Lesson 03 Demo 03

Using Inheritance in Java

Objective: The Necessity and Significance of Using Inheritance in Java

Tools: Eclipse IDE

Prerequisites: None

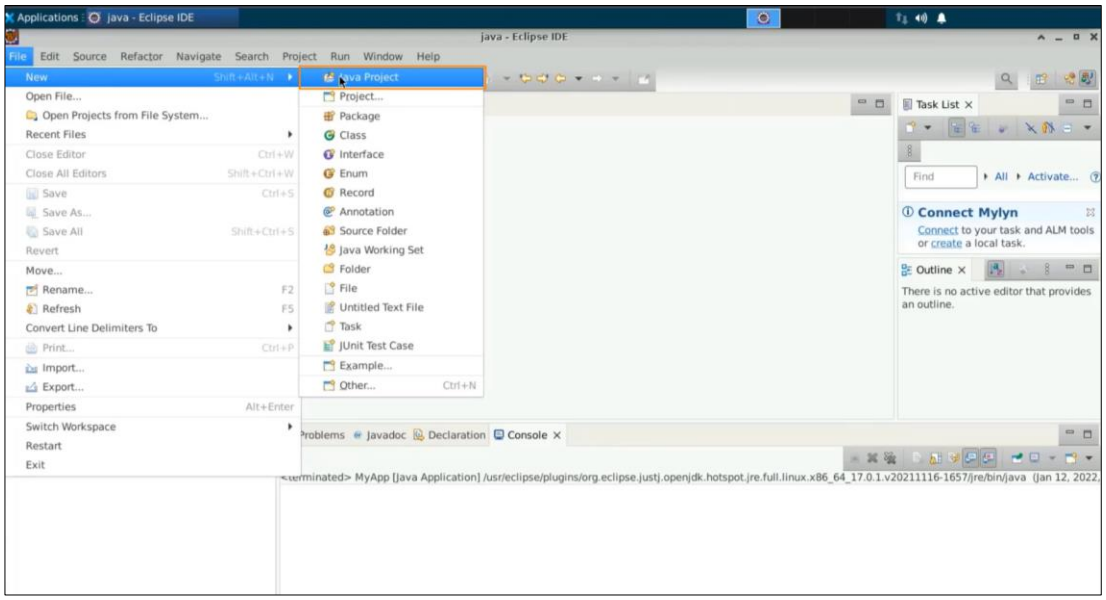
Steps to be followed:

1. Open Eclipse IDE, and open a new Java project and a class
2. Create a class with three attributes
3. Create an object using the new operator and execute the code
4. Create an object with the default constructor
5. Implement the concept of sub-class and superclass
6. Access the attributes of classes
7. Execute the code with example data
8. Understand inheritance rules, their importance, and execute sample code
9. Create a default constructor and parameterized constructor
10. Initialize the arrays

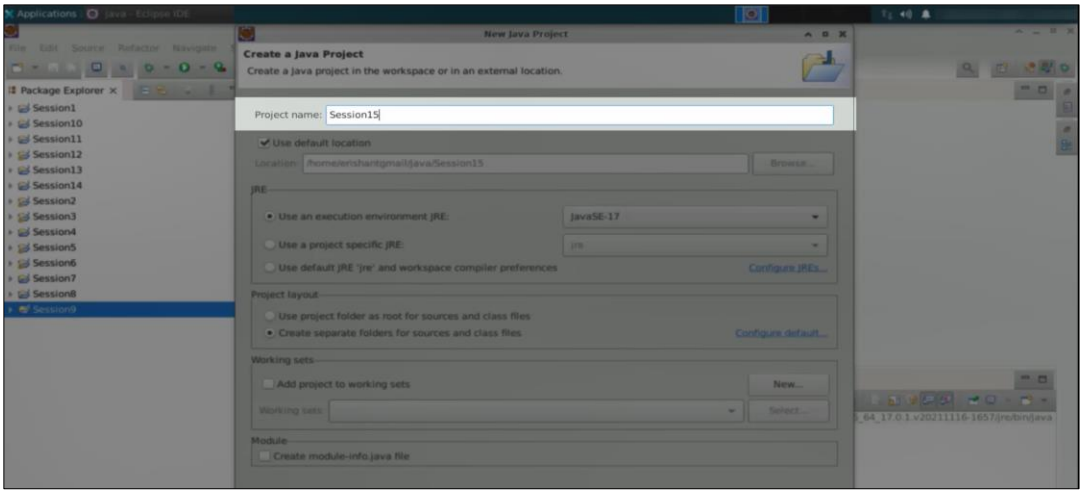
Step 1: Open Eclipse IDE, and open a new Java project and a class

1.1 Open the **Eclipse IDE**

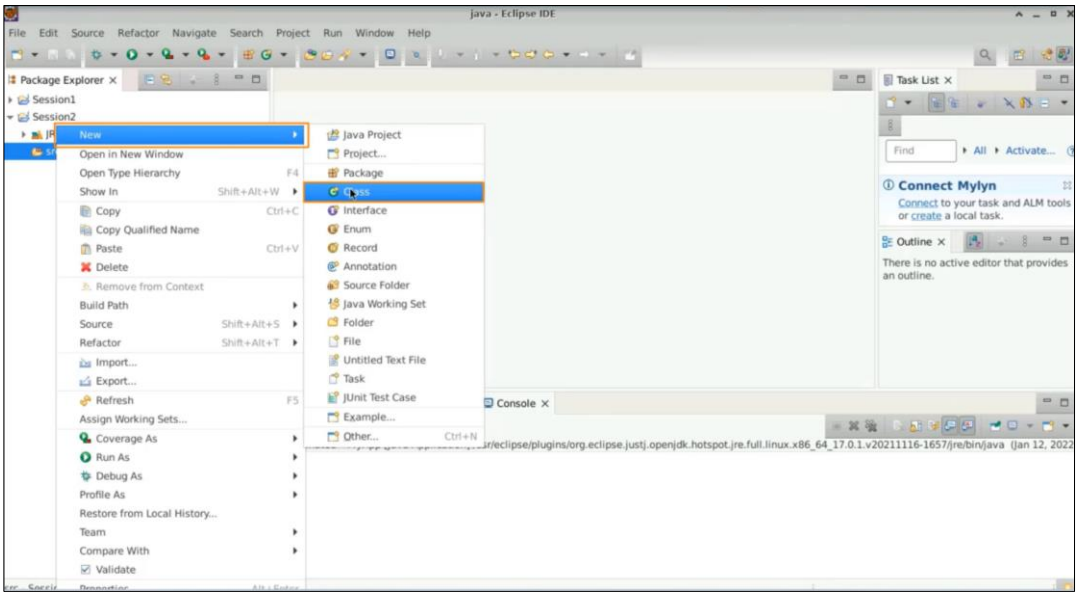
1.2. Select **File**, then **New,** and then **Java project**



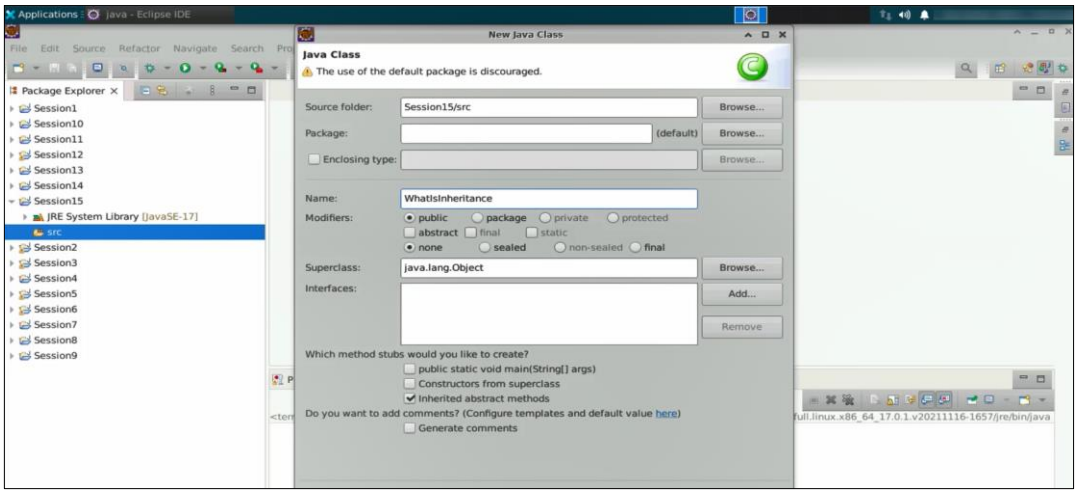
1.3 Name the project **“Session15”,** uncheck **“Create a module info dot Java file”**, and press **Finish**



