Lesson 03 Demo 11

Implementing Object Class in Java

Objective: To implement the use of object class in Java

Tools required: Eclipse IDE

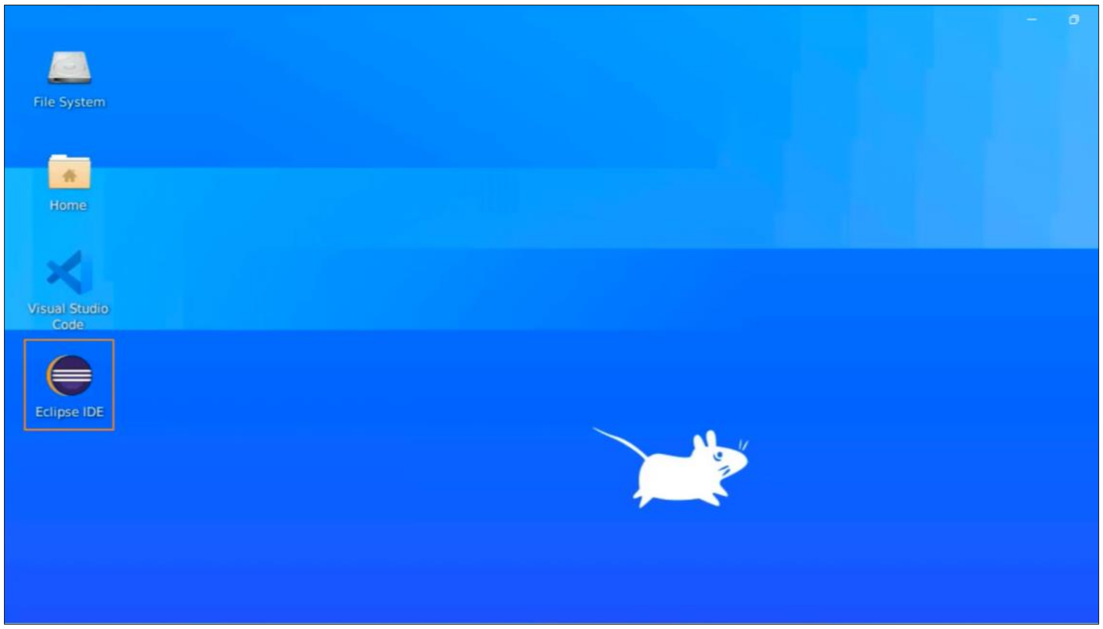
Prerequisites: None

Steps to be followed:

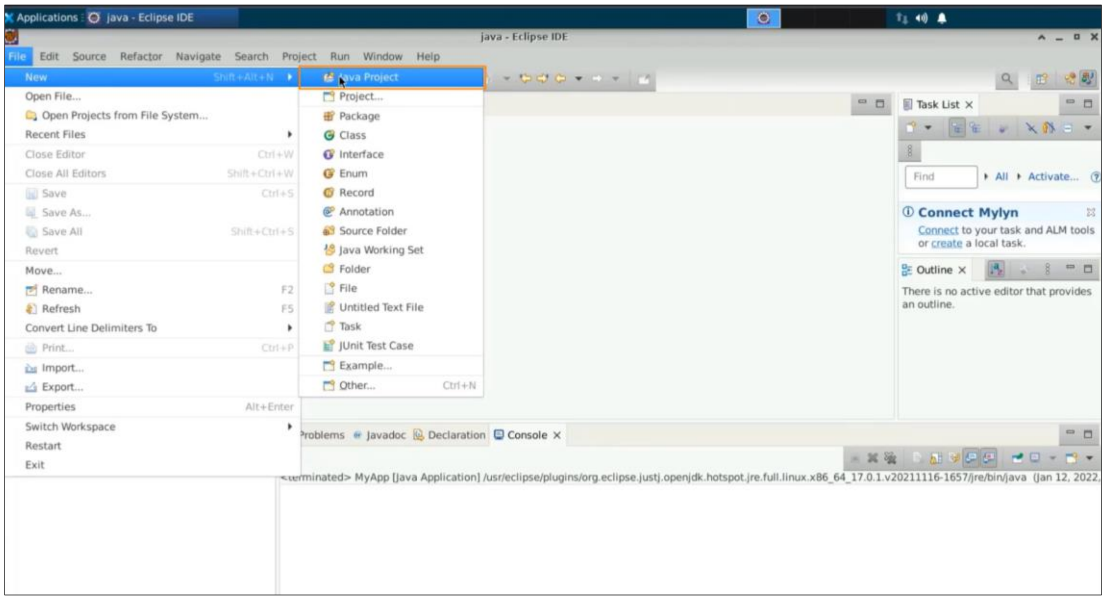
1. Create a class called Object Demo, followed by selecting the main method
2. Create a class named Product with three attributes
3. Create a parameterized constructor and create the object of the product
4. Execute the code with examples and overriding

