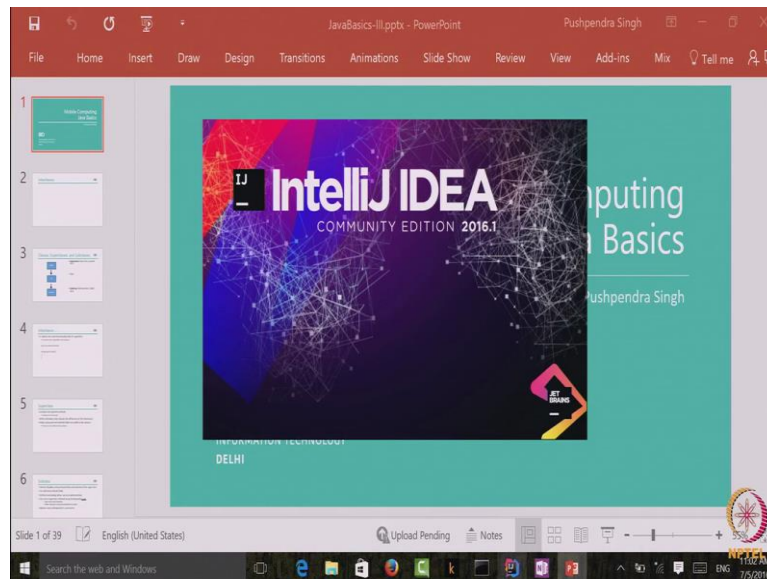


**Mobile Computing**  
**Professor Pushpedra Singh**  
**Indraprasth Institute of Information Technology Delhi**  
**Java Basics**  
**Lecture 03**

Hello, last time we have learnt about strings. Today we will see some programs displaying the concept that we learnt in the lecture.

(Refer Slide Time: 0:26)



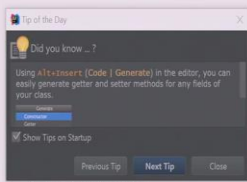
```
ClassDemo - [C:\Users\Pushpendra Singh\IdeaProjects\ClassDemo] - [ClassDemo] - IntelliJ IDEA 2016.1.3
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
New
Open Recent
Close Project
Settings... Ctrl+Alt+S
Project Structure... Ctrl+Alt+Shift+S
Other Settings
Import Settings...
Export Settings...
Export to Eclipse...
Settings Repository...
Save All Ctrl+S
Synchronize Ctrl+Alt+Y
Invalidate Caches / Restart...
Export to HTML...
Print...
Add to Favorites
File Encoding
Line Separators
Make File Read only
Power Save Mode
Exit

EquilateralTriangle e = new EquilateralTriangle(11.5);
Circle c = new Circle("Circle");
GeometryFigure f;

e.setSize();

f.getArea();

f.displayShape();
System.out.println(c.getSize());
```



```
StringDemo - [C:\Users\Pushpendra Singh\IdeaProjects\StringDemo] - [StringDemo] - IntelliJ IDEA 2016.1.3
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
StringDemo
Main

package in.ac.iiitd.nptel;

public class Main {

    public static void main(String[] args) {
        // write your code here
        String m1 = "HelloWorld";

        System.out.println(m1.substring(0, 3));
        System.out.println(m1.substring(1, 3));

        String m2 = m1.substring(0, 3) + "p!";

        String m3 = "Help!";

        String m4 = "HelloWorld";
        String m5 = "HelloWorld";

        System.out.println("m1: " + m1);
```

```
StringDemo - [C:\Users\Pushpendra Singh\IdeaProjects\StringDemo] - [StringDemo] - IntelliJ IDEA 2016.1.3
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
StringDemo
Main

String m4 = "HelloWorld";
String m5 = "HelloWorld";

System.out.println("m1: " + m1);
System.out.println("m2: " + m2);
System.out.println("m3: " + m3);
System.out.println("m4: " + m4);
System.out.println("m5: " + m5);

boolean b1 = (m1 == m2);
boolean b2 = m1.equals(m2);
boolean b3 = (m2 == m3);
boolean b4 = m2.equals(m3);
boolean b5 = (m4 == m5);
boolean b6 = m4.equals(m5);
boolean b7 = (m1 == m4);
boolean b8 = m1.equals(m4);

System.out.println("Result " + b1 + " " + b2);
System.out.println("Result " + b3 + " " + b4);
```

```
StringDemo - [C:\Users\Pushpendra Singh\IdeaProjects\StringDemo] - StringDemo - ...\src\ac\in\id\prtel\Main.java - IntelliJ IDEA 2016.1.3
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
StringDemo
Main.java
System.out.println("m1: " + m1);
System.out.println("m2: " + m2);
System.out.println("m3: " + m3);
System.out.println("m4: " + m4);
System.out.println("m5: " + m5);

boolean b1 = (m1 == m2);
boolean b2 = m1.equals(m2);
boolean b3 = (m2 == m3);
boolean b4 = m2.equals(m3);
boolean b5 = (m4 == m5);
boolean b6 = m4.equals(m5);
boolean b7 = (m1 == m4);
boolean b8 = m1.equals(m4);

System.out.println("Result " + b1 + " " + b2);
System.out.println("Result " + b3 + " " + b4);
System.out.println("Result " + b5 + " " + b6);
System.out.println("Result " + b7 + " " + b8);
}

StringDemo - [C:\Users\Pushpendra Singh\IdeaProjects\StringDemo] - StringDemo - ...\src\ac\in\id\prtel\Main.java - IntelliJ IDEA 2016.1.3
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
StringDemo
Main.java
boolean b1 = (m1 == m2);
boolean b2 = m1.equals(m2);
boolean b3 = (m2 == m3);
boolean b4 = m2.equals(m3);
boolean b5 = (m4 == m5);
boolean b6 = m4.equals(m5);
boolean b7 = (m1 == m4);
boolean b8 = m1.equals(m4);

System.out.println("Result " + b1 + " " + b2);
System.out.println("Result " + b3 + " " + b4);
System.out.println("Result " + b5 + " " + b6);
}

Run Main
"C:\Program Files\Java\jdk1.8.0_92\bin\java" ...
Hel
e1
m1: HelloWorld
m2: Help!
m3: Help!
m4: HelloWorld
m5: HelloWorld
Result false false
Compilation completed successfully in 23.757ms (moments ago)
StringDemo - [C:\Users\Pushpendra Singh\IdeaProjects\StringDemo] - StringDemo - ...\src\ac\in\id\prtel\Main.java - IntelliJ IDEA 2016.1.3
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
StringDemo
Main.java
Run Main
"C:\Program Files\Java\jdk1.8.0_92\bin\java" ...
Hel
e1
m1: HelloWorld
m2: Help!
m3: Help!
m4: HelloWorld
m5: HelloWorld
Result false false
Compilation completed successfully in 23.757ms (moments ago)
StringDemo - [C:\Users\Pushpendra Singh\IdeaProjects\StringDemo] - StringDemo - ...\src\ac\in\id\prtel\Main.java - IntelliJ IDEA 2016.1.3
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
StringDemo
Main.java
Run Main
"C:\Program Files\Java\jdk1.8.0_92\bin\java" ...
Hel
e1
m1: HelloWorld
m2: Help!
m3: Help!
m4: HelloWorld
m5: HelloWorld
Result false false
Compilation completed successfully in 23.757ms (moments ago)
```