1.4 With a **Session15** on the src, do a right-click and create a **new class**

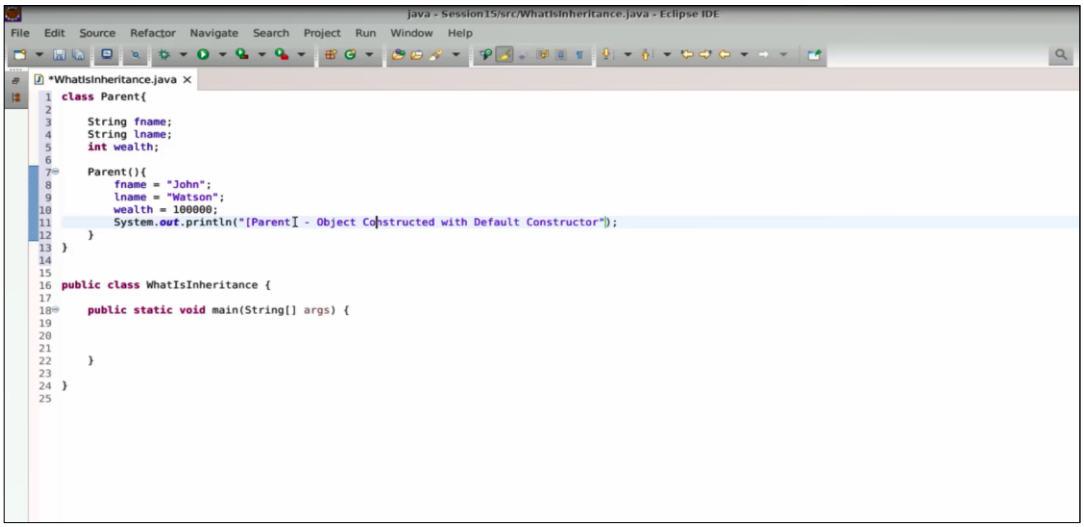


1.5 Name this class as an **WhatIsInheritance**, then select the **main method,** and then select **finish**

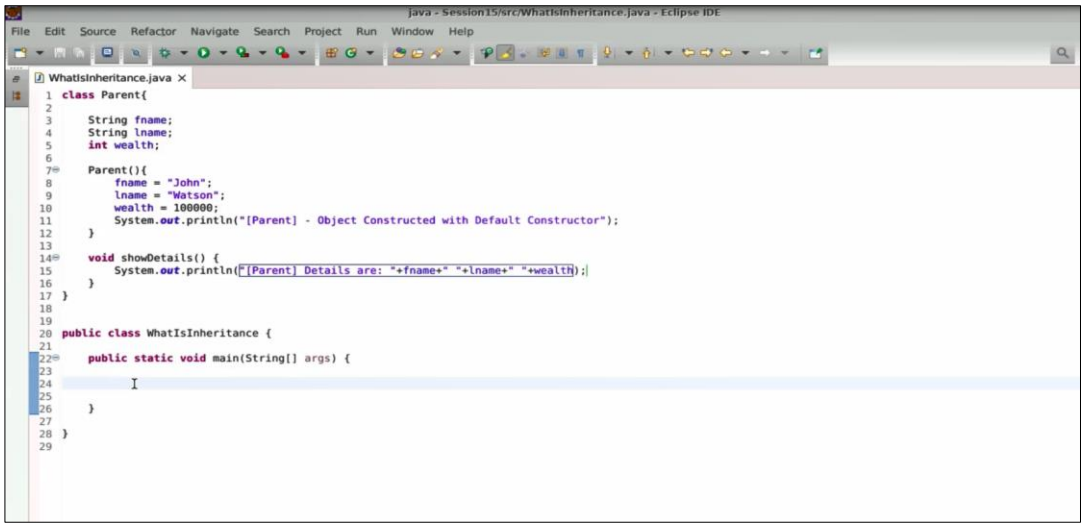


Step 2: Create a class with three attributes

2.1 Let us write a class called parent and define three attributes F name, last name, and an integer wealth. Now in the constructor of the parent initialize the first name to John, the last name to Watson, and the wealth to some 100,000. Also, use the SQL statement saying the parent, object is constructed with the default constructor

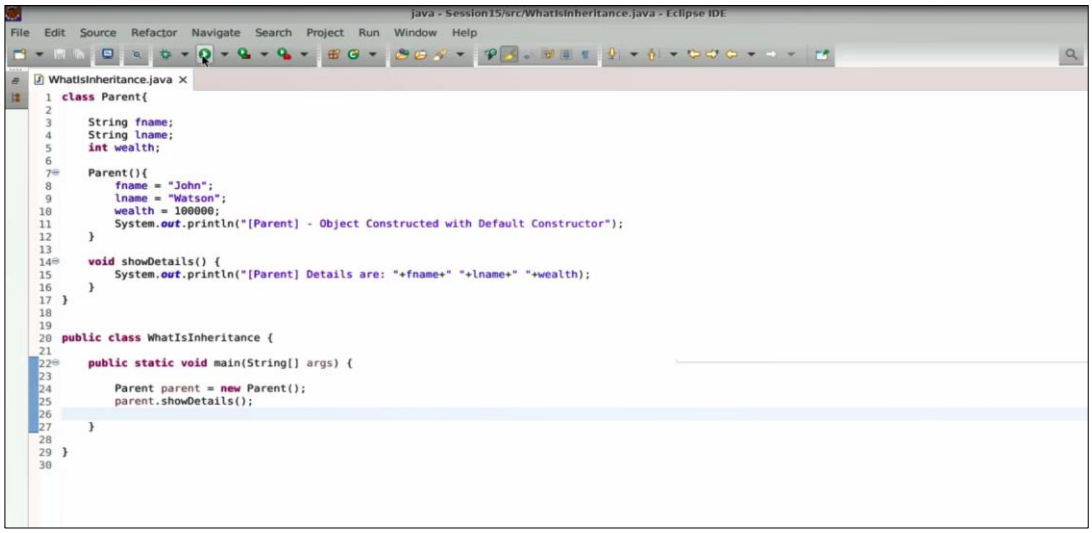


2.2 Now type "**show**." The parent or simply create a method called "**showDetails**." Here you are going to execute the print statement on the parent data. Use this like a parent, and say "**details are**" plus the name, space, the last name, and a space called wealth. This is how you create a parent class.

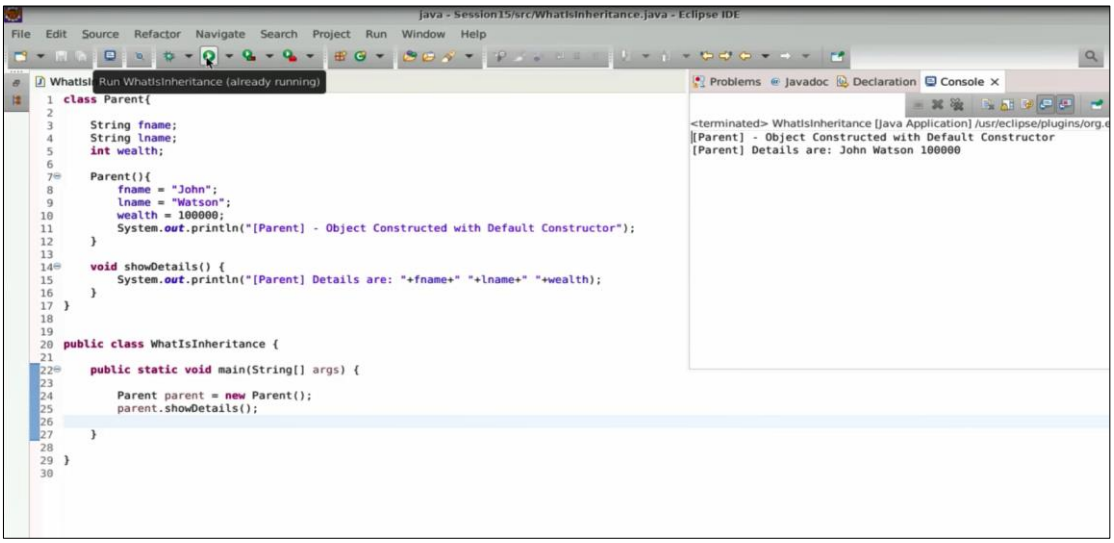


Step 3: Create an object using the new operator and execute the code

3.1 Now in the main method You are going to create the object of parent. Here you are with the new operator will create the object of **parent** by using the default constructor execution. Let us say you want to see the details inside the pane and object, you can simply say show the details. You will execute this method called show details

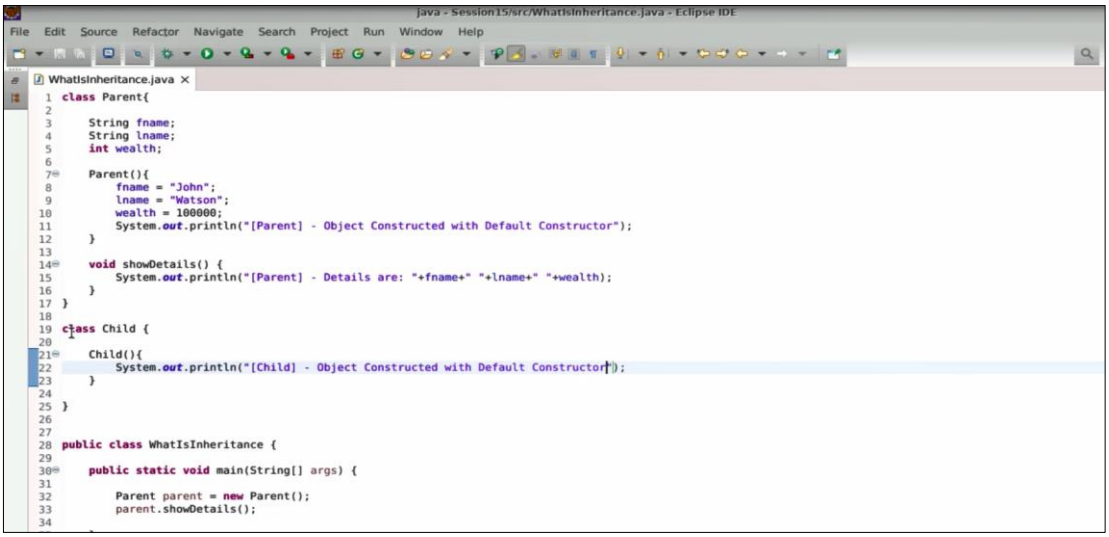


3.2 Run this code and you will see from the parent you are able to understand that there are some default construction, which is being used and it happened for the parent object. The next thing is details which is like John Watson and 100,000

