Lesson 03 Demo 10

Using this and Super in Java

Objective: To differentiate the use of **this** and **super** in java

Tools required: Eclipse IDE

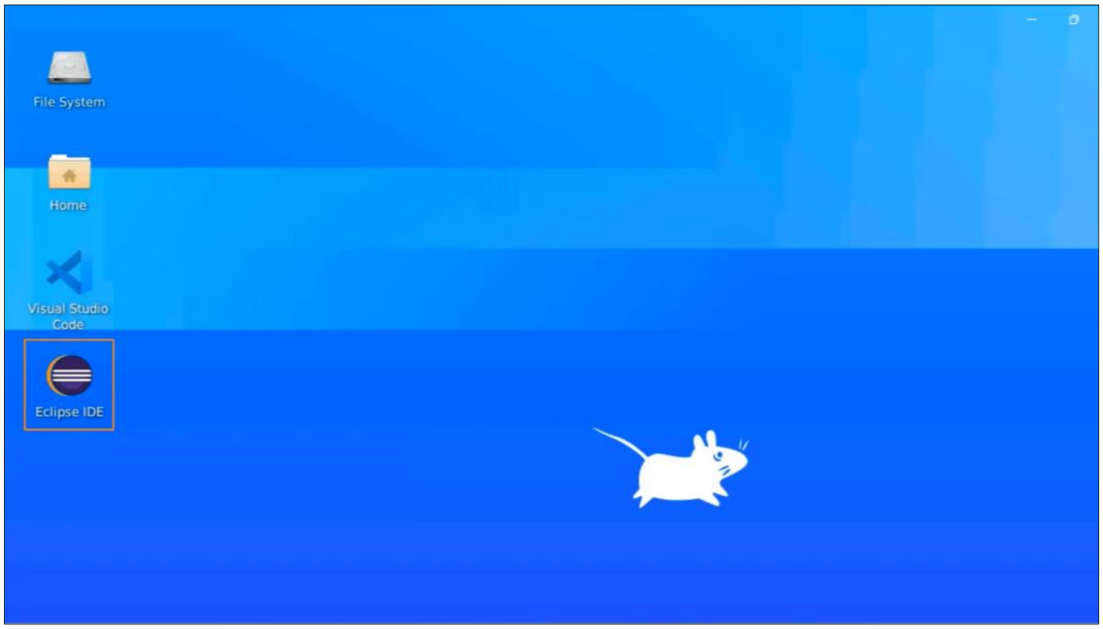
Prerequisites: None

Steps to be followed:

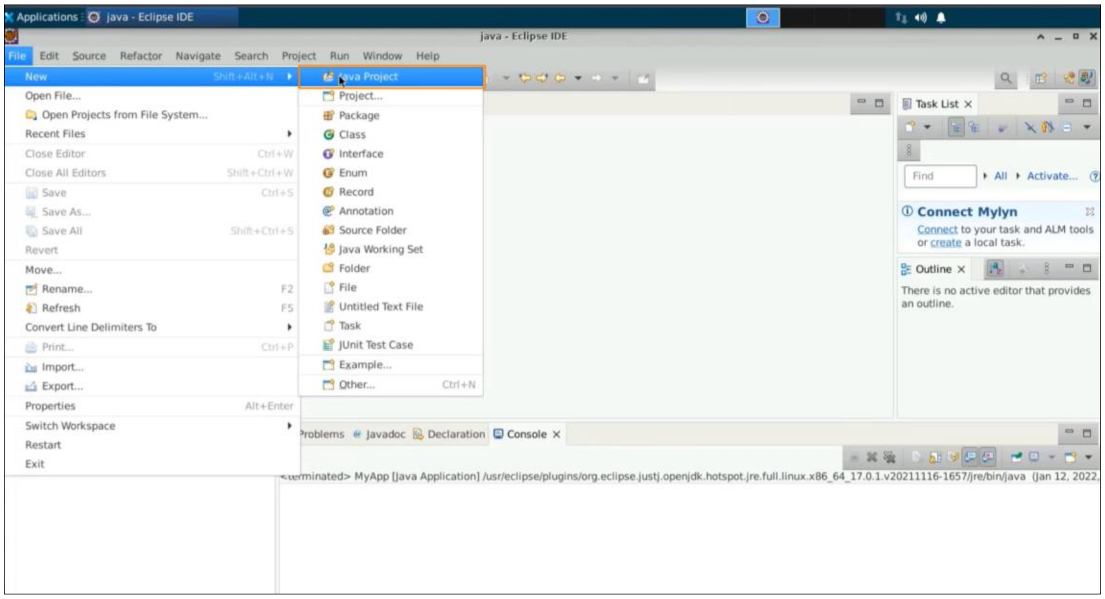
1. Create a class called **ThisDemo**, followed by selecting the main method
2. Create a class called **User** and a default constructor
3. Create two user objects, the first one to hold the hash code of the user object and the second one for a new user
4. Execute the code with examples
5. Implement constructor chaining and create a user object
6. Create a class **SuperDemo** with two classes parent and child
7. Create multiple constructors for the parent constructor and use parameterized constructors
8. Create the object of the child with the default constructor while reusing the code
9. Create an inheritance relationship
10. Implement a super execution call and override the default behavior

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

1.1 Open the **Eclipse IDE**



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

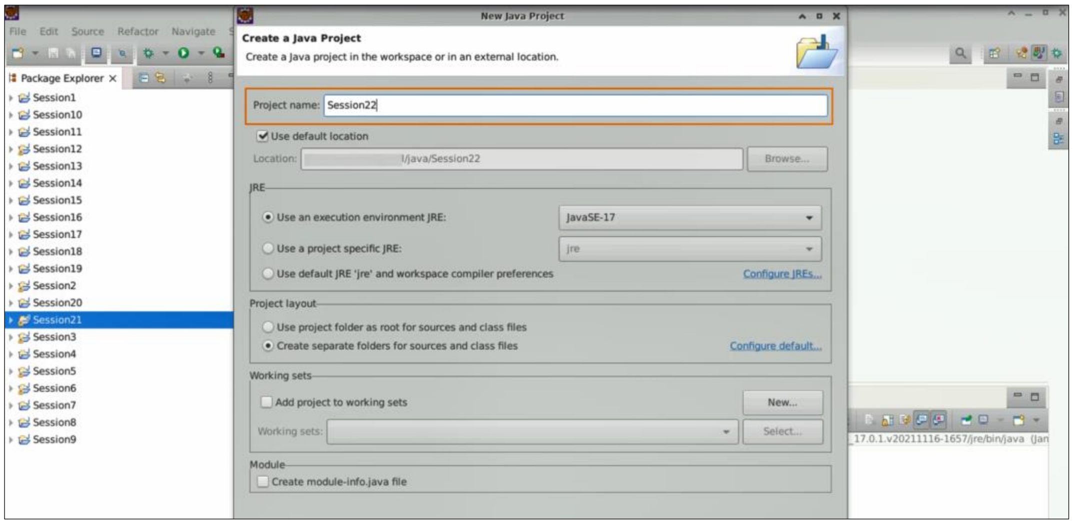


Connect to your task and ALM tools or create a local task.

There is no active editor that provides an outline.

```
<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 **“Session22”,** uncheck **“Create a module info.Java file”**, and Click on **Finish**



Create a Java Project
Create a Java project in the workspace or in an external location.

Project name: Session22

☑ Use default location

Location: [redacted]l/java/Session22

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

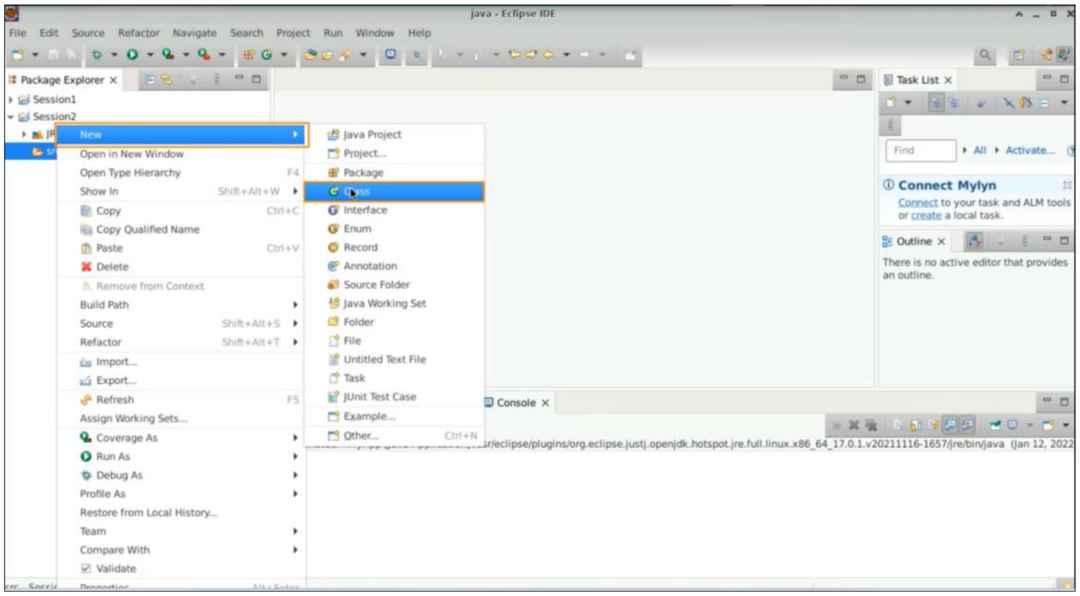
Working sets:

Module

☐ Create module-info.java file

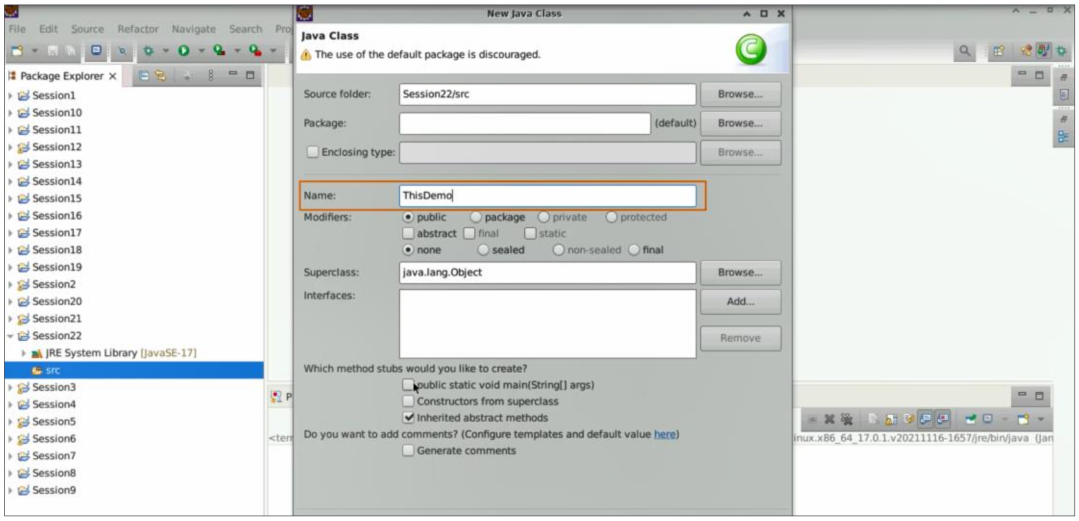
17.0.1.v20211116-1657/jre/bin/java (Jan

1.4 With a **Session22** on the **src**, do a right-click and create a **new class**



...r/eclipse/plugins/org.eclipse.justj.openjdk.hotspot.jre.full.linux.x86_64_17.0.1.v20211116-1657/jre/bin/java (Jan 12, 2022

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



Java Class

The use of the default package is discouraged.

Source folder: Session22/src

Package: (default)

Enclosing type:

Name: ThisDemo

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?

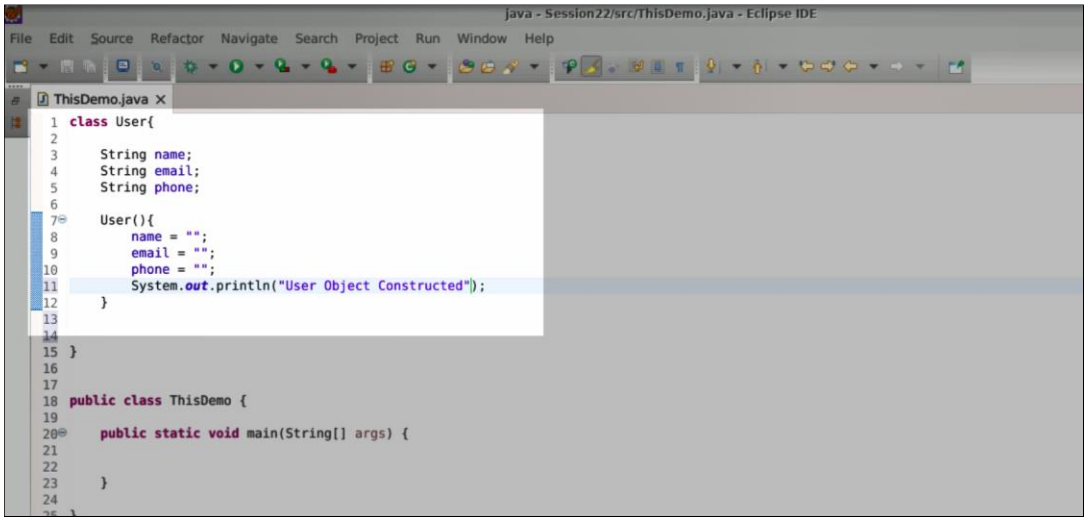
- public static void main(String[] args)
- Constructors from superclass
- [✓] Inherited abstract methods

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

- Generate comments

Step 2: Create a class called User and a default constructor

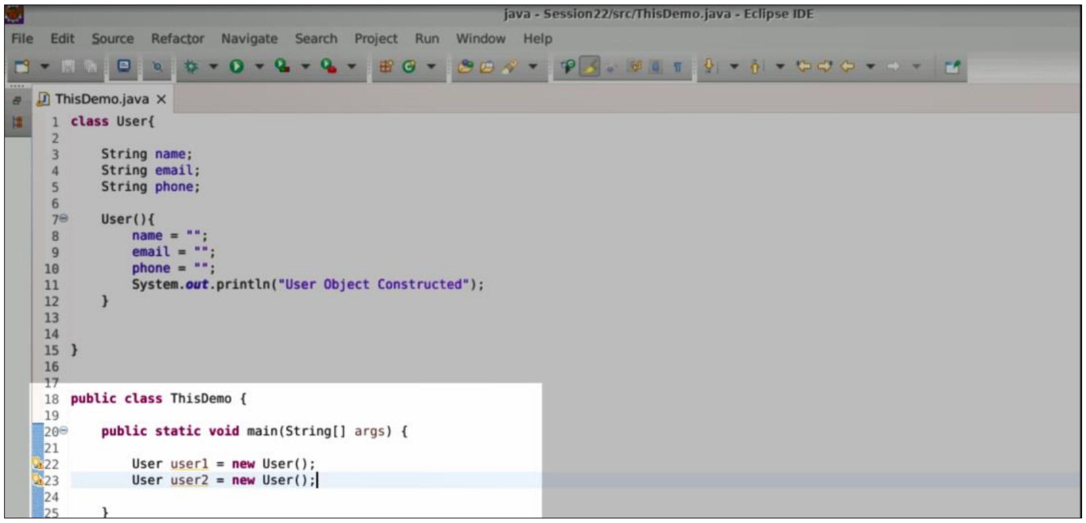
2.1 Create a class called User. For a User, you will have a **name**, an **email**, and a **phone number** as three attributes. Create a default constructor. In the default constructor, initialize all three strings to empty values. With this, write **System.out.println("Object constructed")**


java - Session22/src/ThisDemo.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
ThisDemo.java

