Lesson 03 Demo 06

Implementing Run Time Polymorphism

Objective: Depicting the concept of polymorphism in Java

Tools required: Eclipse IDE

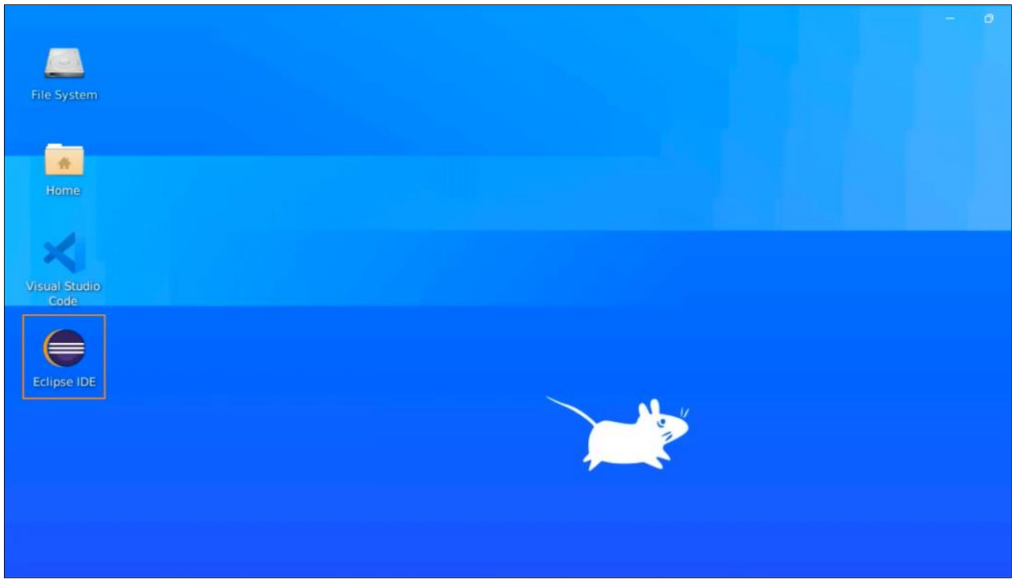
Prerequisites: None

Steps to be followed:

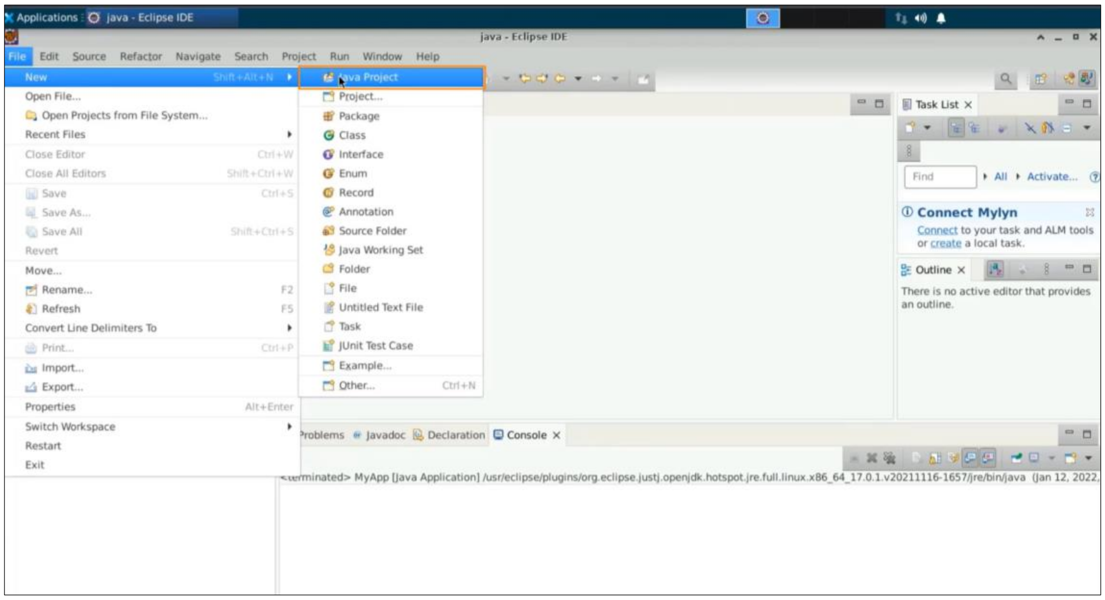
1. Create a class called Polymorphic statement, followed by selecting the main method
2. Write a class called CA with a method named show, and a subclass called CB with its own show method
3. Create a reference variable of the parent to hold the hash code of the object of the child
4. Execute the show method
5. Understand the concept of downcasting

Step 1: Create a class called Polymorphic statement, followed by selecting the main method

1.1 Open the **Eclipse IDE**

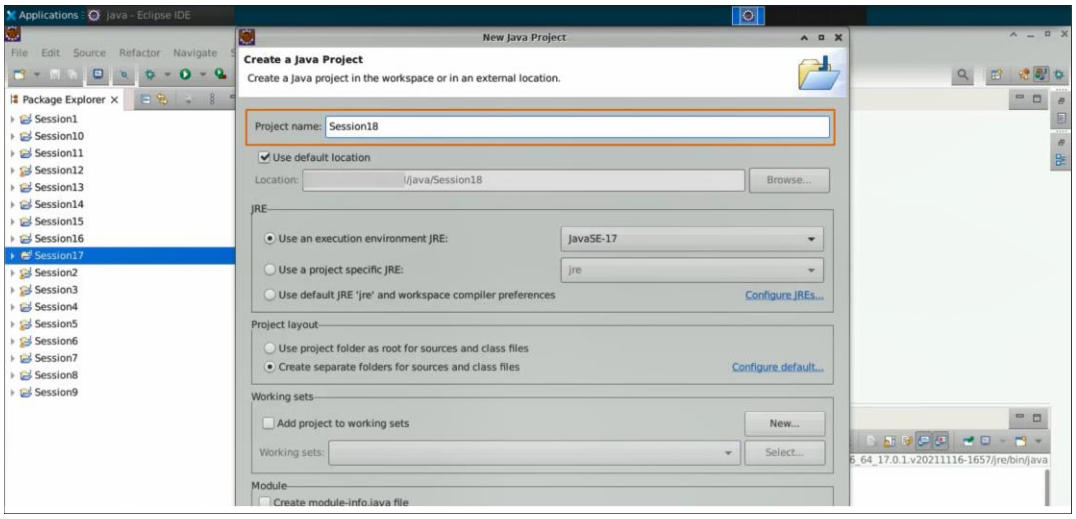


1.2. Select **File**, then **New,** and then **Java project**



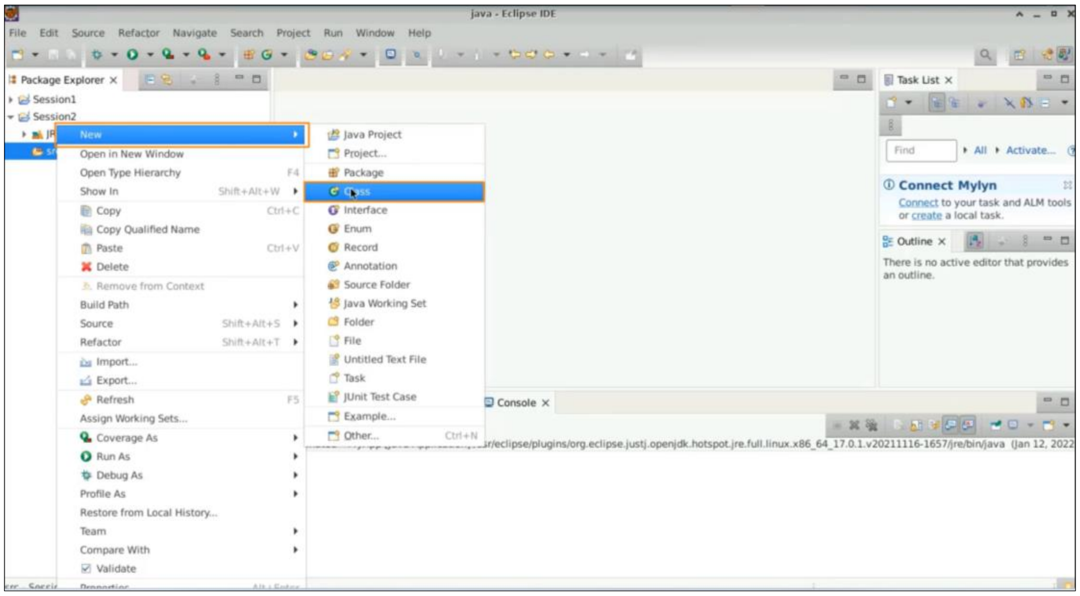
```
<terminated> MyApp [Java Application] /usr/eclipse/plugins/org.eclipse.justj.openjdk.hotspot.jre.full.linux.x86_64_17.0.1.v20211116-1657/jre/bin/java (Jan 12, 2022,
```

1.3 Name the project **“Session18”,** uncheck **“Create a module info dot Java file”**, and press **Finish**

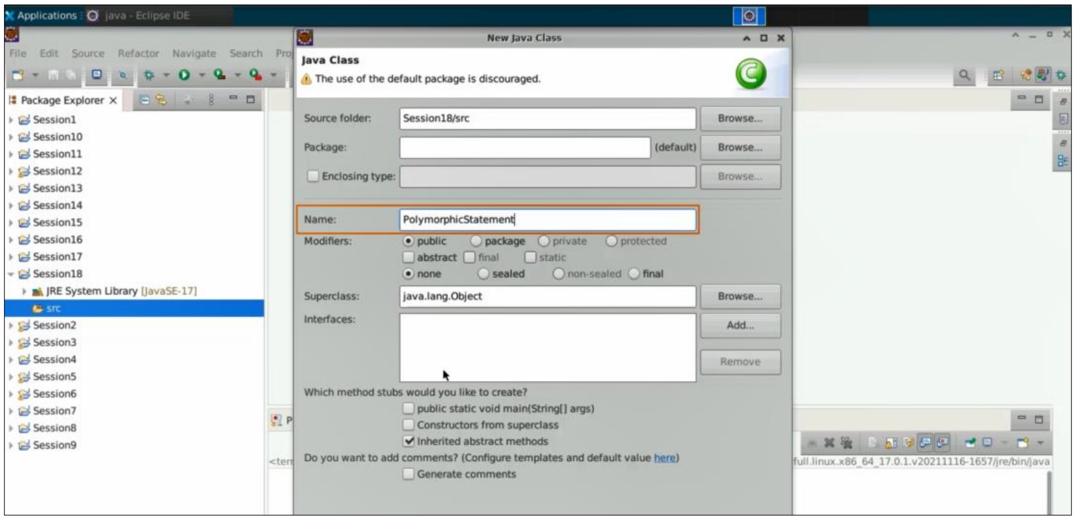


Project name: Session18
Use default location
Location: /java/Session18
JRE
Use an execution environment JRE: JavaSE-17
Use a project specific JRE: jre
Use default JRE 'jre' and workspace compiler preferences
Project layout
Use project folder as root for sources and class files
Create separate folders for sources and class files
Working sets
Add project to working sets
Working sets:
Module
Create module-info.java file

1.4 With a **Session18** on the **src**, do a right-click and create a **new class**



1.5 Name this class as a **PolymorphicStatement**, then select the **main method,** and then select **finish**



The use of the default package is discouraged.