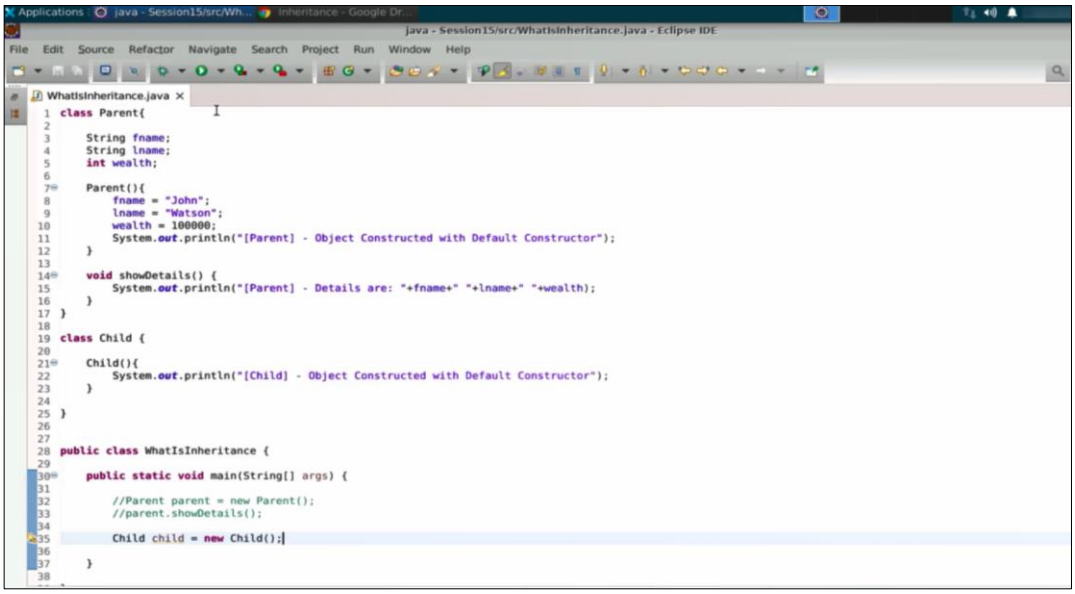


Step 4: Create an object with the default constructor

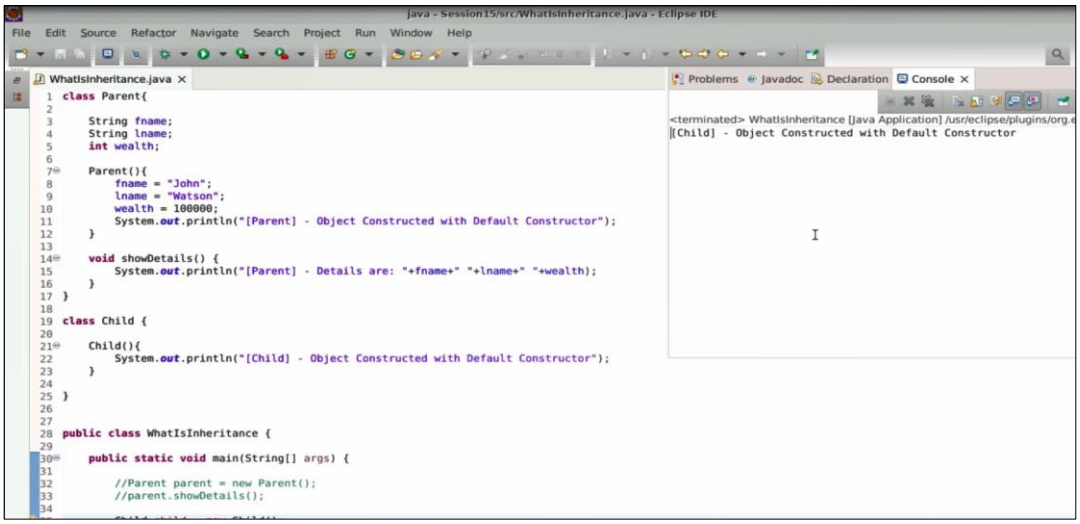
4.1 Now write one more class here and this class goes like a child. The child over here as of now, has no attributes and you are going to come up and create the constructor with a print statement, and this print statement goes like child object constructed with default constructor



4.2 When you create another class child, it has no attribute. There is just a constructor, so comment on the two statements in line number 32 and 33. Which means now you are not creating a parent object. You're going to create an object of a child

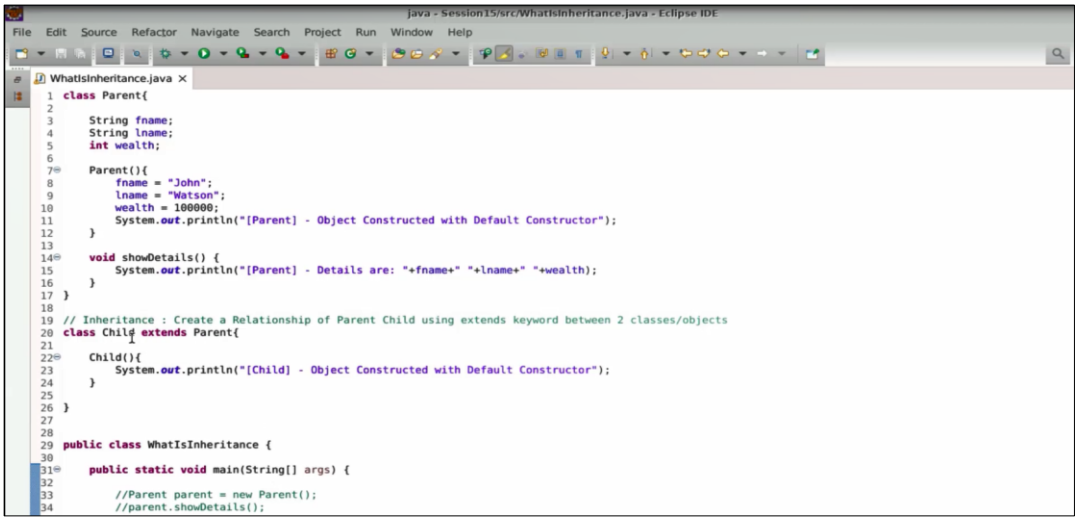


4.3 Run this code and you see that there is this child object constructed with the default constructor. it is the object which has no data within it, it is an empty object. It is just a storage container with the constructor

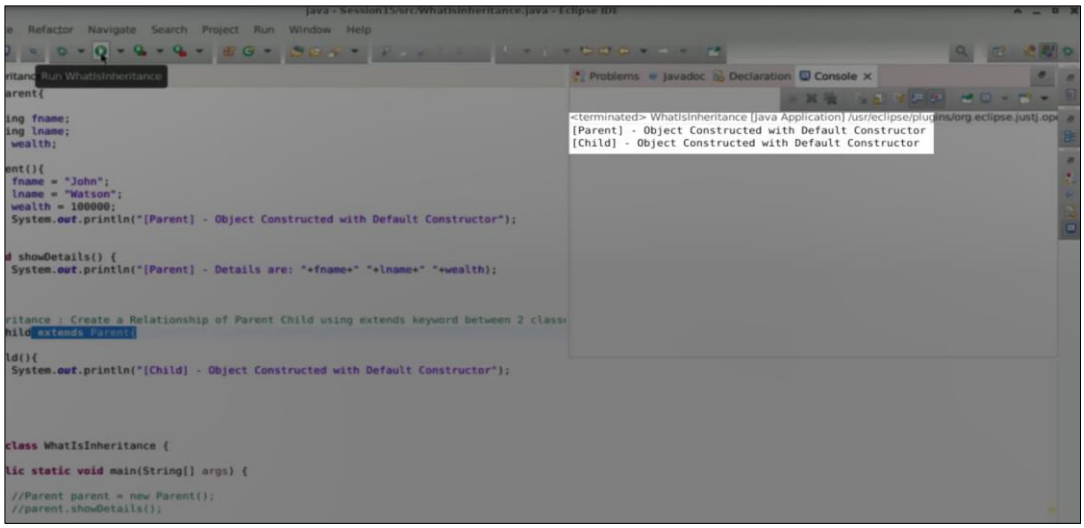


Step 5: Implement the concept of sub-class and superclass

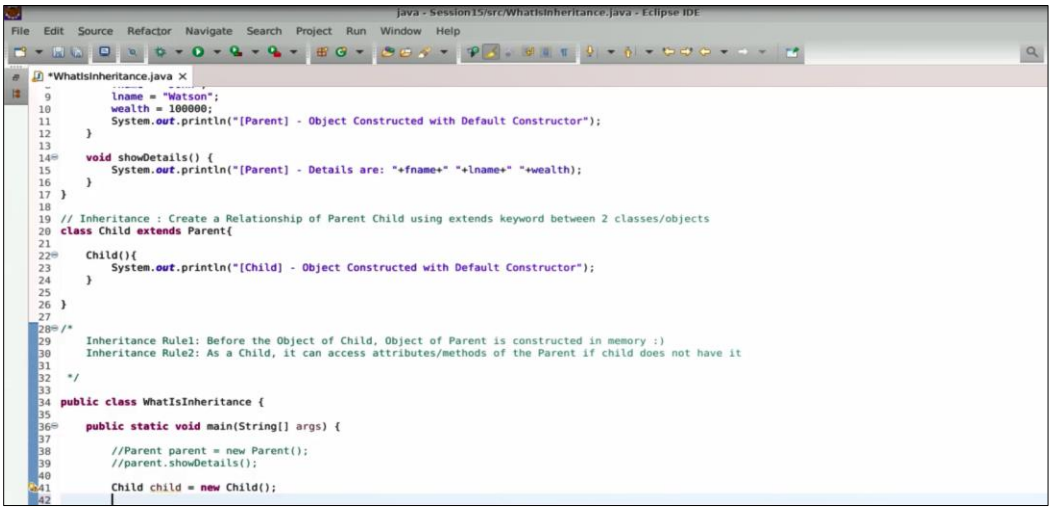
5.1 Now if you need to relate. The class parent with the class child through inheritance. You will simply come here and add that the child extends parent. You are relating to objects or two classes. Where **child** is known as the subclass, and **parent** is the superclass or the other way round



