler will get confused. Right. And then this program will throw an error.


Compile-time Polymorphism

- This type of polymorphism in C++ can be achieved by function Overloading.
- Overloading by changing number of arguments or type of arguments is called compile-time polymorphism.

Handwritten notes:

```
float max1(int a, int b)
int max1(int a, int b)
```

Handwritten call: max1(7, 8)



So the arguments when you use you have to be bit careful. Right. So the argument should be different. And the number of arguments. Suppose I use let us say in C.

In the second program I use one more argument in C that will work okay. So overloading is nothing but changing the number of arguments or type of

arguments which is called the compile time polymorphism. So anyway from the program I will explain one more time what is compile time polymorphism.

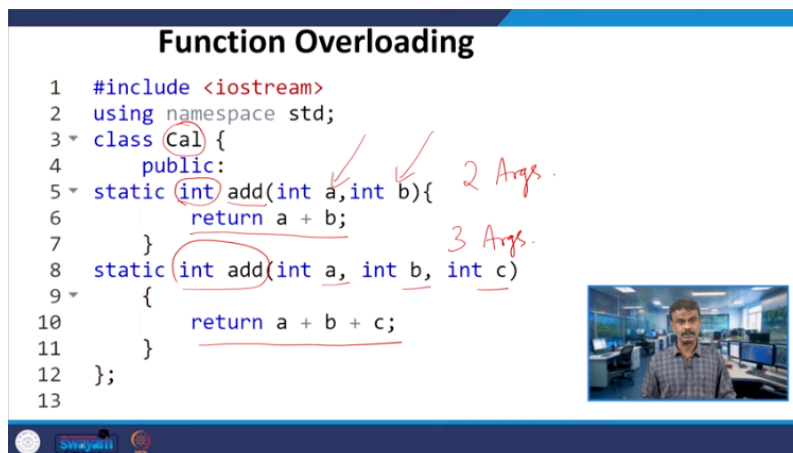
Suppose you take this particular example right. So here

You see the name of the class is. You are calculating something cal. Right. And here you have a function name called add. Right.

Whose return type is integer. Correct. And then here you have. Two arguments. Both are.

Data type. integer right both are having data type integer so two arguments you are using right you are using two arguments so here you are returning a plus b so addition of two numbers right you are returning and another function so let us say int add so which is having a b and c right so here you have So let us say three arguments all are having integer data type right. So here you are returning A plus B plus C right. So now this is three arguments.

So this is three arguments. So you have two functions right of same name. So these are all member functions. Under the class calculation. Right.



So, Cal. Simply Cal. Right. The class Cal. So, you have two member functions.

One is this with the two arguments. Another one is this with three arguments. But the beauty over here is both are having same name. Right. So, one is returning A plus B. That means sum of two numbers.

Another one is returning sum of three numbers. Right. So now when you go to the main program. Right. So you have an object.


Object called C. Whose class is Cal. Right. So you are instantiating an object under CAL. Name of the object is C. So now this C will invoke one addition function. Right.

```
14 int main(void) {
15     Cal C; // class object declaration.
16     cout<<C.add(10, 20)<<endl;
17     cout<<C.add(12, 20, 23);
18     return 0;
19 }
20
```

stdout

30

55



One of the functions. Both are same name. So 10 comma 20. You are passing two values. So here you are passing two values.

And the next one you are passing how many values? 3 values. So now the question is which one will be called? So for the first case line number 16 you have 2 values. Right.


So the compiler will decide to call the 2 argument function. Because you have passed 2 values. Right. So A will take what is the value? A will take 10.

B will take 20. So in this case A will take 10. And B will take 20. And then there is an addition. Return A plus B. Alright.

Function Overloading

```
1 #include <iostream>
2 using namespace std;
3 class Cal {
4     public:
5     static int add(int a, int b) {
6         return a + b;
7     }
8     static int add(int a, int b, int c) {
9         return a + b + c;
10    }
11 };
12
13
```

Handwritten notes: $a \rightarrow 10$, $b \rightarrow 20$, 2 Args., 3 Args.



So the addition will happen. We know the output should be 30. Alright. So the second case. You are passing three arguments.

Right. Three values in fact. You are passing three values. 12, 20 and 23. Passed by values.

Correct. So that means since you have passed three values the three argument member function will be called. So A will take the value how much? 12. So here in this case A will take the value 12. B will take the value how much? B will take the value how much? 20. And C will take the value 23.


Right. So these three will be added. Return A plus B plus C. Right. 35 plus 20, 55.