Step 1: Create a class called Object Demo, followed by selecting the main method

1.1 Open the **Eclipse IDE**

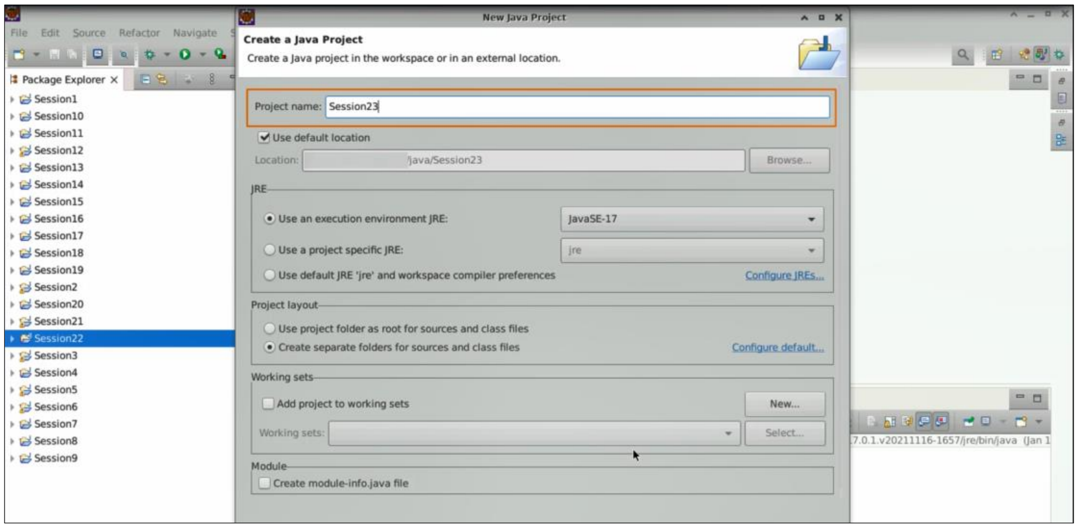


1.2. Select **File**, then **New,** and then **Java project**



```
<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 **"Session23",** uncheck **"Create a module info.Java file"**, and press **Finish**



Project name: Session23

[✓] Use default location

Location: java/Session23

(●) Use an execution environment JRE: JavaSE-17

( ) Use a project specific JRE: jre

( ) Use default JRE 'jre' and workspace compiler preferences

( ) Use project folder as root for sources and class files

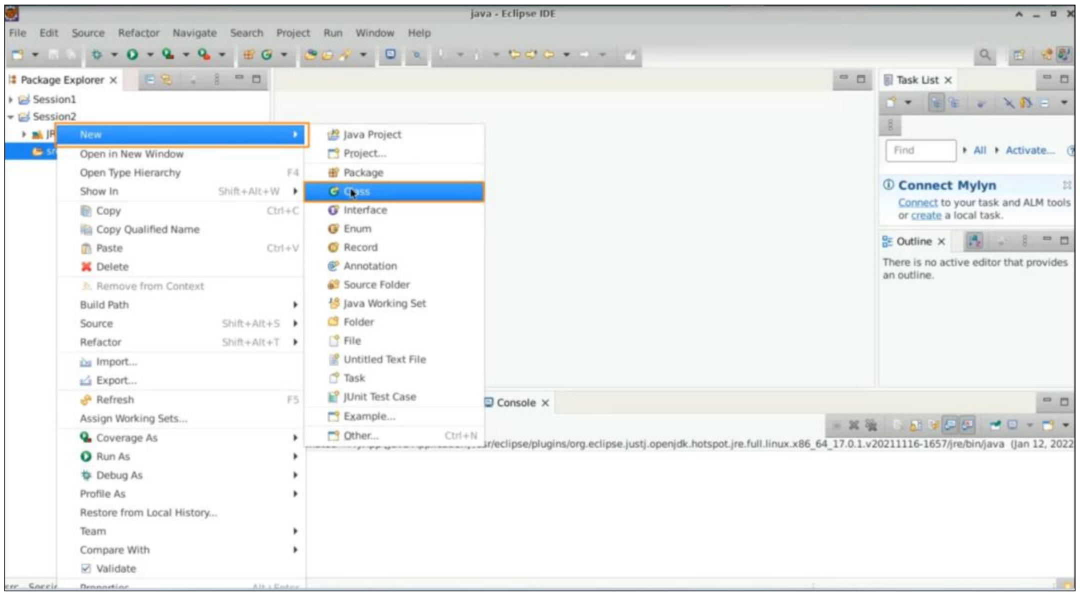
(●) Create separate folders for sources and class files

[ ] Add project to working sets

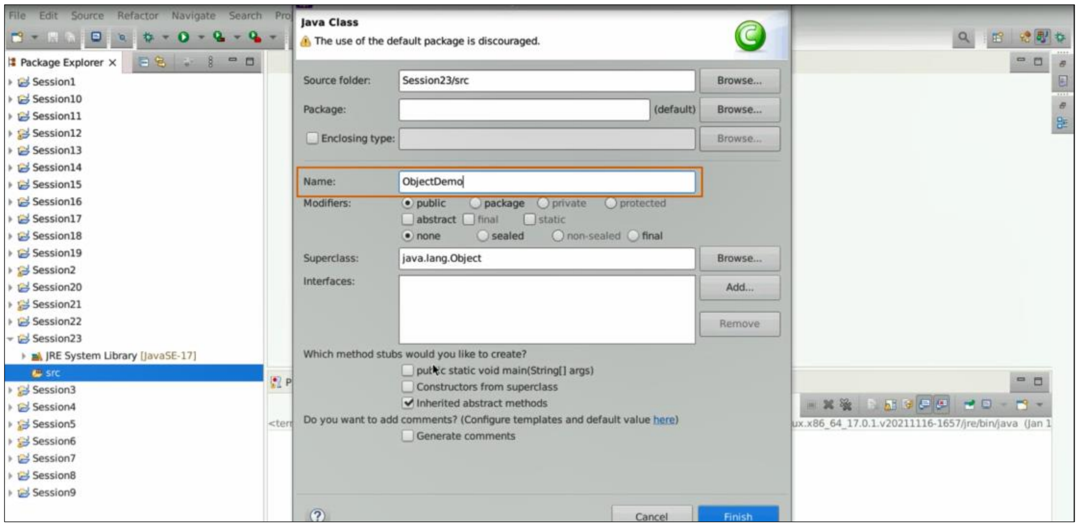
Working sets:

[ ] Create module-info.java file

1.4 With a **Session2** on the **src**, do a right-click and create a **new class**



1.5 Name this class as an **ObjectDemo**, then select the **main method,** and then select **Finish**


Java Class

The use of the default package is discouraged.

Source folder: Session23/src

Package: (default)

Name: ObjectDemo

Modifiers: [✓ public] package private protected

Superclass: java.lang.Object

Which method stubs would you like to create?

public static void main(String[] args)

Constructors from superclass

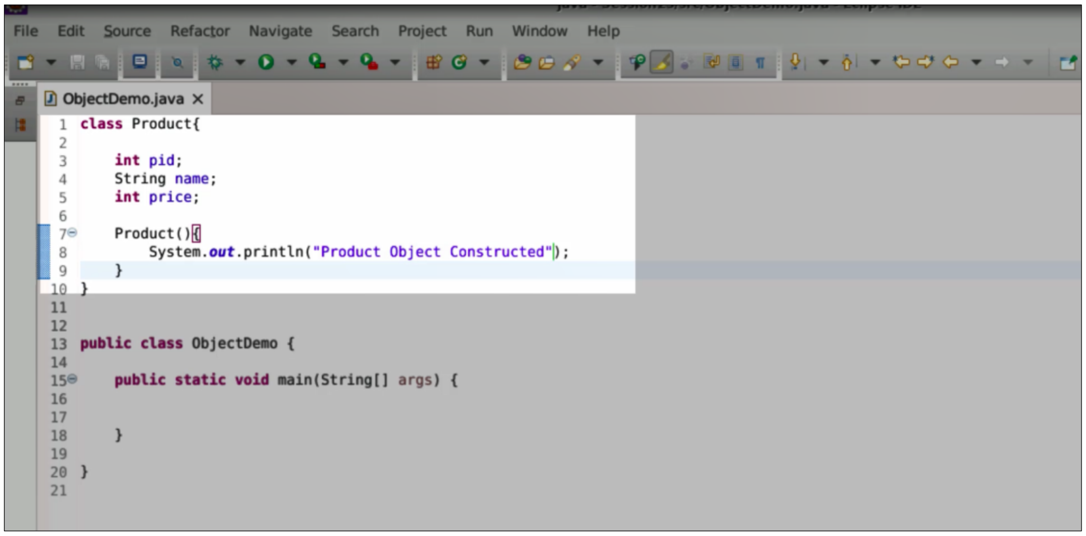
[✓ Inherited abstract methods]

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

Generate comments

Step 2: Create a class named Product with three attributes

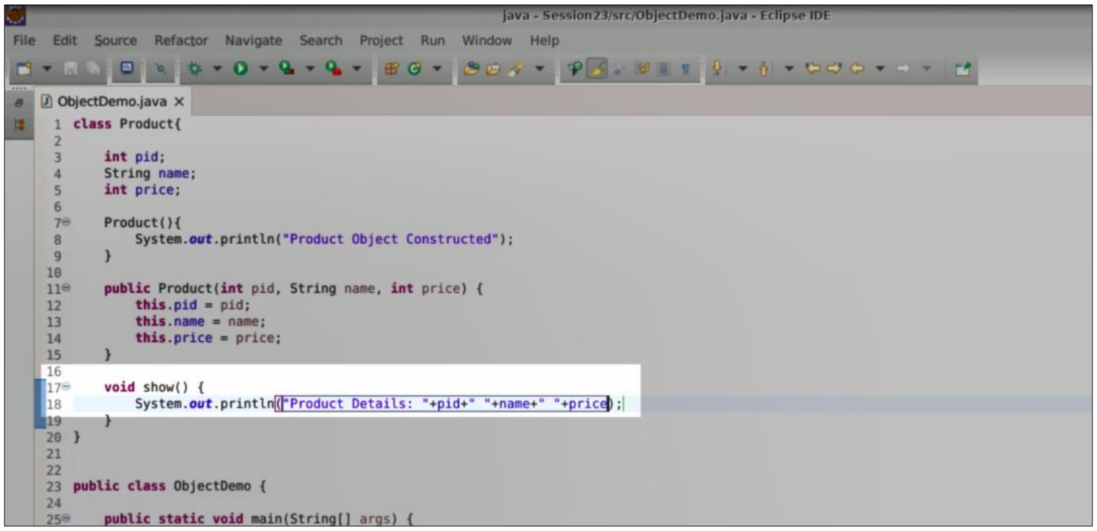
2.1 Create a class named **Product**. This class should have three attributes: **pid**, **name**, and **price**. In the constructor, print a message indicating that a Product object has been constructed


ObjectDemo.java
File Edit Source Refactor Navigate Search Project Run Window Help

