Lesson 03 Demo 09

Implementing Final Variables and Methods in Java

Objective: Using the keyword Final in Java

Tools required: Eclipse IDE

Prerequisites: None

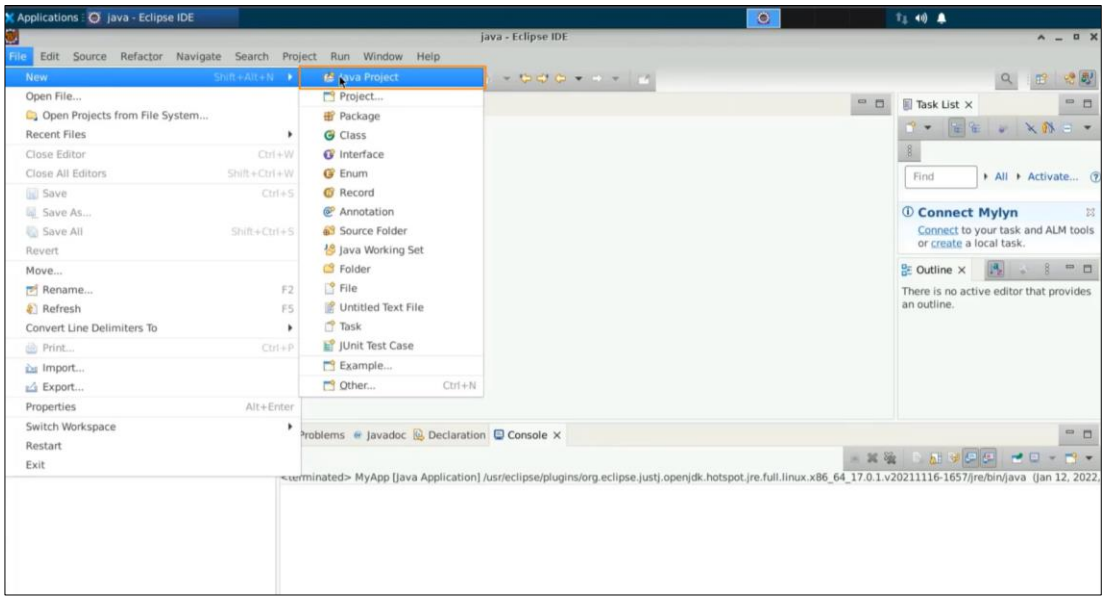
Steps to be followed:

1. Create a class called FinalKeyword, followed by selecting the main method
2. Define a normal variable and final variable with examples
3. Create a class with a method pay and inherit this class
4. Override and customize the pay method
5. Mark the method as final to limit redefining methods

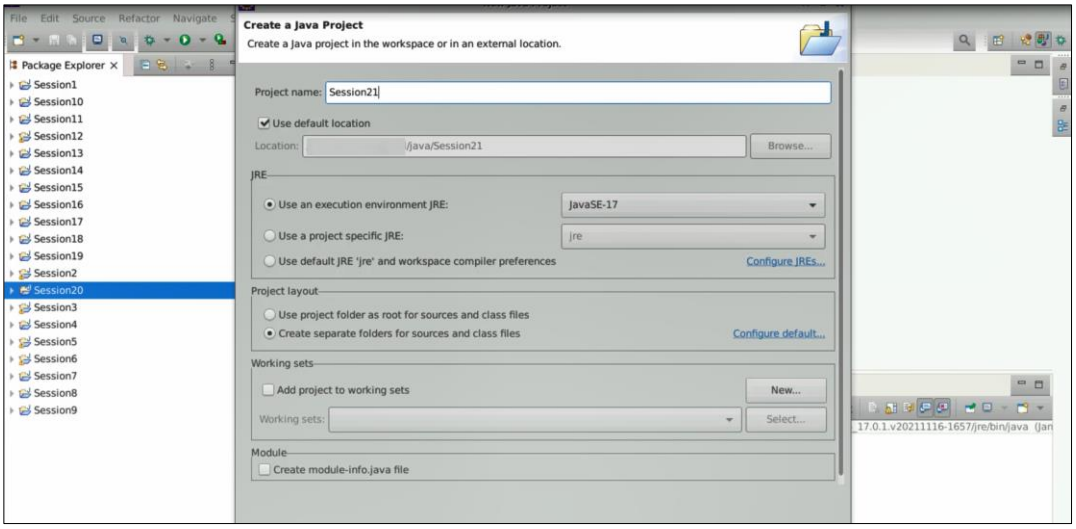
Step 1: Create a class called FinalKeyword, followed by selecting the main method