The image consists of two screenshots of an IDE window. The top screenshot shows the following code:

```
boolean b1 = (m1 == m2);
boolean b2 = m1.equals(m2);
boolean b3 = (m2 == m3);
boolean b4 = m2.equals(m3);
boolean b5 = (m4 == m5);
boolean b6 = m4.equals(m5);
boolean b7 = (m1 == m4);
boolean b8 = m1.equals(m4);

System.out.println("Result " + b1 + " " + b2);
System.out.println("Result " + b3 + " " + b4);
System.out.println("Result " + b5 + " " + b6);
```

The bottom screenshot shows the same code with the following additions:

```
System.out.println(m1.substring(0, 3));
System.out.println(m1.substring(1, 3));

String m2 = m1.substring(0, 3) + "!";

String m3 = "Help!";

String m4 = "HelloWorld";
String m5 = "HelloWorld";
```

The output console in both screenshots shows the following text:

```
Hel
el
m1: HelloWorld
m2: Help!
m3: Help!
m4: HelloWorld
m5: HelloWorld
Result false false
Result false true
```

For running java programs you have to install an IDE such as intelliJ or eclipse both of these IDE are available online. Let us see, i have made it here a very simple program for just showing some concept of the strings that we learnt the other day. Here I initialize string M one to helloworld then I tried to print two substrings of M 1.

After that a sign another string variable M 2 to a part of M 1 plus a new string and then I sign another string M 3 and then M 4 and M 5. We want to see how does the sub string function works and we also want to see that what happens when we compare strings using double equal to sign and when compare a strings using the equals method as recommended.

So in this program as you may have guessed M 1 is a sub strings of the hello world it takes the first three characters 0, 1, 2 so in a sub string method called the first position is inclusive but the second position is not inclusive. So in M 2 we are taking first three letters for example H E L and then we are combining them P exclamation mark. Which together means help.

We are assigning M 3 with help exclamation mark directly then we are assigning M 4 and M 5 again with the hello world.