Source folder: Session18/src

Package: (default)

Enclosing type:

Name: PolymorphicStatement

Modifiers: [✓ public] package private protected
abstract final static
[✓ none] sealed non-sealed final

Superclass: java.lang.Object

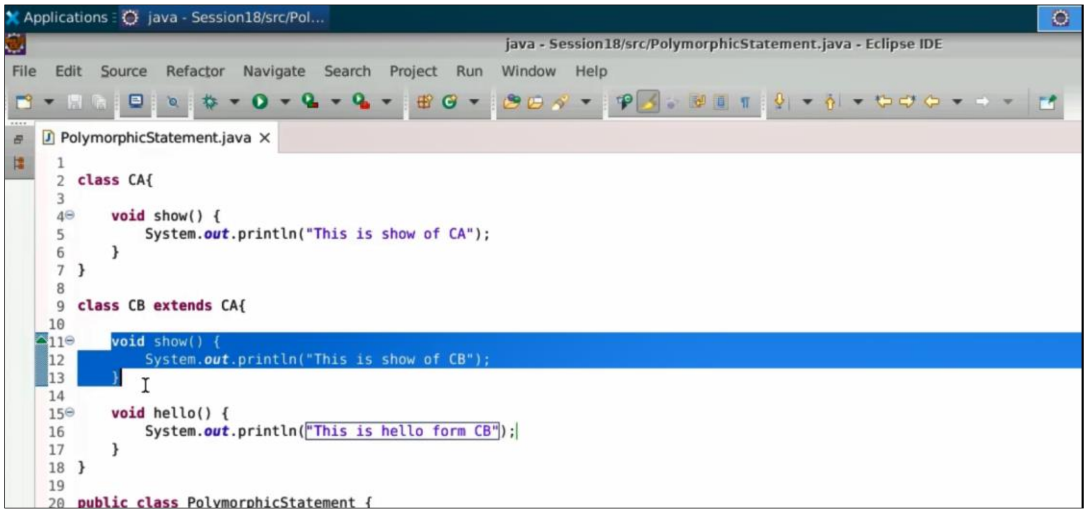
Interfaces:

Which method stubs would you like to create?
- [ ] public static void main(String[] args)
- [ ] Constructors from superclass
- [x] Inherited abstract methods

Do you want to add comments? (Configure templates and default value here)
- [ ] Generate comments

Step 2: Write a class called CA with a method named show, and a subclass called CB with its own show method

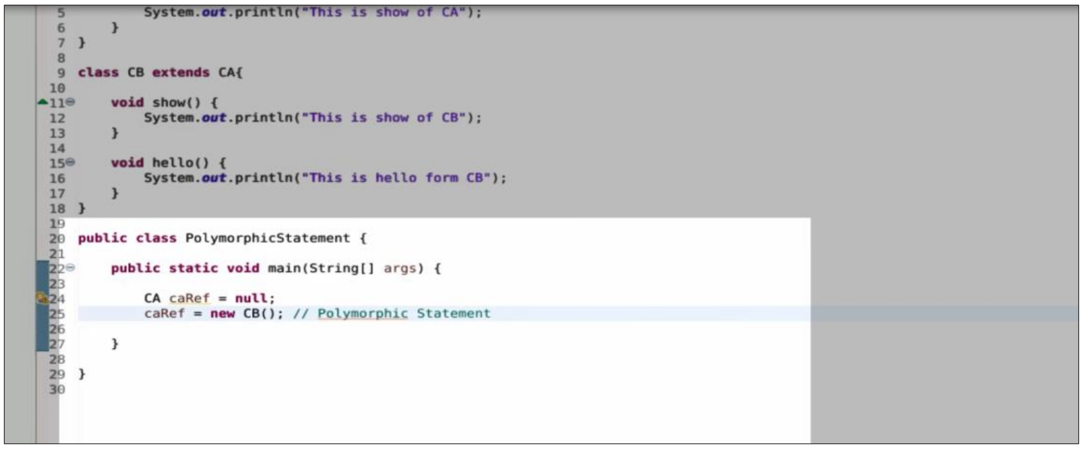
2.1 Let us write a class called **CA** with a method called **show**, which can be written as **this is show of CA**. Next, write a class called **CB**, which is the child of **CA** and has a method called **show**. It can be written as **show of CB** and has a method called **hello**. You can write **this is hello from CB**


Applications
java - Session18/src/Pol...
java - Session18/src/PolymorphicStatement.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
PolymorphicStatement.java

```java

class CA{

    void show() {
        System.out.println("This is show of CA");
    }
}

class CB extends CA{

    void show() {
        System.out.println("This is show of CB");
    }

    void hello() {
        System.out.println("This is hello form CB");
    }
}

public class PolymorphicStatement {
```

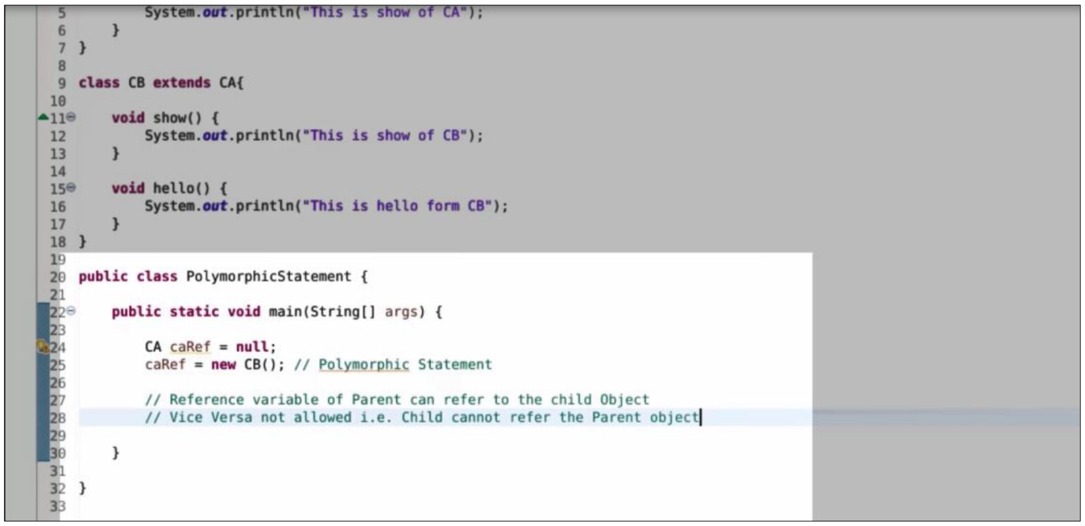
Step 3: Create a reference variable of the parent to hold the hash code of the object of the child

3.1 The polymorphic statement is as follows: you will create the reference variable of the parent, such as **caRef**, which initially points to **nothing (null).** This reference variable will then hold the hash code of the child object. Such a statement is referred to as a polymorphic statement



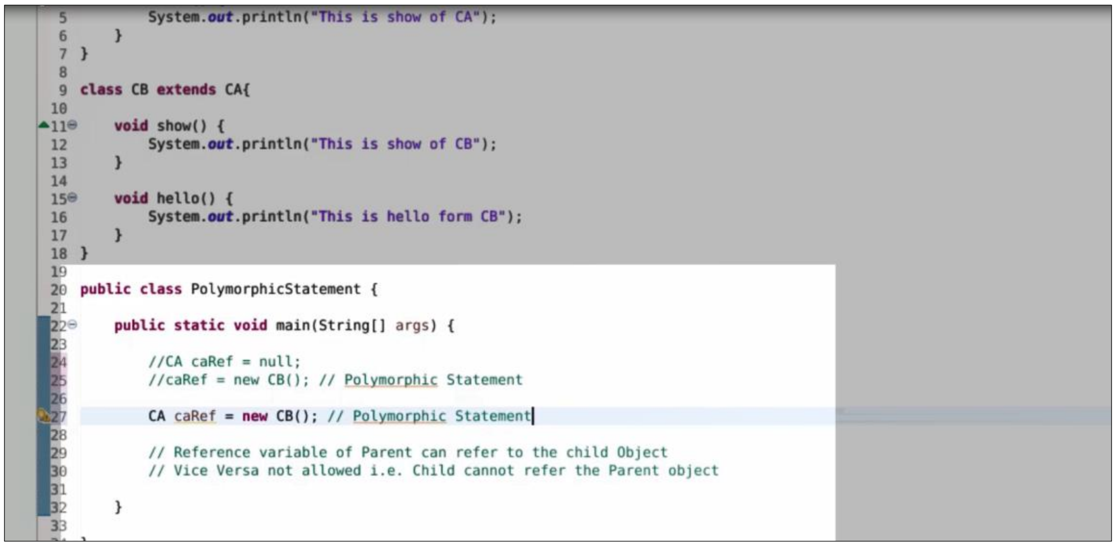
```java
        System.out.println("This is show of CA");
    }
}

class CB extends CA{

    void show() {
        System.out.println("This is show of CB");
    }

    void hello() {
        System.out.println("This is hello form CB");
    }
}

public class PolymorphicStatement {

    public static void main(String[] args) {

        CA caRef = null;
        caRef = new CB(); // Polymorphic Statement

    }

}
```

3.2 The polymorphic statement is the statement where reference variable of parent can refer to the child object. Vice Versa, not allowed that is, the child cannot refer to the parent object



```
        System.out.println("This is show of CA");
    }
}

class CB extends CA{

    void show() {
        System.out.println("This is show of CB");
    }

    void hello() {
        System.out.println("This is hello form CB");
    }
}

public class PolymorphicStatement {

    public static void main(String[] args) {

        CA caRef = null;
        caRef = new CB(); // Polymorphic Statement

        // Reference variable of Parent can refer to the child Object
        // Vice Versa not allowed i.e. Child cannot refer the Parent object

    }

}
```

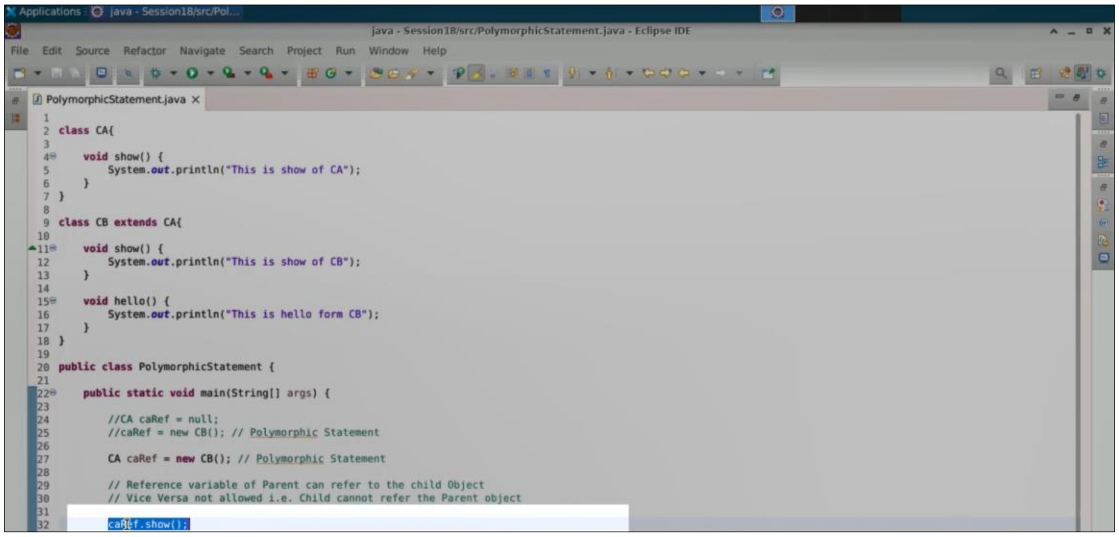
3.3 Now the same statement can also be written in one single line of code. This means, the **CA**, **caRef** is a new object of **CB**, this is a single line where the instruction is written and this is known as the polymorphic statement



```
        System.out.println("This is show of CA");
    }
}

class CB extends CA{

    void show() {
        System.out.println("This is show of CB");
    }

    void hello() {
        System.out.println("This is hello form CB");
    }
}

public class PolymorphicStatement {

    public static void main(String[] args) {

        //CA caRef = null;
        //caRef = new CB(); // Polymorphic Statement

        CA caRef = new CB(); // Polymorphic Statement

        // Reference variable of Parent can refer to the child Object
        // Vice Versa not allowed i.e. Child cannot refer the Parent object

    }