1.1 Open the **Eclipse IDE**

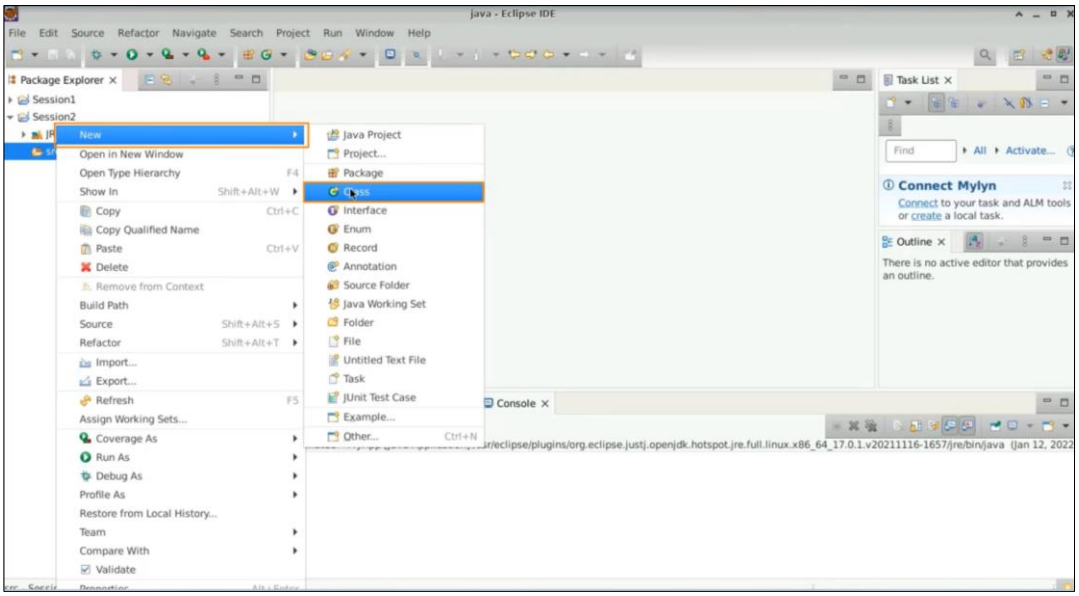
1.2. Select **File**, then **New,** and then **Java project**



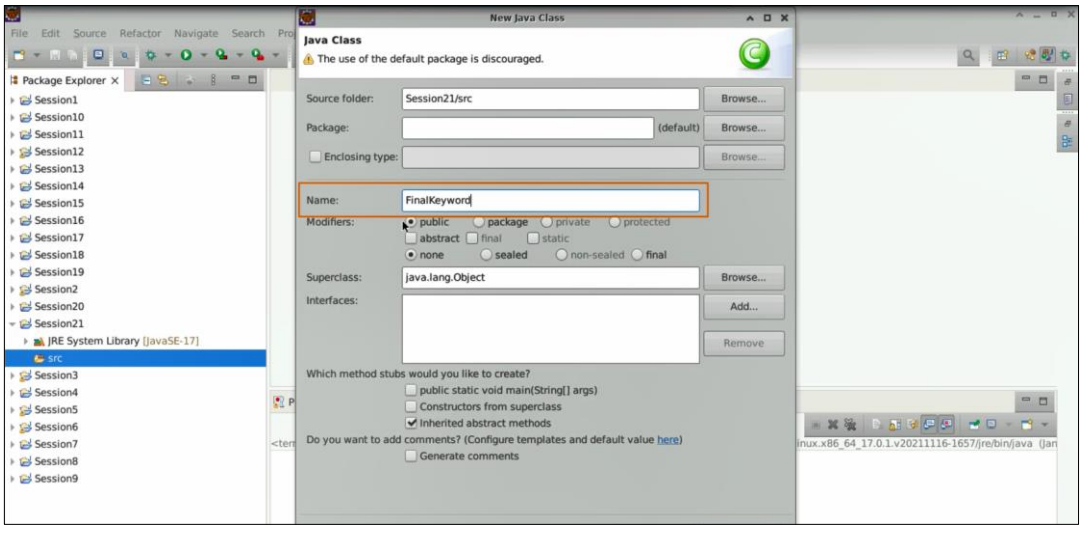
1.3 Name the project **“Session21”,** uncheck **“Create a module info dot Java file”**, and press **Finish**