```
class Product{

    int pid;
    String name;
    int price;

    Product(){
        System.out.println("Product Object Constructed");
    }
}


public class ObjectDemo {

    public static void main(String[] args) {


    }

}
```

Step 3: Create a parameterized constructor and create the object of the product

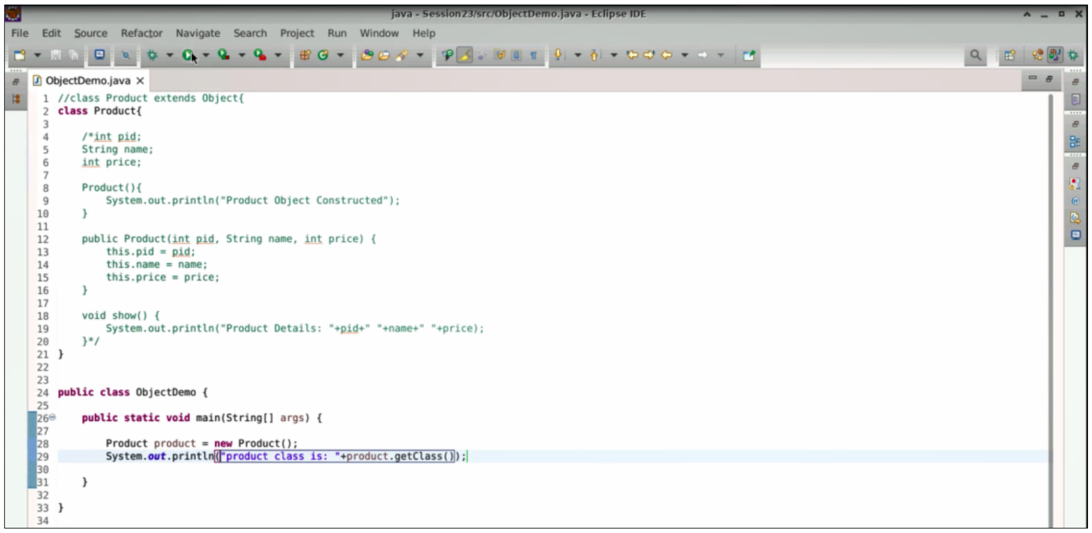
3.1 You can also create a parameterized constructor by right-clicking in your class, selecting Source, and generating the constructor using the fields. Now, you have the Product object, and you can create a show method which will print the object details. Let us write the print statement as **Product details: + pid +   + name +   + price."**


java - Session23/src/ObjectDemo.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
ObjectDemo.java

```
class Product{

    int pid;
    String name;
    int price;