```

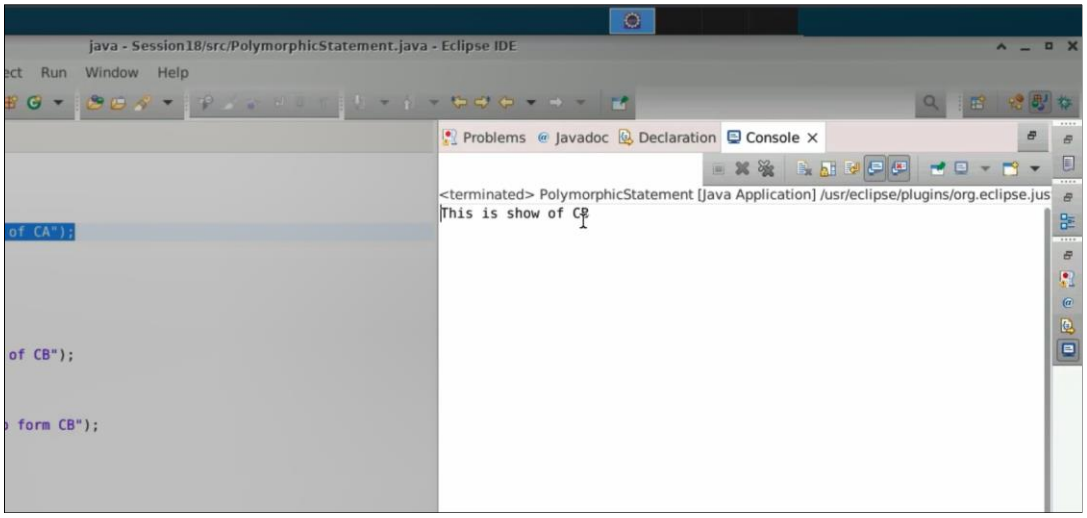
Step 4: Execute the show method

4.1 If you execute the show method, then it means, that you are trying to execute the show method on **CA**



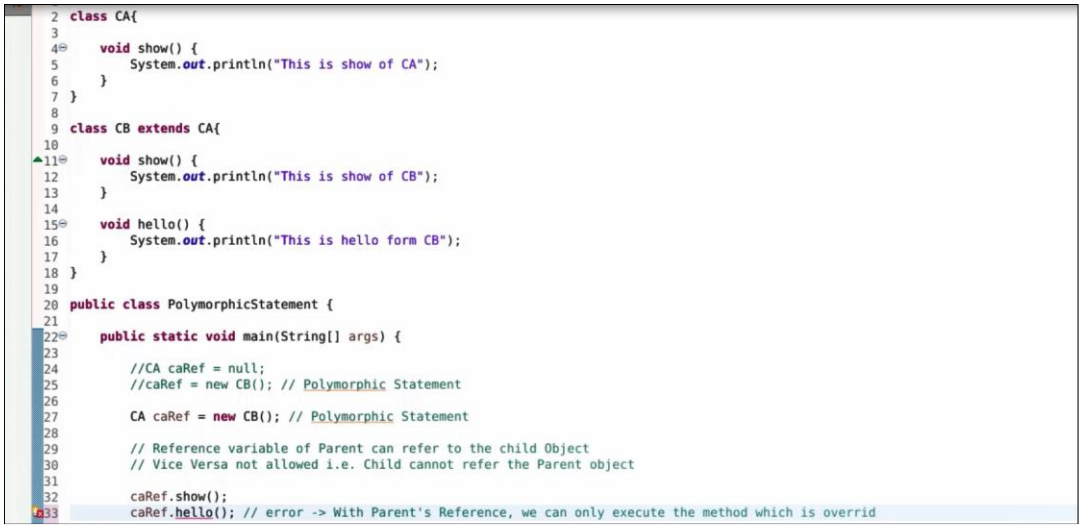
```
class CA{

    void show() {
        System.out.println("This is show of CA");
    }
}

class CB extends CA{

    void show() {
        System.out.println("This is show of CB");
    }

    void hello() {
        System.out.println("This is hello form CB");
    }
}

public class PolymorphicStatement {

    public static void main(String[] args) {

        //CA caRef = null;
        //caRef = new CB(); // Polymorphic Statement

        CA caRef = new CB(); // Polymorphic Statement

        // Reference variable of Parent can refer to the child Object
        // Vice Versa not allowed i.e. Child cannot refer the Parent object

        caRef.show();
```

4.2 This means the output should ideally come as "**this is show of CA**", but what you see is the output comes as "**this is show of CB**".



```
<terminated> PolymorphicStatement [Java Application] /usr/eclipse/plugins/org.eclipse.jus
This is show of CB
```

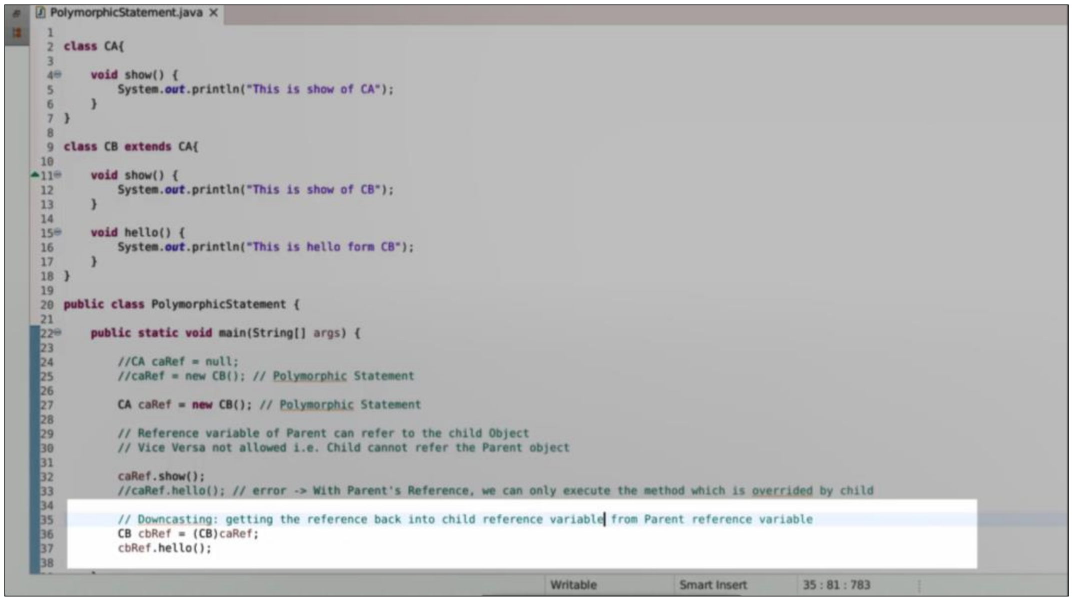
4.3 The reason is very simple: there are two show methods, one belonging to **CA** and one to **CB**. The object is of the class **CB** and not of **CA**, so overriding comes into action with the polymorphic statement. You will not be able to execute the method called hello, as this is an error. With the parent's reference, we can only execute the method that is overridden by the child



```
class CA{

    void show() {
        System.out.println("This is show of CA");
    }
}

class CB extends CA{

    void show() {
        System.out.println("This is show of CB");
    }

    void hello() {
        System.out.println("This is hello form CB");
    }
}

public class PolymorphicStatement {

    public static void main(String[] args) {

        //CA caRef = null;
        //caRef = new CB(); // Polymorphic Statement

        CA caRef = new CB(); // Polymorphic Statement

        // Reference variable of Parent can refer to the child Object
        // Vice Versa not allowed i.e. Child cannot refer the Parent object

        caRef.show();
        caRef.hello(); // error -> With Parent's Reference, we can only execute the method which is overrid
```

Step 5: Understand the concept of downcasting

5.1 If you want to access this method, you can use **downcasting**. You can create a reference variable of **CB** and then cast the parent reference into the child reference variable. With the **CB** reference, you will be able to execute the hello method. This concept is known as **downcasting**, which means getting the reference back into the child reference variable from the parent reference variable

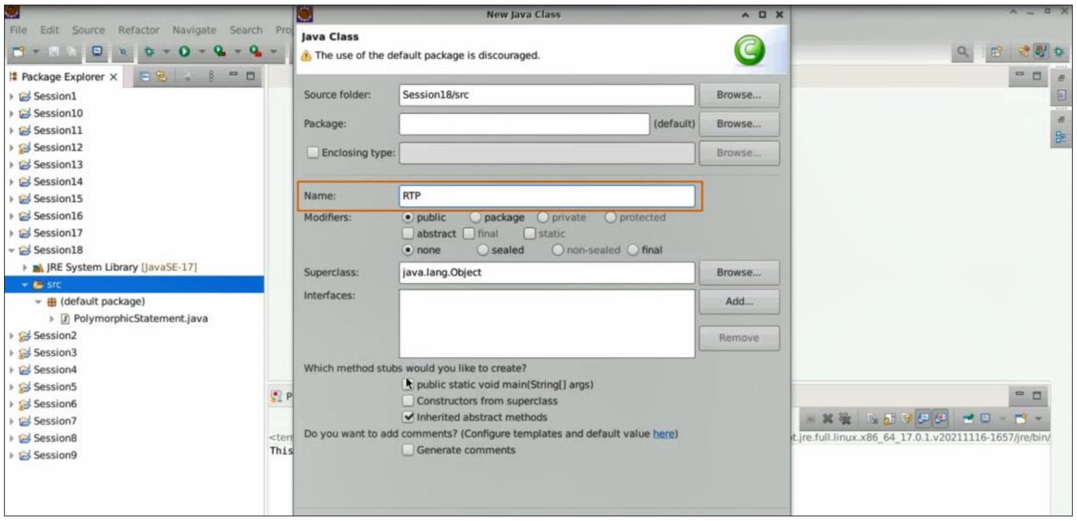


PolymorphicStatement.java

```java

class CA{

    void show() {
        System.out.println("This is show of CA");
    }
}

class CB extends CA{

    void show() {
        System.out.println("This is show of CB");
    }

    void hello() {
        System.out.println("This is hello form CB");
    }
}

public class PolymorphicStatement {

    public static void main(String[] args) {

        //CA caRef = null;
        //caRef = new CB(); // Polymorphic Statement

        CA caRef = new CB(); // Polymorphic Statement

        // Reference variable of Parent can refer to the child Object
        // Vice Versa not allowed i.e. Child cannot refer the Parent object

        caRef.show();
        //caRef.hello(); // error -> With Parent's Reference, we can only execute the method which is overrided by child

        // Downcasting: getting the reference back into child reference variable from Parent reference variable
        CB cbRef = (CB)caRef;
        cbRef.hello();

```

Writable | Smart Insert | 35 : 81 : 783

5.2 Next, let us understand the concept called runtime polymorphism. Let us take another use case, for which you will write a new class called **RuntimePolymorphism** or RTP for short



New Java Class

Java Class

The use of the default package is discouraged.