1.4 With a **Session21** on the src, do a right-click and create a **new class**

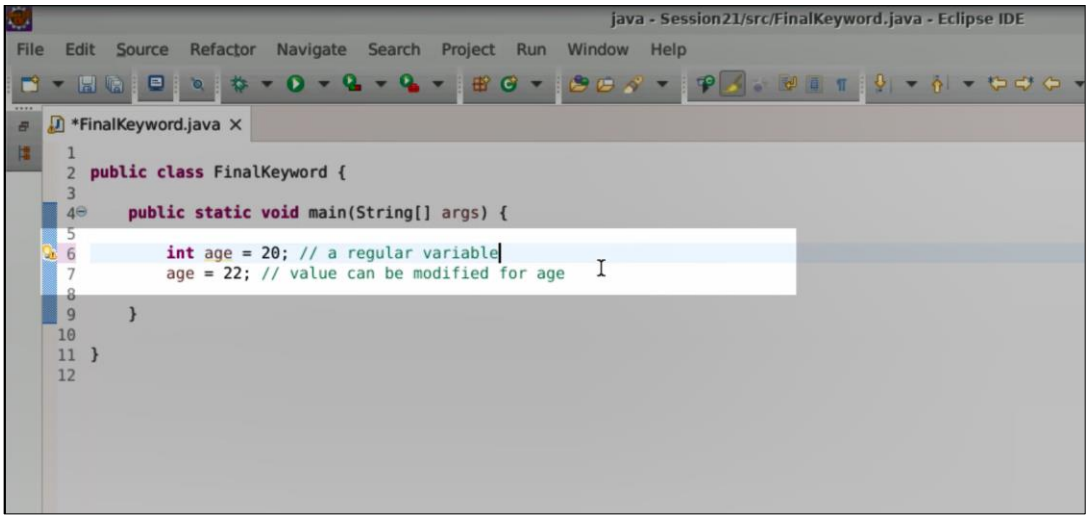


1.5 Name this class as a **FinalKeyword**, then select the **main method,** and then select **finish**

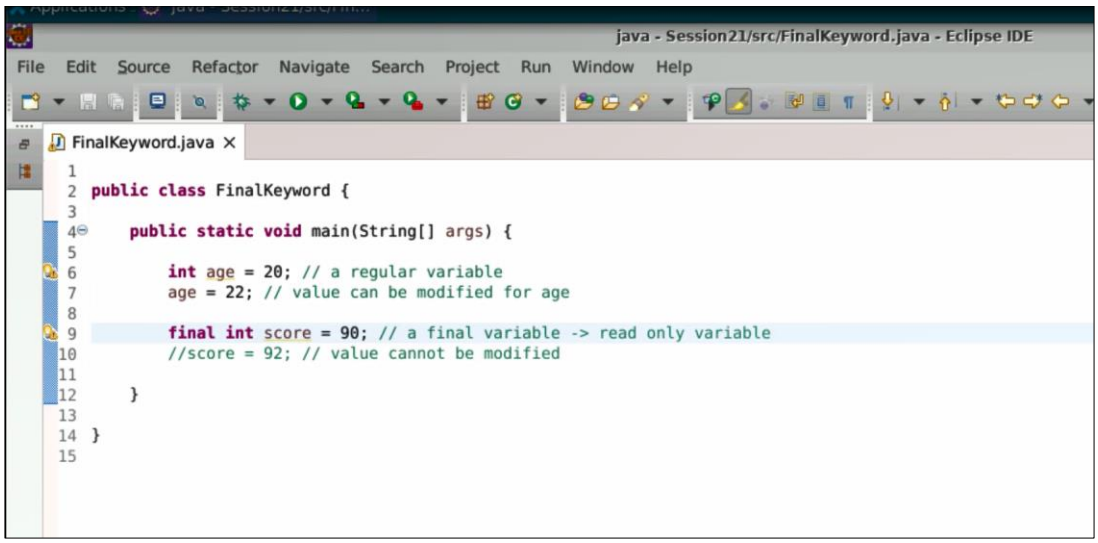


Step 2: Define a normal variable and final variable with examples

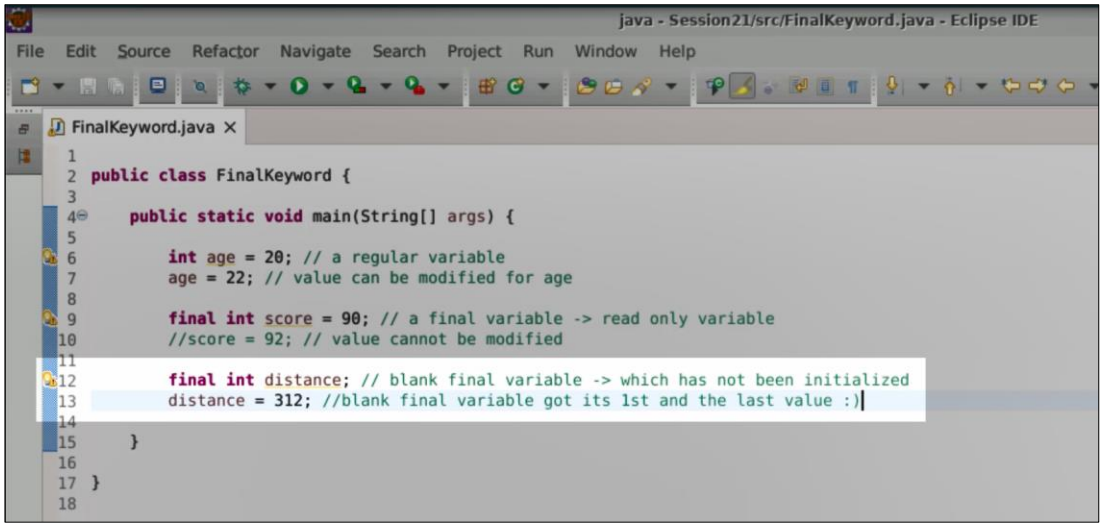
2.1 What is a normal variable? If you create a variable named **age**, you can manipulate its contents; the value can be modified. This describes a normal or regular variable



2.2 Let us see if you can create a variable that can be marked as **final**. The moment you add a variable called score with a value like 90, you will not be able to update it. If you mark your variable as **final**, this becomes a final variable. Its value cannot be modified. Here, you are trying to create a constant. The constant terminology is replaced with the **final** keyword. This is now a read-only variable; you can read it, but you cannot update it. **Final** is a keyword