Function Overloading

```
1  #include <iostream>
2  using namespace std;
3  class Cal {
4  public:
5  static int add(int a, int b) {
6      return a + b;
7  }
8  static int add(int a, int b, int c)
9  {
10     return a + b + c;
11 }
12 };
13
```

Handwritten annotations:

- For the 2-argument function: $a \rightarrow 10$, $b \rightarrow 20$, result 30 . Labeled "2 Args."
- For the 3-argument function: $a \rightarrow 12$, $b \rightarrow 20$, $c \rightarrow 23$. Labeled "3 Args."



So the next answer I have to get 55. This is the output I have to get. Right. So if you look at the output, you are getting this 30 and 55. Right.

So now the question is how it is being chosen. So when you are calling at 10 comma 20, when this object C is invoking at. Right. So 10 comma 20. The compiler will decide during compilation time.

Right. So during compilation time, compiler will decide. Right. Since you have passed two values. Right.

So the two argument member function will be called. Right. So this will be decided during compile time. Right. And similarly, in line number 17, you have passed three values.

Right. So the compiler will decide to call three argument function. Right. Three argument member function. So these two will be decided during the compile time.

Therefore, this is called compile time polymorphism or this is another name called early binding. Or you can also call this concept as static binding. All are same name, static binding, early binding and compile time polymorphism. All right. So the main idea, the beauty of the object oriented programming, you can use the same function name as many times as possible.

```
14 int main(void) {
15     Cal C; // class object declaration.
16     cout<<C.add(10, 20)<<endl;
17     cout<<C.add(12, 20, 23);
18     return 0;
19 }
20
```

stdout


30 ✓

55 ✓

Early binding

2 vals.

3 vals



Only thing it has to follow this rule. All right. So you have to keep in mind the number of arguments or the type of argument should be changed. So you call this as compile time polymorphism. So the next one is runtime polymorphism.

All right. So it is decided during runtime. We have talked about inheritance. Extensively we studied about the single inheritance. So if you recall, so we know what is meant by single inheritance.

So there you have base class and derived class. I will talk in terms of C++. Assume that like a previous program, you have the member function or method in base class as well as derived class. The same name. Assume that dot add right.

Run-time Polymorphism

- Run-time polymorphism is also known as Dynamic Method Dispatch.
- This type of polymorphism can be achieved using function overriding.
- In this approach a call to the overridden function is resolved during run-time.
- Method Overriding is done when a child or a derived class has a function with same name, parameters and return type as the parent or the superclass, then that function overrides the function in the base class.



Assume that similar to or otherwise you assume that you are using some factorial or ncr right. So this function let us assume that it is available both in base class as well as derived class. Now you have created an object. So let us assume you have created an object under derived class because that is what the example we had seen. So, now the question is when this object calls let us say ncr this is a function member function right.

So, this is a member function. So, when this is calling the function right when the object is invoking the function the object that you have created under derived class. So, now the question is which one will be called because now assume that both are public All right all the entities are public like data member and methods or member data or member functions are public in base class. So now the question is just now we have studied right in one of the examples.

So when you are creating an object in derived class this can also access the data member or methods or member function or member data right so can be accessed if it is in public. Right. Suppose the names are same. Right. Which one will be called?

Right. Suppose the names are same, which one will be called? So this will be decided during runtime. The previous case, the previous example, compile time polymorphism, I said that dot add function. So which will be decided by the compiler during compile time?

Whereas here in this case, slightly opposite. So this will be decided during runtime. So, this polymorphism concept, this can be achieved by using function overriding. So, we have seen overloading in the case of compile time

polymorphism and this is called function overriding. The same function name, so that may appear as many times, more than one time or more than two times or several times.

So, now the question is which one will be called? Sometimes it is a combination of your compile time and runtime. Possible, right? The arguments are different. Which one will be called?

So now the question is, assume that base class, same function is there. And the derived class, same function name is there. So which one will be called? So this will be decided during runtime. And this can be achieved using function overriding.

And I also said this will be the overridden functions is resolved during runtime that is very important right. So you call this as runtime polymorphism. So assume that you have a same function name in both superclass and subclass right or base class and derived class. And also here in this case the number of parameters return type can be same. Whereas we have some constraint in compile time polymorphism.

So here not a problem. Both the functions may be identical. Inside you are doing something extra. So when you are writing I am in the base class. See out I am in the base class.

And here the other one, see out, I am in the derived class. But when you are writing the function, right, the number of arguments, the return type, all are identical, right? So even if it is identical, right, so this can be achieved, right? So which one has to be chosen can be decided by the compiler during runtime, right? right.

So, therefore, this is called the runtime polymorphism, right. So, here you have seen, right, the concepts like function overriding, right. So, we have seen overloading and here you call it as function overriding and this can be achieved during runtime right and also the superclass and subclass are involving right. So, because in superclass and subclass you can find the same function name.