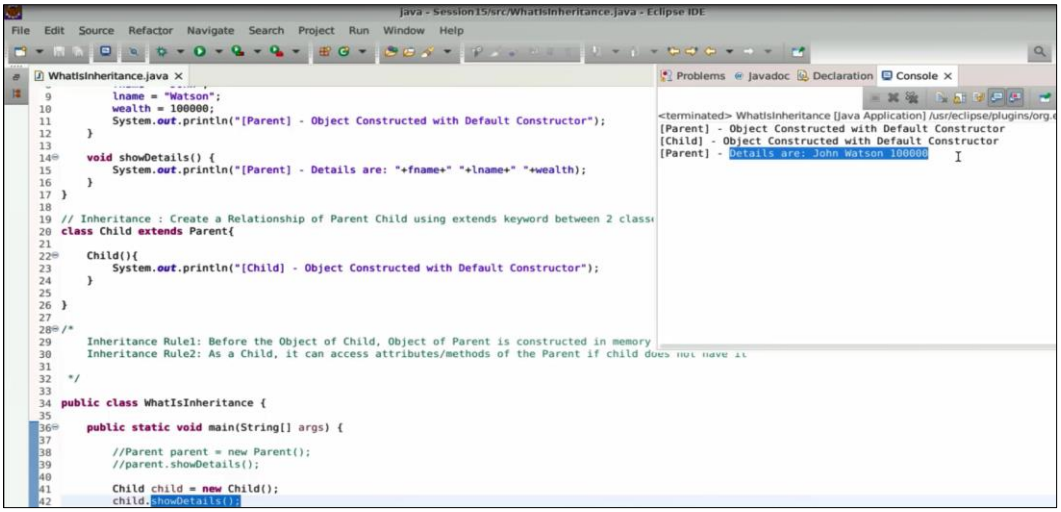
5.2 Run the code and What you see is a magical output. This is also the very **first thumb rule** to inheritance. Inheritance is a parent-child relationship. Which has a rule that before the object of the child object of the parent is constructed in memory. That is why you will get two objects constructed instead of 1 object



5.3 Rule 2 says as a child it can access attributes slash methods of the parent. In case the **child** is not going to have, some attribute or method it will look up into the **parent**, and if the parent has its child can access it

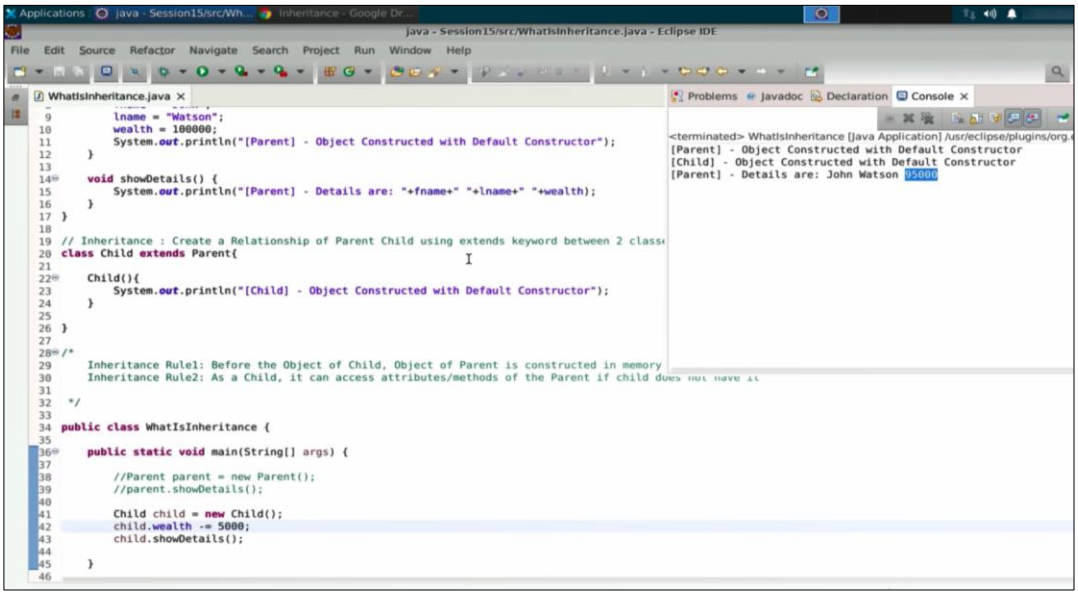


5.4 Now, type "**child.showDetails**." You need to see that you can access the **showDetails** method of the parent. You can access it if you execute it, and you will get to see the details coming up

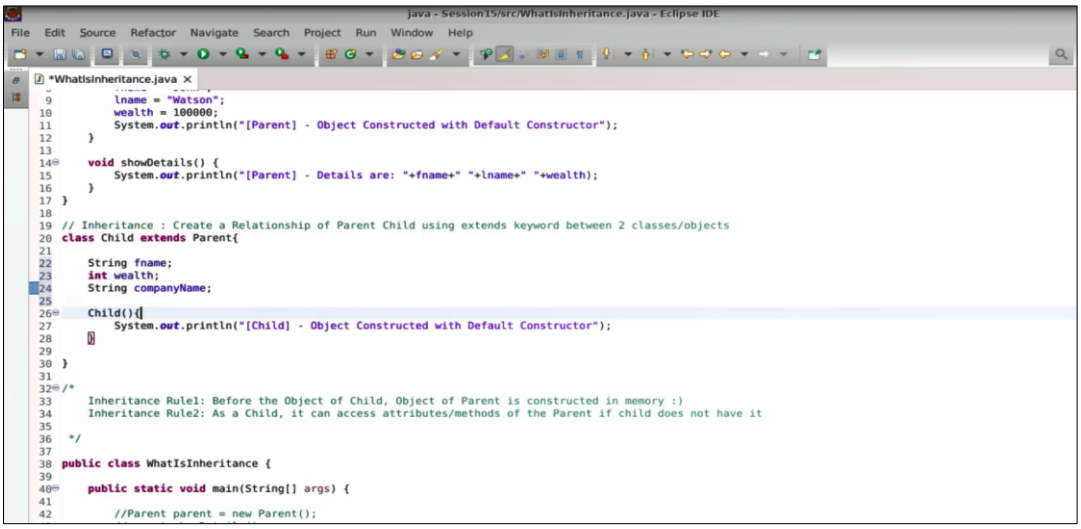


Step 6: Access the attributes of classes

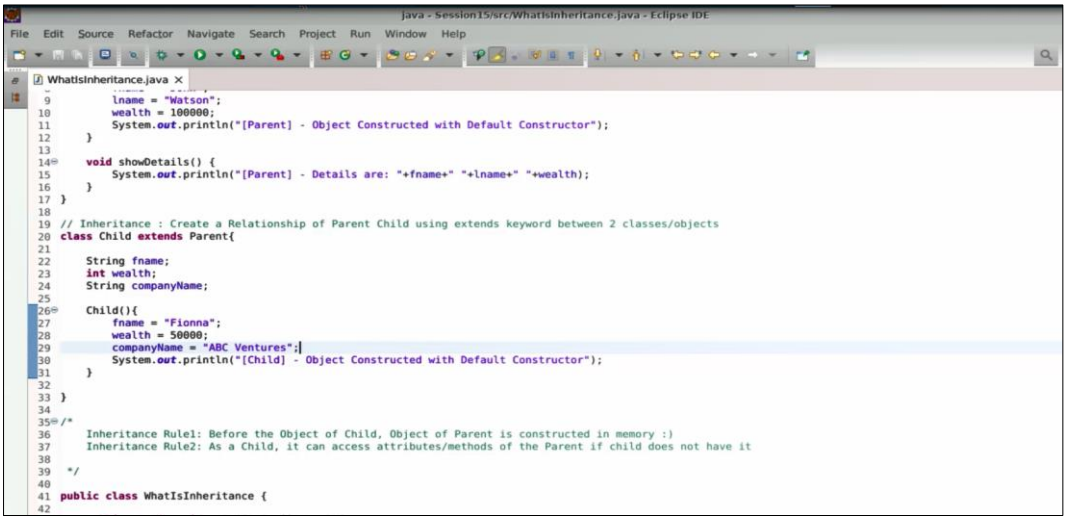
6.1 Now, let us say the child will access the attribute of the parent called wealth. You are going to say "**child.wealth = parent.wealth - 5000**". You are updating the value of the wealth by subtracting 5000 from it, but this is not the child's wealth. Whenever you run the program, you will observe that the wealth has been reduced by 5000 by the child, and this change is reflected in the parent



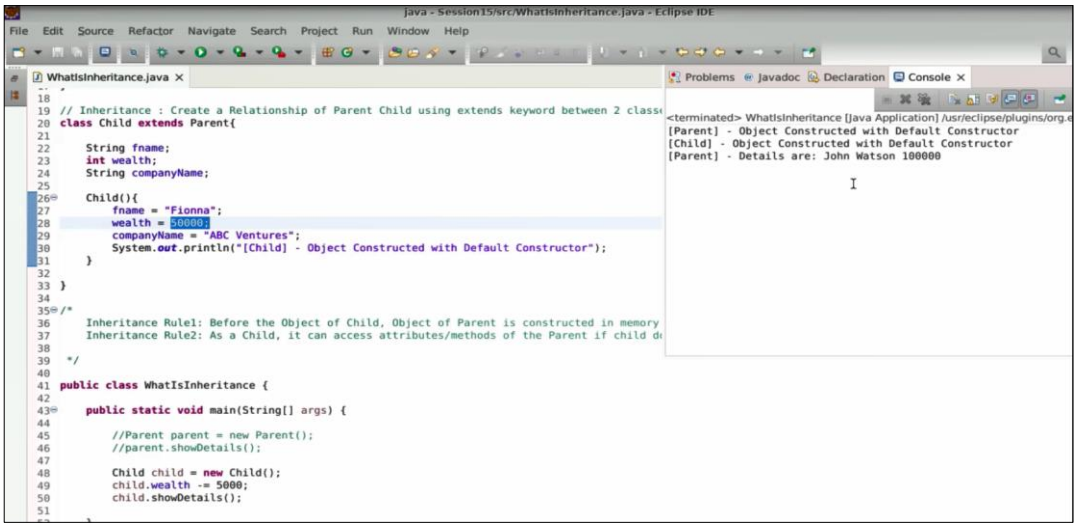
6.2 Now, what if the child has those attributes? Let us add that the child has an attribute called first name, an attribute called wealth, and an additional attribute called company name