Source folder: Session18/src

Package: (default)

Enclosing type:

Name: RTP

Modifiers: [✓ public] package private protected; abstract final static; [✓ none] sealed non-sealed final

Superclass: java.lang.Object

Interfaces:

Which method stubs would you like to create?

[✓] public static void main(String[] args)

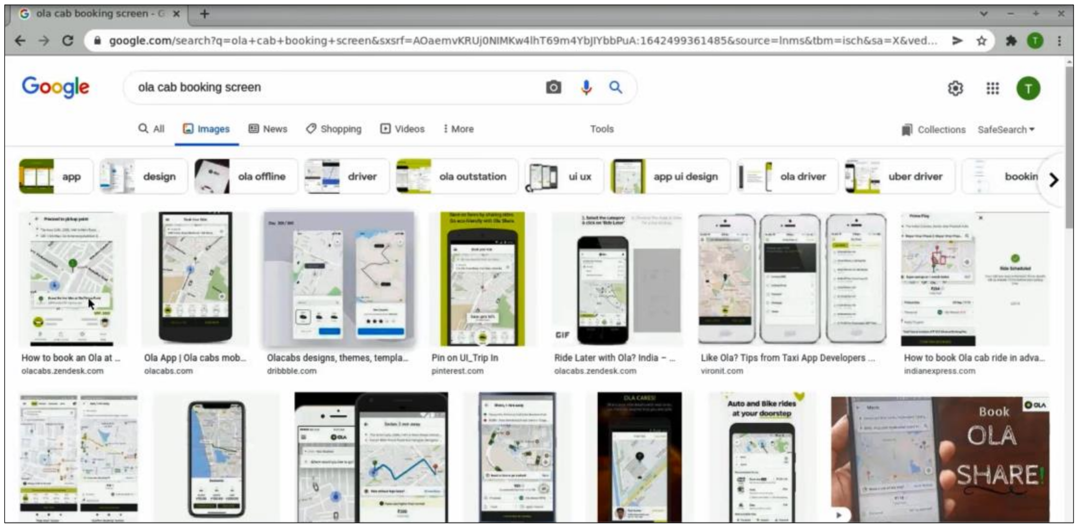
[ ] Constructors from superclass

[✓] Inherited abstract methods

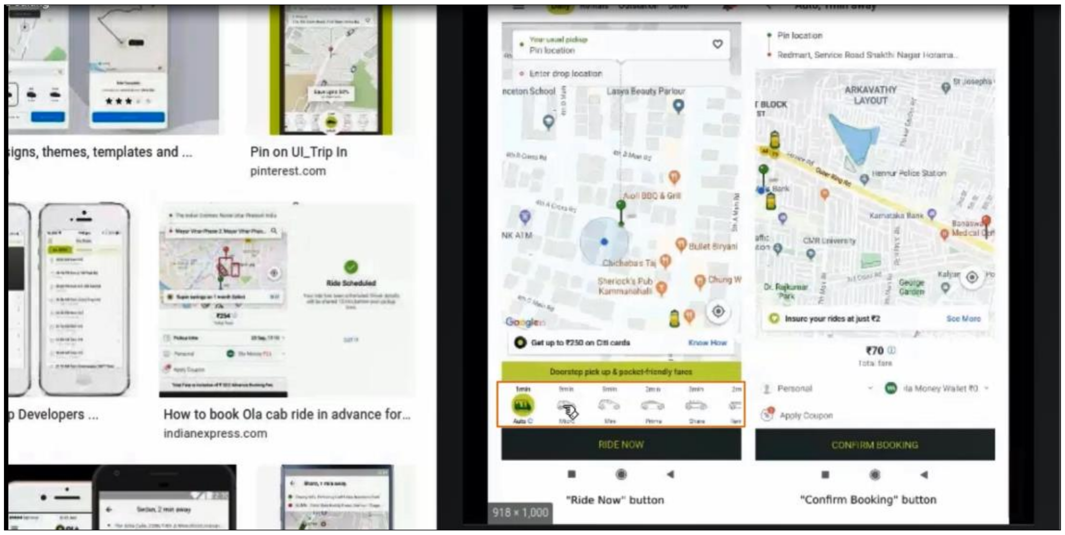
Do you want to add comments? (Configure templates and default value here)

[ ] Generate comments

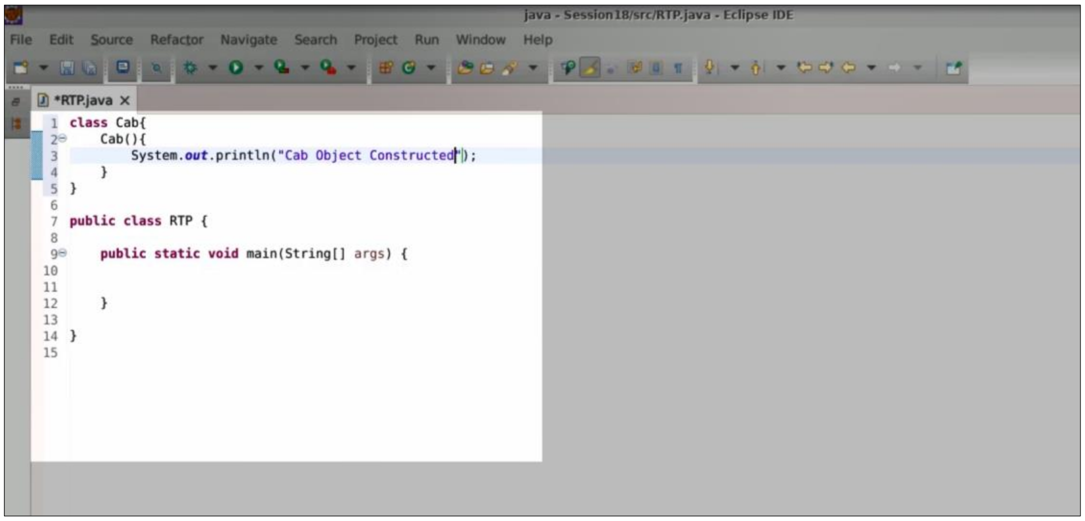
5.3 Next, open up the browser and search for the Ola cab booking screen. Here, you are using a reference called 'ola' in order to book a cab


ola cab booking screen
google.com/search?q=ola+cab+booking+screen
All
Images
News
Shopping
Videos
More
Tools
app
design
ola offline
driver
ola outstation
ui ux
app ui design
ola driver
uber driver
How to book an Ola at ...
Ola App | Ola cabs mob...
Olacabs designs, themes, templa...
Pin on UI_Trip In
Ride Later with Ola? India - ...
Like Ola? Tips from Taxi App Developers ...
How to book Ola cab ride in adva...
Book OLA SHARE!

5.4 As you can notice, there are different types of cabs available such as auto, micro cab, mini cab, shared cab, prime cab, and luxury cab. As a user, you have the option to choose from different cabs and then book them accordingly

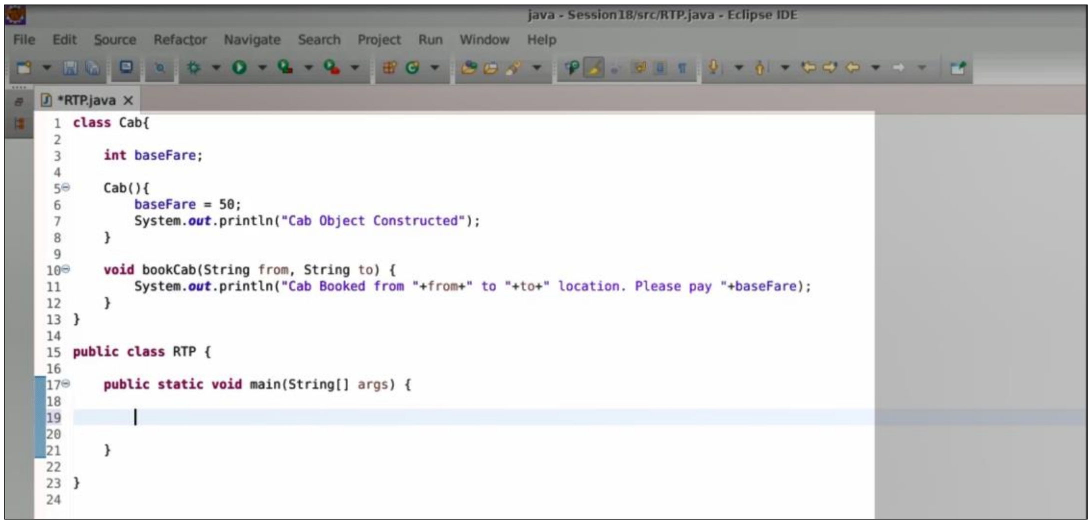


5.5 Let us understand how we can implement a similar use case using the concept of runtime polymorphism. Let us create a parent class called Cab, and within this parent class, there is a constructor that prints out a message indicating that a Cab object has been constructed


java - Session18/src/RTP.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
*RTP.java

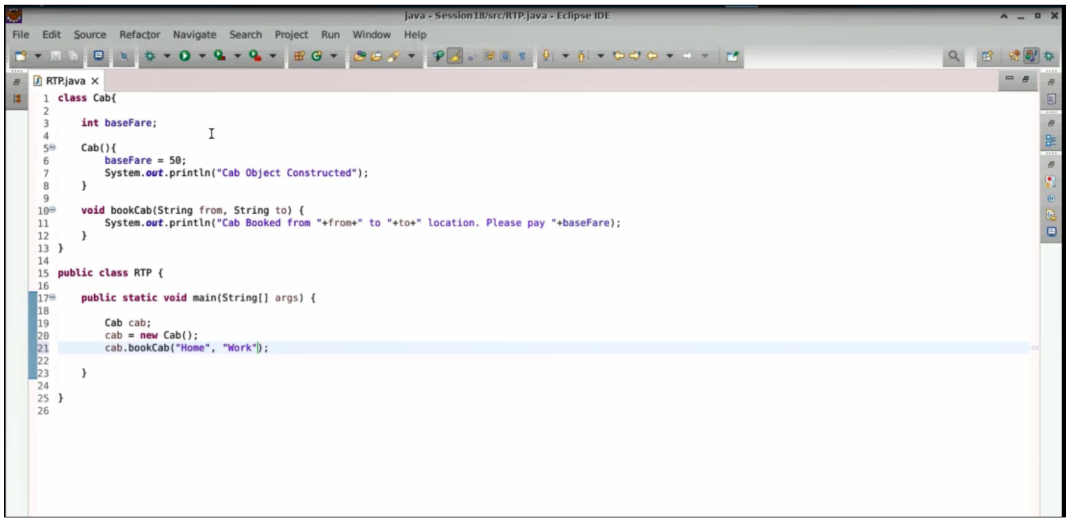
```java
class Cab{
    Cab(){
        System.out.println("Cab Object Constructed");
    }
}

public class RTP {

    public static void main(String[] args) {


    }

}
```