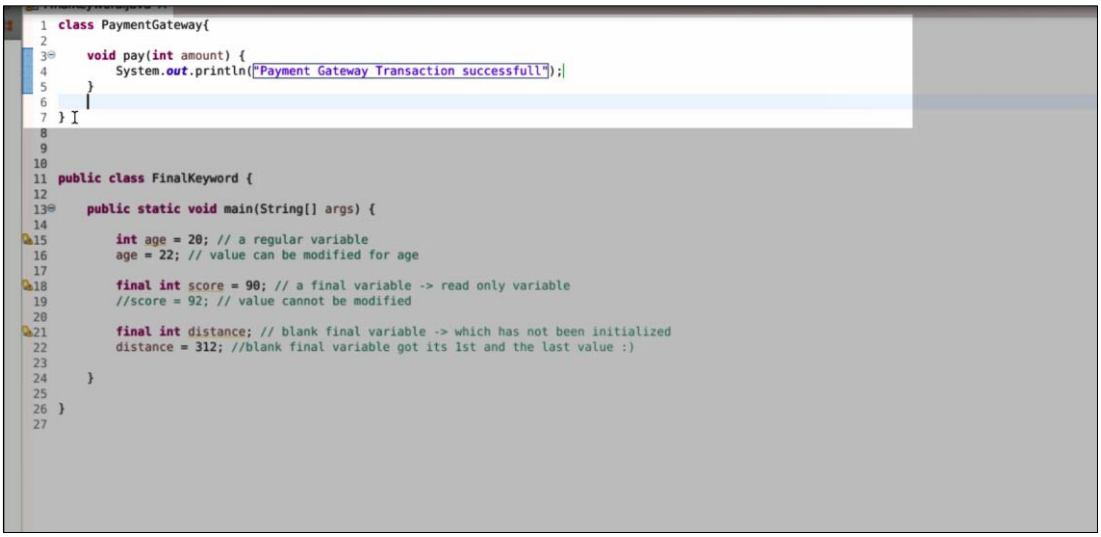


2.3 You can use a variable that has no initial data. For example, you can declare **final int distance;** This is referred to as a blank final variable. When you see a blank final variable, it means the variable has not been initialized. You can later assign a value to the variable, such as **distance = 312;** This is how a blank final variable gets its first and only value

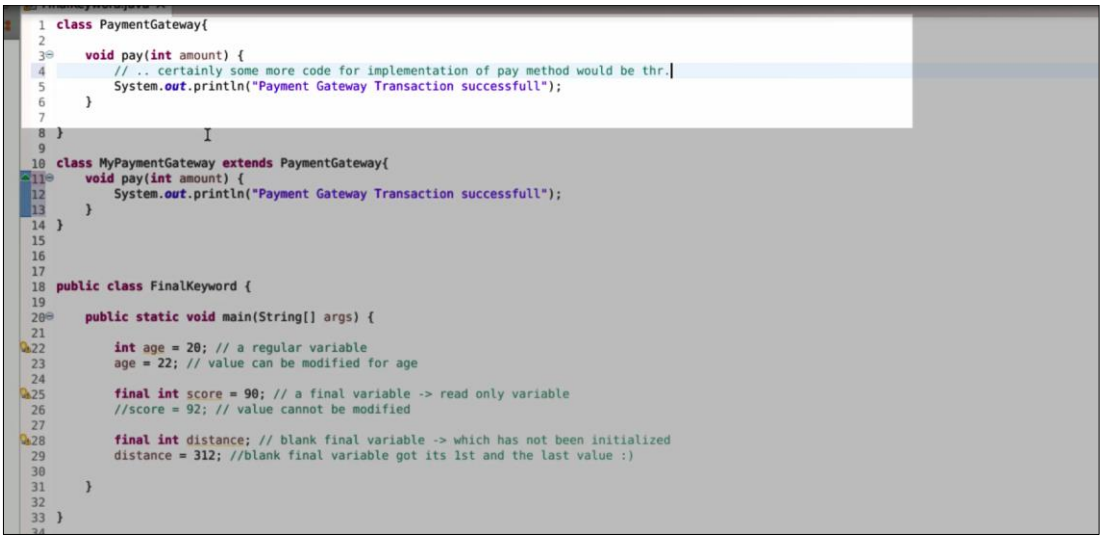


Step 3: Create a class with a method pay and inherit this class

3.1 Now, let us create a class called **PaymentGateway**. This class will have a method called **pay**, which takes one amount as input so that you can perform a transaction. The payment gateway will then output **" Payment Gateway Transaction successful."**