6.3 In the constructor, add the **fname** as "**Fionna**". You will work on the **lname (last name)**, which is "**Watson**". Set the wealth to 50,000 and the **companyName** to "**ABC Ventures**"

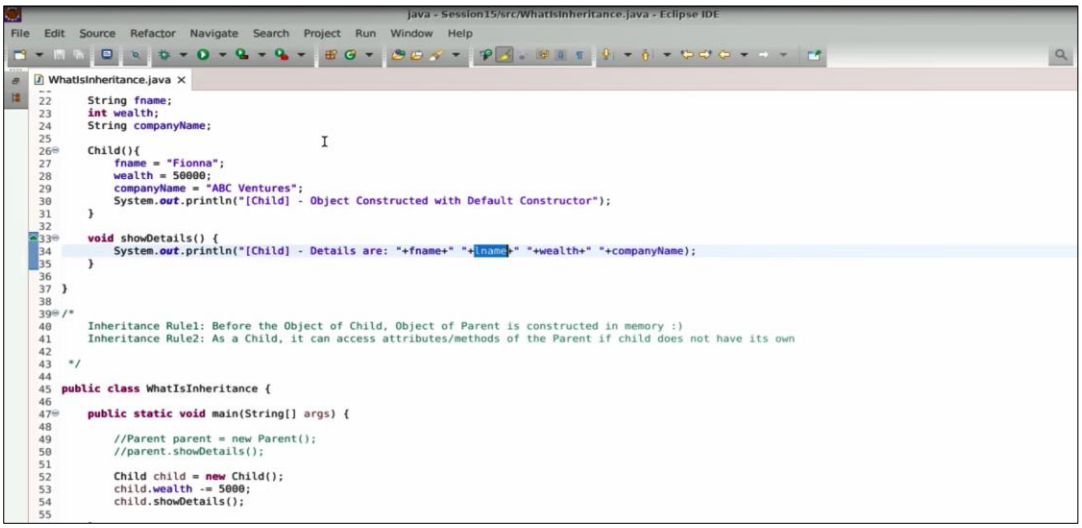


6.4 When you execute the show details method in the parent you will see that the parents' wealth remains unimpacted. When the child has its own attribute or method child going to access its own



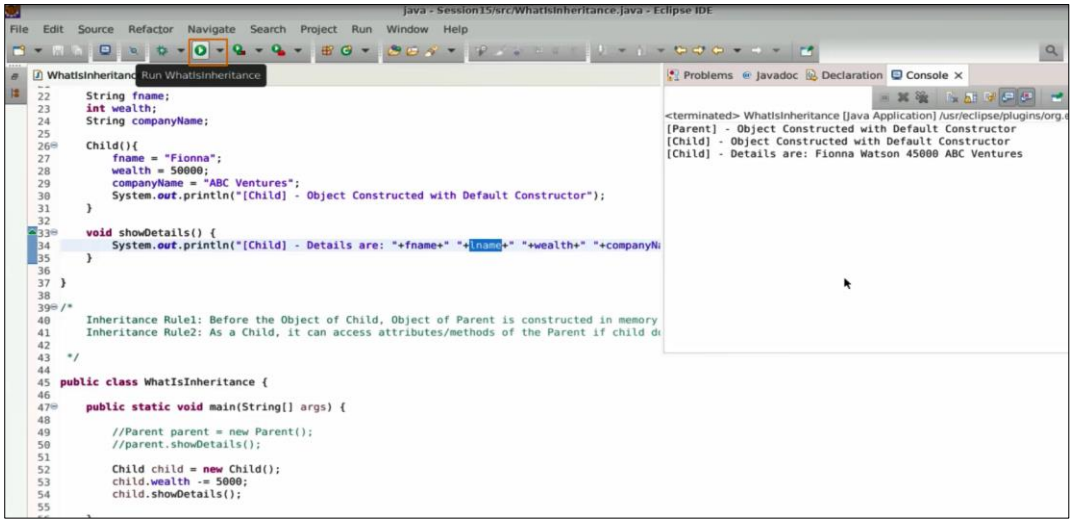
6.5 Let us type **void showDetails() {**

System.out.println("[Child] - Details are: " + fname + " " + lname + " " + wealth + " " + companyName);}

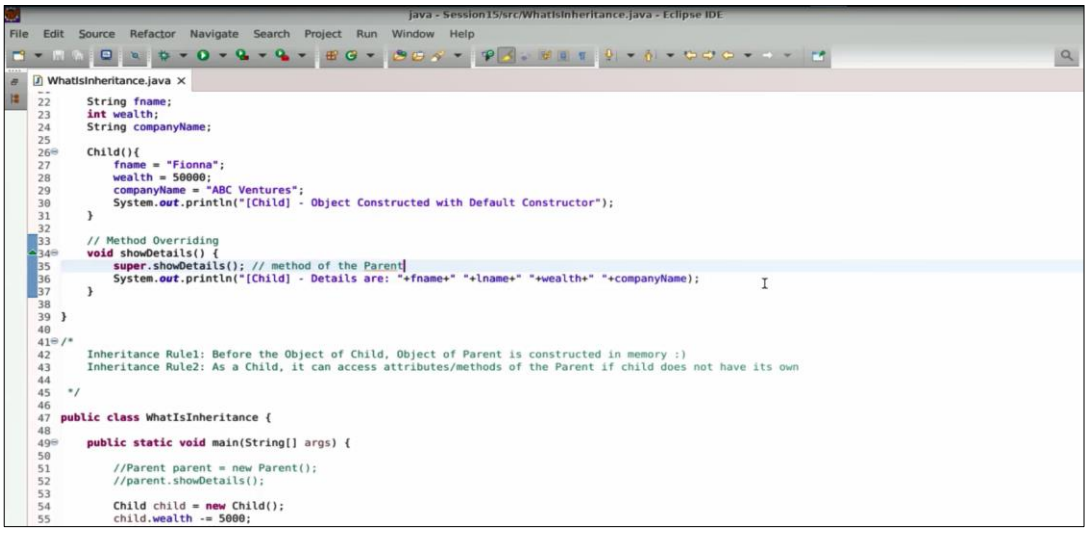


Step 7: Execute the code with example data

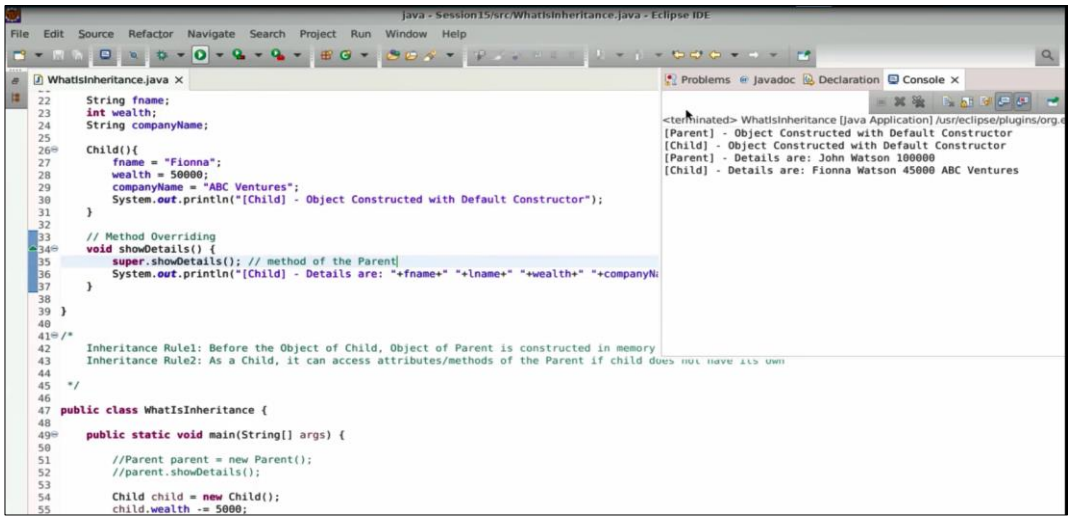
7.1 Run the code and you can see details are Fiona Watson with 45,000 and ABC Ventures. This beautiful concept is also referred to as **method overriding**



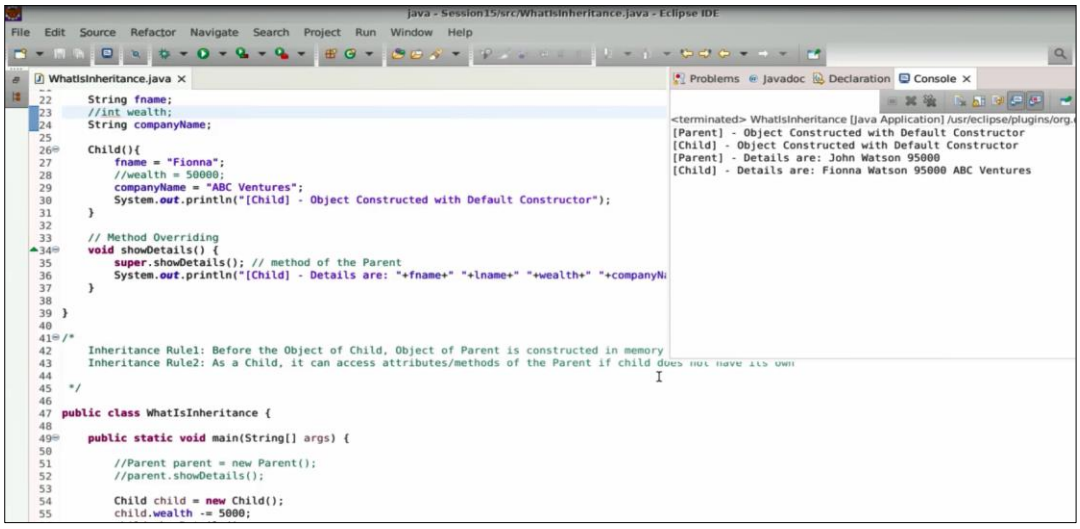
7.2 You can use this reference variable called super which refers to the parent object and You can execute the method called show details. Now this is the method of the parent. You are now exiting the method of the parents



7.3 Run the code, what you will notice is the parent days would come up as John Watson and no impact on the wealth, but child details coming up with the Watson being inherited from the parent and the wealth has been impacted.