We print all the five strings and then we do the comparison in the comparison first we compare using double equal to sign and then we compare using equal sign and we want to see the results we store the result in the Boolean then we print the result. So let us run the program and see the result.

As you see for the first method called where I am taking sub strings of 0 to 3 I get a print out of H E L that is first three letters for second time I only get E L because strings and arrays both starts from position 0 just like another programming languages like C plus plus. Then I print M 1, M 2, M 3, M 4, M 5. Now you can see that the value of M 1, M 4 and M 5 is Hello world while the value of M 2 and M 3 is help followed by an exclamation mark.

Later in our program we are comparing M 1 and M 2 we are comparing M 2 and M 3 we are comparing M 4 and M 5 and we are comparing M 1 and M 4. First by double equal sign and then by the equal methods (())(3:43) then we are printing all the results. Let's see what result are we getting. Comparison of M 1, M 2 by either method result us into false that is not very surprising. Comparison of M 2 and M 3 where both of them have the same value HEL results false when we are using double equal to sign but results true when we are using equals method call.

This is because as we learn in the lecture other day the double equal sign compares the location while equal compares the value. And because locations of the both strings are different that's why the double equal sign gives us false while equal method gives us string. Same thing you can later see when we compare M 4 and M 5. They come true in both insponses. As we learnt in the last class that a strings are immutable in java.

Which means once java has created a string then java keeps it for all the future references that is if I keep creating other string variables M 4, M 5, M 6, M 7 with the value hello world. Java will keep assigning may the same object that it has already stored.

(Refer Slide Time: 5:16)

```
boolean b1 = (m1 == m4);
boolean b6 = m1.equals(m4);

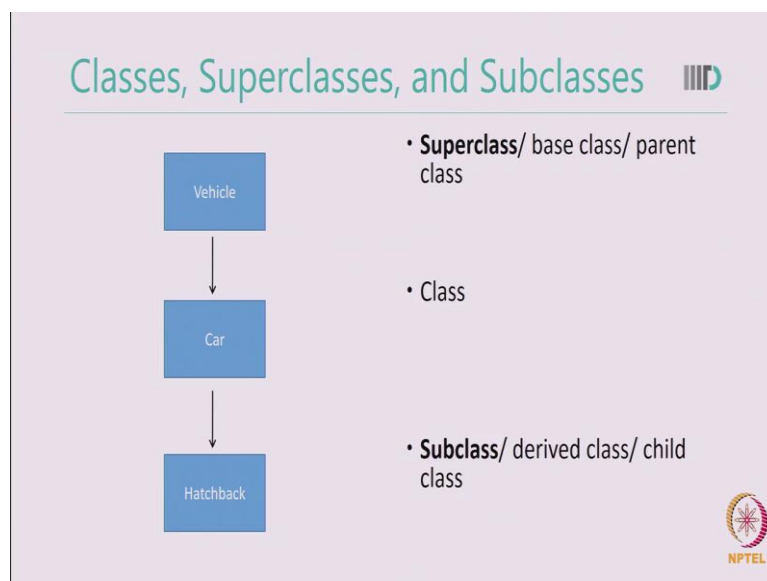
System.out.println("Result " + b1 + " " + b2);
System.out.println("Result " + b3 + " " + b4);
System.out.println("Result " + b5 + " " + b6);
System.out.println("Result " + b7 + " " + b8);

}
}
```

Run Main  
m3: Help!  
m4: HelloWorld  
m5: HelloWorld  
Result false false  
Result false true  
Result true true  
Result true true  
Process finished with exit code 0

That is the reason why we are getting M 4 and M 5 both true by a double equal sign as well as by the value. Not only this if we look at our M 1 which was the first time when we assign this string word hello world even M 1 and M 4 also come out to be same. You may want to extend this program and compare M 1 and M 5 you will get the same result as for M 1 and M 4. The simple example shows you whenever we want to compare values of strings we must always use dot equals. It also shows that a strings are immutable that is once they are created they live in the memory. Therefore if you want strings that you want to muted please use string builder and not the string constructor.

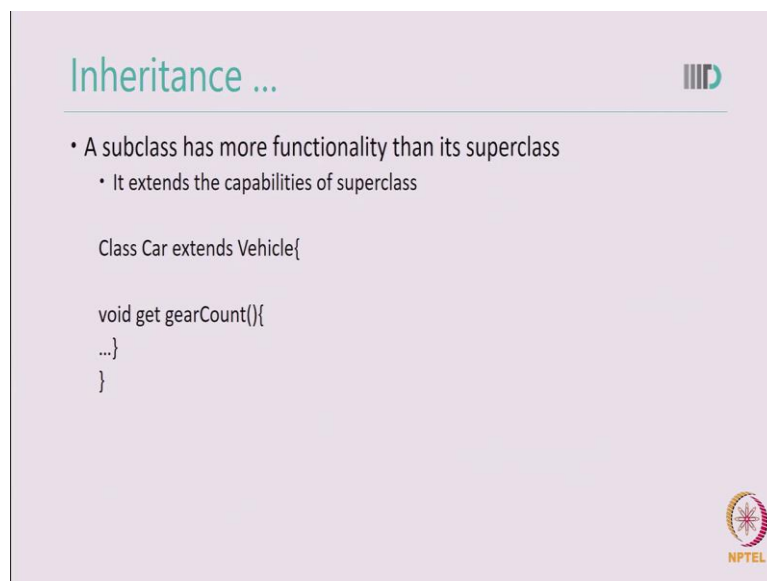
(Refer Slide Time: 6:05)



