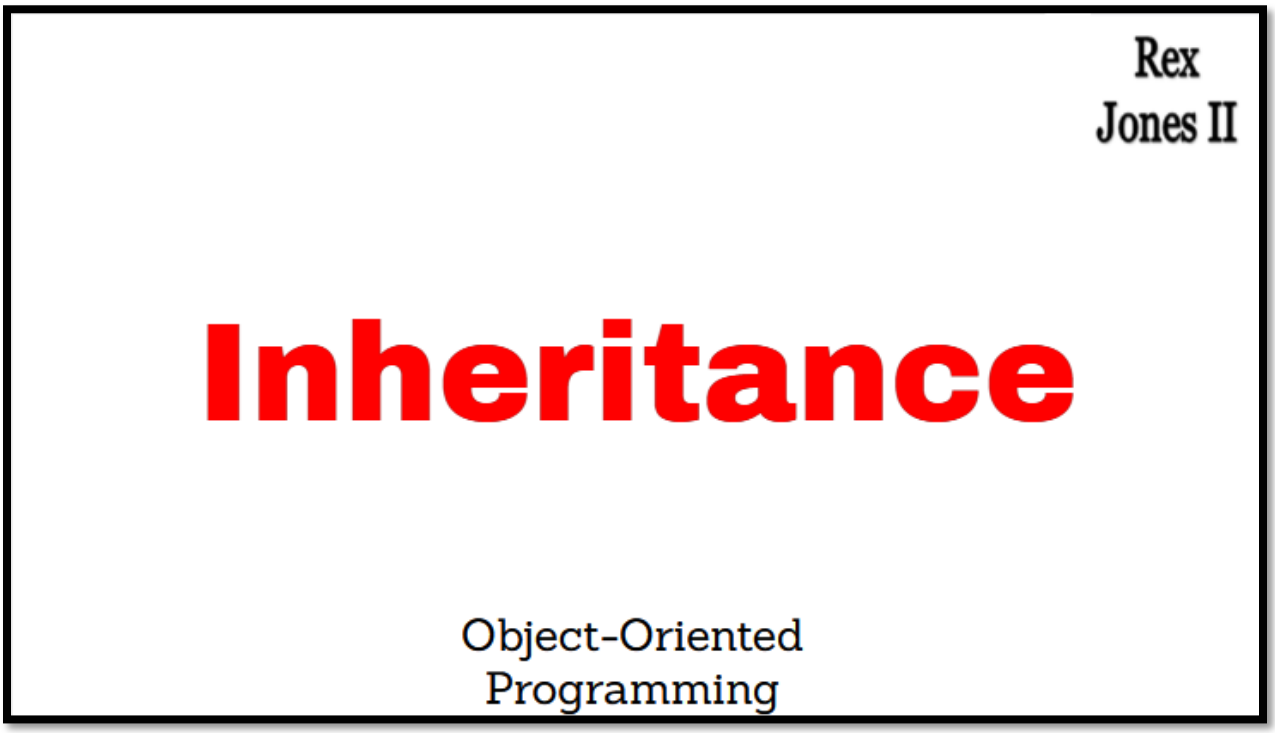
Rex
Jones II

(Transcript) Java Inheritance (OOP)



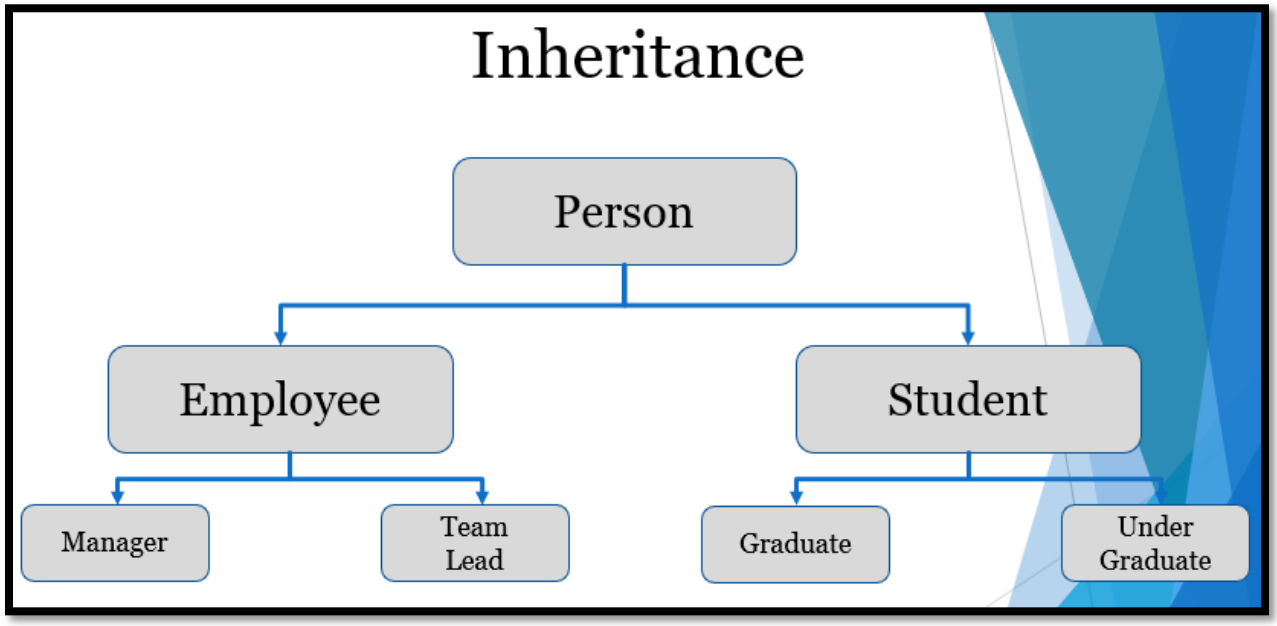
Inheritance is when a class receives the members of another class. It allows us to extend a class then reuse the code. We create a generic class and specialized class. The generic class defines common traits that will be inherited by a specialized class. In Java, the inherited class is called a superclass also known as parent class. The special class which receives an inheritance is called a subclass also known as child class.