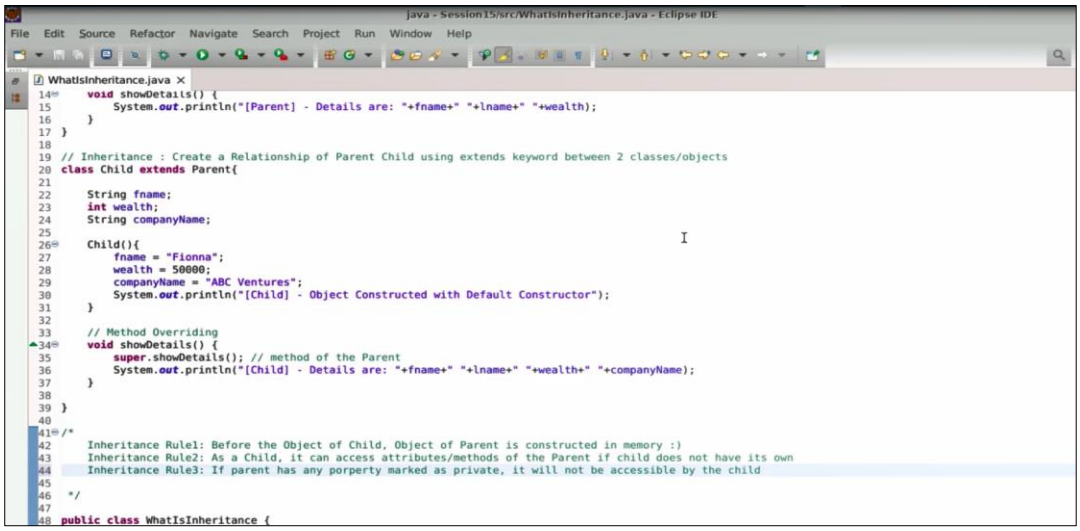


7.4 If the child is not going to have an attribute called wealth, when you run the code, what you would see is that the wealth is going to be inherited from the parent. What belongs to the child, would be accessed. If something is not even there in the parent, then definitely you will get a compile-time error

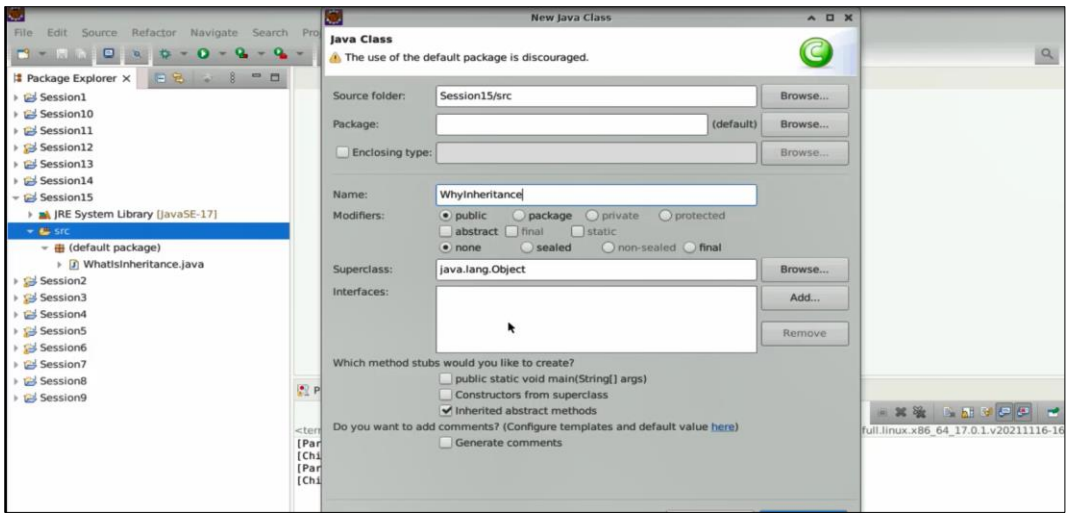


Step 8: Understand inheritance rules, their importance, and execute sample code

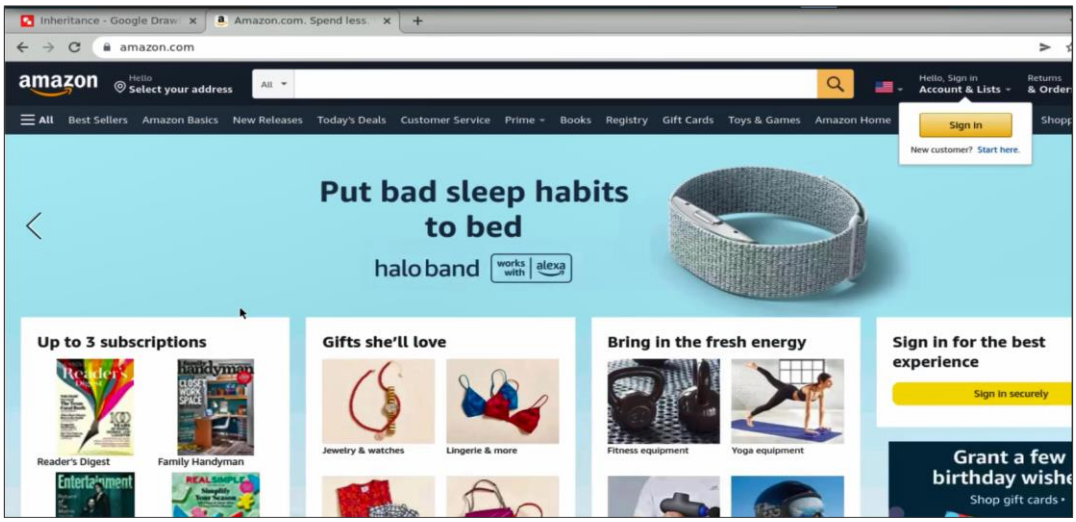
8.1 One last rule of inheritance, which is typical with the link of rule 2. If a parent has any property marked as private. It will not be accessible to the child



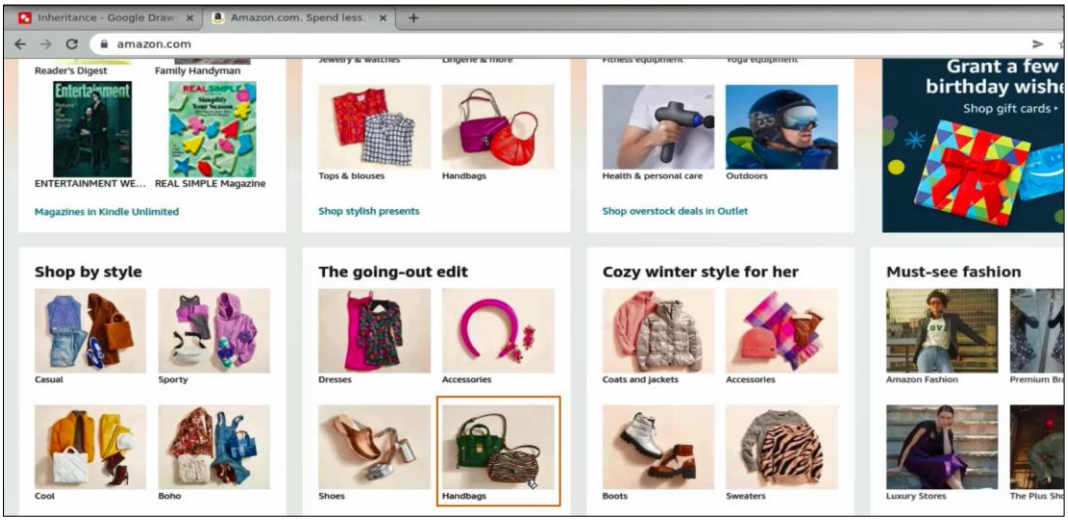
8.2 Now let us proceed ahead and see how the inheritance has a real significance. You are going to write a new class Known as **WhyInheritance** With the main method



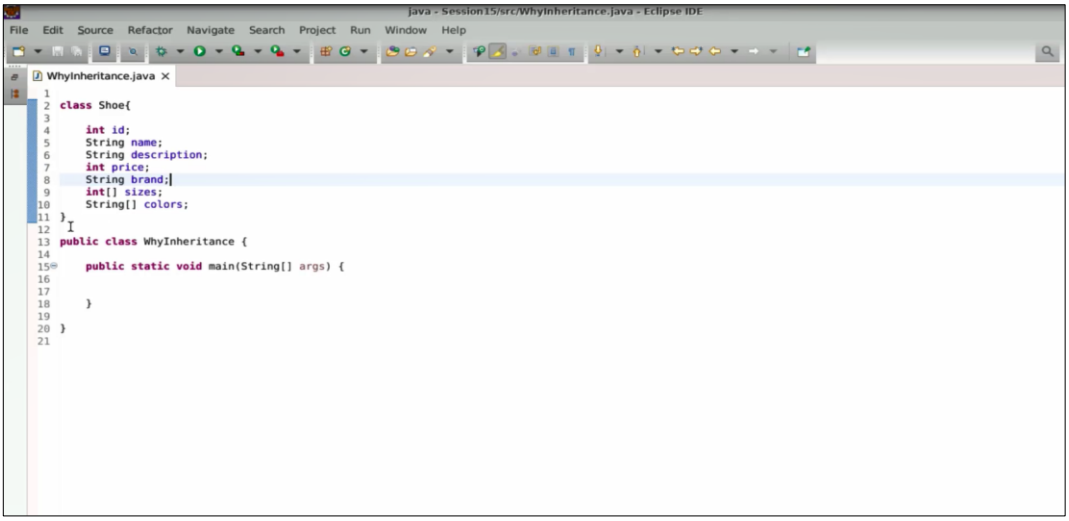
8.3 Let us take a use case of an e-commerce store. You are going to open something known as Amazon