Now let's move on as we see suppose we have a class called vehicle. Now you may see that there could be different types of vehicles. For example car is a vehicle, a truck is a vehicle and the bus is a vehicle. However if you want to see that a car is a vehicle we may want to see that car could form a subclass of the vehicle. Between the cars we again have a choice there could be hatchback car, there could be a sedan car. Which can then be other sub classes of the car we use the hierarchy to simplify our programs.

We define the super class which has a functionality which is common to all the sub classes and we move on down the level by defining more and more functionality. For example every vehicle may have some wheels, every vehicle may have some setting space. However car may have some specialise type of vehicle. A wheels and a car may have specialise type of seating space then for the hatchback we go for more definition into the functionality that we have defined.

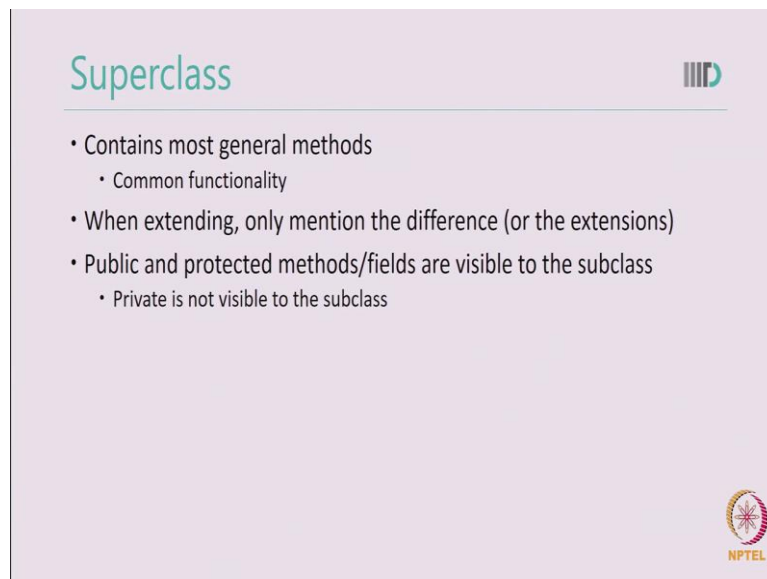
(Refer Slide Time: 7:19)



The slide is titled "Inheritance ..." in a teal font. In the top right corner, there is a logo consisting of three vertical bars of increasing height. Below the title, there is a bulleted list with one item: "A subclass has more functionality than its superclass", which has a sub-bullet: "It extends the capabilities of superclass". Below the list, there is a code snippet: "Class Car extends Vehicle{" followed by an indented block "void get gearCount(){ ...}" and a closing brace "}". In the bottom right corner, there is a circular logo with a gear-like pattern and the text "NPTEL" below it.

So how do we create inheritance? You may see the keyword extends. Once I use the keyword extends my class car becomes a sub class of another class called vehicle.

(Refer Slide Time: 7:40)

