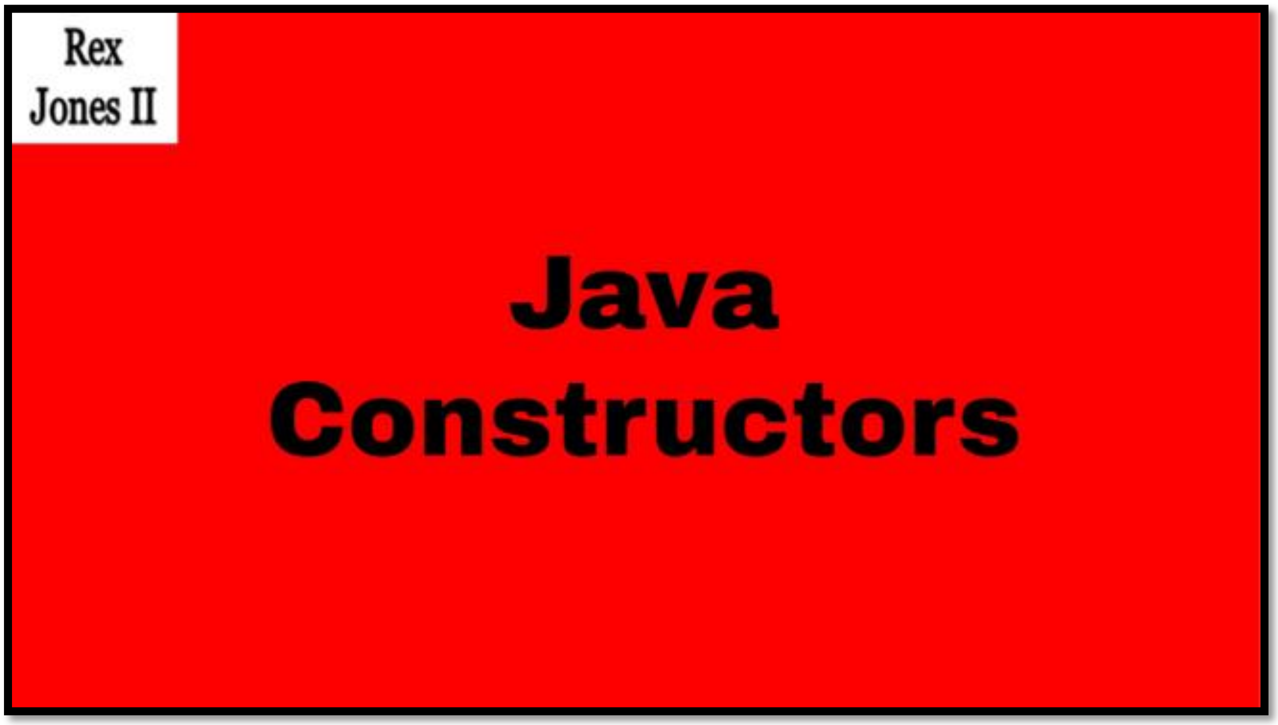
(Transcript)
Java Constructors



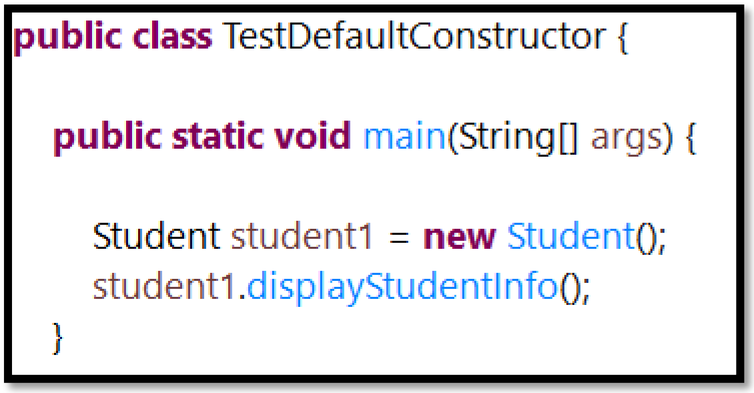
Introduction

In this session, let's talk about constructors. A constructor is a special method that initialize values when creating an object. All classes have constructors because Java automatically provide a default constructor. There are 2 types of constructors: the default constructor and parameterized constructor.