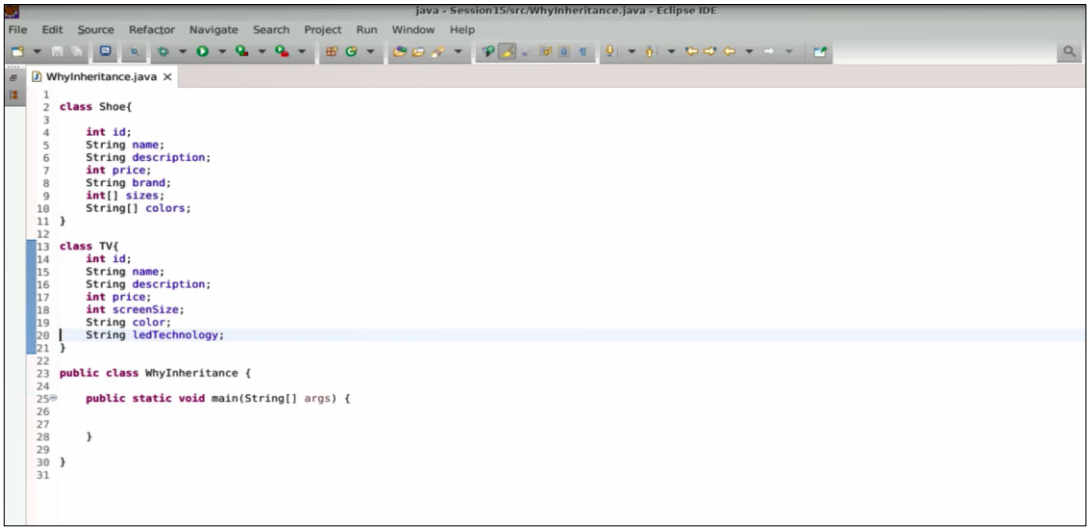
8.4 Now on Amazon you do have various products to be worked on. As you can see there is a product called dress, accessory, shoe, handbag. How is this all managed.



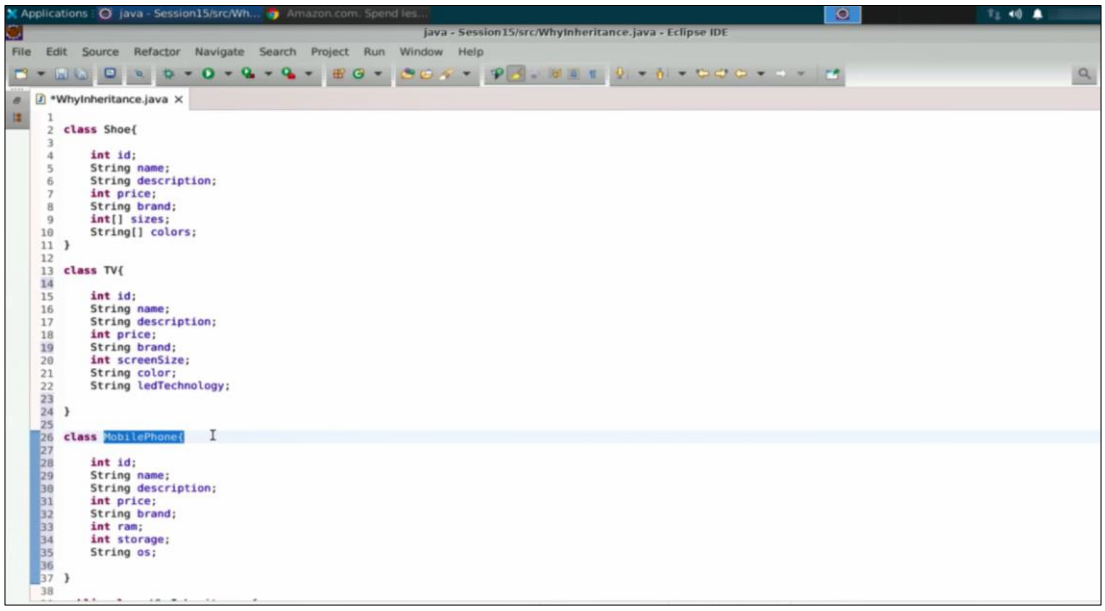
8.5 You will write a class and first You will represent something known as a shoe. To represent a shoe, you need to have an id for the issue. The name and description of the issue. Then there is another attribute called price. Now You need to have something on as an integer array of sizes, brand, and colors



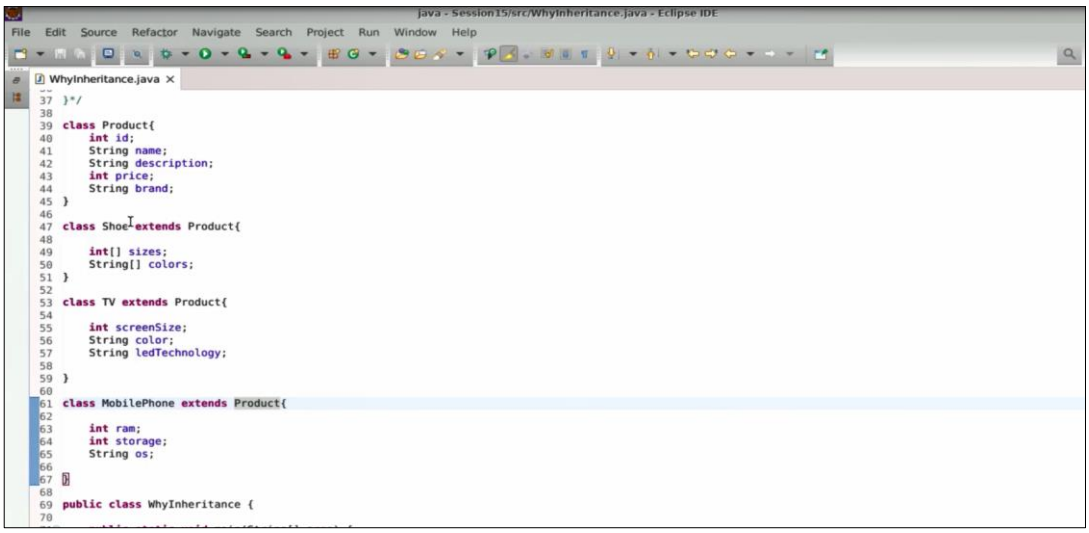
8.6 Now write a class called television it would have some common attributes id, name, description, integer price, screen size, array of sizes, array of colors, and ledtechnology



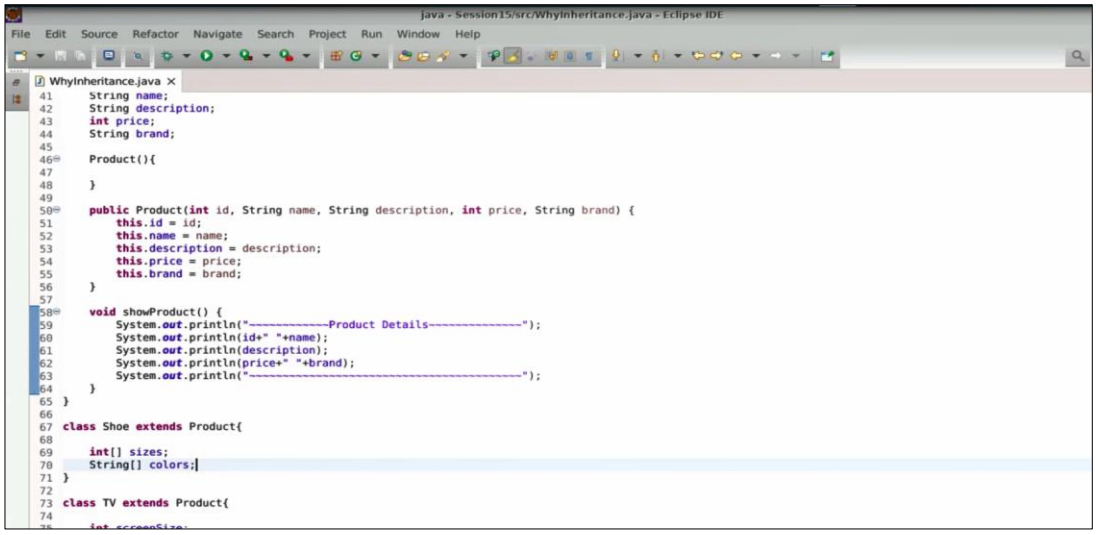
8.7 Let us consider one more object on the E-commerce platform. This is a mobile device or a mobile phone. For a phone, it will have an id, name, description, price, brand, ram, storage, and operating system



8.8 Now, if you are noticing a pattern, the first four or five attributes they are common across all the different objects. when you know that there is similarity when you must create different objects that is where you need inheritance. Create a class known as product. make these common attributes to be a part of product. Eliminate the common attributes. Say shoe extends product, television extends the product, and mobile phone extends the product

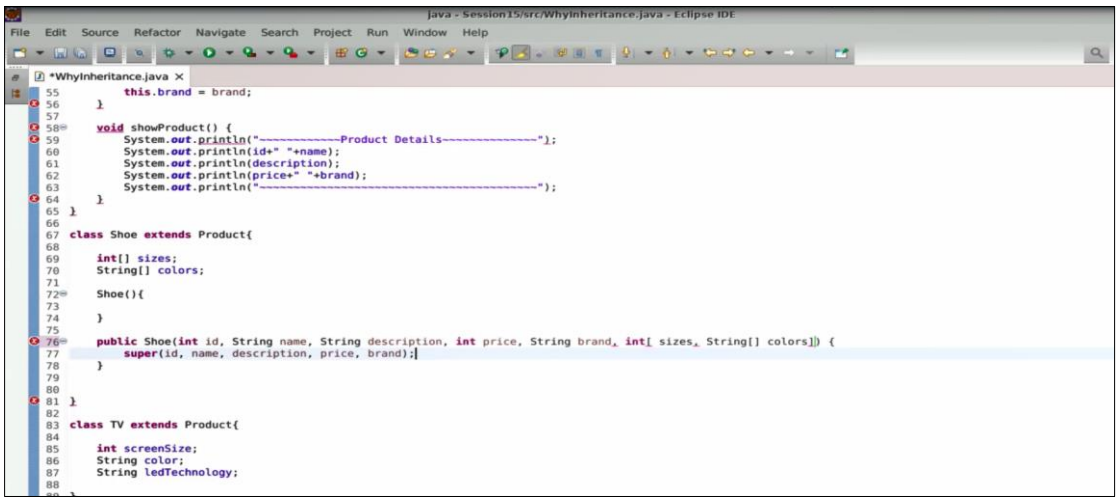


8.9 For the product here, you can use a default constructor or a parameterized constructor. You can add void show the product. You are going to display id, name, add print description and then you can come up with the price and let us say brand. These same attribute initializations and these attributes display part