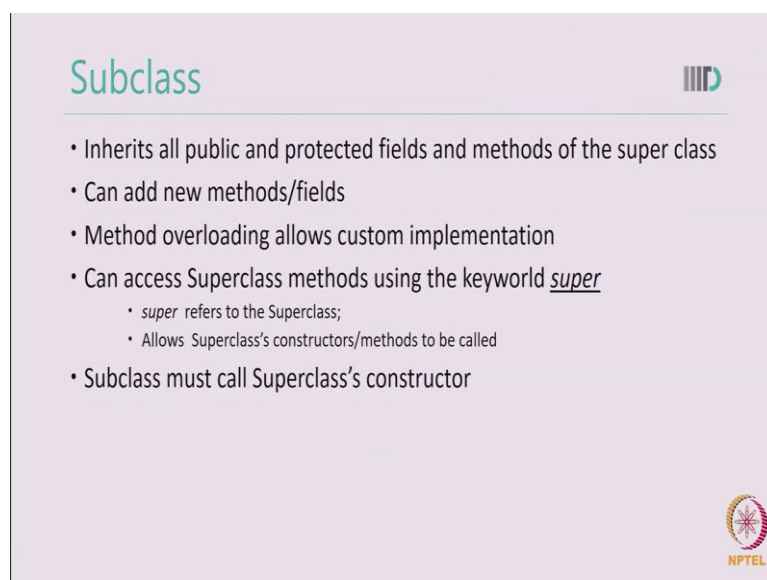


The slide is titled "Superclass" in a teal font at the top left. In the top right corner, there is a logo consisting of three vertical bars of increasing height. The main content is a list of four bullet points: "Contains most general methods" (with a sub-bullet "Common functionality"), "When extending, only mention the difference (or the extensions)", "Public and protected methods/fields are visible to the subclass" (with a sub-bullet "Private is not visible to the subclass"), and "Subclass must call Superclass's constructor". In the bottom right corner, there is the NPTEL logo, which features a circular emblem with a star and the text "NPTEL" below it.

- Contains most general methods
  - Common functionality
- When extending, only mention the difference (or the extensions)
- Public and protected methods/fields are visible to the subclass
  - Private is not visible to the subclass

So a super class like in the previous example vehicle will continue in most general method which are common to all the sub classes and when we extends ideally we should only mention the differences. In last lecture we had also studied about the (())(7:56) modifier such as public, protected and private . When we inherit a class all public and protected methods or fields are visible to the sub class. However private methods or fields are not visible to the sub class.

(Refer Slide Time: 8:15)



The slide is titled "Subclass" in a teal font at the top left. In the top right corner, there is a logo consisting of three vertical bars of increasing height. The main content is a list of five bullet points: "Inherits all public and protected fields and methods of the super class", "Can add new methods/fields", "Method overloading allows custom implementation", "Can access Superclass methods using the keyword super" (with sub-bullets "super refers to the Superclass;" and "Allows Superclass's constructors/methods to be called"), and "Subclass must call Superclass's constructor". In the bottom right corner, there is the NPTEL logo, which features a circular emblem with a star and the text "NPTEL" below it.

- Inherits all public and protected fields and methods of the super class
- Can add new methods/fields
- Method overloading allows custom implementation
- Can access Superclass methods using the keyword super
  - *super* refers to the Superclass;
  - Allows Superclass's constructors/methods to be called
- Subclass must call Superclass's constructor

A subclass inherits all public and protected fields and methods of the super class. But it can also add new methods or fields. These new methods or fields are not visible to the super

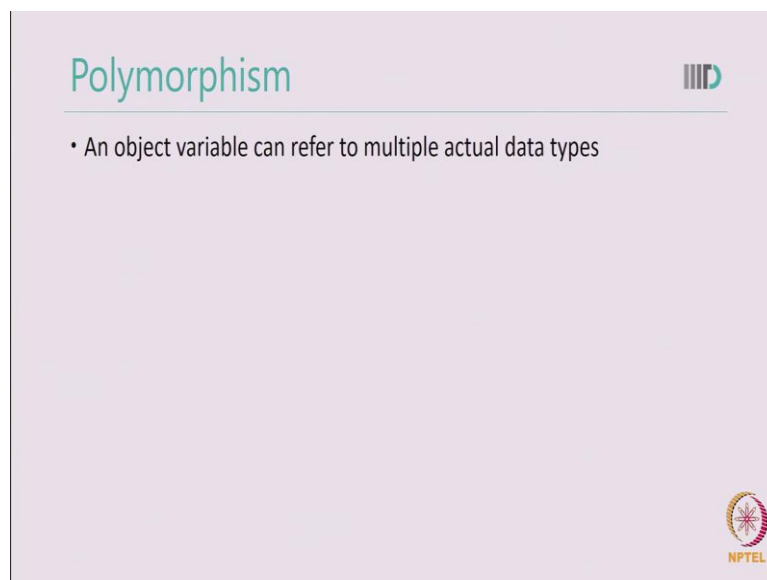


class. We do something which is called method overloading so that we can change the behaviour of a method that has been defined in the super class.

Suppose the super class has a method called get wheels count. Now a car has only four wheels while a truck has more than a four wheels. So if from the vehicle we drive two classes one of type car, one of type truck then it is advisable that we do method over riding so that the get wheels count returns the correct number for two differences of classes.