```java
class User{

    String name;
    String email;
    String phone;

    User(){
        name = "";
        email = "";
        phone = "";
        System.out.println("User Object Constructed");
    }


}


public class ThisDemo {

    public static void main(String[] args) {


    }

```

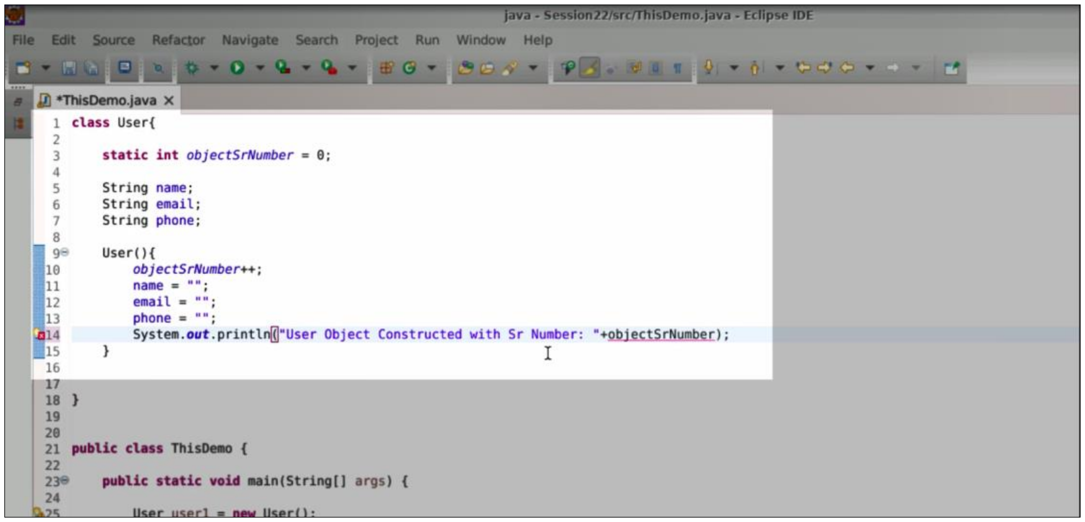
Step 3: Create two user objects, the first one to hold the hash code of the user object and the second one for a new user

3.1 Write **System.out.println("Object constructed")**, not to mention "user object constructed". Let us create two User objects: User **user1 = new User()**. It holds the hash code of the user object. Then add User **user2 = new User()**


java - Session22/src/ThisDemo.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
ThisDemo.java

```
class User{

    String name;
    String email;
    String phone;

    User(){
        name = "";
        email = "";
        phone = "";
        System.out.println("User Object Constructed");
    }


}


public class ThisDemo {

    public static void main(String[] args) {

        User user1 = new User();
        User user2 = new User();

    }
```

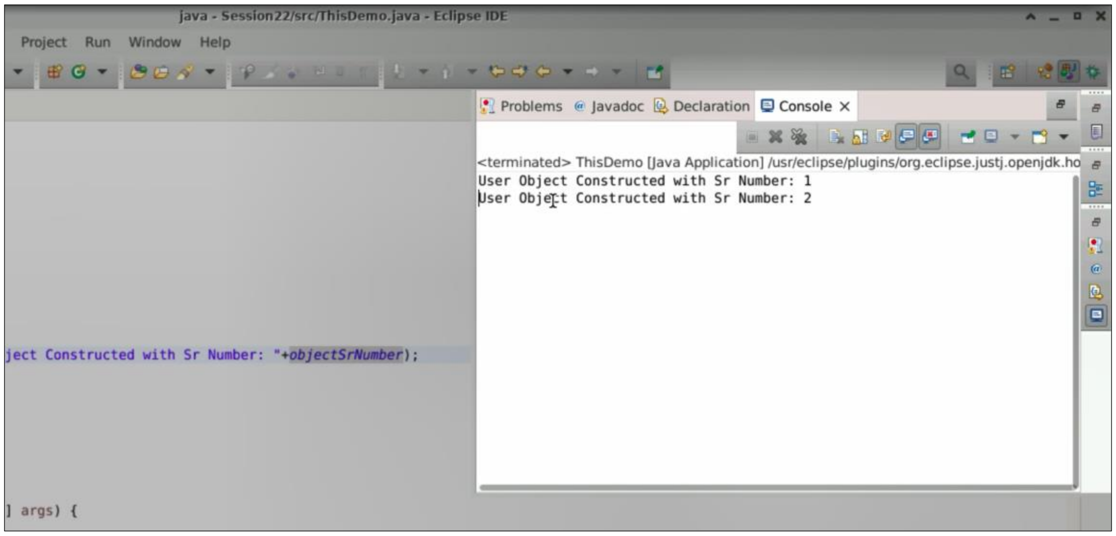
3.2 To make things more meaningful, you will use the static keyword. Declare **static int objectSerialNumber**, and by default, this value is **0**. Whenever an object is created, increment **objectSerialNumber** by 1. You can print **("User object constructed with Sr Number: " + objectSrNumber);**


java - Session22/src/ThisDemo.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
*ThisDemo.java

```
class User{

    static int objectSrNumber = 0;

    String name;
    String email;
    String phone;

    User(){
        objectSrNumber++;
        name = "";
        email = "";
        phone = "";
        System.out.println("User Object Constructed with Sr Number: "+objectSrNumber);
    }


}


public class ThisDemo {

    public static void main(String[] args) {

        User user1 = new User();
```

Step 4: Execute the code with examples

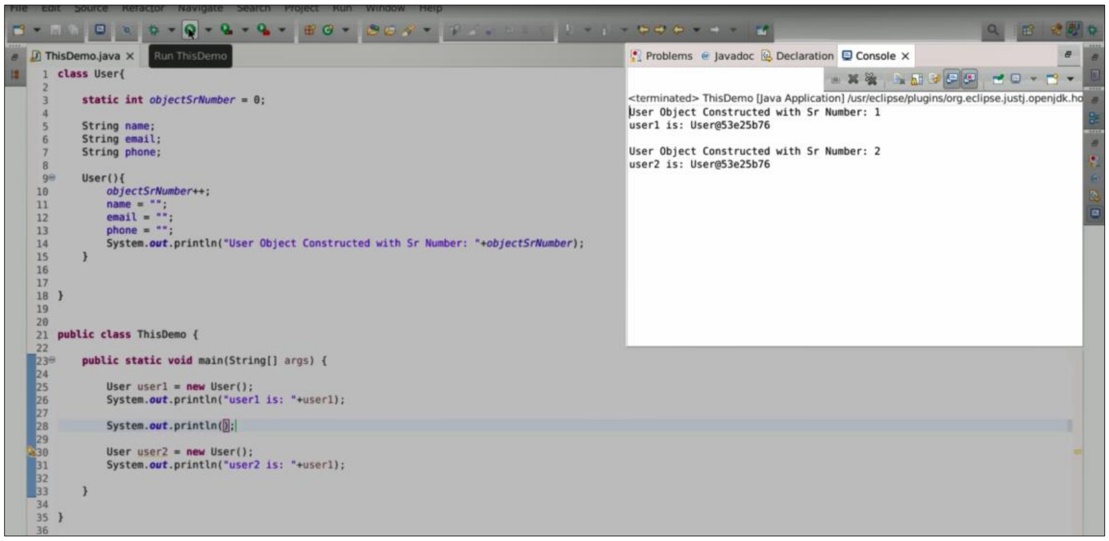
4.1 If you run the program, you will see that a user object is constructed with serial number 1, and another user object is constructed with serial number 2. Essentially, it is just a counter that you are using



```
ject Constructed with Sr Number: "+objectSrNumber);