Default Constructor

Starting with the default constructor also known as no-arg constructor. We have 3 comments: Constructors Initialize The Object When Creating The Object, Constructors Have The Same Name As The Class Name, and Constructors Have No Return Type. We have a field called studentID and studentName. In addition, there's a method called displayStudentInfo. The method will print the Student id and print the Student name.

We are going to test the default constructor by creating an object. An object is created using the keyword new for new Student(). To access the new object, we must assign the object to a variable of type Student with a variable name of student1. Now we can use our object. Let's call the method student1.displayStudentInfo().

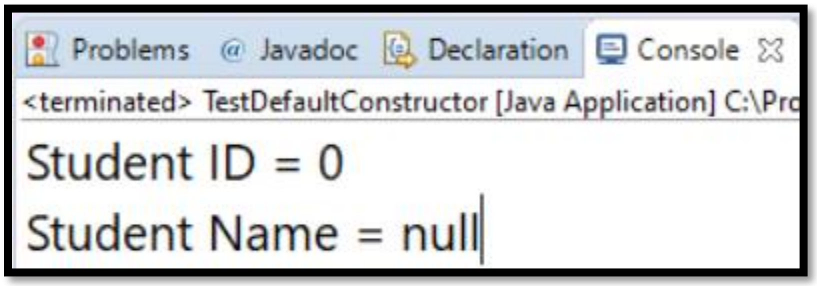


```
public class TestDefaultConstructor {

   public static void main(String[] args) {

      Student student1 = new Student();
      student1.displayStudentInfo();
   }
```

Before assigning values, the default constructor initializes default values: numeric types default to zero, reference types such as String default to null, and boolean default to false. Let's run. As expected, we see Student ID is 0 and Student Name is null. The values are 0 and null because they do not have a value yet.

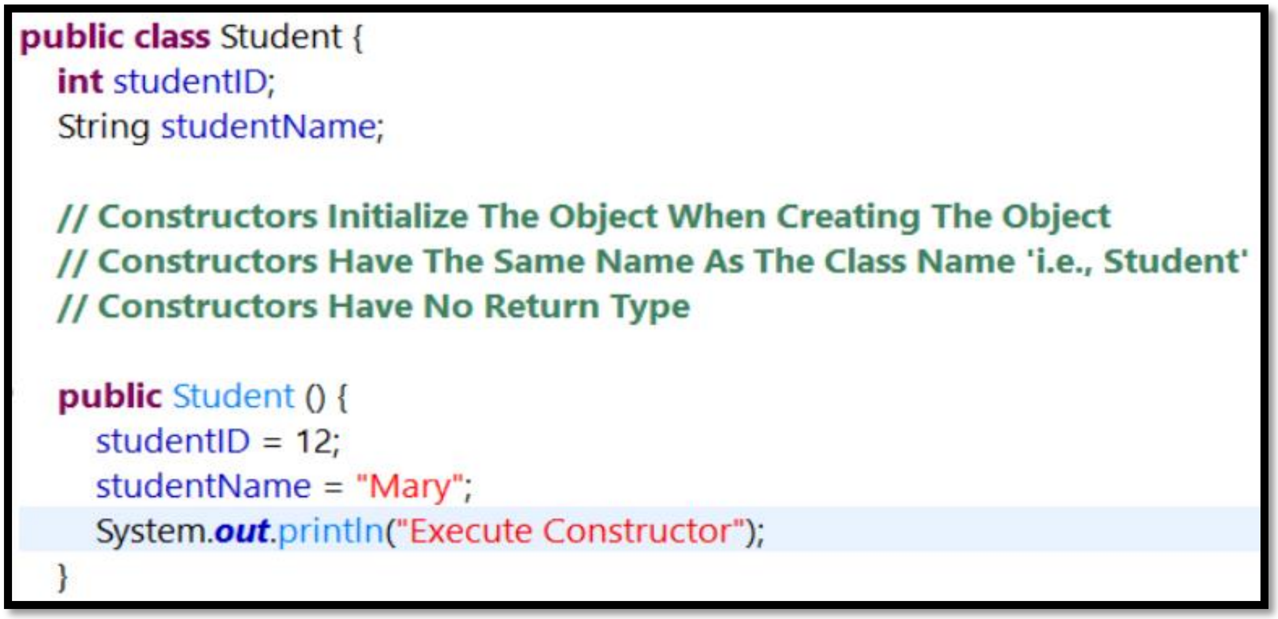


```
<terminated> TestDefaultConstructor [Java Application] C:\Pro
Student ID = 0
Student Name = null
```

Now, let's initialize our own values.