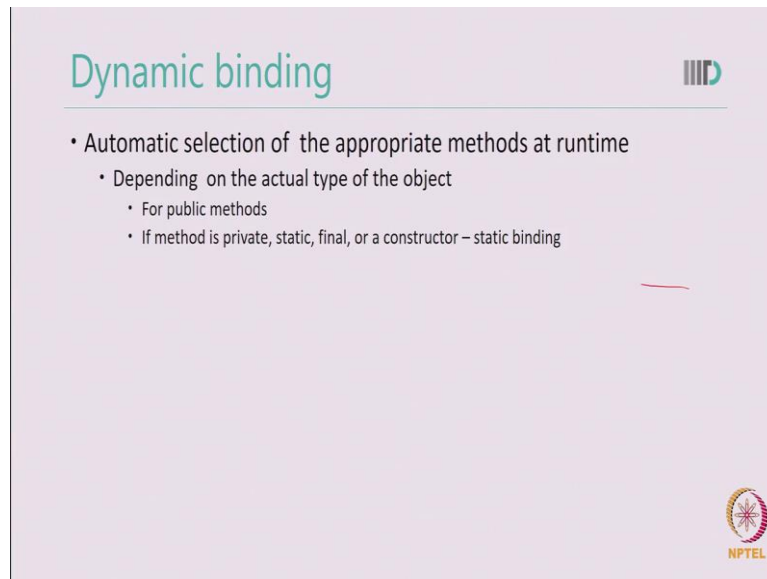
Each subclass can access super class method using the key word super. Super refers to the super class and allow super class constructor or method to be called as well. Sub classes must call super class constructor.

(Refer Slide Time: 9:27)



Polymorphism, polymorphism is concept in java where an object variable can refer to multiple actual data types. For example in the previous case where my vehicle was super class and from that suppose I drive two sub classes one of type car and one of type truck. I can actually have a variable of type vehicle which can refer at some point of time to (9:56) to an object of type car at other time to an object of type truck.

(Refer Slide Time: 10:04)



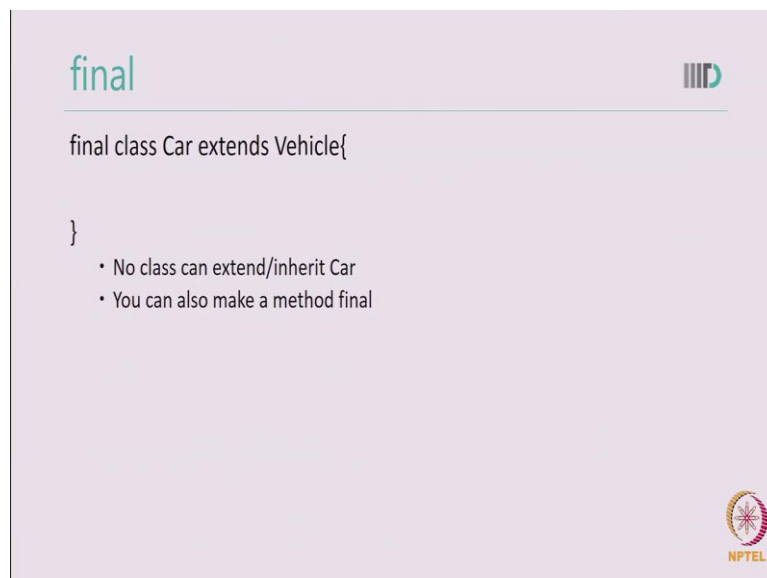
**Dynamic binding**

- Automatic selection of the appropriate methods at runtime
  - Depending on the actual type of the object
    - For public methods
    - If method is private, static, final, or a constructor – static binding

NPTEL

Java determines at the run time how to do the correct binding? So for example there is a method called get wheels count in both truck sub car and car sub class. But if my variable is referring towards truck sub class then the method of the truck sub class will run at the runtime however if the variable is referring to a type of car then the method of car sub class will run at that time.

(Refer Slide Time: 10:36)



**final**

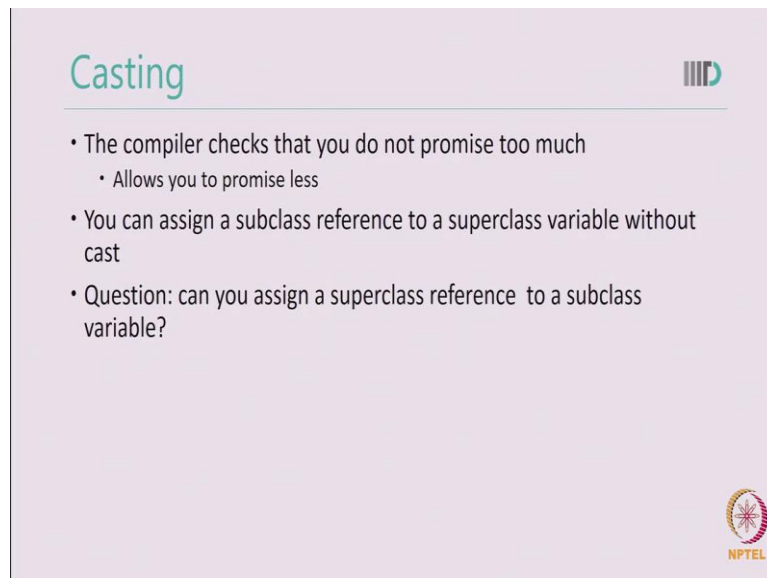
```
final class Car extends Vehicle{  
  
}
```

- No class can extend/inherit Car
- You can also make a method final

NPTEL

If we don't, if we don't want a class to be extended we must declare that class to be final once we declare class to the final cannot be extended and it generates a programming error. Similarly we can also make a method final so that method behaviour cannot be changed.