] args) {
```

```
<terminated> ThisDemo [Java Application] /usr/eclipse/plugins/org.eclipse.justj.openjdk.ho
User Object Constructed with Sr Number: 1
User Object Constructed with Sr Number: 2
```

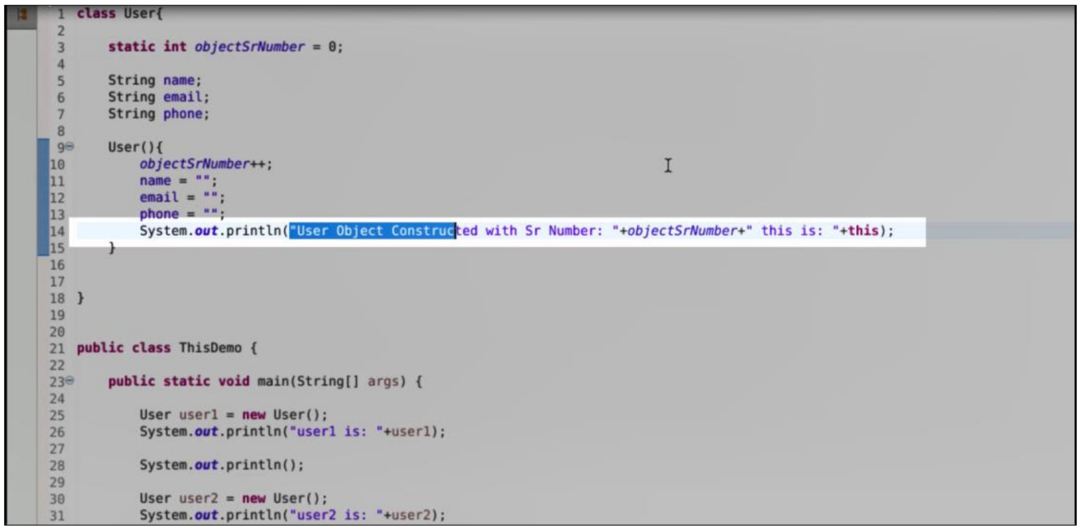
4.2 Now, after this, print **user1 is + user1**, and print **user2** is User two. In between, add an empty print line, and run the code again. You will see that **user1** is a user object with a hash code like **53e25b76**. Similarly, **user2** refers to another hash code, indicating that it is a separate memory location



```java
class User{

    static int objectSrNumber = 0;

    String name;
    String email;
    String phone;

    User(){
        objectSrNumber++;
        name = "";
        email = "";
        phone = "";
        System.out.println("User Object Constructed with Sr Number: "+objectSrNumber);
    }


}


public class ThisDemo {

    public static void main(String[] args) {

        User user1 = new User();
        System.out.println("user1 is: "+user1);

        System.out.println();

        User user2 = new User();
        System.out.println("user2 is: "+user1);

    }

}
```

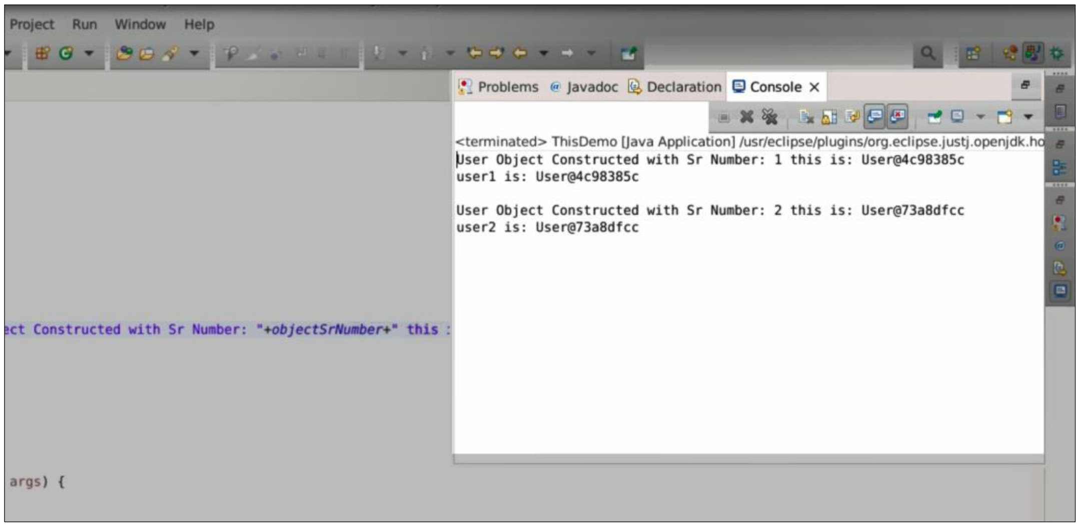
```
<terminated> ThisDemo [Java Application] /usr/eclipse/plugins/org.eclipse.justj.openjdk.ho
User Object Constructed with Sr Number: 1
user1 is: User@53e25b76

User Object Constructed with Sr Number: 2
user2 is: User@53e25b76
```

4.3 Similarly, **user2** refers to another hash code, which means it is an entirely separate memory location. Here you can see it is a different memory location. Inside this constructor, add **this is: + this**. That means, along with "**user object constructed**," you will also be printing something known as a reference variable



```
class User{

    static int objectSrNumber = 0;

    String name;
    String email;
    String phone;

    User(){
        objectSrNumber++;
        name = "";
        email = "";
        phone = "";
        System.out.println("User Object Constructed with Sr Number: "+objectSrNumber+" this is: "+this);
    }


}


public class ThisDemo {

    public static void main(String[] args) {

        User user1 = new User();
        System.out.println("user1 is: "+user1);

        System.out.println();

        User user2 = new User();
        System.out.println("user2 is: "+user2);
```

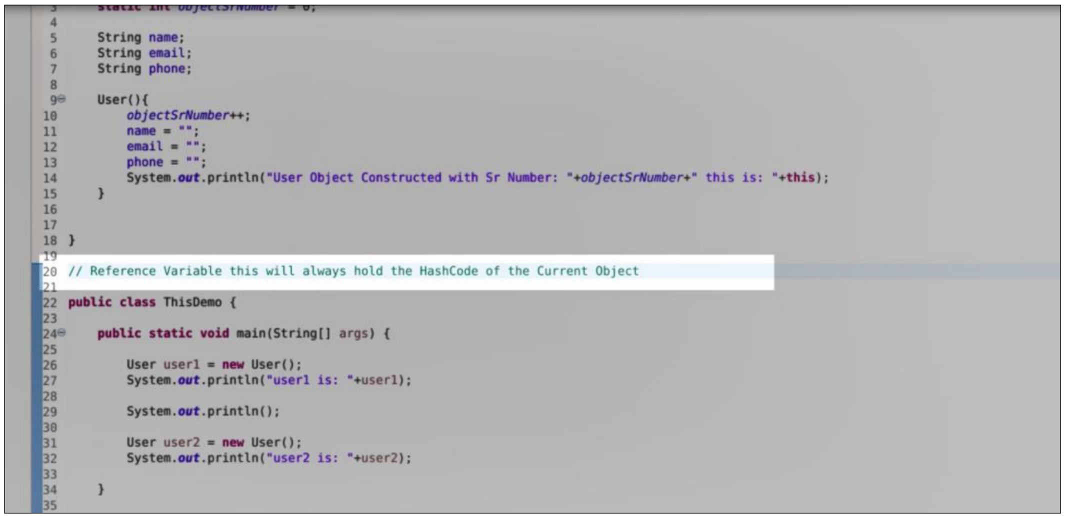
4.4 What you see is that when you create the very first object with the reference variable **user1**, whatever is in **user1** is the same as inside this as the reference variable. This contains the same hash code that **user1** is containing



```
ect Constructed with Sr Number: "+objectSrNumber+" this :

args) {
```

```
<terminated> ThisDemo [Java Application] /usr/eclipse/plugins/org.eclipse.justj.openjdk.ho
User Object Constructed with Sr Number: 1 this is: User@4c98385c
user1 is: User@4c98385c

User Object Constructed with Sr Number: 2 this is: User@73a8dfcc
user2 is: User@73a8dfcc
```

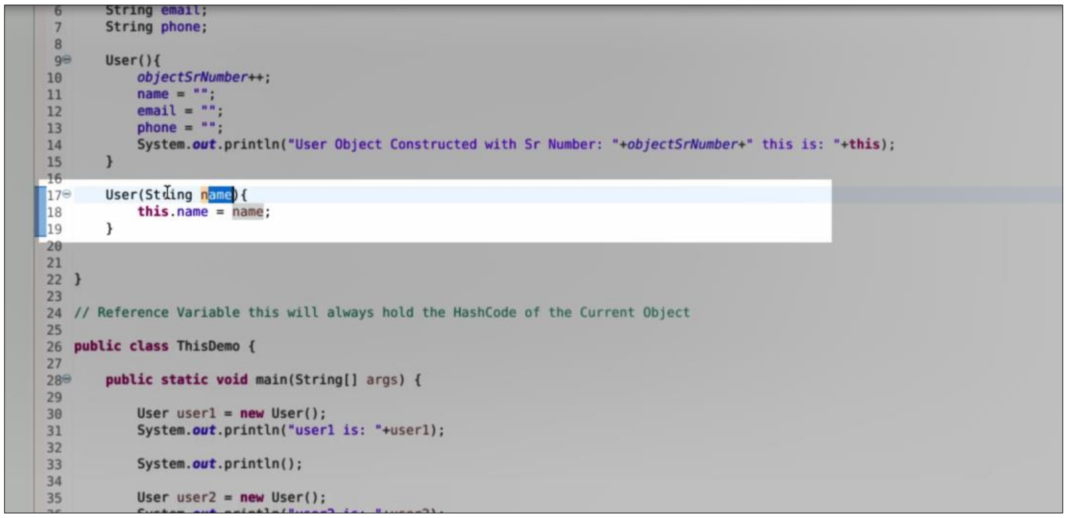
4.5 The reference variable **'this'** always holds the memory address of the current object. It refers to whichever object is currently in action



```
    static int objectSrNumber = 0;

    String name;
    String email;
    String phone;

    User(){
        objectSrNumber++;
        name = "";
        email = "";
        phone = "";
        System.out.println("User Object Constructed with Sr Number: "+objectSrNumber+" this is: "+this);
    }


}

// Reference Variable this will always hold the HashCode of the Current Object

public class ThisDemo {

    public static void main(String[] args) {

        User user1 = new User();
        System.out.println("user1 is: "+user1);

        System.out.println();

        User user2 = new User();
        System.out.println("user2 is: "+user2);