    Product(){
        System.out.println("Product Object Constructed");
    }

    public Product(int pid, String name, int price) {
        this.pid = pid;
        this.name = name;
        this.price = price;
    }

    void show() {
        System.out.println("Product Details:  "+pid+"  "+name+"  "+price);
    }
}


public class ObjectDemo {

    public static void main(String[] args) {
```

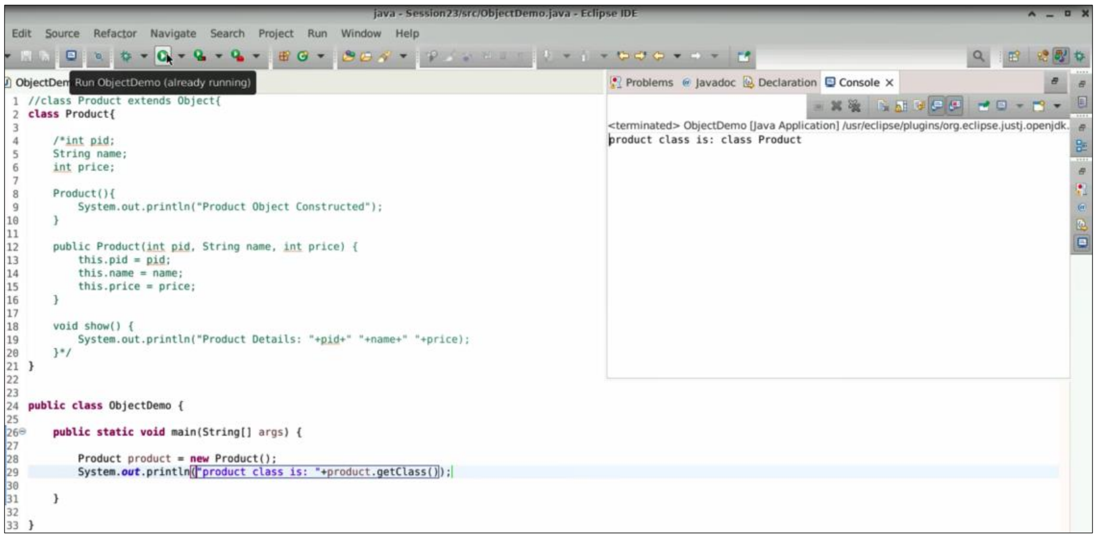
Step 4: Execute the code with examples and overriding

4.1 Write the product, product as a new product, and you create the object of the product. Before doing anything, first, comment on all of the code inside the product. You don't have to mention an extended object, whether you extend or do not extend the objects, your class will always be the child of this object. With the product, let us come here and write **"product class is", product.getClass()** as shown below:



```
//class Product extends Object{
class Product{

    /*int pid;
    String name;
    int price;

    Product(){
        System.out.println("Product Object Constructed");
    }

    public Product(int pid, String name, int price) {
        this.pid = pid;
        this.name = name;
        this.price = price;
    }

    void show() {
        System.out.println("Product Details: "+pid+" "+name+" "+price);
    }*/
}


public class ObjectDemo {

    public static void main(String[] args) {

        Product product = new Product();
        System.out.println("product class is: "+product.getClass());

    }

}
```

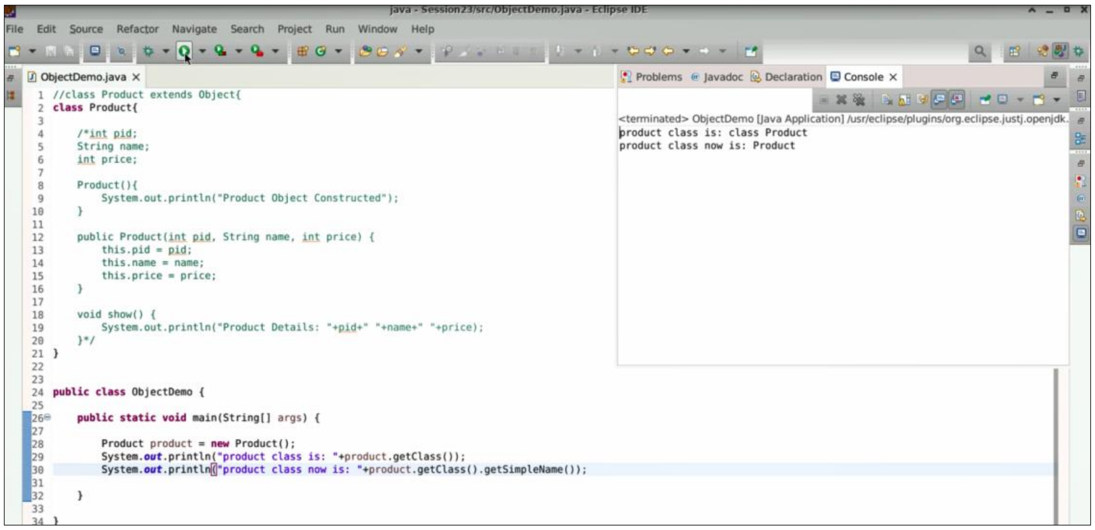
4.2 Let us run this code, it says the product class is class product, hence it is even coming up with the keyword class



```
//class Product extends Object{
class Product{

    /*int pid;
    String name;
    int price;

    Product(){
        System.out.println("Product Object Constructed");
    }

    public Product(int pid, String name, int price) {
        this.pid = pid;
        this.name = name;
        this.price = price;
    }

    void show() {
        System.out.println("Product Details: "+pid+" "+name+" "+price);
    }*/
}


public class ObjectDemo {

    public static void main(String[] args) {

        Product product = new Product();
        System.out.println("product class is: "+product.getClass());

    }
}
```

```
<terminated> ObjectDemo [Java Application] /usr/eclipse/plugins/org.eclipse.justj.openjdk.
product class is: class Product
```

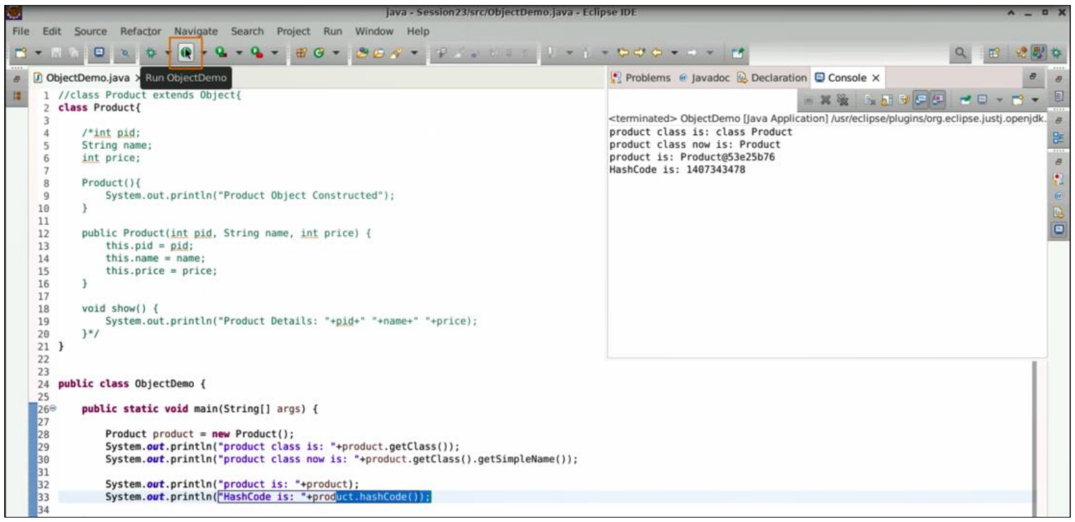
4.3 Next, write print the product class now is get **class.get** the simple name. When you write get the simple name, it will display just the name of the class to which this reference variable belongs. Hence, this get class method is given by the object parent of any class