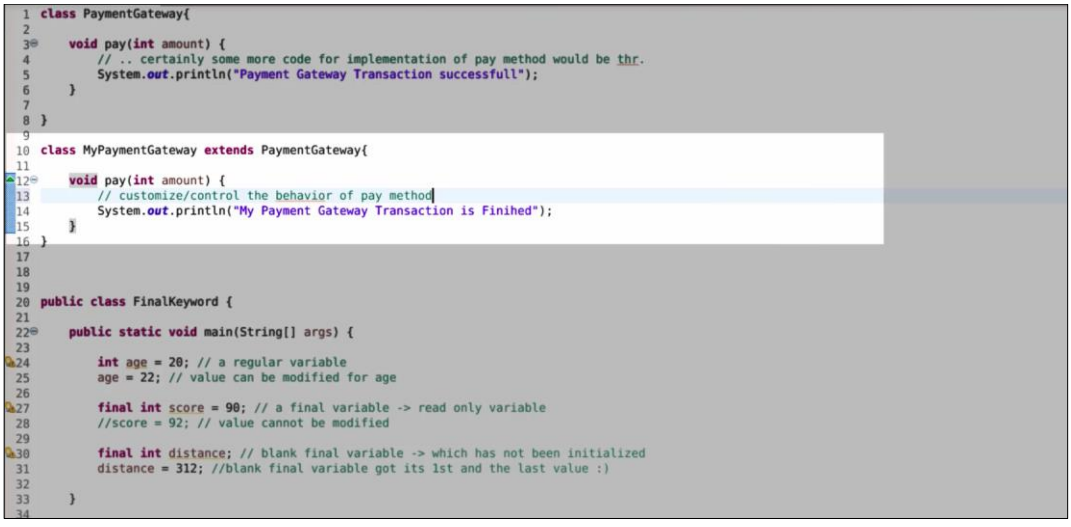
3.2 Inherit this class by creating **MyGateway** which extends **PaymentGateway**. Now, when you inherit from **PaymentGateway**, you can override the pay method. Certainly, some more code for the implementation of the pay method would be there; this is just an assumption



```
class PaymentGateway{

    void pay(int amount) {
        // .. certainly some more code for implementation of pay method would be thr.
        System.out.println("Payment Gateway Transaction successfull");
    }

}

class MyPaymentGateway extends PaymentGateway{
    void pay(int amount) {
        System.out.println("Payment Gateway Transaction successfull");
    }
}


public class FinalKeyword {

    public static void main(String[] args) {

        int age = 20; // a regular variable
        age = 22; // value can be modified for age

        final int score = 90; // a final variable -> read only variable
        //score = 92; // value cannot be modified

        final int distance; // blank final variable -> which has not been initialized
        distance = 312; //blank final variable got its 1st and the last value :)

    }
}
```

Step 4: Override and customize the pay method

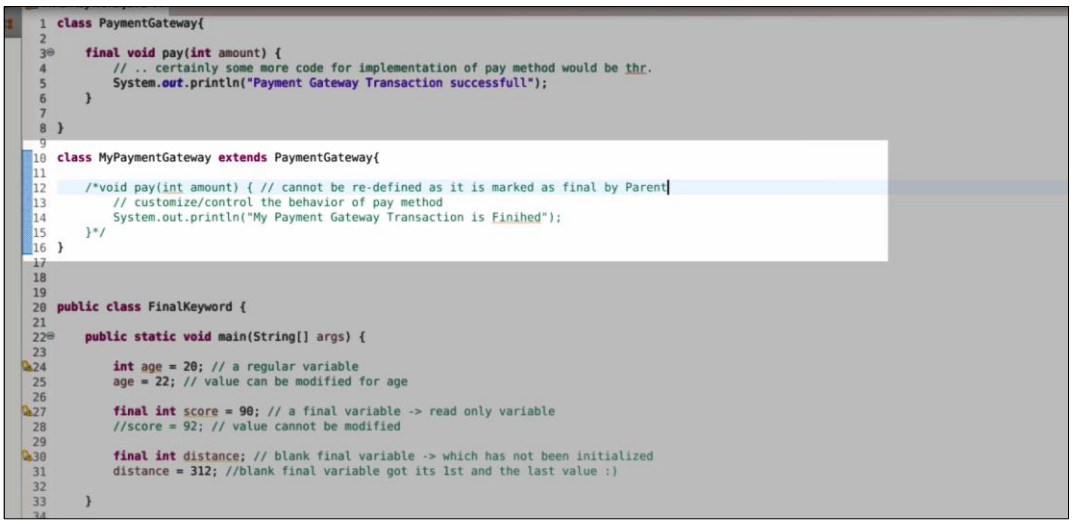
4.1 Now you can add my payment gateway Transaction is finished. You have overridden the pay method, and you are trying to customize it or maybe control it, customize slash control the behavior of the Pay method. This can be sometimes a, you can say security concern