5.6 Let us create some attributes. There is a **bookCab** method that requires you to fill in your source location with strings **'from'** and **'to'**, which indicate where you are traveling from and to. For the Cab class, let us take another attribute called **baseFare**, and set it to 50 bucks. Thus, a Cab object is created. Next, print the message: **System.out.println("Cab booked from " + from + " to " + to + " location. Please pay " + baseFare);**


java - Session18/src/RTP.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
*RTP.java

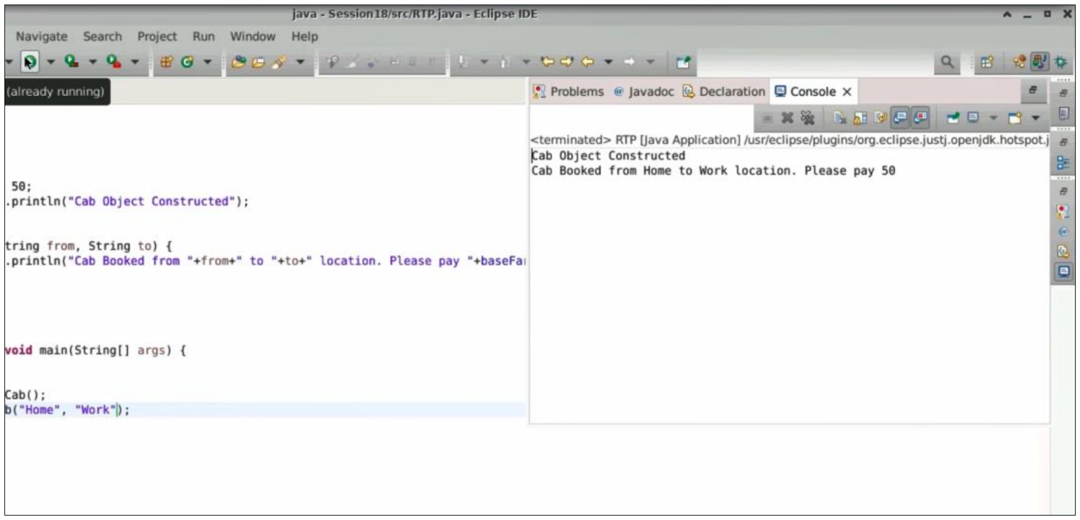
```java
class Cab{

    int baseFare;

    Cab(){
        baseFare = 50;
        System.out.println("Cab Object Constructed");
    }

    void bookCab(String from, String to) {
        System.out.println("Cab Booked from "+from+" to "+to+" location. Please pay "+baseFare);
    }
}

public class RTP {

    public static void main(String[] args) {

        

    }

}
```

5.7 If you want to create an object of Cab, let us define a reference variable **cab**, then write it as cab is a new instance of Cab. Here, instead of one line, you have just written it in two lines. Then, write **cab.bookCab("home", "work");**


java - Session18/src/RTP.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
RTP.java

```java
class Cab{

    int baseFare;

    Cab(){
        baseFare = 50;
        System.out.println("Cab Object Constructed");
    }

    void bookCab(String from, String to) {
        System.out.println("Cab Booked from "+from+" to "+to+" location. Please pay "+baseFare);
    }
}

public class RTP {

    public static void main(String[] args) {

        Cab cab;
        cab = new Cab();
        cab.bookCab("Home", "Work");

    }

}
```

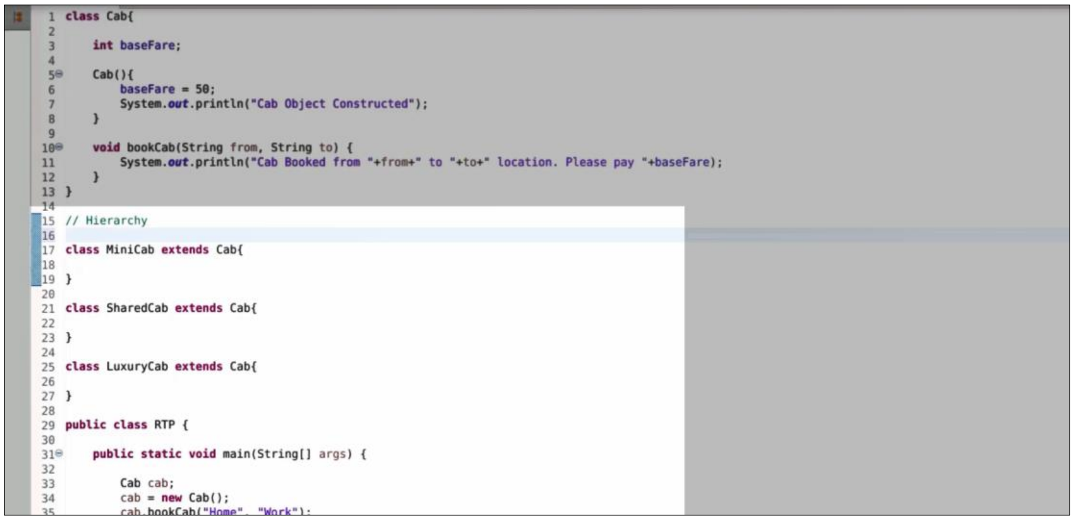
5.8 Run this program. It is a very basic Cab object, which is constructed with the method execution. Then, it shows that the cab is booked from home to work location and asks to please pay 50 bucks. You need to implement inheritance so that you can extend this regular Cab. The cab is a common cab, but it's not decided what kind of cab it is


java - Session18/src/RTP.java - Eclipse IDE
Navigate Search Project Run Window Help
(already running)
Problems Javadoc Declaration Console

```
 50;
.println("Cab Object Constructed");

tring from, String to) {
.println("Cab Booked from "+from+" to "+to+" location. Please pay "+baseFa

void main(String[] args) {

Cab();
b("Home", "Work");
```

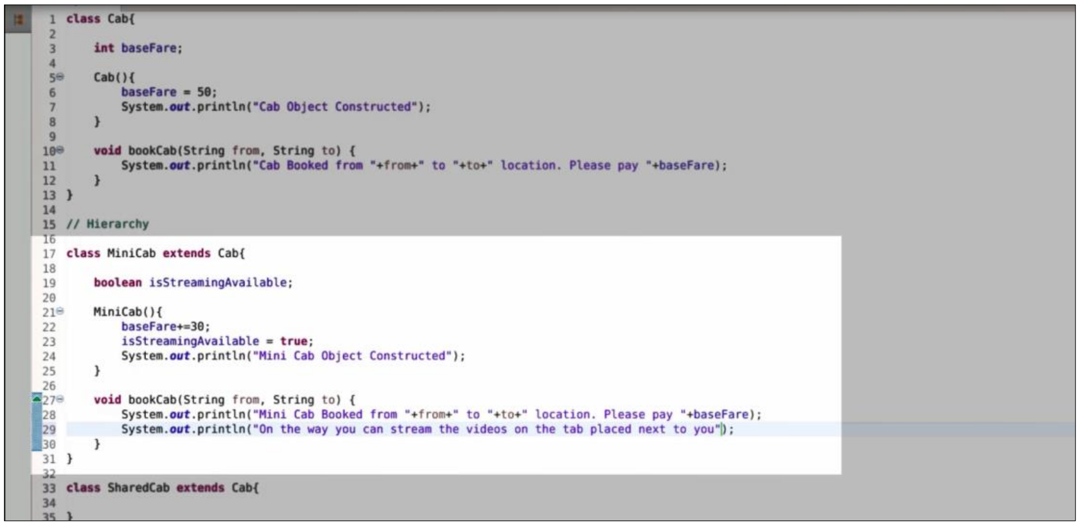
```
<terminated> RTP [Java Application] /usr/eclipse/plugins/org.eclipse.justj.openjdk.hotspot.j
Cab Object Constructed
Cab Booked from Home to Work location. Please pay 50
```

5.9 Let us write a **MiniCab** class, which is the child of Cab. Then, a **SharedCab** class, which is also a child of Cab, and a **LuxuryCab** class, which is another child of Cab. This means there is one parent class with multiple children classes, creating a hierarchical inheritance



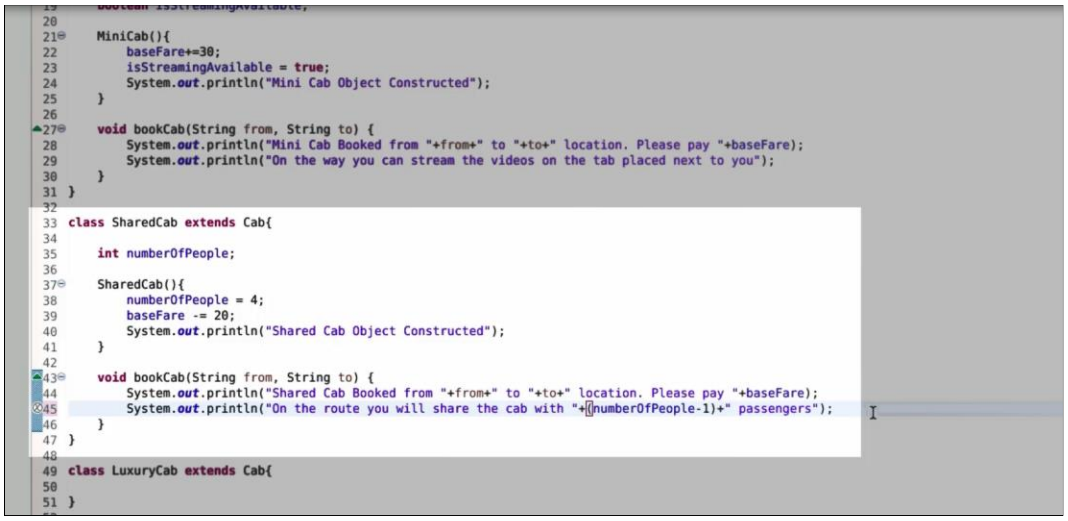
```java
class Cab{

    int baseFare;

    Cab(){
        baseFare = 50;
        System.out.println("Cab Object Constructed");
    }

    void bookCab(String from, String to) {
        System.out.println("Cab Booked from "+from+" to "+to+" location. Please pay "+baseFare);
    }
}

// Hierarchy

class MiniCab extends Cab{

}

class SharedCab extends Cab{

}

class LuxuryCab extends Cab{

}

public class RTP {

    public static void main(String[] args) {

        Cab cab;
        cab = new Cab();
        cab.bookCab("Home", "Work");
```

5.10 In the **MiniCab** class, let us create a constructor. For the **MiniCab**, print out **'MiniCab object constructed'.** Then, override the **bookCab** method to print 'MiniCab booked.' In the constructor, set the base fare to be incremented by 30 bucks. Let us add a boolean attribute **isStreamingAvailable**, which can be set to **true**. You can print **'Is streaming available: true.'** A MiniCab object is created, and you can specify that **"On the way, you can stream videos on the tablet placed next to you"**