    }

```

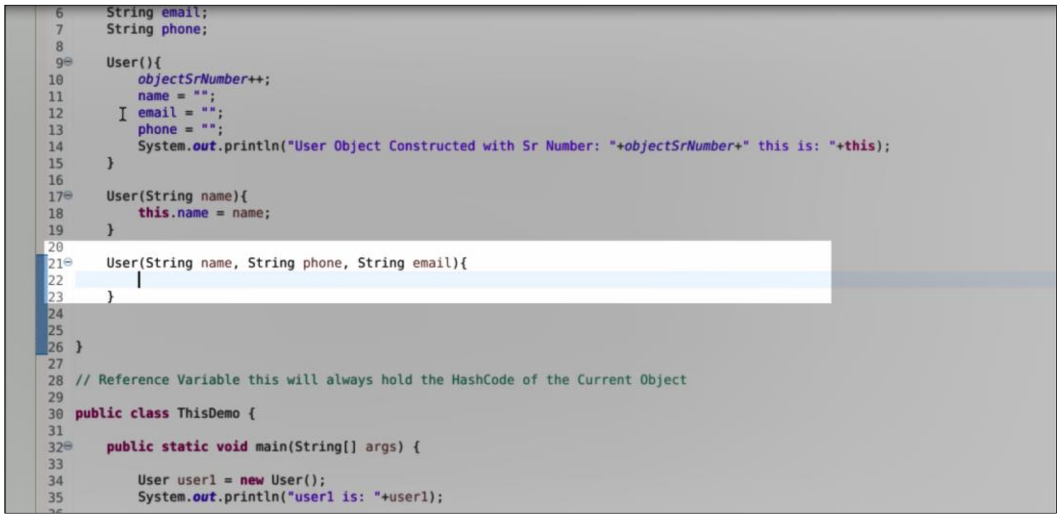
Step 5: Implement constructor chaining and create a user object

5.1 Now, let us explore another usage of **'this'** for constructor chaining and how it is used. We are going to create another user object with a name. Here, **'this.name'** refers to the name of the current object, where **'name'** on the right-hand side is the name of your attribute for this input variable



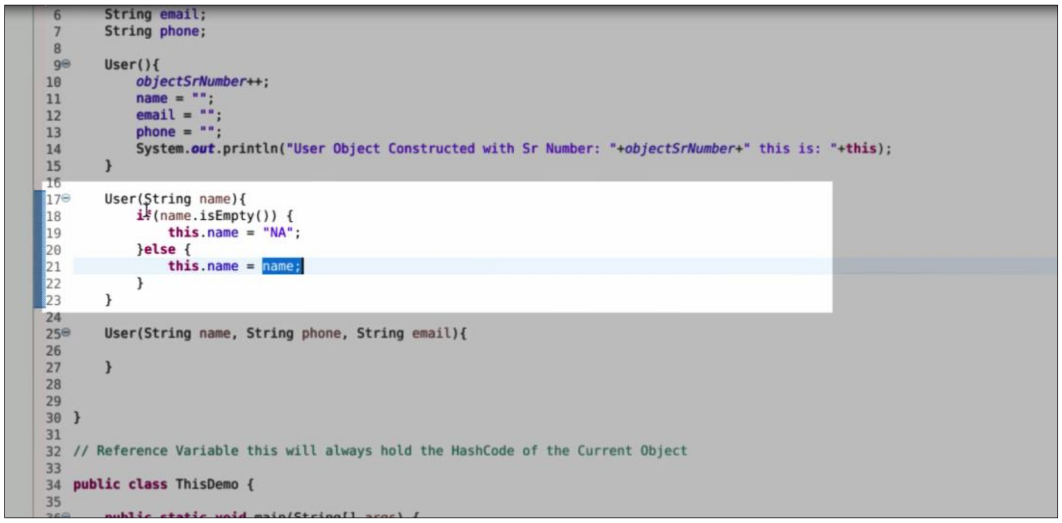
```
    String email;
    String phone;

    User(){
        objectSrNumber++;
        name = "";
        email = "";
        phone = "";
        System.out.println("User Object Constructed with Sr Number: "+objectSrNumber+" this is: "+this);
    }

    User(String name){
        this.name = name;
    }


}

// Reference Variable this will always hold the HashCode of the Current Object

public class ThisDemo {

    public static void main(String[] args) {

        User user1 = new User();
        System.out.println("user1 is: "+user1);

        System.out.println();

        User user2 = new User();
        System.out.println("user2 is: "+user2);
```

5.2 Now, let us consider a User class with three constructors: one that takes a String parameter for **name**, another that takes String parameters for **name**, **phone**, and **email**, and a third constructor



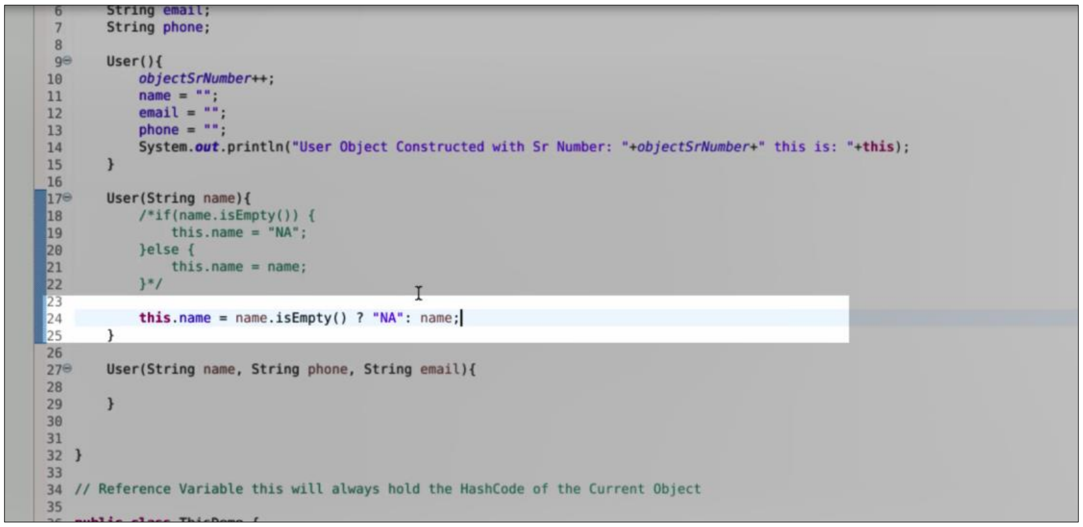
```
    String email;
    String phone;

    User(){
        objectSrNumber++;
        name = "";
        email = "";
        phone = "";
        System.out.println("User Object Constructed with Sr Number: "+objectSrNumber+" this is: "+this);
    }

    User(String name){
        this.name = name;
    }

    User(String name, String phone, String email){
        
    }


}

// Reference Variable this will always hold the HashCode of the Current Object

public class ThisDemo {

    public static void main(String[] args) {

        User user1 = new User();
        System.out.println("user1 is: "+user1);
```

5.3 If you are just substituting the name, then you will need to check the validation. If the name is empty, you can handle it in the else part. For example, if the name is empty, you might want to indicate that **this.name** is not available



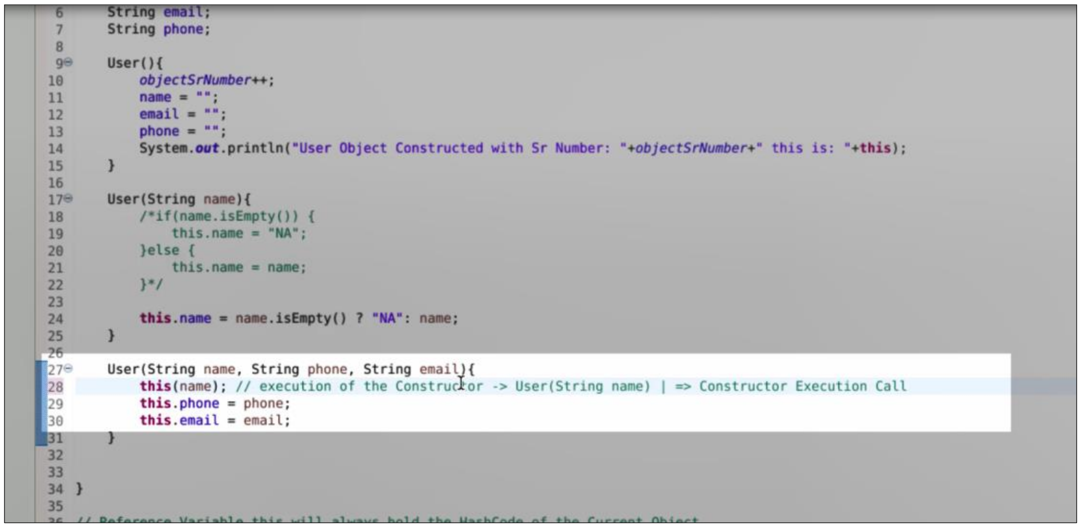
```
    String email;
    String phone;

    User(){
        objectSrNumber++;
        name = "";
        email = "";
        phone = "";
        System.out.println("User Object Constructed with Sr Number: "+objectSrNumber+" this is: "+this);
    }

    User(String name){
        if(name.isEmpty()) {
            this.name = "NA";
        }else {
            this.name = name;
        }
    }

    User(String name, String phone, String email){

    }


}

// Reference Variable this will always hold the HashCode of the Current Object

public class ThisDemo {

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

5.4 Alternatively, instead of explicitly writing **'this'**, you can simply **use 'this.name = name.isEmpty() ? "NA" : name;'**. This condition checks if the **'name'** is empty. If it is, it assigns **'NA'** to **'this.name';** otherwise, it assigns the **'name'** value



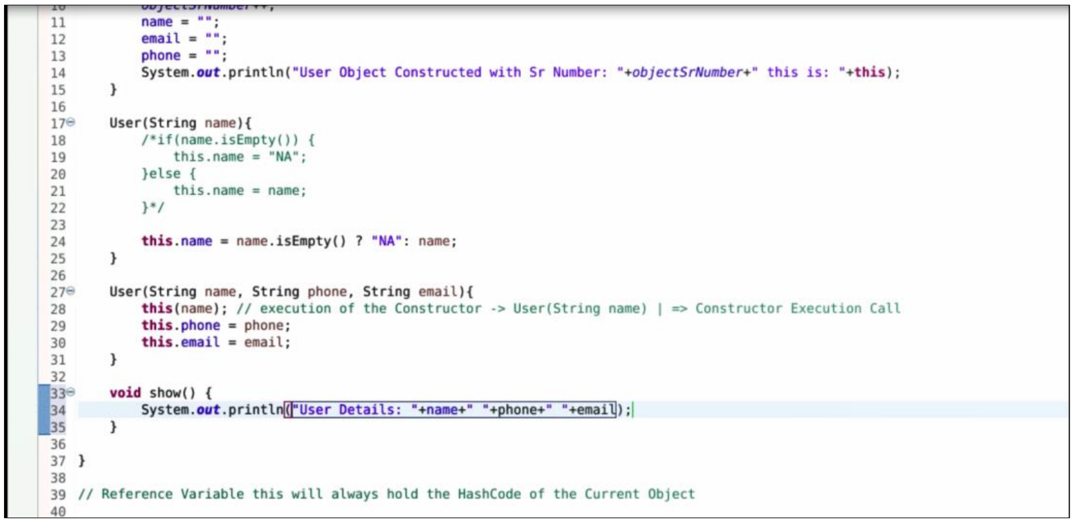
```
    String email;
    String phone;

    User(){
        objectSrNumber++;
        name = "";
        email = "";
        phone = "";
        System.out.println("User Object Constructed with Sr Number: "+objectSrNumber+" this is: "+this);
    }

    User(String name){
        /*if(name.isEmpty()) {
            this.name = "NA";
        }else {
            this.name = name;
        }*/

        this.name = name.isEmpty() ? "NA": name;
    }

    User(String name, String phone, String email){

    }


}

// Reference Variable this will always hold the HashCode of the Current Object

public class ThisDemo {
```

5.5 In the constructor, which is the user constructor with three inputs, you will execute **'this'** and pass the **'name'**, calling it as the execution of the constructor **'Username'**. Then, you will add **'this.phone'** and **'this.email'** as **'phone'** and **'email'**, respectively. You can make the constructor execute from some other constructor using the reference 'this'. Here, you call this a constructor execution call