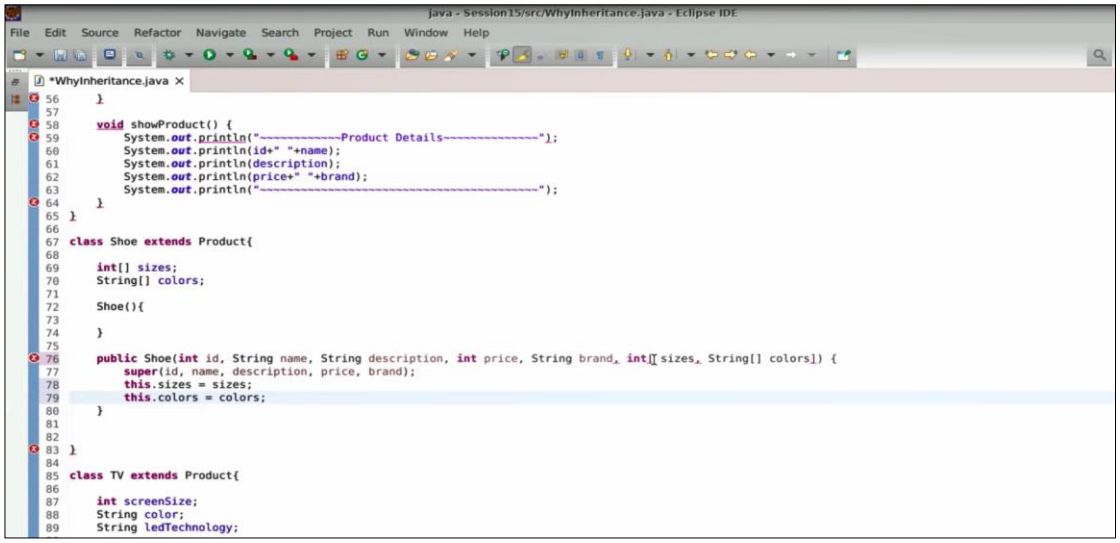
Step 9: Create a default constructor and parameterized constructor

9.1 Let us show you how so you can create a shoe default constructor. And for my parameterized constructor, you will just say source, generate the constructor from the superclass. You will type generate and what you see is a magical part, for the shoe you will take all the attributes. But for the attributes called ID, name, description, Price, and the brand, this would be, right away passed to the parent for the initialization

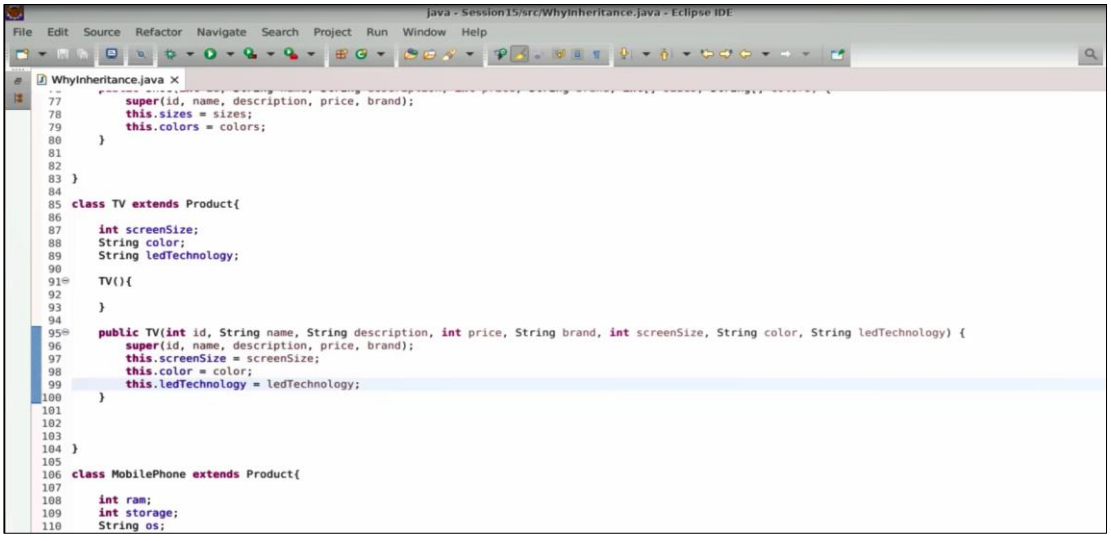


Step 10: Initialize the arrays

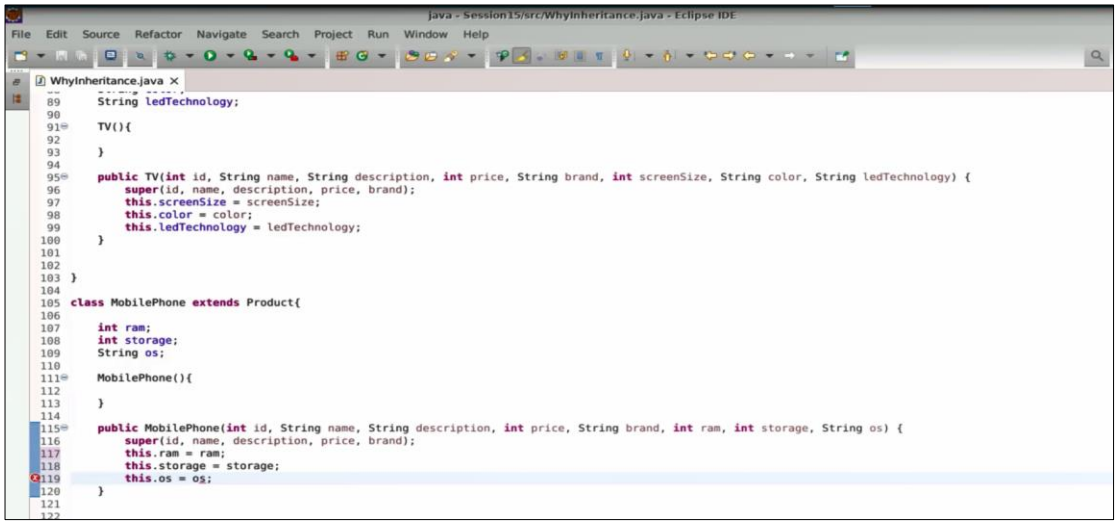
10.1 Regarding my array of sizes or my array of colors, you can initialize them like this: **this.sizes** and **this.colors**. A point to notice now is that the attribute initialization has reduced drastically. In previous examples, you had to initialize all these attributes, but now you are just initializing 2 of them



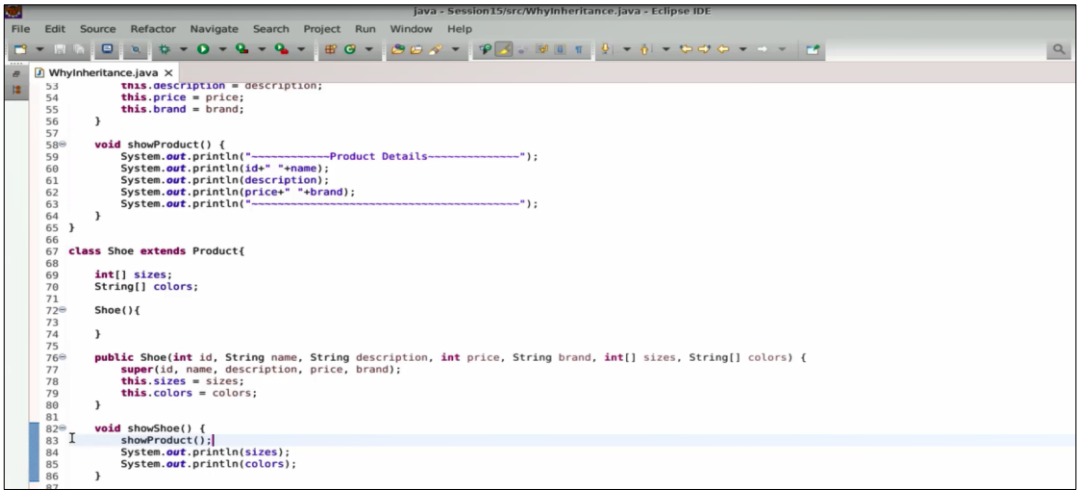
10.2 Let us take it from the television. Create a constructor through the parent class. Execute the parent constructor, and then set **this.screenSize** to screenSize. Then you have **this.color** as color. After that, set **this.ledTechnology** to ledTechnology. That is how you save development time



10.3 For the mobile phone, you can now right-click and select **Source**, then Generate Constructor from Parent. Take your additional attributes and put them inside the constructor. Here you go: **this.ram = ram**, **this.storage = storage**, and **this.os = os**



10.4 Now again, create a method called "**void showShoe**." Inside the method, you would display the additional attributes, such as sizes and colors. You can simply call "**showProduct**" in the same way for the television and mobile phone. You now understand the use case of inheritance, the rules it comes with, and why you need it. You need inheritance because it saves development time. In addition to this, inheritance provides several benefits, including the ability to implement design patterns with the help of a concept called runtime polymorphism



By following the above steps, you have successfully used Inheritance in Java and understood its necessity and significance.