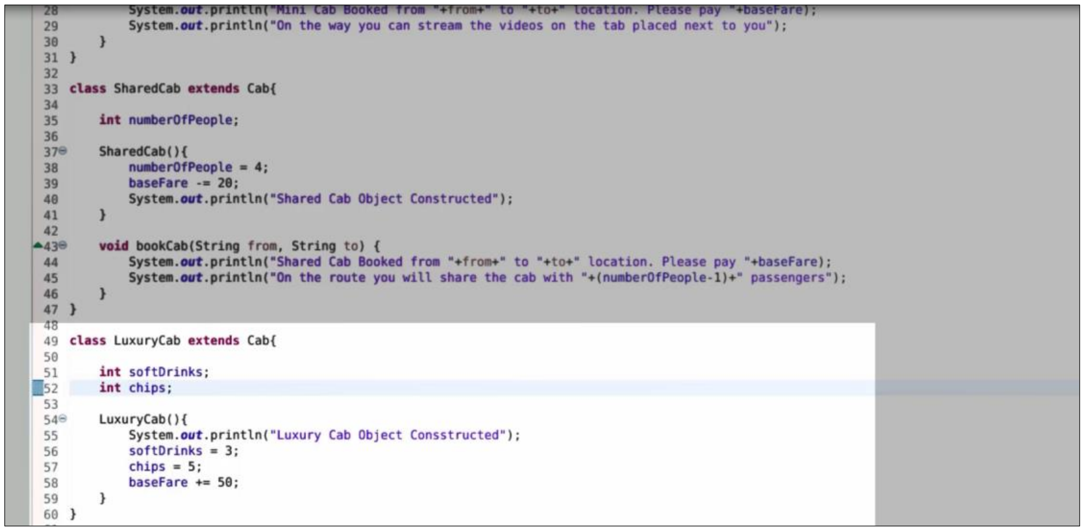
```java
class Cab{

    int baseFare;

    Cab(){
        baseFare = 50;
        System.out.println("Cab Object Constructed");
    }

    void bookCab(String from, String to) {
        System.out.println("Cab Booked from "+from+" to "+to+" location. Please pay "+baseFare);
    }
}

// Hierarchy

class MiniCab extends Cab{

    boolean isStreamingAvailable;

    MiniCab(){
        baseFare+=30;
        isStreamingAvailable = true;
        System.out.println("Mini Cab Object Constructed");
    }

    void bookCab(String from, String to) {
        System.out.println("Mini Cab Booked from "+from+" to "+to+" location. Please pay "+baseFare);
        System.out.println("On the way you can stream the videos on the tab placed next to you");
    }
}

class SharedCab extends Cab{

}
```

5.11 For the **SharedCab**, you can have different attributes, such as the number of people. Let us create the **SharedCab** object and print '**SharedCab object constructed**'. The number of people can be 4, and you can override the same **bookCab** method. However, here the base fare will be decremented by 20 bucks. Print **System.out.println("Shared Cab booked from " + from + " to " + to + " location. Please pay " + baseFare); System.out.println("On the route, you will share the cab with " + (numberOfPeople - 1) + " passengers");**



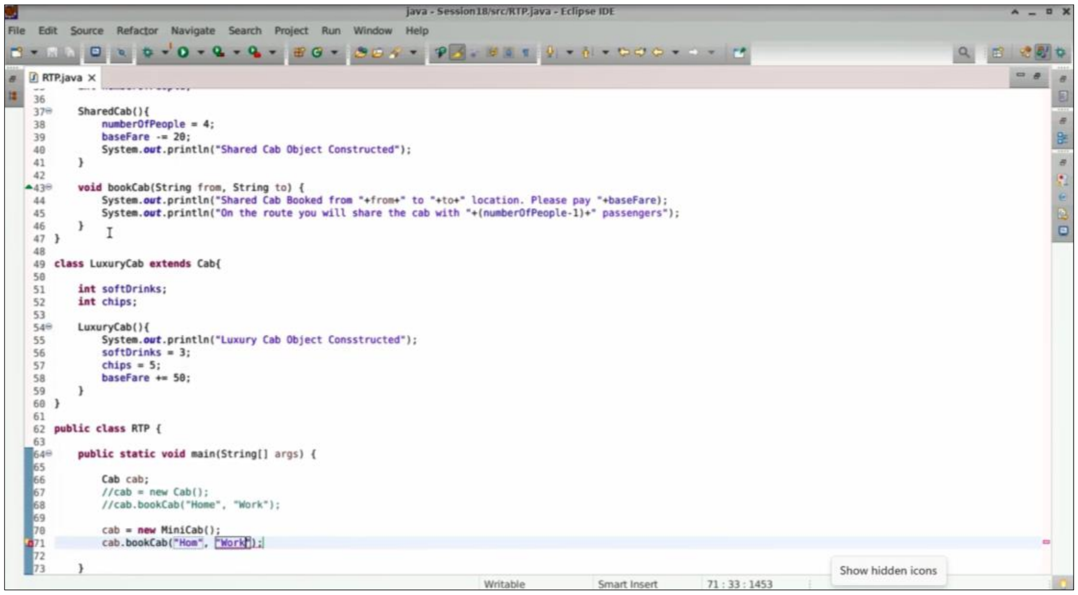
```
    boolean isStreamingAvailable;

    MiniCab(){
        baseFare+=30;
        isStreamingAvailable = true;
        System.out.println("Mini Cab Object Constructed");
    }

    void bookCab(String from, String to) {
        System.out.println("Mini Cab Booked from "+from+" to "+to+" location. Please pay "+baseFare);
        System.out.println("On the way you can stream the videos on the tab placed next to you");
    }
}

class SharedCab extends Cab{

    int numberOfPeople;

    SharedCab(){
        numberOfPeople = 4;
        baseFare -= 20;
        System.out.println("Shared Cab Object Constructed");
    }

    void bookCab(String from, String to) {
        System.out.println("Shared Cab Booked from "+from+" to "+to+" location. Please pay "+baseFare);
        System.out.println("On the route you will share the cab with "+(numberOfPeople-1)+" passengers");
    }
}

class LuxuryCab extends Cab{

}
```

5.12 In the **LuxuryCab**, you will have more features available. Let us set the Boolean attribute **softDrinks** to **true**. Then, set another Boolean attribute **chips** to **true**. Let us create a **LuxuryCab** constructor. In this constructor, set the base fare to be incremented by 50 bucks. You can also initialize these attributes here, such as setting the number of soft drinks to three and the number of chips to five



```
        System.out.println("Mini Cab Booked from "+from+" to "+to+" location. Please pay "+baseFare);
        System.out.println("On the way you can stream the videos on the tab placed next to you");
    }
}

class SharedCab extends Cab{

    int numberOfPeople;

    SharedCab(){
        numberOfPeople = 4;
        baseFare -= 20;
        System.out.println("Shared Cab Object Constructed");
    }

    void bookCab(String from, String to) {
        System.out.println("Shared Cab Booked from "+from+" to "+to+" location. Please pay "+baseFare);
        System.out.println("On the route you will share the cab with "+(numberOfPeople-1)+" passengers");
    }
}

class LuxuryCab extends Cab{

    int softDrinks;
    int chips;

    LuxuryCab(){
        System.out.println("Luxury Cab Object Consstructed");
        softDrinks = 3;
        chips = 5;
        baseFare += 50;
    }
}
```

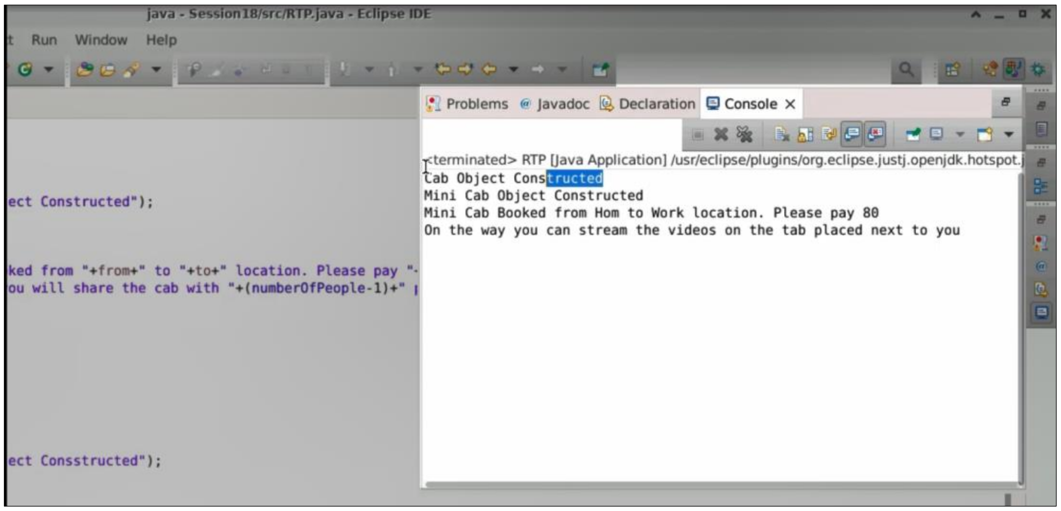
5.13 Now, let us write that the Cab object will not refer to a Cab instance. Instead, the reference variable of the parent class, which is Cab, will now point to a MiniCab object. This is a polymorphic statement. With the Cab reference, you will execute the bookCab method



java - Session18/src/RTP.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
RTP.java
Show hidden icons
Writable
Smart Insert
71 : 33 : 1453

```java

    SharedCab(){
        numberOfPeople = 4;
        baseFare -= 20;
        System.out.println("Shared Cab Object Constructed");
    }

    void bookCab(String from, String to) {
        System.out.println("Shared Cab Booked from "+from+" to "+to+" location. Please pay "+baseFare);
        System.out.println("On the route you will share the cab with "+(numberOfPeople-1)+" passengers");
    }
}

class LuxuryCab extends Cab{

    int softDrinks;
    int chips;

    LuxuryCab(){
        System.out.println("Luxury Cab Object Consstructed");
        softDrinks = 3;
        chips = 5;
        baseFare += 50;
    }
}

public class RTP {

    public static void main(String[] args) {

        Cab cab;
        //cab = new Cab();
        //cab.bookCab("Home", "Work");

        cab = new MiniCab();
        cab.bookCab("Hom", "Work");

    }
```