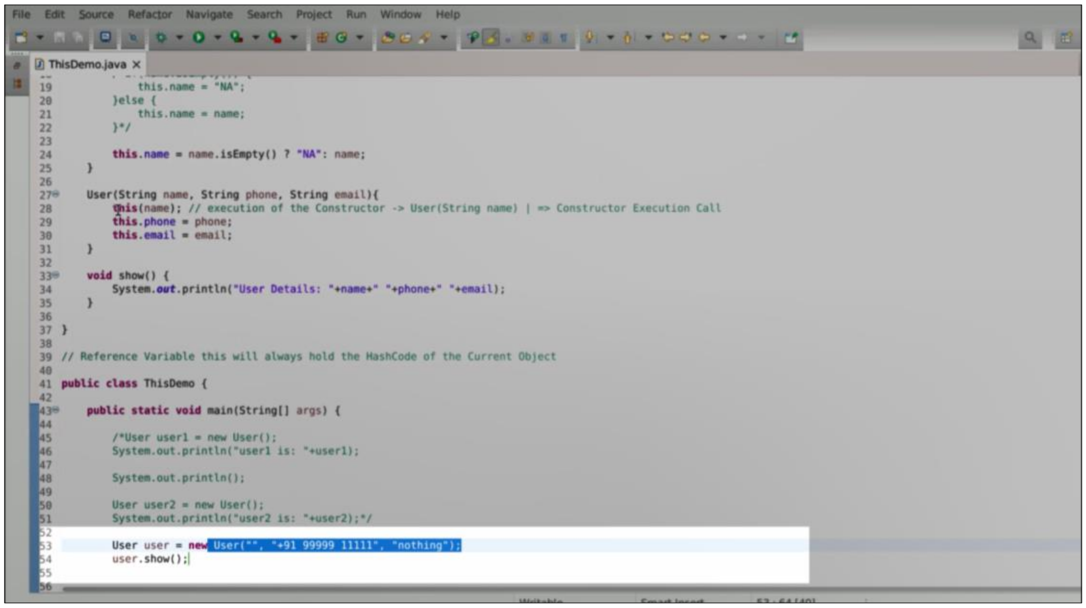
```
    String email;
    String phone;

    User(){
        objectSrNumber++;
        name = "";
        email = "";
        phone = "";
        System.out.println("User Object Constructed with Sr Number: "+objectSrNumber+" this is: "+this);
    }

    User(String name){
        /*if(name.isEmpty()) {
            this.name = "NA";
        }else {
            this.name = name;
        }*/

        this.name = name.isEmpty() ? "NA": name;
    }

    User(String name, String phone, String email){
        this(name); // execution of the Constructor -> User(String name) | => Constructor Execution Call
        this.phone = phone;
        this.email = email;
    }


}

// Reference Variable this will always hold the HashCode of the Current Object
```

5.6 Now, after the interactions, we will define a method called **'show'** that displays user details such as the name, phone number, and email



```java
        name = "";
        email = "";
        phone = "";
        System.out.println("User Object Constructed with Sr Number: "+objectSrNumber+" this is: "+this);
    }

    User(String name){
        /*if(name.isEmpty()) {
            this.name = "NA";
        }else {
            this.name = name;
        }*/

        this.name = name.isEmpty() ? "NA": name;
    }

    User(String name, String phone, String email){
        this(name); // execution of the Constructor -> User(String name) | => Constructor Execution Call
        this.phone = phone;
        this.email = email;
    }

    void show() {
        System.out.println("User Details: "+name+" "+phone+" "+email);
    }

}

// Reference Variable this will always hold the HashCode of the Current Object
```

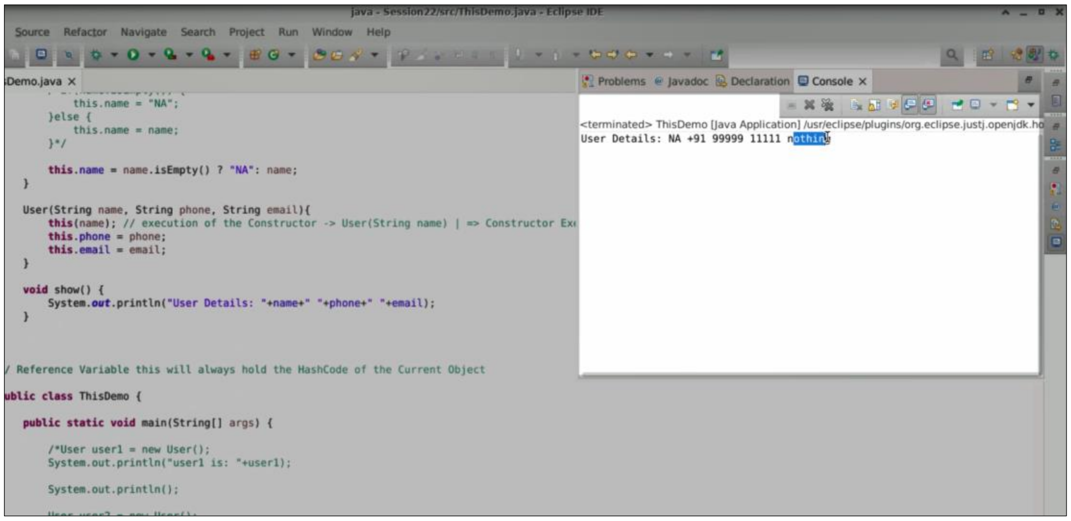
5.7 Let us create a user object that is like user, user three is a new user, or do a comment here, Let us not mix the fundamentals. You will say user as a new user, for the name you are going to pass something known as empty. For the phone, you will pass some number, and for the email, you are going to pass. Let's see what happens when you say **user.show**



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

```java
            this.name = "NA";
        }else {
            this.name = name;
        }*/

        this.name = name.isEmpty() ? "NA": name;
    }

    User(String name, String phone, String email){
        this(name); // execution of the Constructor -> User(String name) | => Constructor Execution Call
        this.phone = phone;
        this.email = email;
    }

    void show() {
        System.out.println("User Details: "+name+" "+phone+" "+email);
    }

}

// Reference Variable this will always hold the HashCode of the Current Object

public class ThisDemo {

    public static void main(String[] args) {

        /*User user1 = new User();
        System.out.println("user1 is: "+user1);

        System.out.println();

        User user2 = new User();
        System.out.println("user2 is: "+user2);*/

        User user = new User("", "+91 99999 11111", "nothing");
        user.show();
```

5.8 Run the program and here you are with the NA phone number and nothing. You are now able to execute your constructors from the other constructors. You can create some basic algorithms and then you can try working on it

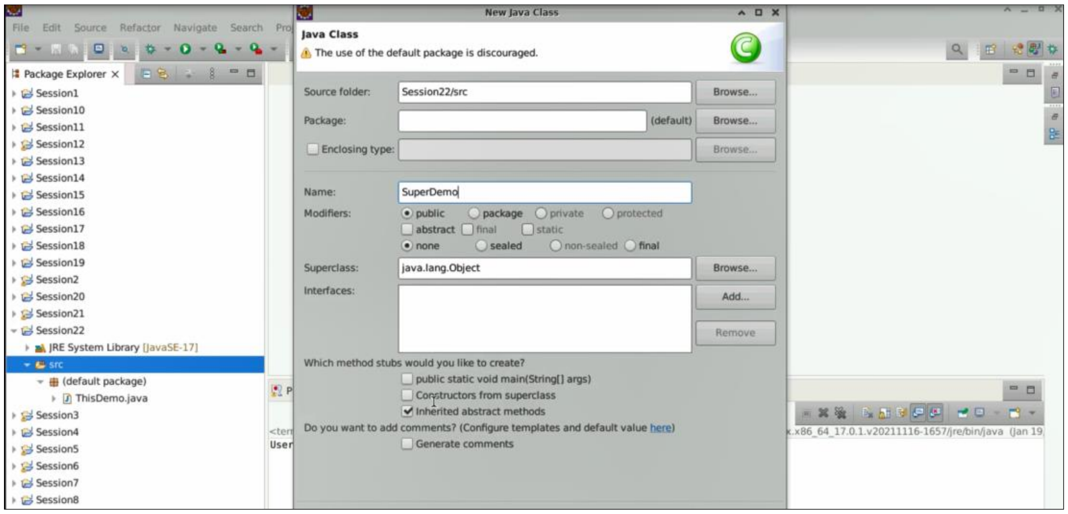


```
            this.name = "NA";
        }else {
            this.name = name;
        }*/

        this.name = name.isEmpty() ? "NA": name;
    }

    User(String name, String phone, String email){
        this(name); // execution of the Constructor -> User(String name) | => Constructor Ex
        this.phone = phone;
        this.email = email;
    }

    void show() {
        System.out.println("User Details: "+name+" "+phone+" "+email);
    }


/ Reference Variable this will always hold the HashCode of the Current Object

ublic class ThisDemo {

    public static void main(String[] args) {

        /*User user1 = new User();
        System.out.println("user1 is: "+user1);

        System.out.println();
```

```
<terminated> ThisDemo [Java Application] /usr/eclipse/plugins/org.eclipse.justj.openjdk.ho
User Details: NA +91 99999 11111 nothing
```

Step 6: Create a class SuperDemo with two classes parent and child

6.1 Now let us see what is meant by super. Let us write a new class by the name **SuperDemo**



Java Class

The use of the default package is discouraged.

Source folder: Session22/src

Package: (default)

Enclosing type:

Name: SuperDemo

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)

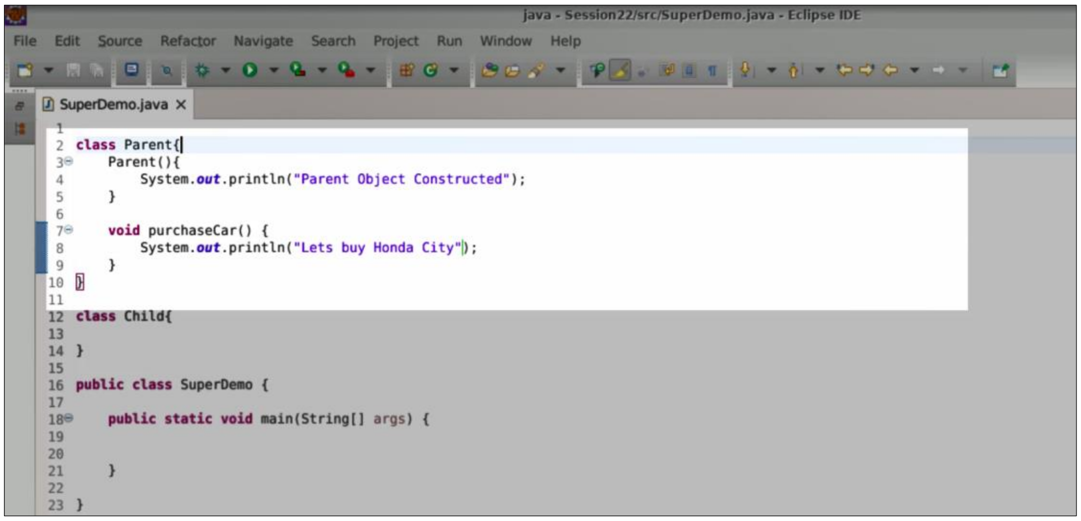
Constructors from superclass

[✓ Inherited abstract methods]

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

Generate comments

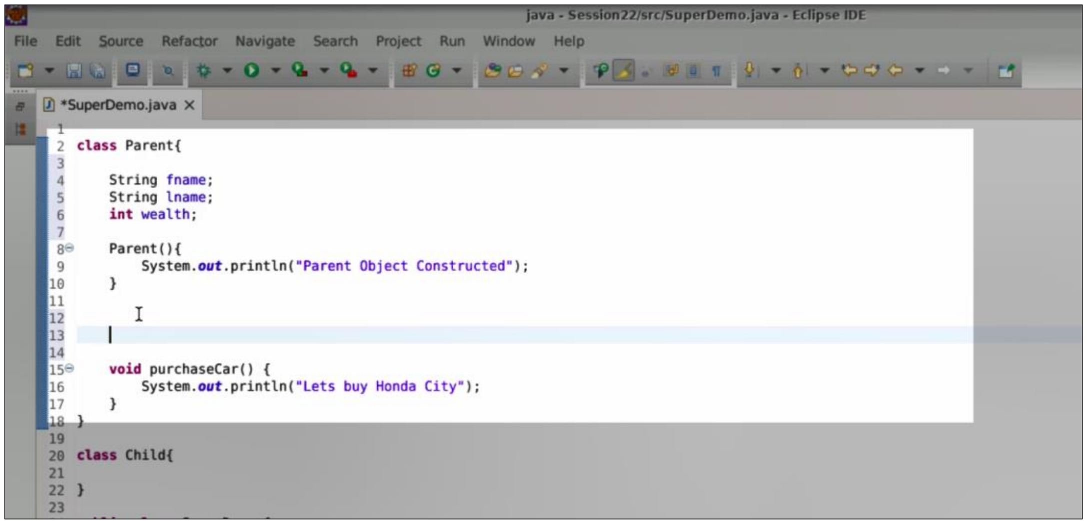
6.2 Here create two classes, let us say a class called Parent, and a class called Child. For the parent, you have a parent constructor. Simply print the parent object constructed. Here parent has a method called purchasing a car. And here parents say, let us buy Honda City


java - Session22/src/SuperDemo.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
SuperDemo.java

```

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

    void purchaseCar() {
        System.out.println("Lets buy Honda City");
    }
}

class Child{

}

public class SuperDemo {

    public static void main(String[] args) {


    }

}
```