How do we define a constructor? We define a constructor just like a method but with the same name as the class name. We write public Student (). Notice, there is no return type such as void. Declare studentID = to 12 and studentName = to "Mary", how about a print statement sysout("Execute Constructor").

Rex
Jones II

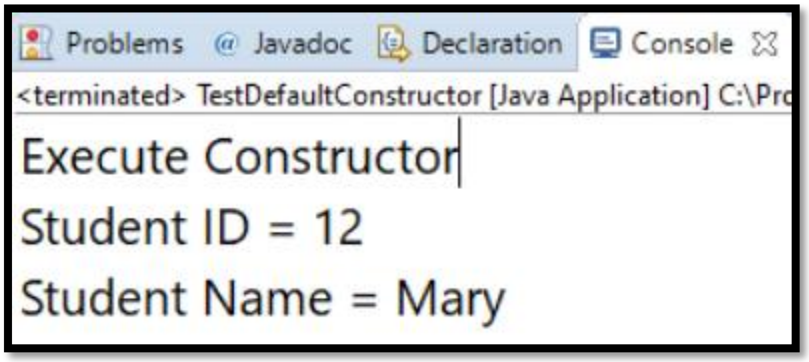


```
public class Student {
  int studentID;
  String studentName;

  // Constructors Initialize The Object When Creating The Object
  // Constructors Have The Same Name As The Class Name 'i.e., Student'
  // Constructors Have No Return Type

  public Student () {
    studentID = 12;
    studentName = "Mary";
    System.out.println("Execute Constructor");
  }
```

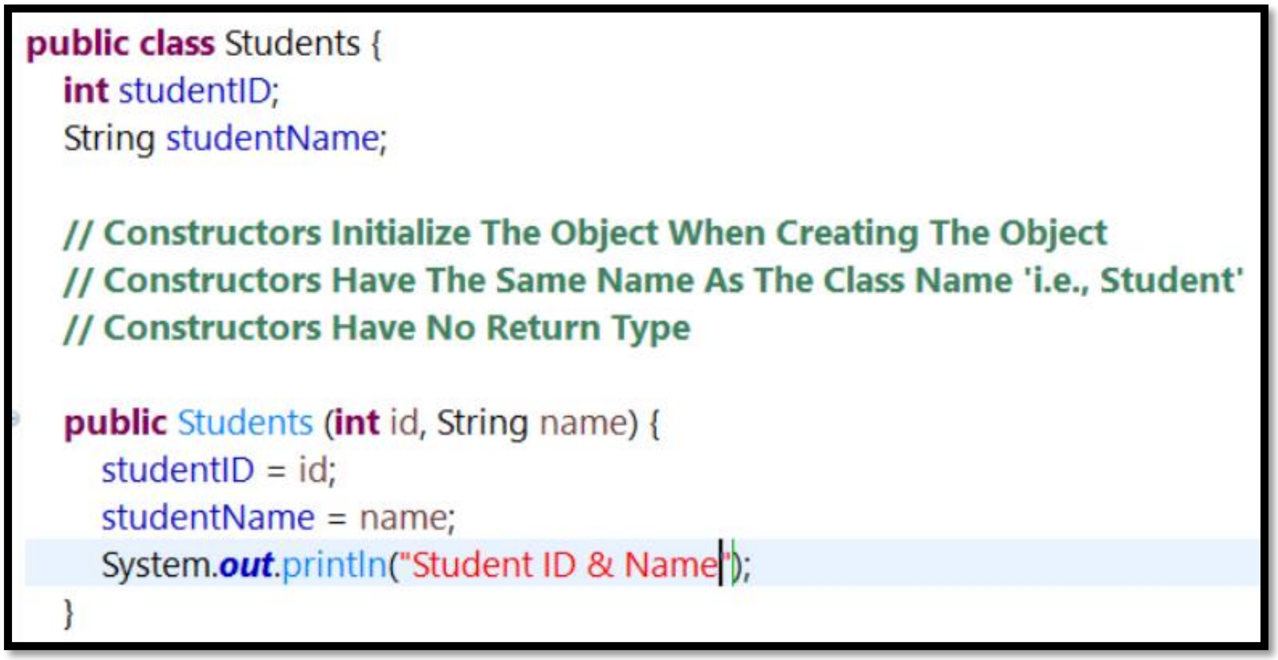
Now let's go back to the Test class. Line 7 creates the object, sets the values, and show us how the constructor will be executed when creating the object. Let's Run. We see Execute Constructor / Student ID = 12 / Student Name = Mary.



```
<terminated> TestDefaultConstructor [Java Application] C:\Pro
Execute Constructor
Student ID = 12
Student Name = Mary
```