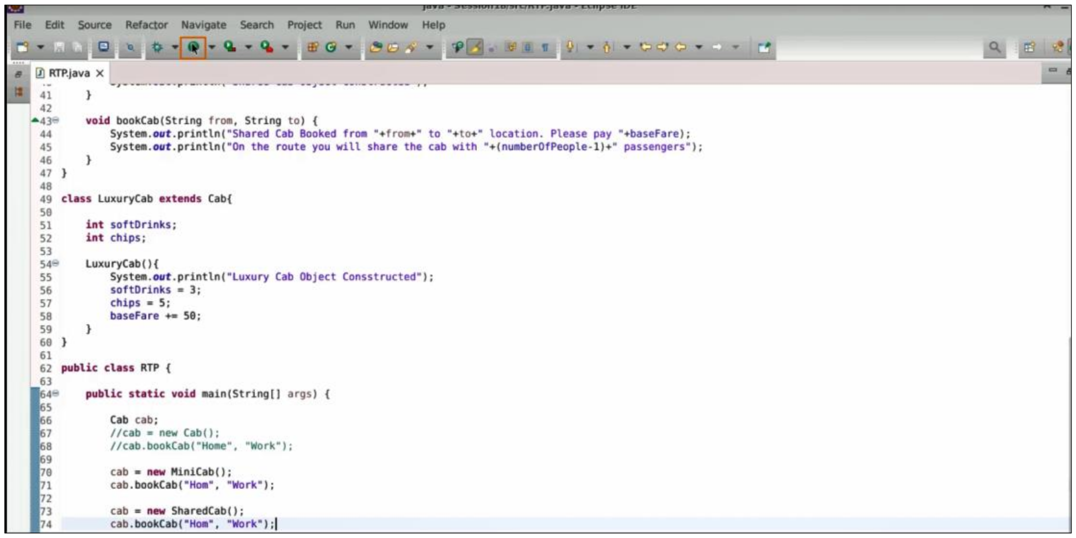
5.14 Run the code. Let us set the **'from'** location to home and the **'to'** location to work. When you book the cab, you will notice that the parent object is constructed first, followed by the child object. Then, you will see a message saying **'Mini Cab booked from Home to Work location, please pay 80 bucks,'** and **'On the way, you can stream videos on the tablet placed next to you'**



```
ect Constructed");

ked from "+from+" to "+to+" location. Please pay "
ou will share the cab with "+(numberOfPeople-1)+" 

ect Consstructed");
```

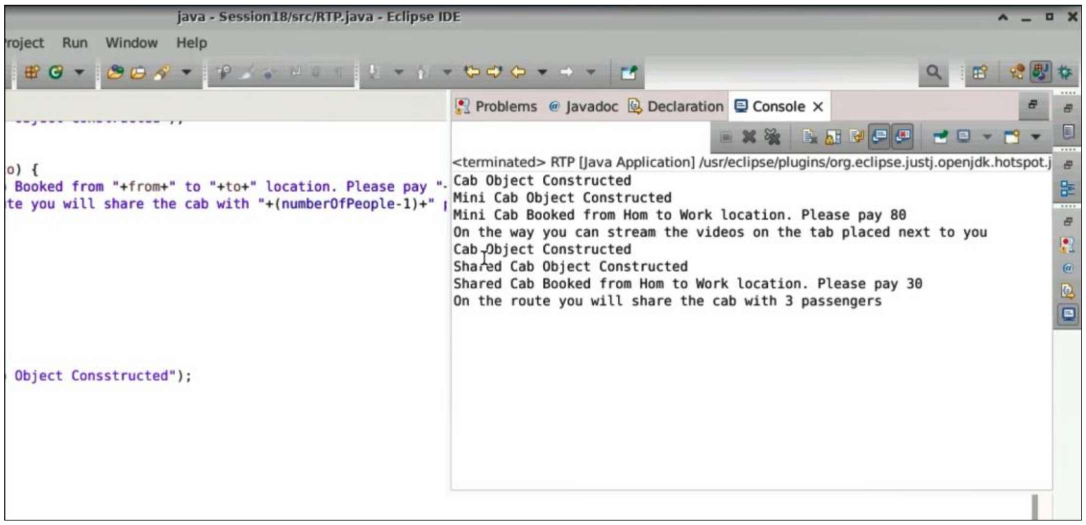
```
<terminated> RTP [Java Application] /usr/eclipse/plugins/org.eclipse.justj.openjdk.hotspot.j
Cab Object Constructed
Mini Cab Object Constructed
Mini Cab Booked from Hom to Work location. Please pay 80
On the way you can stream the videos on the tab placed next to you
```

5.15 In case you wish to change, you can say the same reference variable cab can also refer to the shared cab



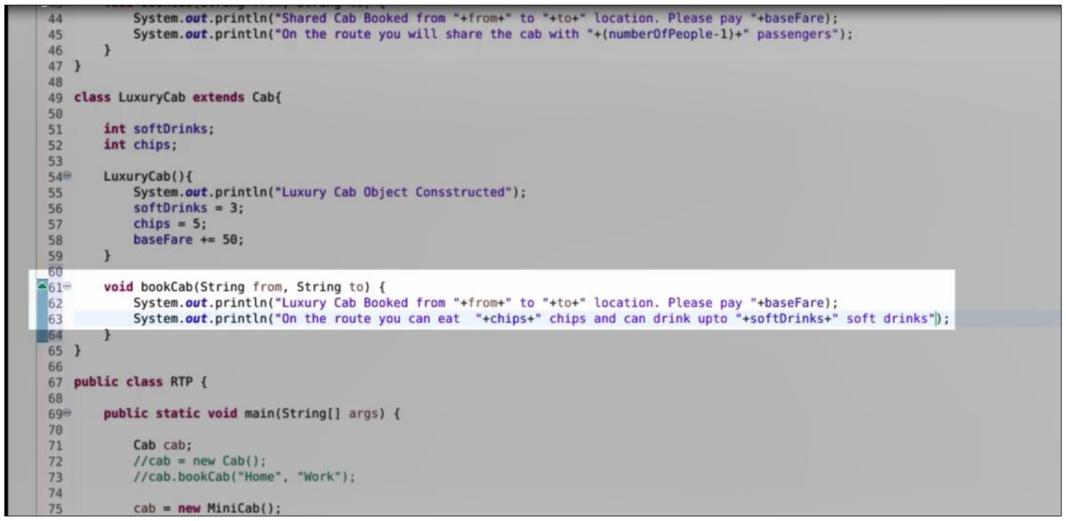
```java
    }

    void bookCab(String from, String to) {
        System.out.println("Shared Cab Booked from "+from+" to "+to+" location. Please pay "+baseFare);
        System.out.println("On the route you will share the cab with "+(numberOfPeople-1)+" passengers");
    }
}

class LuxuryCab extends Cab{

    int softDrinks;
    int chips;

    LuxuryCab(){
        System.out.println("Luxury Cab Object Consstructed");
        softDrinks = 3;
        chips = 5;
        baseFare += 50;
    }
}

public class RTP {

    public static void main(String[] args) {

        Cab cab;
        //cab = new Cab();
        //cab.bookCab("Home", "Work");

        cab = new MiniCab();
        cab.bookCab("Hom", "Work");

        cab = new SharedCab();
        cab.bookCab("Hom", "Work");
```

5.16 Then if you execute the same method on the book cab, you will notice that there is another cab object and the shared object, so this is the shared cab that is booked, were you can share the Cab with three passengers



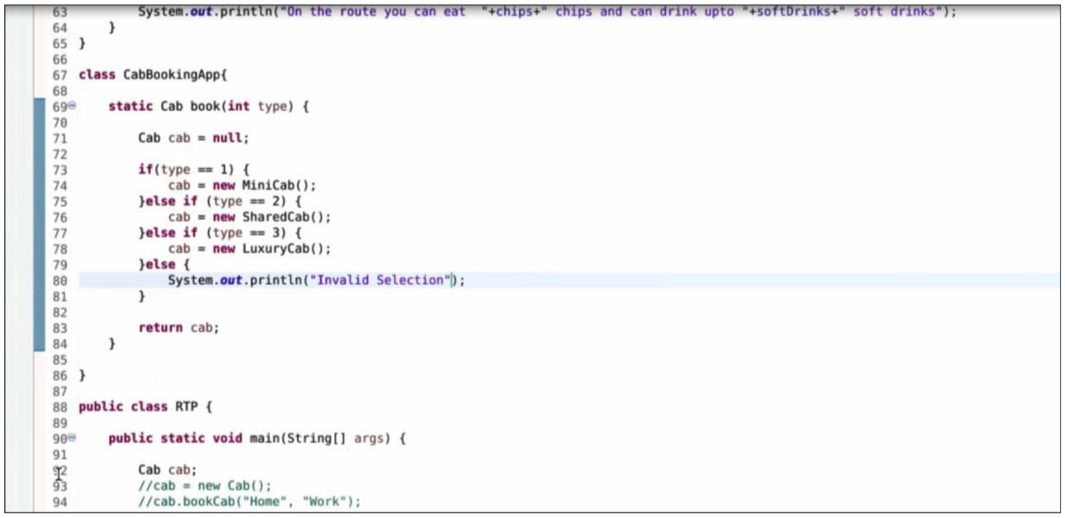
```
<terminated> RTP [Java Application] /usr/eclipse/plugins/org.eclipse.justj.openjdk.hotspot.j
Cab Object Constructed
Mini Cab Object Constructed
Mini Cab Booked from Hom to Work location. Please pay 80
On the way you can stream the videos on the tab placed next to you
Cab Object Constructed
Shared Cab Object Constructed
Shared Cab Booked from Hom to Work location. Please pay 30
On the route you will share the cab with 3 passengers
```

5.17 Let us now understand how you can implement the exact functionality of this cab booking app, including making a selection. When a luxury cab is booked, on the route, you can eat chips and drink up to the specified number of soft drinks. And that is it



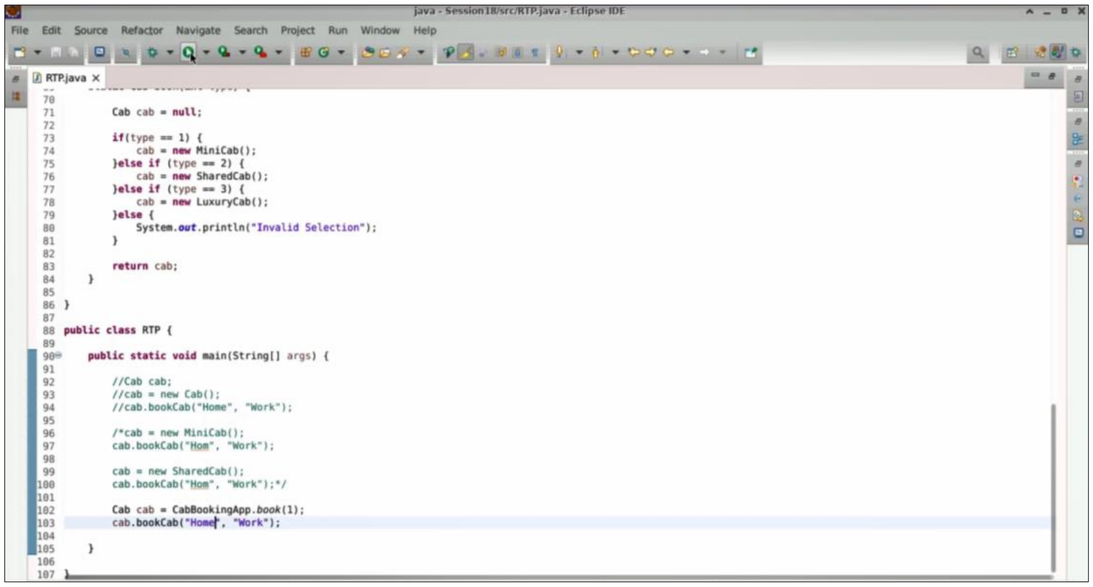
```
        System.out.println("Shared Cab Booked from "+from+" to "+to+" location. Please pay "+baseFare);
        System.out.println("On the route you will share the cab with "+(numberOfPeople-1)+" passengers");
    }
}

class LuxuryCab extends Cab{

    int softDrinks;
    int chips;

    LuxuryCab(){
        System.out.println("Luxury Cab Object Consstructed");
        softDrinks = 3;
        chips = 5;
        baseFare += 50;
    }

    void bookCab(String from, String to) {
        System.out.println("Luxury Cab Booked from "+from+" to "+to+" location. Please pay "+baseFare);
        System.out.println("On the route you can eat  "+chips+" chips and can drink upto "+softDrinks+" soft drinks");
    }
}

public class RTP {

    public static void main(String[] args) {

        Cab cab;
        //cab = new Cab();
        //cab.bookCab("Home", "Work");

        cab = new MiniCab();
```