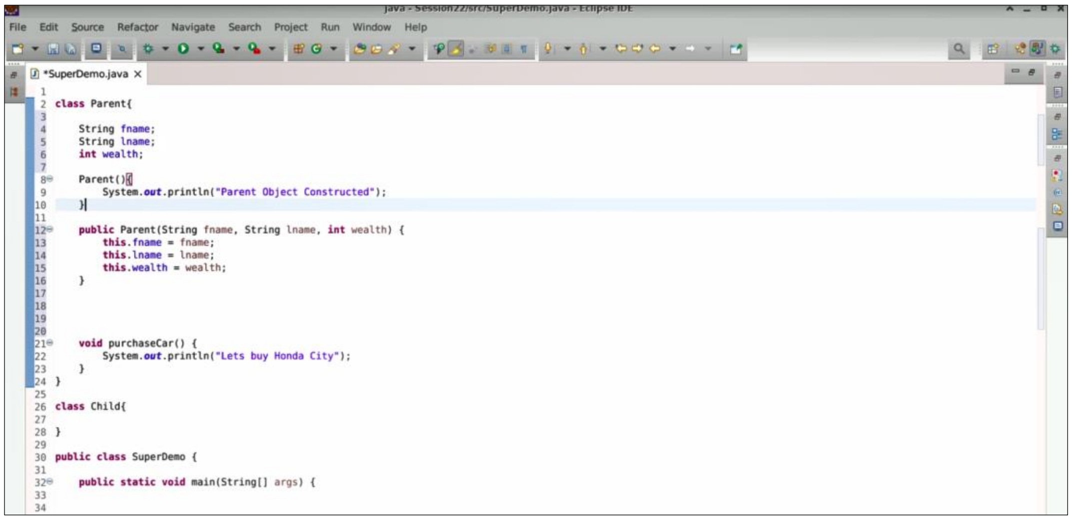
Step 7: Create multiple constructors for the parent constructor and use parameterized constructors

7.1 For the parent constructor you can create multiple constructors for that, let us take an attribute, here you get the first name, and you get a wealth attribute and will also take the last name as another attribute. You have 3 attributes


java - Session22/src/SuperDemo.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
*SuperDemo.java

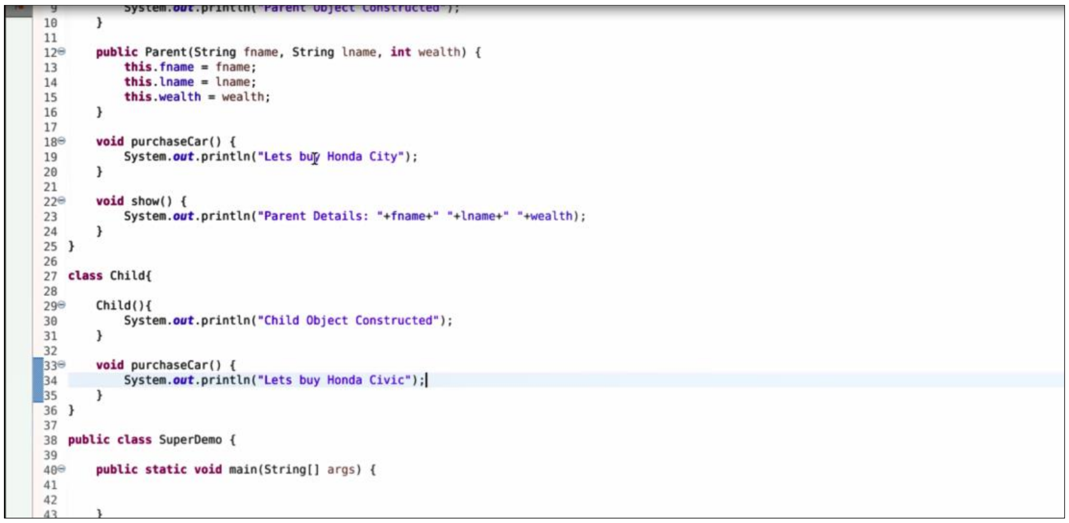
```

class Parent{

    String fname;
    String lname;
    int wealth;

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




    void purchaseCar() {
        System.out.println("Lets buy Honda City");
    }
}

class Child{

}

```

7.2 Here you have a default object construction, and you can use something known as parameterized constructor, generate the constructor using the fields and here you are. You got the constructor with all three details coming in


java - Session22/src/SuperDemo.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
*SuperDemo.java

```java
class Parent{

    String fname;
    String lname;
    int wealth;

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

    public Parent(String fname, String lname, int wealth) {
        this.fname = fname;
        this.lname = lname;
        this.wealth = wealth;
    }



    void purchaseCar() {
        System.out.println("Lets buy Honda City");
    }
}

class Child{

}

public class SuperDemo {

    public static void main(String[] args) {

```

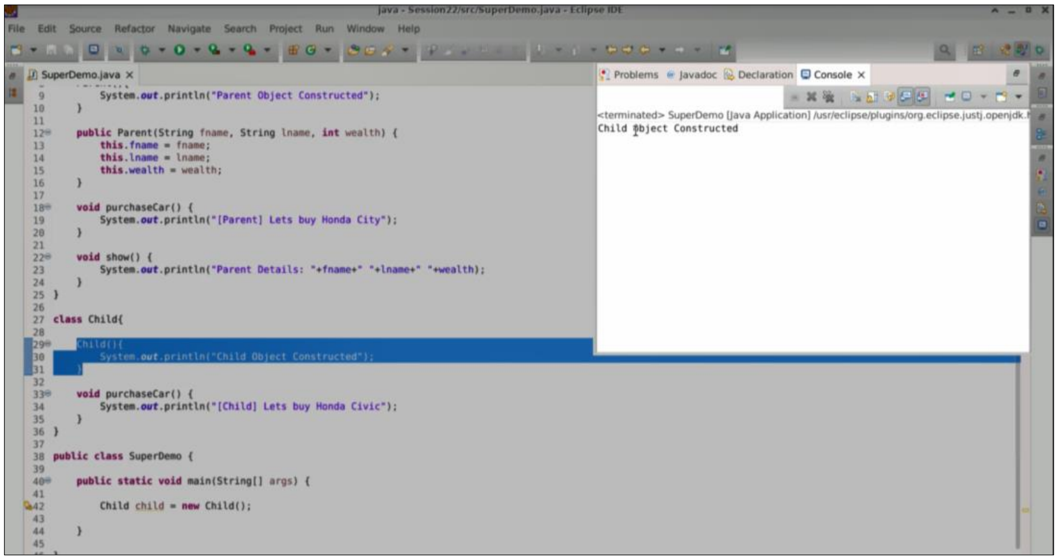
7.3 We will add a method called "**show**" to display parent details like first and last name (likely stored in variables named "**fname**" and "**lname**") and maybe belt level ("**belt**"). Then, within a child class inheriting from the parent, we'll create a default constructor that announces object creation and override a "**purchaseCar**" method to specify a car purchase (Honda Civic here)



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

    public Parent(String fname, String lname, int wealth) {
        this.fname = fname;
        this.lname = lname;
        this.wealth = wealth;
    }

    void purchaseCar() {
        System.out.println("Lets buy Honda City");
    }

    void show() {
        System.out.println("Parent Details: "+fname+" "+lname+" "+wealth);
    }
}

class Child{

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

    void purchaseCar() {
        System.out.println("Lets buy Honda Civic");
    }
}

public class SuperDemo {

    public static void main(String[] args) {


    }
```

Step 8: Create the object of the child with the default constructor while reusing the code

8.1 Now, what is the very first use of a parent? Mean super. Create the object of the child with the default constructor when you run the code, what you will see is that there is this child object constructed

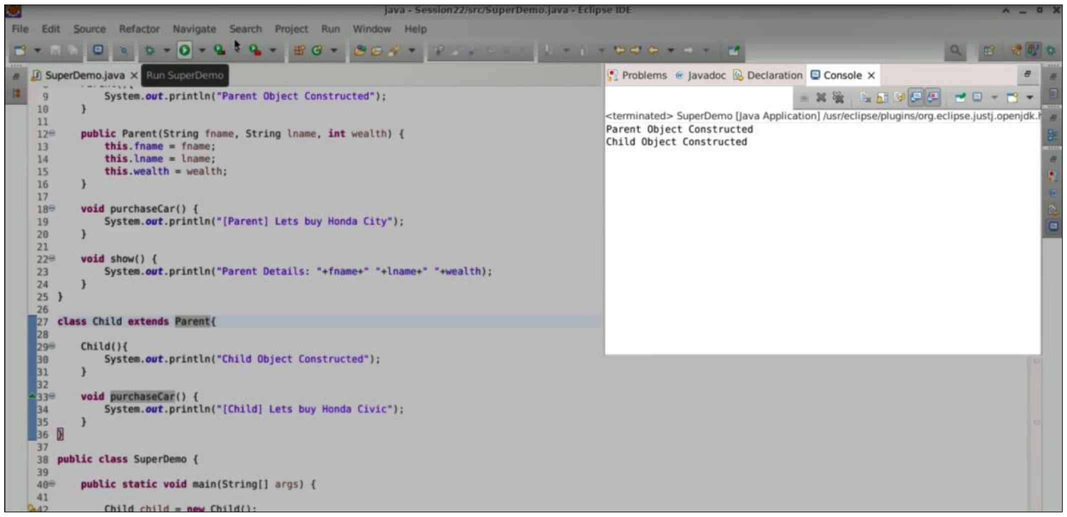

java - Session22/src/SuperDemo.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
SuperDemo.java
Problems Javadoc Declaration Console

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

    public Parent(String fname, String lname, int wealth) {
        this.fname = fname;
        this.lname = lname;
        this.wealth = wealth;
    }

    void purchaseCar() {
        System.out.println("[Parent] Lets buy Honda City");
    }

    void show() {
        System.out.println("Parent Details: "+fname+" "+lname+" "+wealth);
    }
}

class Child{

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

    void purchaseCar() {
        System.out.println("[Child] Lets buy Honda Civic");
    }
}

public class SuperDemo {

    public static void main(String[] args) {

        Child child = new Child();

    }

```

```
<terminated> SuperDemo [Java Application] /usr/eclipse/plugins/org.eclipse.justj.openjdk.
Child Object Constructed
```