Parameterized Constructor

Now, let's look at Parameterized Constructors. The Parameterized Constructor has the same concept as Default Constructor. Initialize the object when creating the object, Same name as class name, and Do not have a return type. However, this type of constructor accepts values from its parameters. We add parameters to a constructor in the same way as a method. Write
public Students (int id, String name) / id and name are the parameters. Assign id to studentID and name to studentName. Print statement sysout("Student ID & Name").

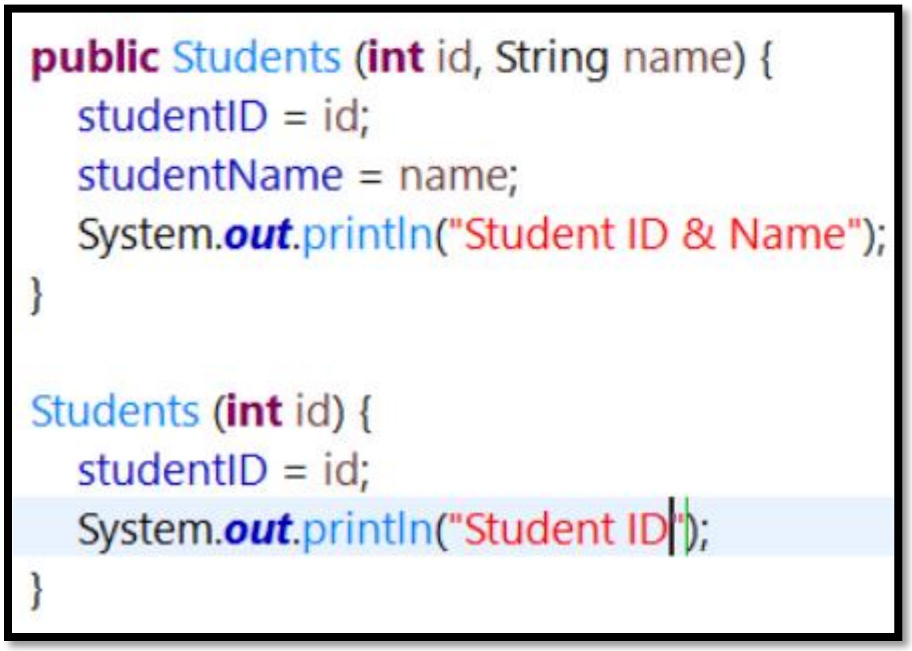


```
public class Students {
   int studentID;
   String studentName;

   // Constructors Initialize The Object When Creating The Object
   // Constructors Have The Same Name As The Class Name 'i.e., Student'
   // Constructors Have No Return Type

   public Students (int id, String name) {
      studentID = id;
      studentName = name;
      System.out.println("Student ID & Name");
   }
```

We can have as many constructors as we want. Just change the parameter list. Also, if we choose to, we can bypass writing the access modifier public and only write the class name Students (). Either way is okay. We can write public or without public. This time, our constructor only include (int id). Write studentID = id / sysout("Student ID").