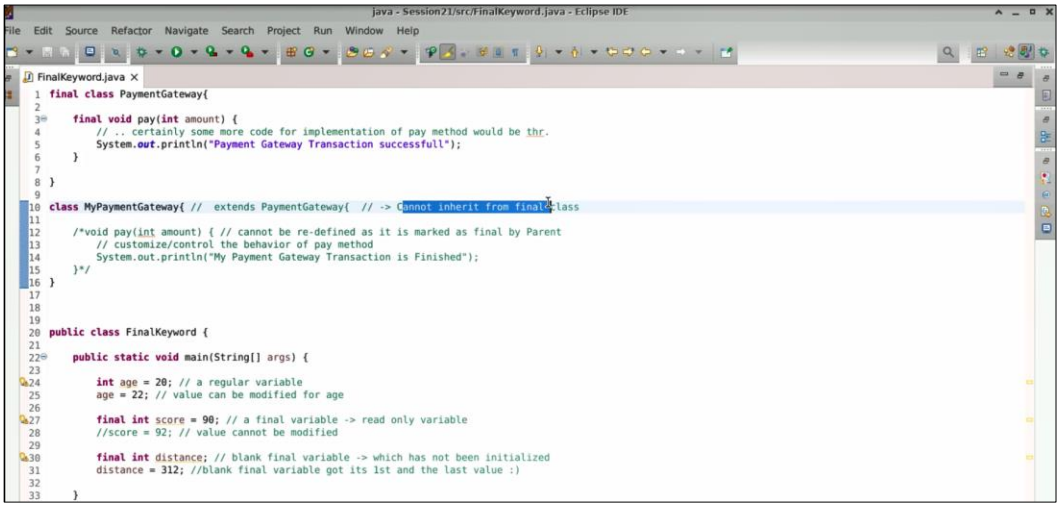
```
class PaymentGateway{

    void pay(int amount) {
        // .. certainly some more code for implementation of pay method would be thr.
        System.out.println("Payment Gateway Transaction successfull");
    }

}

class MyPaymentGateway extends PaymentGateway{

    void pay(int amount) {
        // customize/control the behavior of pay method
        System.out.println("My Payment Gateway Transaction is Finihed");
    }
}


public class FinalKeyword {

    public static void main(String[] args) {

        int age = 20; // a regular variable
        age = 22; // value can be modified for age

        final int score = 90; // a final variable -> read only variable
        //score = 92; // value cannot be modified

        final int distance; // blank final variable -> which has not been initialized
        distance = 312; //blank final variable got its 1st and the last value :)

    }
```

Step 5: Mark the method as final to limit redefining methods

5.1 What you can do is, when you write general classes where you want certain methods not to be overridden, you can mark them as **final**. The moment you mark your pay method as final, no other class inheriting the **PaymentGateway** class will be able to override it. A final method is a method that cannot be redefined by a child class, as it is marked as **final** by the parent class



5.2 If you are giving your code to another organization and you do not want them to inherit your classes and override your methods according to their requirements, you can mark your methods as final to prevent redefinition. You can also mark the class as final. By doing this, you stop anyone from inheriting your class, effectively preventing inheritance



By following the above steps, you have successfully Implemented Final Variables and Methods in Java.