```
//class Product extends Object{
class Product{

    /*int pid;
    String name;
    int price;

    Product(){
        System.out.println("Product Object Constructed");
    }

    public Product(int pid, String name, int price) {
        this.pid = pid;
        this.name = name;
        this.price = price;
    }

    void show() {
        System.out.println("Product Details: "+pid+" "+name+" "+price);
    }*/
}


public class ObjectDemo {

    public static void main(String[] args) {

        Product product = new Product();
        System.out.println("product class is: "+product.getClass());
        System.out.println("product class now is: "+product.getClass().getSimpleName());

    }
}
```

```
<terminated> ObjectDemo [Java Application] /usr/eclipse/plugins/org.eclipse.justj.openjdk.
product class is: class Product
product class now is: Product
```

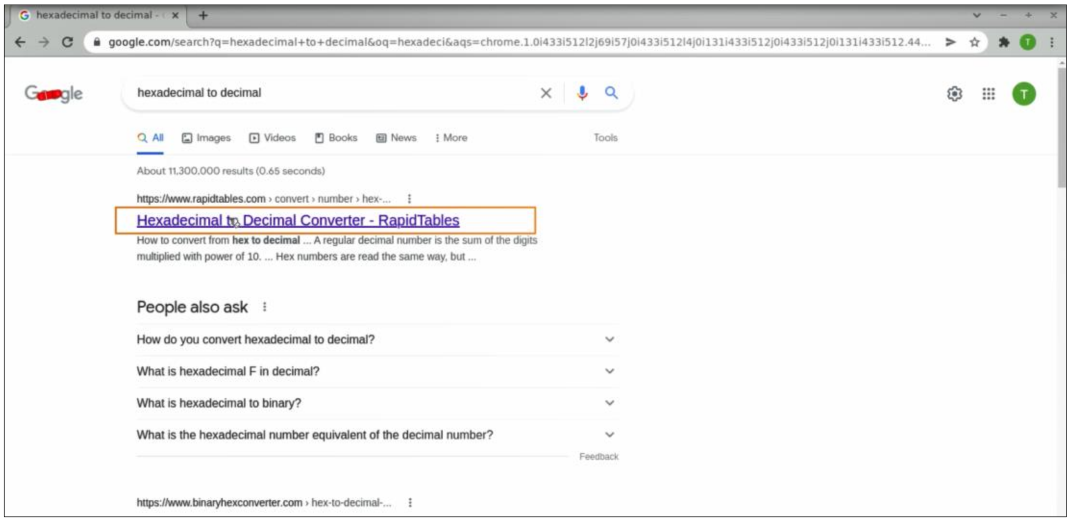
4.4 Similarly, if you wish to see the hash code, print the reference variable. Let us write **System.out.println("Product is: " + product)**. And thereafter, write **System.out.println("Hash code is: " + product.hashCode())**. The hash code, let us print down the hash code of the product as well. Run the code. A number is shown as a decimal value or a number value, but it is a hexadecimal value



```
//class Product extends Object{
class Product{

    /*int pid;
    String name;
    int price;

    Product(){
        System.out.println("Product Object Constructed");
    }

    public Product(int pid, String name, int price) {
        this.pid = pid;
        this.name = name;
        this.price = price;
    }

    void show() {
        System.out.println("Product Details: "+pid+" "+name+" "+price);
    }*/
}


public class ObjectDemo {

    public static void main(String[] args) {

        Product product = new Product();
        System.out.println("product class is: "+product.getClass());
        System.out.println("product class now is: "+product.getClass().getSimpleName());

        System.out.println("product is: "+product);
        System.out.println("HashCode is: "+product.hashCode());

```