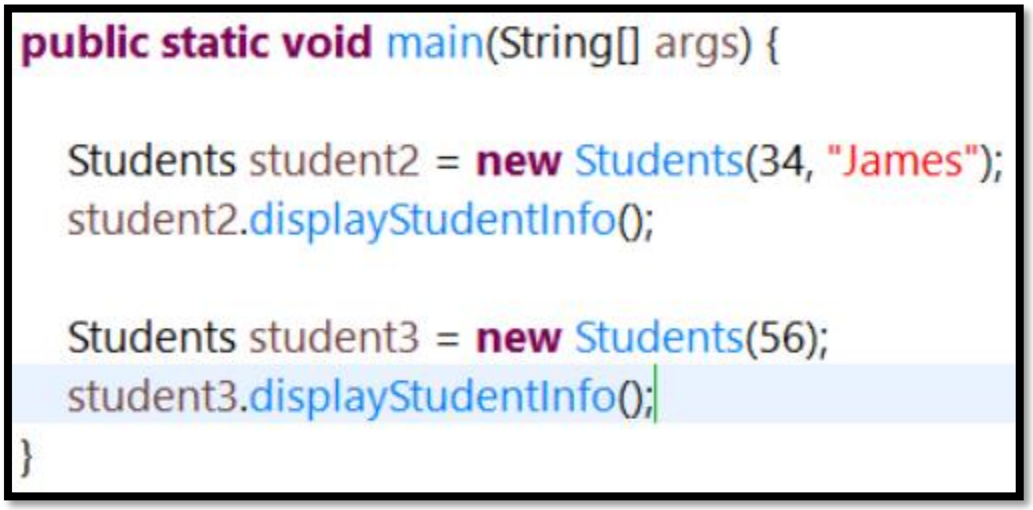
```
public Students (int id, String name) {
   studentID = id;
   studentName = name;
   System.out.println("Student ID & Name");
}

Students (int id) {
   studentID = id;
   System.out.println("Student ID");
}
```

Rex Jones II

Sending values to these constructor. Students student2 = new Students(). Do you see the error? We have an error because an empty Students constructor is not defined. The first and second quick fix want to add an argument. Select the 2nd quick fix and we see 0 and null. Replace 0 with a studentID of 34 and replace the null name with “James”. Call the method student2.displayStudentInfo().

Let’s create one more object Students student3 = new Students(56). 56 is the Student id. Let’s call the method student3.displayStudentInfo().

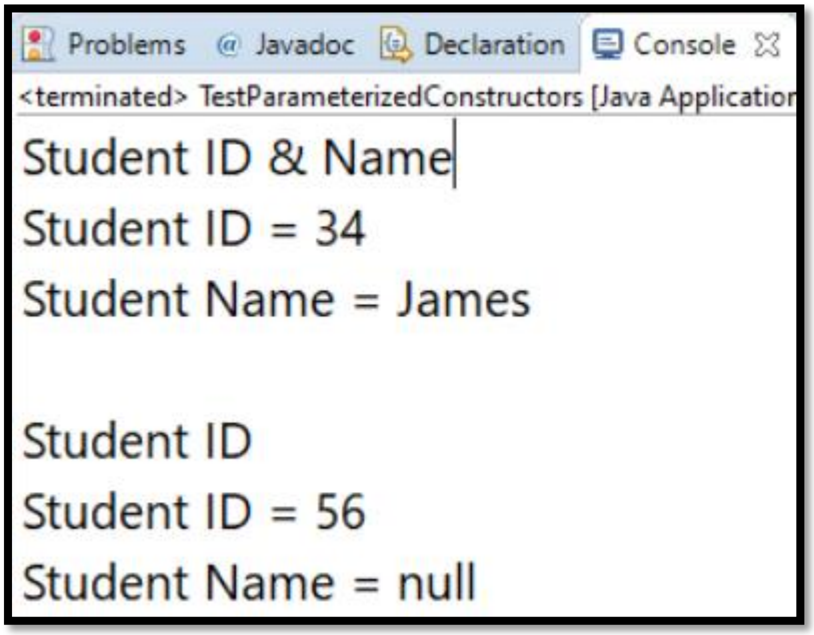


```
public static void main(String[] args) {

    Students student2 = new Students(34, "James");
    student2.displayStudentInfo();

    Students student3 = new Students(56);
    student3.displayStudentInfo();
}
```

Let’s Run.
Student ID & Name: Student ID = 34 and we see Student Name = James
Student ID: Student ID = 56 / Student Name = null because the constructor did not have a student name.



```
<terminated> TestParameterizedConstructors [Java Applicatio
Student ID & Name
Student ID = 34
Student Name = James

Student ID
Student ID = 56
Student Name = null
```

That’s it for Constructors. If you are new to my videos, consider subscribing to my channel and clicking the bell icon. You follow me on Twitter, connect with my on LinkedIn and Facebook. Also, the code and transcript will be added to GitHub.

Social Media Contact

- YouTube https://www.youtube.com/c/RexJonesII/videos
- Twitter https://twitter.com/RexJonesII
- LinkedIn https://www.linkedin.com/in/rexjones34/
- GitHub https://github.com/RexJonesII/Free-Videos
- Facebook http://facebook.com/JonesRexII