So, here it is also called dynamic binding, late binding is also called late binding and you can also call this dynamic binding. these are the other names of runtime polymorphism late binding and dynamic binding whereas in the case of compile

Right. So, this is going to be like a base class. And here we have member data called P. Right. Whose data type is integer. Right.

So we have a member data called P whose data type is integer right and then you have so this is I will call it as a member data or data member whatever way you write fine member data and you have one member function called display right. So if you look there is no argument and return type is void and see out statement class P So this is a class P, you are putting a semicolon. So you have class P. And now class Q is publicly inheriting P. So that means P is the base class and Q is inheriting the properties of P. The inheritance and from the inheritance I am explaining what do you mean by runtime polymorphism. So here, so this is also common statement derived.

This is what I mean. Here no argument. There is also no argument. Here the return type is void. There is also return type void.

But inside the function you can change. That is what exactly I told. Right. So here c out class q. Here c out class q. So now the question is. Right.

Suppose q. Right. I have an object q1. So, this is a C++ code only I put like this Q1. I am creating an object Q1. So, now Q1 which will call the function.

So, now assume that Q1 is calling the function display function. Now the question is which one will be chosen? So Q1 is under Q. It is the subclass. So you have an object Q1. Now assume that Q1 is invoking display.

So this we will extensively study. So this is just an introductory class of polymorphism. So now when your object Q1 is invoking display. So, which function will be called, right? Which member function will be called?

So, this will be decided during runtime, right? So, this will be decided during runtime. You can think of, all right? Anyway, extensively we are going to study. So, I am giving like a reading assignment.

So, you can write a program and think. So, which one will be called? So, extensively anyway we are going to study so many because we can create like this. Let us say P, P1. Right.

And then P1 dot display because I have not introduced constructor and all. So we can play with that. So P1 dot display. Right. So now the question is when P1 is invoking the same function name display, which one will be called?

right. So, what we can do? So, once extensively we study about the polymorphism and when we run certain programs. So, you can understand, right. So, which one will be called, right.

So, this is about the polymorphism. So, in the introductory lecture. So, so far we have studied the classes and objects. So, now you can able to write some sample programs or the basic programs on the classes and objects.

Run-time Polymorphism

```

class P
// base class declaration.
{
    int p;
    public:
    void display()
    {
        cout<< "Class P ";
    }
};

```

Handwritten notes:
 - Red arrow from `int p;` to "member data"
 - Red circle around `void display()`
 - Red circle around `cout<< "Class P ";`
 - Red text: `P p1;`
 - Red text: `p1.display();`
 - Red text: `Q (q1);`

```

class Q : public P
// derived class declaration.
{
    int q;
    public:
    void display()
    {
        cout<<"Class Q";
    }
};

```

Handwritten notes:
 - Red circle around `int q;`
 - Red circle around `void display()`
 - Red circle around `cout<<"Class Q";`
 - Red text: `q1.display();`

So, what you can do? You can start writing the code, right. So, during the basic programming, right, the basic programming like procedural oriented programming, you might have returned some code, right. You started with hello world, you started with addition of two numbers, right or finding out the number whether it is prime or not. So, what you can do?

Now, you have to create an object and classes. Right. So let us assume you are writing a program for finding out whether the given number is prime or not. Let us assume you are using completely the object oriented programming, whatever you have studied in the last five lectures. Right.

So create a class. Right. Make a class and create an object. And from there, write member functions, create various objects. Right.

So write a member data and member function and write the logic. So as I said, when you are doing it in object-oriented programming, you can elegantly do that. You can split the programs, right? You can split a program into several classes. So this facility is available in object-oriented.

And when you look at the last main program, it may look like three or four lines, right? So that means you have made everything with the different classes. So with the help of classes and objects, you can start write the basic programming. So whatever you have studied in procedural oriented programming, you start writing with the help of classes and objects. And you now have a little knowledge on inheritance and polymorphism.

So try to do that. You might have studied function overloading in procedural oriented programming. So here you have the concept of object oriented

programming. right and the very basic concepts like classes, objects, inheritance and polymorphism. So, you can try to write whatever the code that you have studied in the basic programming.

So, try to write and then extensively the next class extensively we study the classes, objects and and then I will introduce right extensive extensive classes and objects because we have to introduce the block inside the class constructor and destructor more importantly right the constructor and destructor because when you are using C++ the memory management is very important right so whenever you are not using certain objects how I can flush it right so this we will study right in the next few classes so classes and objects And you will study constructors and distractors. So, with this I am concluding the fifth lecture. Thank you very much.