```
<terminated> ObjectDemo [Java Application] /usr/eclipse/plugins/org.eclipse.justj.openjdk.
product class is: class Product
product class now is: Product
product is: Product@53e25b76
HashCode is: 1407343478
```

4.5 Write hexadecimal to decimal and search for online conversion


google.com/search?q=hexadecimal+to+decimal&oq=hexadeci&aqs=chrome.1.0i433i512l2j69i57j0i433i512l4j0i131i433i512j0i433i512j0i131i433i512.44...

hexadecimal to decimal

All Images Videos Books News More Tools

About 11,300,000 results (0.65 seconds)

https://www.rapidtables.com › convert › number › hex-...

Hexadecimal to Decimal Converter - RapidTables

How to convert from **hex to decimal** ... A regular decimal number is the sum of the digits multiplied with power of 10. ... Hex numbers are read the same way, but ...

People also ask

How do you convert hexadecimal to decimal?

What is hexadecimal F in decimal?

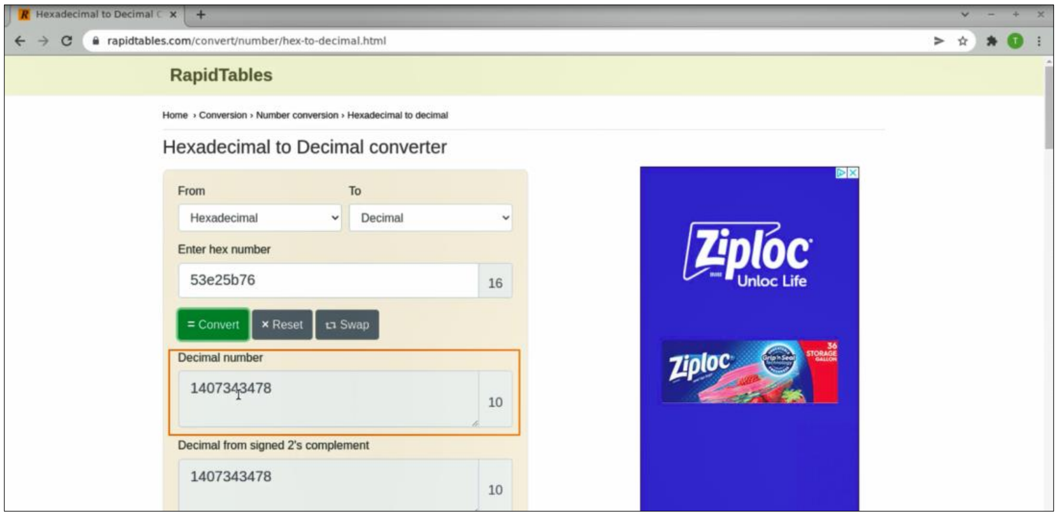
What is hexadecimal to binary?

What is the hexadecimal number equivalent of the decimal number?

Feedback

https://www.binaryhexconverter.com › hex-to-decimal-...

4.6 Copy the hexadecimal number and paste it here. Select convert, you will see the decimal the value comes as **1407343478**, which is the same over here


rapidtables.com/convert/number/hex-to-decimal.html

RapidTables

Home › Conversion › Number conversion › Hexadecimal to decimal

Hexadecimal to Decimal converter

From: Hexadecimal To: Decimal

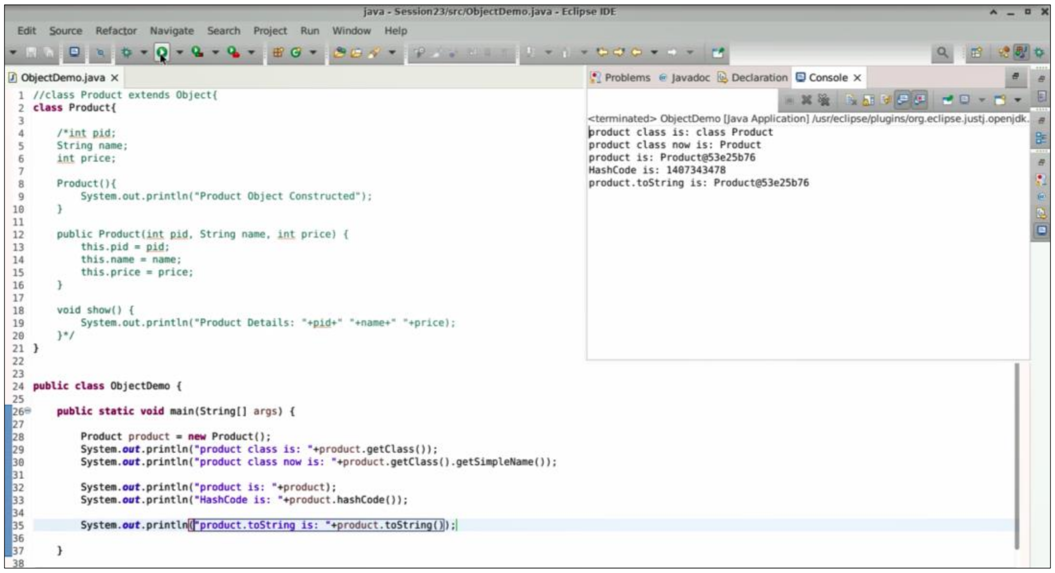
Enter hex number: 53e25b76 16

= Convert × Reset ⇆ Swap

Decimal number: 1407343478 10

Decimal from signed 2's complement: 1407343478 10

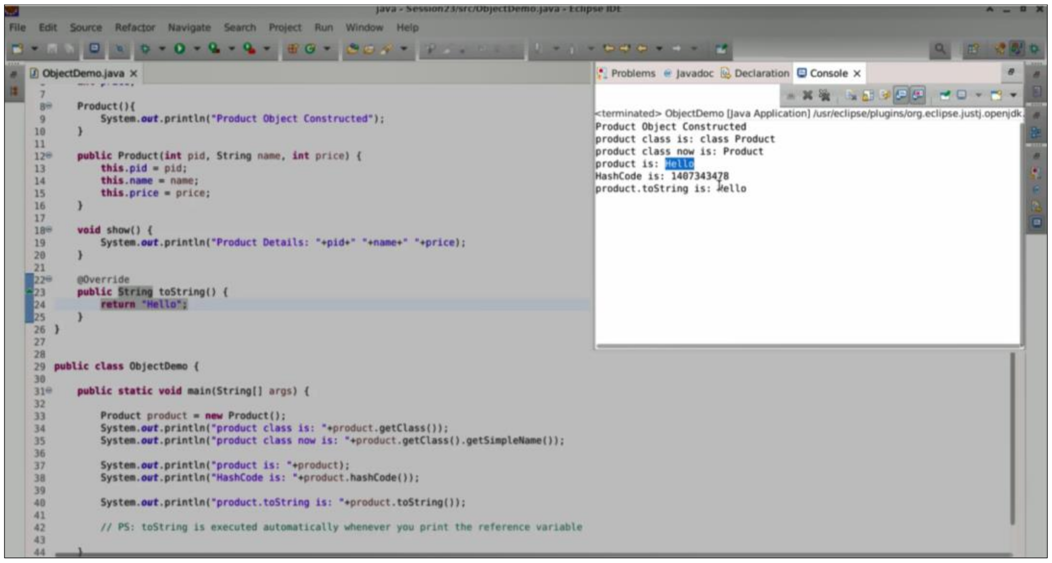
4.7 Write **product.toString() + product.toString(),** so **toString** is a string representation of your object. When you run the program, you will notice that by default, the **toString** method returns the name of the class followed by the **@** symbol and the hash code



```java
//class Product extends Object{
class Product{

    /*int pid;
    String name;
    int price;

    Product(){
        System.out.println("Product Object Constructed");
    }

    public Product(int pid, String name, int price) {
        this.pid = pid;
        this.name = name;
        this.price = price;
    }

    void show() {
        System.out.println("Product Details: "+pid+" "+name+" "+price);
    }*/
}


public class ObjectDemo {

    public static void main(String[] args) {

        Product product = new Product();
        System.out.println("product class is: "+product.getClass());
        System.out.println("product class now is: "+product.getClass().getSimpleName());

        System.out.println("product is: "+product);
        System.out.println("HashCode is: "+product.hashCode());

        System.out.println("product.toString is: "+product.toString());