5.18 In another class, it goes like this: the cab booking app. You can write a method called **bookCab** and pass the type as an integer, making this method static. Here, the cab is initialized to null. Let us specify that if the type is equal to 1, then let the cab be an object of **MiniCab**. Else if the type is 2, then the cab should be an object of **SharedCab**. And if the type is 3, then the same reference variable should point to a **LuxuryCab**. This is known as runtime polymorphism, where dynamically the same reference variable refers to a **MiniCab**, **SharedCab**, or **LuxuryCab**



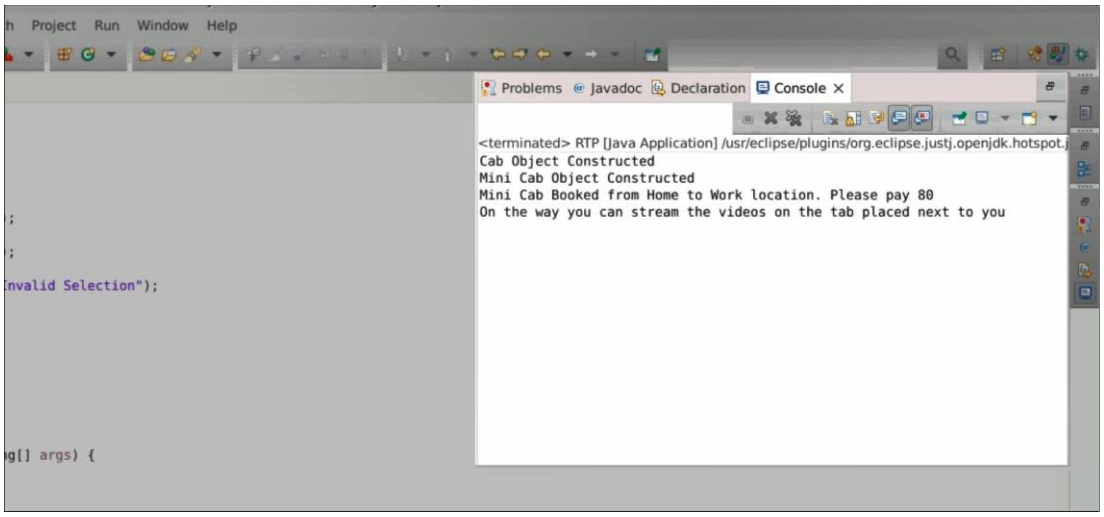
```
        System.out.println("On the route you can eat  "+chips+" chips and can drink upto "+softDrinks+" soft drinks");
    }
}

class CabBookingApp{

    static Cab book(int type) {

        Cab cab = null;

        if(type == 1) {
            cab = new MiniCab();
        }else if (type == 2) {
            cab = new SharedCab();
        }else if (type == 3) {
            cab = new LuxuryCab();
        }else {
            System.out.println("Invalid Selection");
        }

        return cab;
    }

}

public class RTP {

    public static void main(String[] args) {

        Cab cab;
        //cab = new Cab();
        //cab.bookCab("Home", "Work");
```

5.19 Let us comment out this entire part. Let us define cab as the name of the **CabBookingApp** class and call **CabBookingApp.book(1).** When you call **CabBookingApp.book(1)**, you will create a **MiniCab** object and return this cab. Let us execute **bookCab** from 'home' to 'work'


java - Session18/src/RTP.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
RTP.java

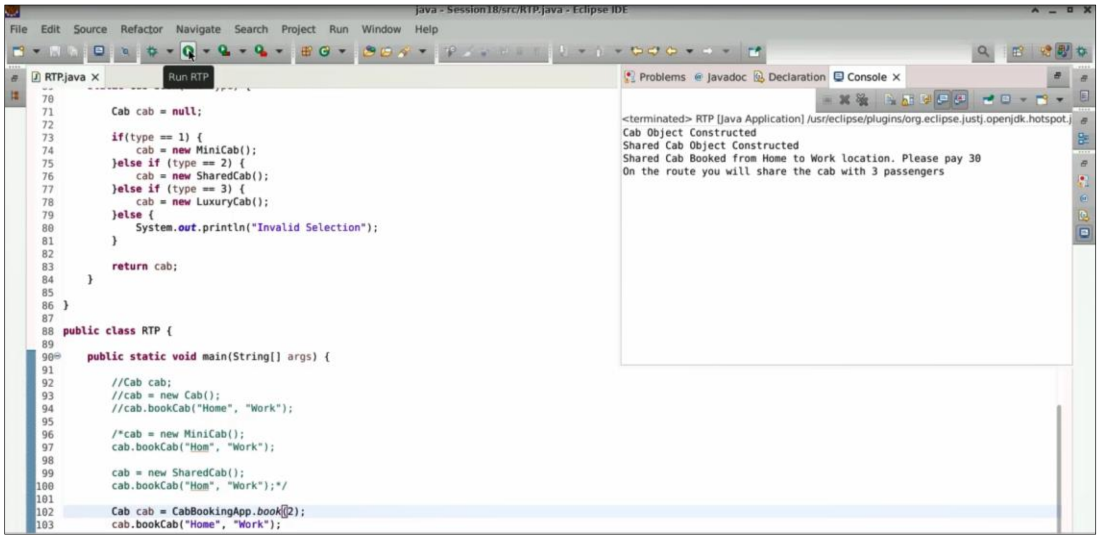
```
        Cab cab = null;

        if(type == 1) {
            cab = new MiniCab();
        }else if (type == 2) {
            cab = new SharedCab();
        }else if (type == 3) {
            cab = new LuxuryCab();
        }else {
            System.out.println("Invalid Selection");
        }

        return cab;
    }

}

public class RTP {

    public static void main(String[] args) {

        //Cab cab;
        //cab = new Cab();
        //cab.bookCab("Home", "Work");

        /*cab = new MiniCab();
        cab.bookCab("Hom", "Work");

        cab = new SharedCab();
        cab.bookCab("Hom", "Work");*/

        Cab cab = CabBookingApp.book(1);
        cab.bookCab("Home", "Work");

    }

}
```

5.20 When you run this code you will notice that it shows, there is a minicab that is booked.



```
<terminated> RTP [Java Application] /usr/eclipse/plugins/org.eclipse.justj.openjdk.hotspot.j
Cab Object Constructed
Mini Cab Object Constructed
Mini Cab Booked from Home to Work location. Please pay 80
On the way you can stream the videos on the tab placed next to you
```

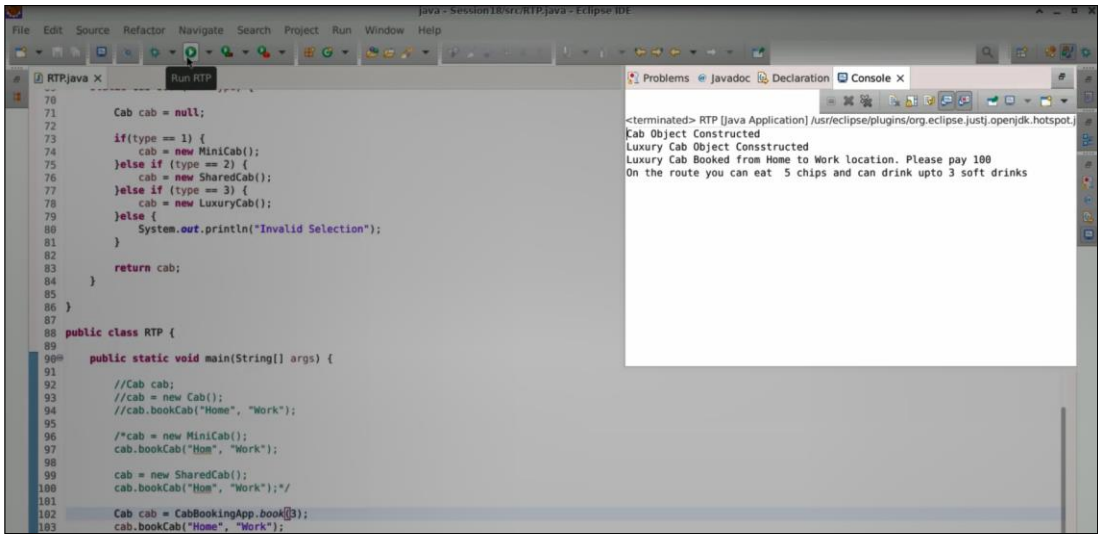
5.21 The moment you pass the value as two, it will change the behavior at runtime, and it shows that it is a shared cab that is booked


java - Session18/src/RTP.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
RTP.java
Run RTP
Problems Javadoc Declaration Console

```
        Cab cab = null;

        if(type == 1) {
            cab = new MiniCab();
        }else if (type == 2) {
            cab = new SharedCab();
        }else if (type == 3) {
            cab = new LuxuryCab();
        }else {
            System.out.println("Invalid Selection");
        }

        return cab;
    }

}

public class RTP {

    public static void main(String[] args) {

        //Cab cab;
        //cab = new Cab();
        //cab.bookCab("Home", "Work");

        /*cab = new MiniCab();
        cab.bookCab("Hom", "Work");

        cab = new SharedCab();
        cab.bookCab("Hom", "Work");*/

        Cab cab = CabBookingApp.book(2);
        cab.bookCab("Home", "Work");
```

```
<terminated> RTP [Java Application] /usr/eclipse/plugins/org.eclipse.justj.openjdk.hotspot.j
Cab Object Constructed
Shared Cab Object Constructed
Shared Cab Booked from Home to Work location. Please pay 30
On the route you will share the cab with 3 passengers
```

5.22 If you pass three, it will give you a luxury cab booking. Hence, runtime polymorphism is implemented with the help of inheritance and the overriding fundamentals


java - Session18/src/RTP.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
RTP.java
Run RTP
Problems Javadoc Declaration Console

```
        Cab cab = null;

        if(type == 1) {
            cab = new MiniCab();
        }else if (type == 2) {
            cab = new SharedCab();
        }else if (type == 3) {
            cab = new LuxuryCab();
        }else {
            System.out.println("Invalid Selection");
        }

        return cab;
    }

}

public class RTP {

    public static void main(String[] args) {

        //Cab cab;
        //cab = new Cab();
        //cab.bookCab("Home", "Work");

        /*cab = new MiniCab();
        cab.bookCab("Hom", "Work");

        cab = new SharedCab();
        cab.bookCab("Hom", "Work");*/

        Cab cab = CabBookingApp.book(3);
        cab.bookCab("Home", "Work");
```

```
<terminated> RTP [Java Application] /usr/eclipse/plugins/org.eclipse.justj.openjdk.hotspot.j
Cab Object Constructed
Luxury Cab Object Consstructed
Luxury Cab Booked from Home to Work location. Please pay 100
On the route you can eat  5 chips and can drink upto 3 soft drinks
```

By following the above steps, you have successfully implemented Run Time Polymorphism and understood its working with an example in Java.