Step 9: Create an inheritance relationship

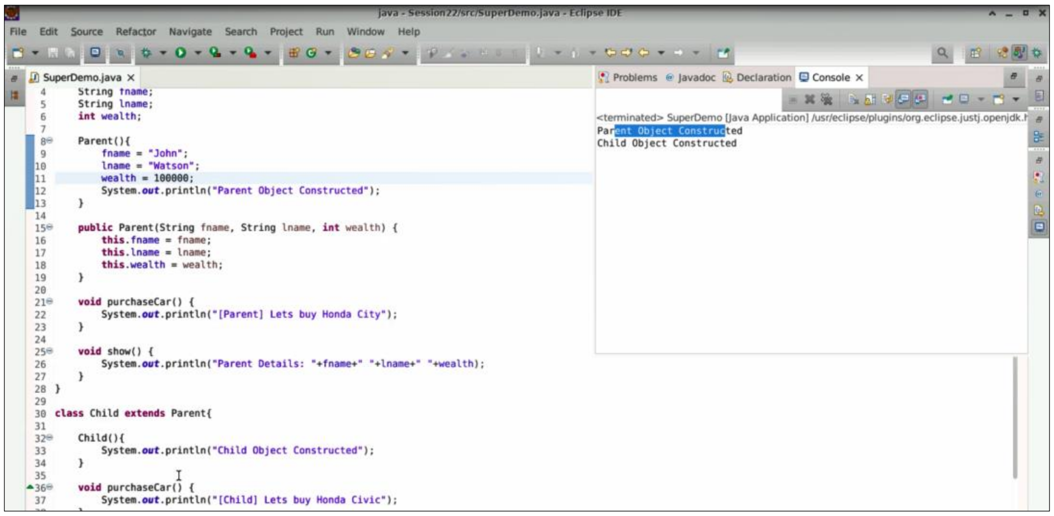
9.1 Since there is no inheritance relationship map. And you will add extends the parent. Now you have an inheritance relationship mark, which typically means when you run this program before the object of child and object of parent will be constructed


java - Session22/src/SuperDemo.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
SuperDemo.java
Run SuperDemo
Problems Javadoc Declaration Console

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

    public Parent(String fname, String lname, int wealth) {
        this.fname = fname;
        this.lname = lname;
        this.wealth = wealth;
    }

    void purchaseCar() {
        System.out.println("[Parent] Lets buy Honda City");
    }

    void show() {
        System.out.println("Parent Details: "+fname+" "+lname+" "+wealth);
    }
}

class Child extends Parent{

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

    void purchaseCar() {
        System.out.println("[Child] Lets buy Honda Civic");
    }
}

public class SuperDemo {

    public static void main(String[] args) {

        Child child = new Child();
```

```
<terminated> SuperDemo [Java Application] /usr/eclipse/plugins/org.eclipse.justj.openjdk.
Parent Object Constructed
Child Object Constructed
```

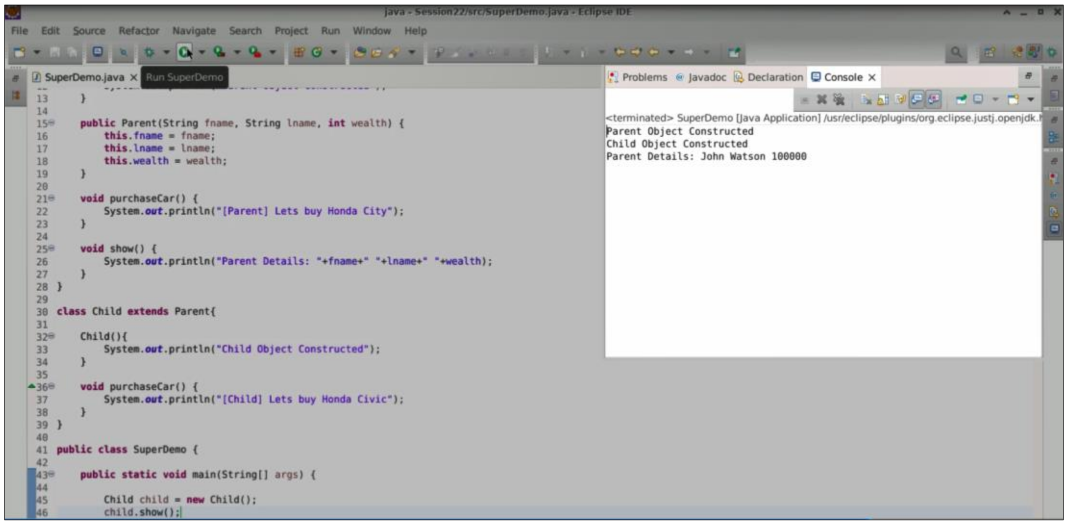
9.2 In your default constructor you say **fname** is John, and **lname** is Watson, and you have a wealth of 100,000. Whenever you create the object of a child. Your object of parent will be created but it will be created with the default constructor


java - Session22/src/SuperDemo.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
SuperDemo.java
Problems Javadoc Declaration Console

```
    String fname;
    String lname;
    int wealth;

    Parent(){
        fname = "John";
        lname = "Watson";
        wealth = 100000;
        System.out.println("Parent Object Constructed");
    }

    public Parent(String fname, String lname, int wealth) {
        this.fname = fname;
        this.lname = lname;
        this.wealth = wealth;
    }

    void purchaseCar() {
        System.out.println("[Parent] Lets buy Honda City");
    }

    void show() {
        System.out.println("Parent Details: "+fname+" "+lname+" "+wealth);
    }
}

class Child extends Parent{

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

    void purchaseCar() {
        System.out.println("[Child] Lets buy Honda Civic");
```

```
<terminated> SuperDemo [Java Application] /usr/eclipse/plugins/org.eclipse.justj.openjdk.
Parent Object Constructed
Child Object Constructed
```

9.3 If you type **child.show**, it is going to show John Watson with 100,000

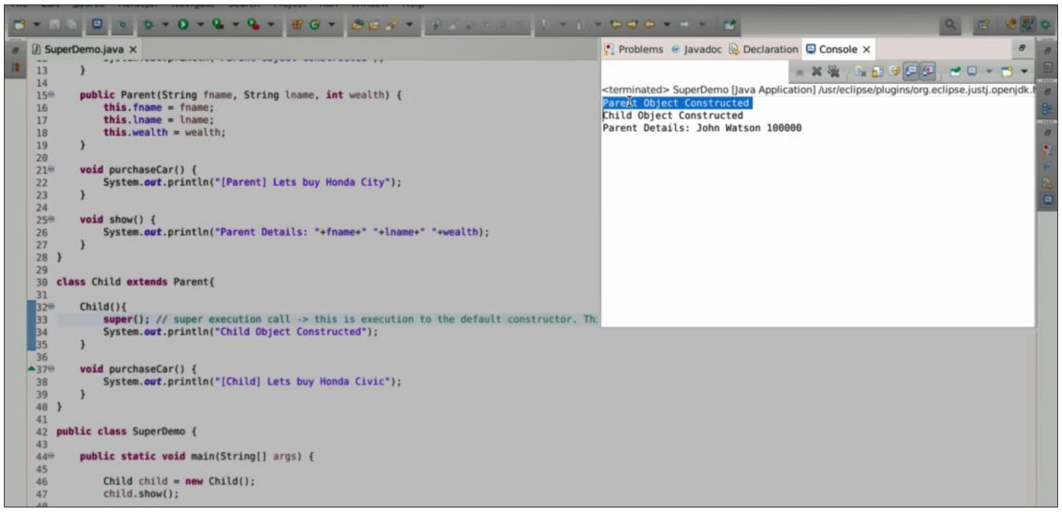

java - Session22/src/SuperDemo.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
SuperDemo.java
Run SuperDemo
Problems Javadoc Declaration Console

```
    }

    public Parent(String fname, String lname, int wealth) {
        this.fname = fname;
        this.lname = lname;
        this.wealth = wealth;
    }

    void purchaseCar() {
        System.out.println("[Parent] Lets buy Honda City");
    }

    void show() {
        System.out.println("Parent Details: "+fname+" "+lname+" "+wealth);
    }
}

class Child extends Parent{

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

    void purchaseCar() {
        System.out.println("[Child] Lets buy Honda Civic");
    }
}

public class SuperDemo {

    public static void main(String[] args) {

        Child child = new Child();
        child.show();
```

```
<terminated> SuperDemo [Java Application] /usr/eclipse/plugins/org.eclipse.justj.openjdk.
Parent Object Constructed
Child Object Constructed
Parent Details: John Watson 100000
```

Step 10: Implement a super execution call and override the default behavior

10.1 Now in the child constructor automatically there is a super execution call. This super call Is taken care of by the compiler automatically. You need not embed this



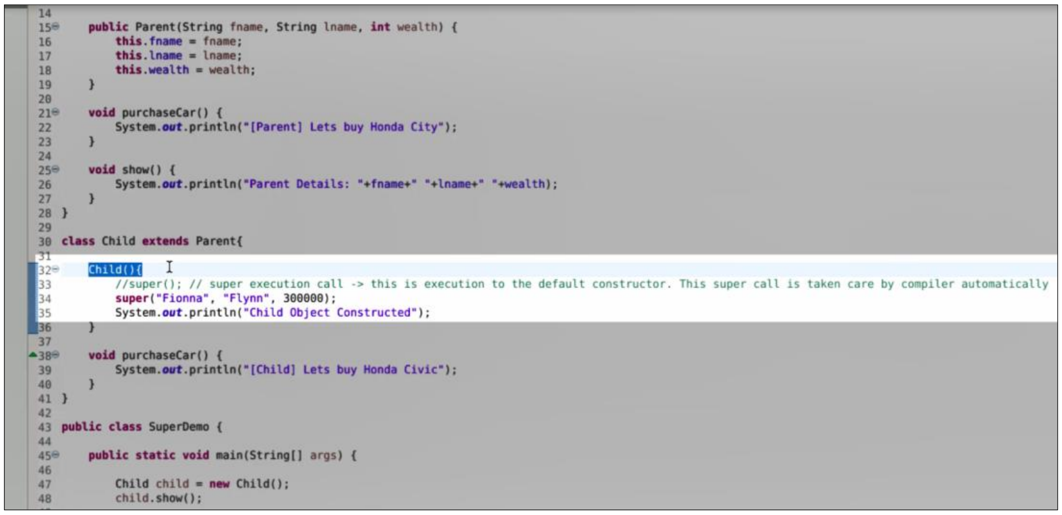
SuperDemo.java

```java
    }

    public Parent(String fname, String lname, int wealth) {
        this.fname = fname;
        this.lname = lname;
        this.wealth = wealth;
    }

    void purchaseCar() {
        System.out.println("[Parent] Lets buy Honda City");
    }

    void show() {
        System.out.println("Parent Details: "+fname+" "+lname+" "+wealth);
    }
}

class Child extends Parent{

    Child(){
        super(); // super execution call -> this is execution to the default constructor. Th
        System.out.println("Child Object Constructed");
    }

    void purchaseCar() {
        System.out.println("[Child] Lets buy Honda Civic");
    }
}