    }
```

```
<terminated> ObjectDemo [Java Application] /usr/eclipse/plugins/org.eclipse.justj.openjdk.
product class is: class Product
product class now is: Product
product is: Product@53e25b76
HashCode is: 1407343478
product.toString is: Product@53e25b76
```

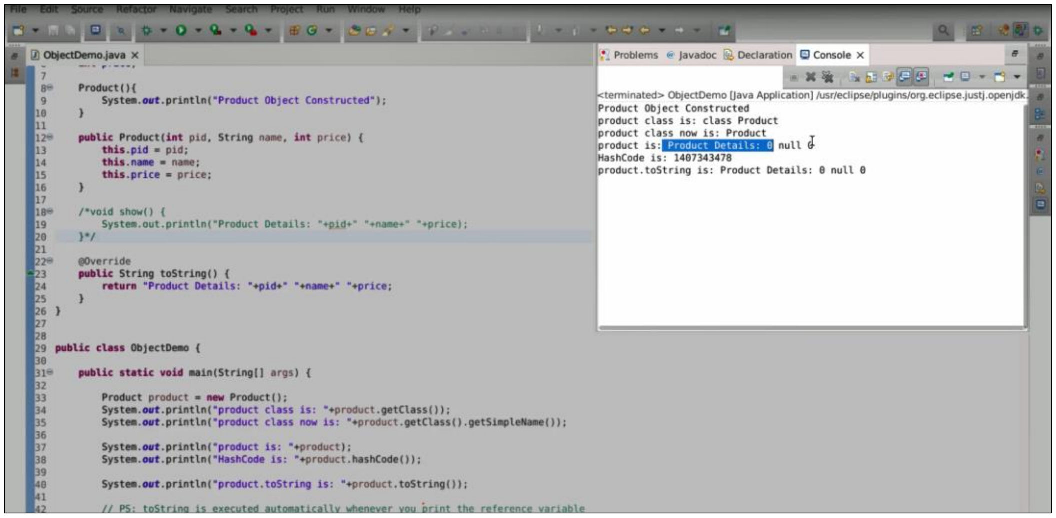
4.8 The **toString()** will be executed automatically whenever you print your reference variable. Let us write, **'PS: toString is executed automatically whenever you print the reference variable**' Let us uncomment this code and override a method called **toString()** here. You are overriding the **toString()** method of the parent Object class, and here you will give a return value. What you want to return is **'hello**.' Run the code, and you will get the output as **'hello'**



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

    public Product(int pid, String name, int price) {
        this.pid = pid;
        this.name = name;
        this.price = price;
    }

    void show() {
        System.out.println("Product Details: "+pid+" "+name+" "+price);
    }

    @Override
    public String toString() {
        return "Hello";
    }
}


public class ObjectDemo {

    public static void main(String[] args) {

        Product product = new Product();
        System.out.println("product class is: "+product.getClass());
        System.out.println("product class now is: "+product.getClass().getSimpleName());

        System.out.println("product is: "+product);
        System.out.println("HashCode is: "+product.hashCode());

        System.out.println("product.toString is: "+product.toString());

        // PS: toString is executed automatically whenever you print the reference variable

    }
```

```
<terminated> ObjectDemo [Java Application] /usr/eclipse/plugins/org.eclipse.justj.openjdk.
Product Object Constructed
product class is: class Product
product class now is: Product
product is: Hello
HashCode is: 1407343478
product.toString is: Hello
```

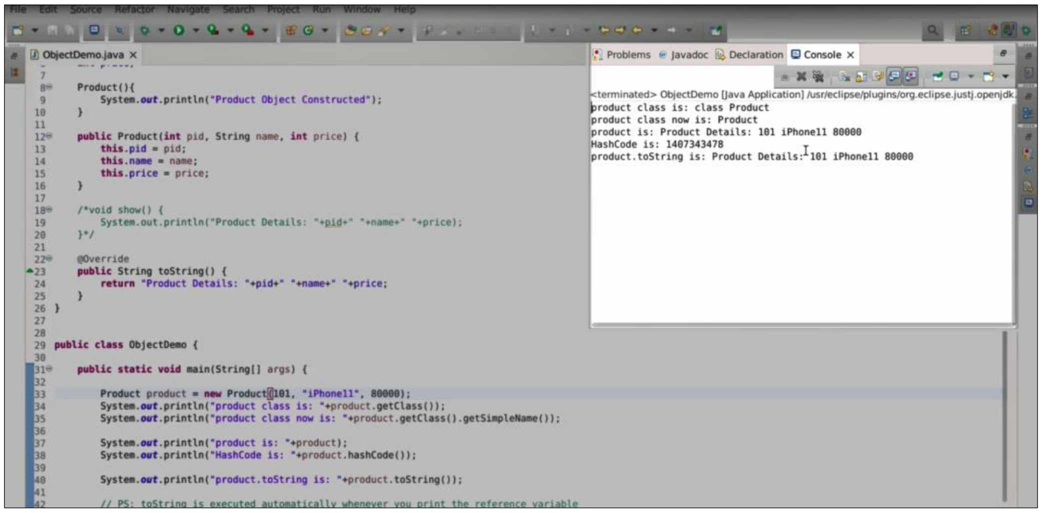
4.9 Rather than saying **'hello**,' you can return the product details and do not need to create a show method anymore. To display the data inside the object, you will use the **toString()** method, which you can override. By default, it will show that the product details are zero, null, and zero because you created a default product object



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

    public Product(int pid, String name, int price) {
        this.pid = pid;
        this.name = name;
        this.price = price;
    }

    /*void show() {
        System.out.println("Product Details: "+pid+" "+name+" "+price);
    }*/

    @Override
    public String toString() {
        return "Product Details: "+pid+" "+name+" "+price;
    }
}


public class ObjectDemo {

    public static void main(String[] args) {

        Product product = new Product();
        System.out.println("product class is: "+product.getClass());
        System.out.println("product class now is: "+product.getClass().getSimpleName());

        System.out.println("product is: "+product);
        System.out.println("HashCode is: "+product.hashCode());

        System.out.println("product.toString is: "+product.toString());

        // PS: toString is executed automatically whenever you print the reference variable
```