(Refer Slide Time: 10:57)



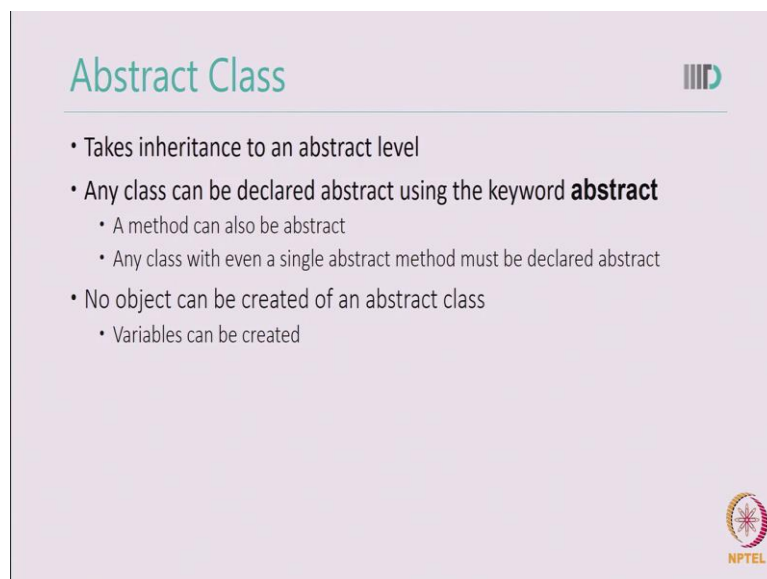
The slide is titled "Casting" in a teal font at the top left. In the top right corner, there is a logo consisting of three vertical bars of increasing height. The main content consists of three bullet points:

- The compiler checks that you do not promise too much
  - Allows you to promise less
- You can assign a subclass reference to a superclass variable without cast
- Question: can you assign a superclass reference to a subclass variable?

In the bottom right corner, there is a circular logo with a star-like pattern and the text "NPTEL" below it.

If we tried to change the reference of one class to another we need to do explicit casting so for example we can assign a sub class reference to a super class variable without casting. However if you want to do it other way down we have to do explicit casting. The compiler checks that you are not promising too much. Which means that what the variable is referring to is the functionality that is implemented and not more than that.

(Refer Slide Time: 11:27)



The slide is titled "Abstract Class" in a teal font at the top left. In the top right corner, there is a logo consisting of three vertical bars of increasing height. The main content consists of three bullet points:

- Takes inheritance to an abstract level
- Any class can be declared abstract using the keyword **abstract**
  - A method can also be abstract
  - Any class with even a single abstract method must be declared abstract
- No object can be created of an abstract class
  - Variables can be created

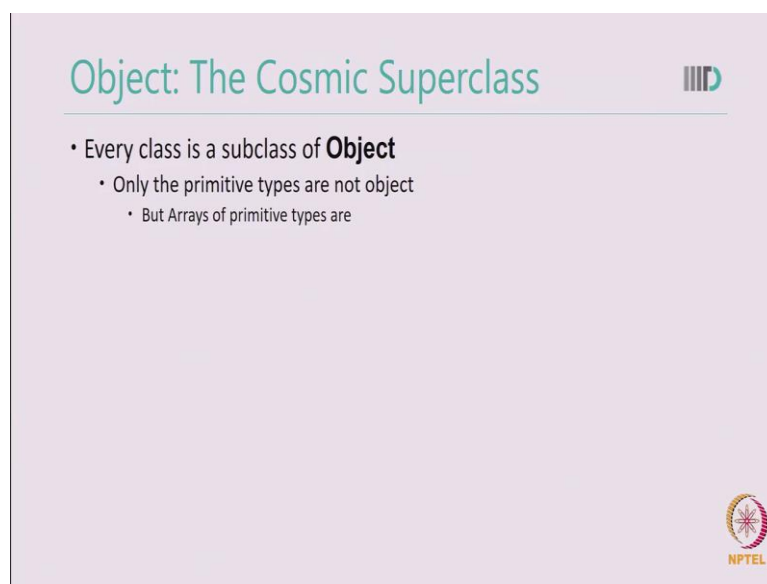
In the bottom right corner, there is a circular logo with a star-like pattern and the text "NPTEL" below it.