public class SuperDemo {

    public static void main(String[] args) {

        Child child = new Child();
        child.show();
```

Console

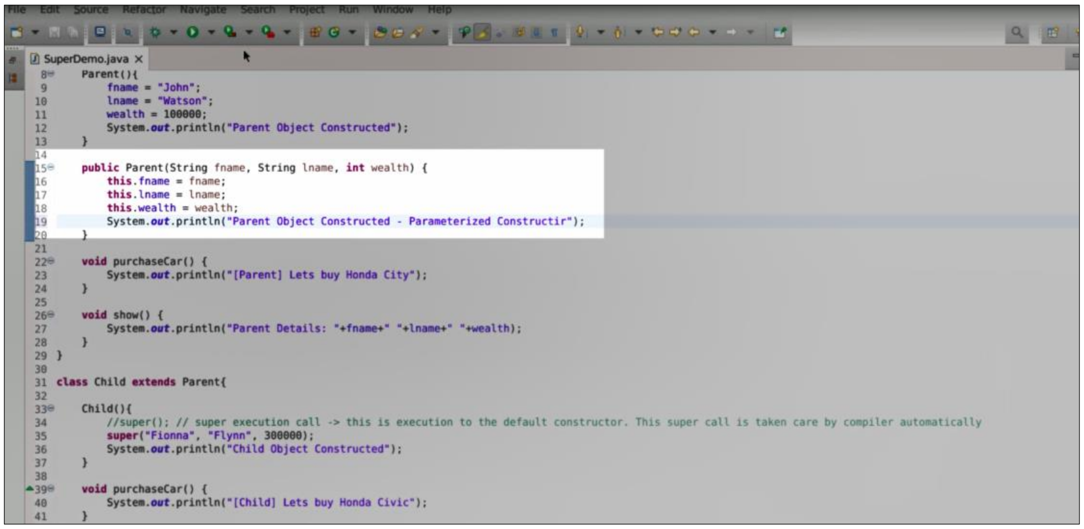
```
<terminated> SuperDemo [Java Application] /usr/eclipse/plugins/org.eclipse.justj.openjdk.
Parent Object Constructed
Child Object Constructed
Parent Details: John Watson 100000
```

10.2 You can override this default behavior. You can execute your super and you can pass the details here. For example, type Fiona. And you got 300,000. Your child object is getting constructed with the default constructor but, before the object of child, object of parents should be created. It is a kind of giving an instruction that please use the parameterized constructor from the parent rather than the default constructor



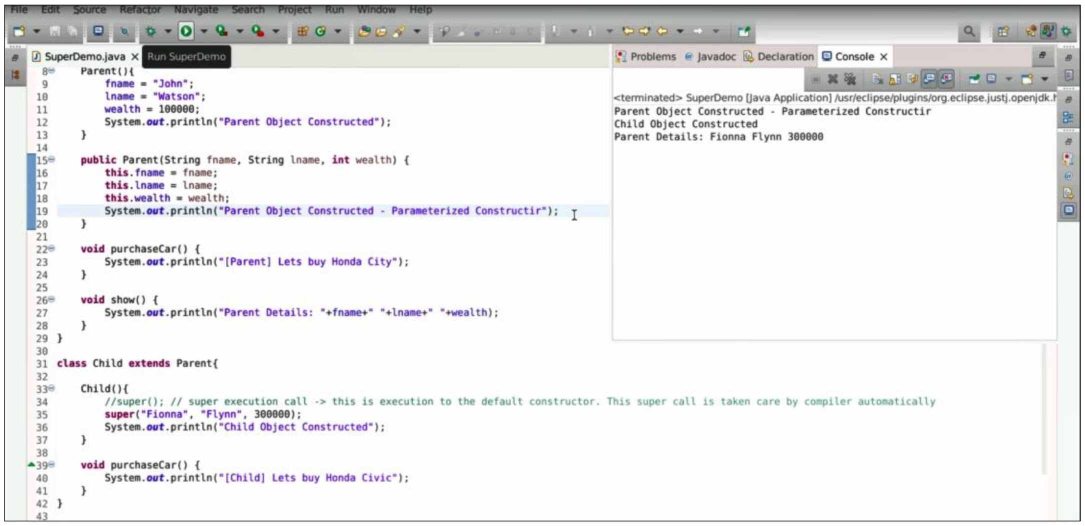
```java

    public Parent(String fname, String lname, int wealth) {
        this.fname = fname;
        this.lname = lname;
        this.wealth = wealth;
    }

    void purchaseCar() {
        System.out.println("[Parent] Lets buy Honda City");
    }

    void show() {
        System.out.println("Parent Details: "+fname+" "+lname+" "+wealth);
    }
}

class Child extends Parent{

    Child(){
        //super(); // super execution call -> this is execution to the default constructor. This super call is taken care by compiler automatically
        super("Fionna", "Flynn", 300000);
        System.out.println("Child Object Constructed");
    }

    void purchaseCar() {
        System.out.println("[Child] Lets buy Honda Civic");
    }
}

public class SuperDemo {

    public static void main(String[] args) {

        Child child = new Child();
        child.show();
```

10.3 Type the print statement that says parent object constructed. This is parameterized constructor



```
    Parent(){
        fname = "John";
        lname = "Watson";
        wealth = 100000;
        System.out.println("Parent Object Constructed");
    }

    public Parent(String fname, String lname, int wealth) {
        this.fname = fname;
        this.lname = lname;
        this.wealth = wealth;
        System.out.println("Parent Object Constructed - Parameterized Constructir");
    }

    void purchaseCar() {
        System.out.println("[Parent] Lets buy Honda City");
    }

    void show() {
        System.out.println("Parent Details: "+fname+" "+lname+" "+wealth);
    }
}

class Child extends Parent{

    Child(){
        //super(); // super execution call -> this is execution to the default constructor. This super call is taken care by compiler automatically
        super("Fionna", "Flynn", 300000);
        System.out.println("Child Object Constructed");
    }

    void purchaseCar() {
        System.out.println("[Child] Lets buy Honda Civic");
    }
```

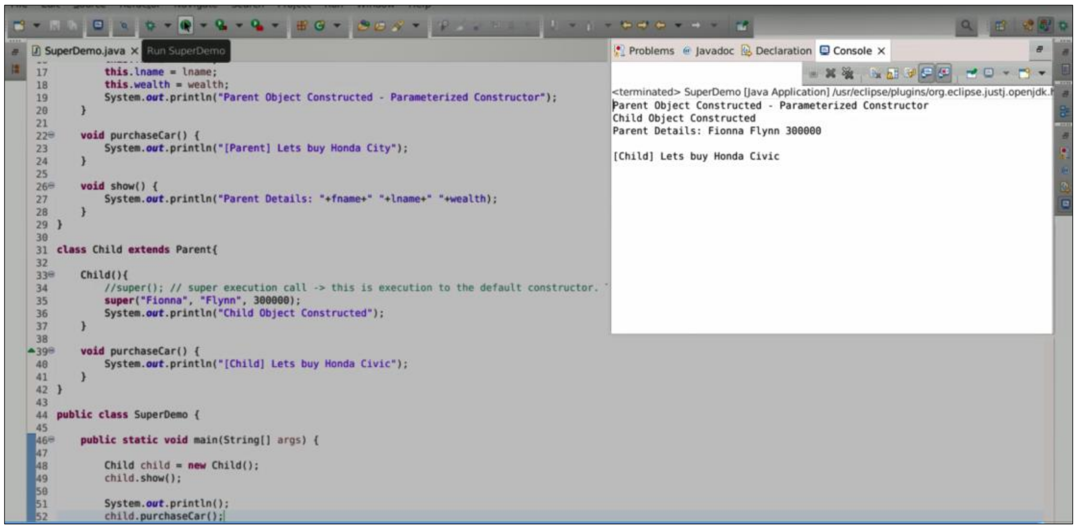
10.4 Now when you run the program, you can see the parent object is constructed with the parameterized constructor rather than the default constructor



```
    Parent(){
        fname = "John";
        lname = "Watson";
        wealth = 100000;
        System.out.println("Parent Object Constructed");
    }

    public Parent(String fname, String lname, int wealth) {
        this.fname = fname;
        this.lname = lname;
        this.wealth = wealth;
        System.out.println("Parent Object Constructed - Parameterized Constructir");
    }

    void purchaseCar() {
        System.out.println("[Parent] Lets buy Honda City");
    }

    void show() {
        System.out.println("Parent Details: "+fname+" "+lname+" "+wealth);
    }
}

class Child extends Parent{

    Child(){
        //super(); // super execution call -> this is execution to the default constructor. This super call is taken care by compiler automatically
        super("Fionna", "Flynn", 300000);
        System.out.println("Child Object Constructed");
    }

    void purchaseCar() {
        System.out.println("[Child] Lets buy Honda Civic");
    }
}
```

```
<terminated> SuperDemo [Java Application] /usr/eclipse/plugins/org.eclipse.justj.openjdk.h
Parent Object Constructed - Parameterized Constructir
Child Object Constructed
Parent Details: Fionna Flynn 300000
```

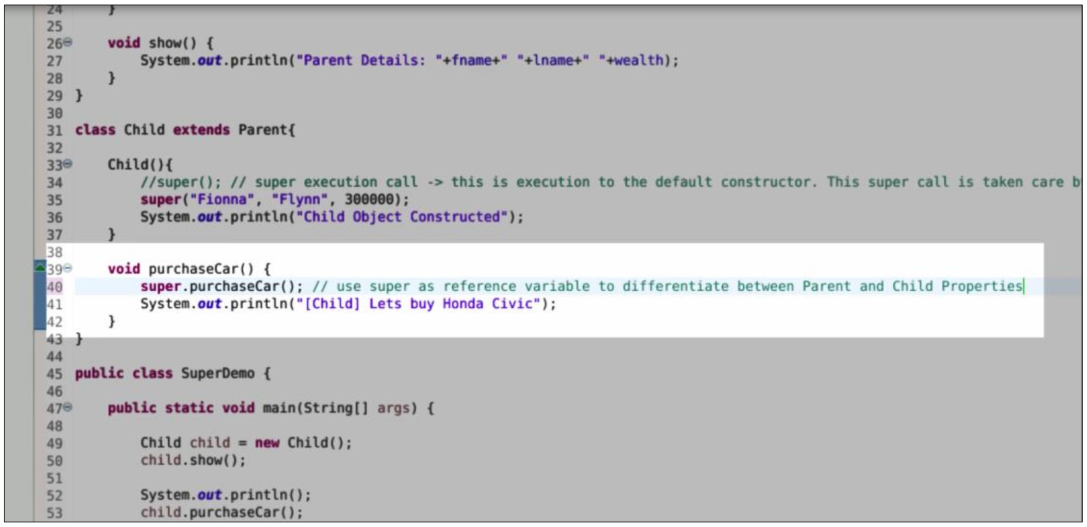
10.5 Now come here and type with the child, Execute the purchase car. This is overriding, as you have seen earlier as well. Let us buy a Honda Civic, which means that the parent's definition will not be executed. This is the definition of the child which will be executed



```java
            this.lname = lname;
            this.wealth = wealth;
            System.out.println("Parent Object Constructed - Parameterized Constructor");
        }

        void purchaseCar() {
            System.out.println("[Parent] Lets buy Honda City");
        }

        void show() {
            System.out.println("Parent Details: "+fname+" "+lname+" "+wealth);
        }
}

class Child extends Parent{

        Child(){
            //super(); // super execution call -> this is execution to the default constructor. 
            super("Fionna", "Flynn", 300000);
            System.out.println("Child Object Constructed");
        }

        void purchaseCar() {
            System.out.println("[Child] Lets buy Honda Civic");
        }
}

public class SuperDemo {

        public static void main(String[] args) {

            Child child = new Child();
            child.show();

            System.out.println();
            child.purchaseCar();
```

```
<terminated> SuperDemo [Java Application] /usr/eclipse/plugins/org.eclipse.justj.openjdk.
Parent Object Constructed - Parameterized Constructor
Child Object Constructed
Parent Details: Fionna Flynn 300000

[Child] Lets buy Honda Civic
```

10.6 If you wish to execute the parent's method from the same method of **purchaseCar**, you can add super to the **purchaseCar** method. Here, use super as a reference variable to differentiate between parent and child properties. This execution call will execute the constructor in the same class, whereas the super execution call will execute the constructor from the parent class



```java
        }

        void show() {
            System.out.println("Parent Details: "+fname+" "+lname+" "+wealth);
        }
}

class Child extends Parent{

        Child(){
            //super(); // super execution call -> this is execution to the default constructor. This super call is taken care b
            super("Fionna", "Flynn", 300000);
            System.out.println("Child Object Constructed");
        }

        void purchaseCar() {
            super.purchaseCar(); // use super as reference variable to differentiate between Parent and Child Properties
            System.out.println("[Child] Lets buy Honda Civic");
        }
}

public class SuperDemo {

        public static void main(String[] args) {

            Child child = new Child();
            child.show();

            System.out.println();
            child.purchaseCar();
```

By following these steps, you have successfully differentiated the use of **this** and **super** in Java