```
<terminated> ObjectDemo [Java Application] /usr/eclipse/plugins/org.eclipse.justj.openjdk.
Product Object Constructed
product class is: class Product
product class now is: Product
product is: Product Details: 0 null 0
HashCode is: 1407343478
product.toString is: Product Details: 0 null 0
```

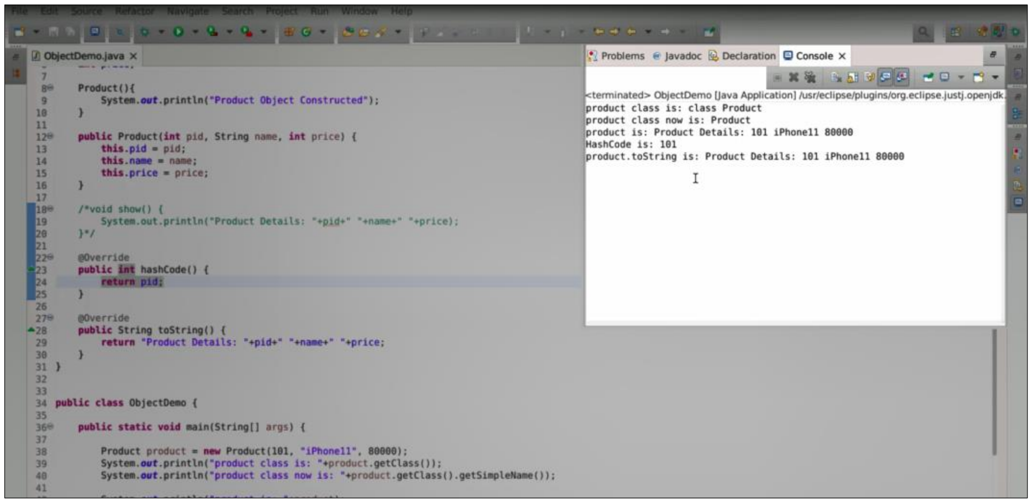
4.10 Let us pass a product ID, a name (such as 'iPhone 11'), and the price. When you run the code, you will see the data inside your object whenever you print your reference variable. This is the benefit of the **toString()** method



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

    public Product(int pid, String name, int price) {
        this.pid = pid;
        this.name = name;
        this.price = price;
    }

    /*void show() {
        System.out.println("Product Details: "+pid+" "+name+" "+price);
    }*/

    @Override
    public String toString() {
        return "Product Details: "+pid+" "+name+" "+price;
    }
}


public class ObjectDemo {

    public static void main(String[] args) {

        Product product = new Product(101, "iPhone11", 80000);
        System.out.println("product class is: "+product.getClass());
        System.out.println("product class now is: "+product.getClass().getSimpleName());

        System.out.println("product is: "+product);
        System.out.println("HashCode is: "+product.hashCode());

        System.out.println("product.toString is: "+product.toString());

        // PS: toString is executed automatically whenever you print the reference variable
```

```
<terminated> ObjectDemo [Java Application] /usr/eclipse/plugins/org.eclipse.justj.openjdk.
product class is: class Product
product class now is: Product
product is: Product Details: 101 iPhone11 80000
HashCode is: 1407343478
product.toString is: Product Details: 101 iPhone11 80000
```

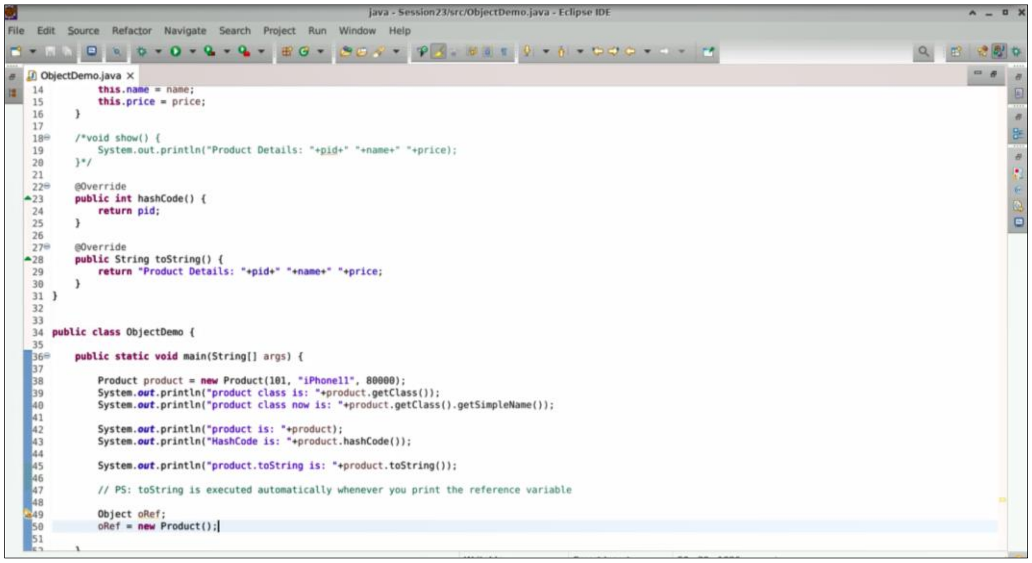
4.11 Similarly, you can override the method called **hashCode()**. Simply type **hashCode()** and then press Ctrl + Spacebar, and it will automatically be implemented. Then, return the product ID. This way, the hash code for your object becomes 101. The Object class in Java provides these methods



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

    public Product(int pid, String name, int price) {
        this.pid = pid;
        this.name = name;
        this.price = price;
    }

    /*void show() {
        System.out.println("Product Details: "+pid+" "+name+" "+price);
    }*/

    @Override
    public int hashCode() {
        return pid;
    }

    @Override
    public String toString() {
        return "Product Details: "+pid+" "+name+" "+price;
    }
}


public class ObjectDemo {

    public static void main(String[] args) {

        Product product = new Product(101, "iPhone11", 80000);
        System.out.println("product class is: "+product.getClass());
        System.out.println("product class now is: "+product.getClass().getSimpleName());
```

```
<terminated> ObjectDemo [Java Application] /usr/eclipse/plugins/org.eclipse.justj.openjdk.
product class is: class Product
product class now is: Product
product is: Product Details: 101 iPhone11 80000
HashCode is: 101
product.toString is: Product Details: 101 iPhone11 80000
```

4.12 If you want, the utility methods can be overridden, and they can be used for general purposes. The object can have this reference variable. And this reference variable can point to your product object. Whenever you create your objects, the parent is by default the class object. And runtime polymorphism works on the root level



```java
        this.name = name;
        this.price = price;
    }

    /*void show() {
        System.out.println("Product Details: "+pid+" "+name+" "+price);
    }*/

    @Override
    public int hashCode() {
        return pid;
    }

    @Override
    public String toString() {
        return "Product Details: "+pid+" "+name+" "+price;
    }
}


public class ObjectDemo {

    public static void main(String[] args) {

        Product product = new Product(101, "iPhone11", 80000);
        System.out.println("product class is: "+product.getClass());
        System.out.println("product class now is: "+product.getClass().getSimpleName());

        System.out.println("product is: "+product);
        System.out.println("HashCode is: "+product.hashCode());

        System.out.println("product.toString is: "+product.toString());

        // PS: toString is executed automatically whenever you print the reference variable

        Object oRef;
        oRef = new Product();
```

By following these steps, you have successfully implemented the use of object class in Java.