Now, let's see abstract classes. Abstract classes take inheritance to another level. (at) In abstract classes we don't define any methods so we only declare that. In order to declare a

class abstract we use the keyword called abstract. Similarly a method can be also an abstract. If a class has even a single abstract method the class must be declared as abstract. Abstract classes means no object can be created out of them and only variables can be created of abstract class.

These variables than refer to the actual instances of sub classes of that particular abstract class. In java we have a class called object which is the super class of all the classes present in java. Which means that whatever class you make in java is automatically a sub class of the super class called object.

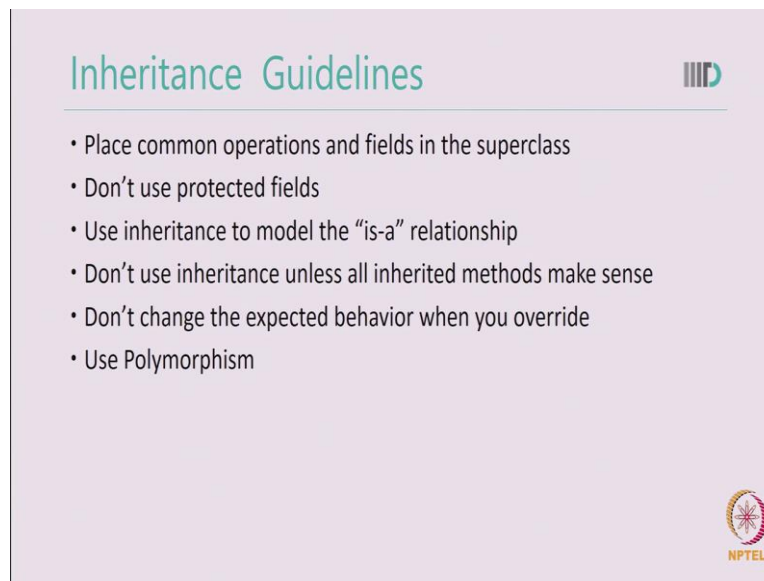
(Refer Slide Time: 12:31)



The slide features a light purple background. At the top left, the title "Object: The Cosmic Superclass" is written in a teal font. To the right of the title is a small icon consisting of three vertical bars of varying heights. Below the title, there is a horizontal line. Underneath the line, there are three bullet points: "• Every class is a subclass of **Object**", "• Only the primitive types are not object", and "• But Arrays of primitive types are". In the bottom right corner, there is a circular logo with a star-like pattern inside, and the text "NPTEL" below it.

Only the primitive types in java are not object. The primitive types are what we studied in the first lecture int, Boolean, etc.

(Refer Slide Time: 12:48)



The slide is titled "Inheritance Guidelines" in a teal font. It features a list of six bullet points: "Place common operations and fields in the superclass", "Don't use protected fields", "Use inheritance to model the 'is-a' relationship", "Don't use inheritance unless all inherited methods make sense", "Don't change the expected behavior when you override", and "Use Polymorphism". The NPTEL logo is visible in the bottom right corner of the slide.

- Place common operations and fields in the superclass
- Don't use protected fields
- Use inheritance to model the "is-a" relationship
- Don't use inheritance unless all inherited methods make sense
- Don't change the expected behavior when you override
- Use Polymorphism

Here are some inheritance guidelines for you. When you are using inheritance place common operation and fields in the super class. Try not to use protected fields. If you want to see whether inheritance is a right approach try to model your problem as a is a relationship. For example car is a vehicle, truck is a vehicle because I can directly see the relationship of is a I can say that yes.

Vehicle may be a super class of car but car even though it has very much common with the jeep. Car is not a jeep and jeep is not a car. So there is no is a relationship. Which means that I should not inherit car from a class jeep or jeep from a class car? The idea is always to make sure that the relationship adjusts where you want to use inheritance. If you cannot see a very clear is a relationship then inheritance may not be the right choice.

Another guide line is to don't use inheritance unless all inherit method make sense. (whi) Let's get back to our example of vehicle and car. In vehicle you should ideally define all the method that can be inherit by a car. So you should not define any method that does not make any sense with reference to the sub class. For example you should only define a method which denotes common functionality. So you can define a method called get wheels count. You can define a method called get seating capacity cause they make sense for all the sub classes of the vehicle. But you should not define a method in a super class which only make sense for certain subclasses but not for another.

Try not to change the expected behaviour when you over write. So if there is a method get wheels count which means that it will return you the number of wheels in a vehicle. Please

keep the same meaning when you over write it in a sub class that is try not to change the implementation to get the seating count from the same method. Java will not stop but this not a good programming practice. So last inheritance guideline is to use polymorphism in your program wherever you can. Thank you!