Hi my name is Rex and I share automation and programming knowledge. In this video and the next few videos, I am going to share knowledge on Java programming. If you are looking to learn or refresh your knowledge, then connect with me on LinkedIn, YouTube, Twitter, Facebook and GitHub. You can download the code and each transcript from GitHub.

With Object-Oriented Programming, Inheritance is known for representing an **"is a"** relationship. For example, an Employee **is a** Person, so the Employee class can be a child class of the Person class. In the same way, a Student **is a** Person and Student can be a child class of the Person class. With this hierarchy, the Person class is the parent of 2 child classes: Employee and Student. The child classes can become a parent to other classes.

When it comes to sharing data, both child classes and each grandchild inherit from the Person parent class. Let's say, the Person class has a variable called name and a method called getName. From left to

right, the Employee, Student, Manager, Team Lead, Graduate, and Undergraduate classes inherit those same members from the Person class.



In Eclipse, we are going to write 2 variables for the Person class.
private String name and private String dateOfBirth. Next, let's add the getter and setter methods by right clicking > Source > Generate Getters and Setters > expand both checkboxes. Eclipse will automatically add these 4 getter and setter methods. Check dateOfBirth and name then click OK. We have all 4 methods.

```
public class Person {
    private String name;
    private String dateOfBirth;

    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getDateOfBirth() {
        return dateOfBirth;
    }
    public void setDateOfBirth(String dateOfBirth) {
        this.dateOfBirth = dateOfBirth;
    }
}
```

Rex Jones II

Next is the Student child class. The purpose of Inheritance is to reuse code by extending a class. Therefore, Student extends the Person class. The keyword extends produces an Inheritance relationship between Student and Person. If I click CTRL + SPACE, we see all of the methods from the Person class. The Student class inherited getDateOfBirth, getName, setDateOfBirth, and setName. Notice, we do not see the private variables name and dateOfBirth. They cannot be seen or accessed by the child class. Inheriting a class does not overrule the access restriction for private properties. Private properties are not visible by outside classes.

One of the benefits of Inheritance, is the public properties of a parent class can be shared by all child classes. After receiving the public properties, a child class can add its own unique members. Now, we are not concerned about creating a variable and method for name or date of birth in the Student class. This helps the Student class focus on specialized variables and methods that are only used by a Student. For example, we can write private int studentID and private String className. The studentID and className are special to the Student class. Generate the methods again: by right clicking > Source > Generate Getters and Setters > check both boxes and click OK.

```
public class Student extends Person {
    private int studentID;
    private String className;

    public int getStudentID() {
        return studentID;
    }
    public void setStudentID(int studentID) {
        this.studentID = studentID;
    }
    public String getClassName() {
        return className;
    }
    public void setClassName(String className) {
        this.className = className;
    }
}
```

We have our methods. We have our parent class and child class. The next class TestInheritance will access both classes. We create a person by writing Person person = new Person (). A student is created by writing Student student = new Student ().

The person object dot has access to all 4 methods defined in the Person class. Let's select getName from the Person class. The student object dot has access to all 8 methods which are defined in the Person class and Student class. Let's also select getName. Do you see how we have access to getName? That's why we are not concerned about the variable being private because the data can still be accessed through a getter method. How about both methods from the Student class? student.getStudentID and student.getClassName.

```
public class TestInheritance {

   public static void main(String[] args) {
      Person person = new Person ();
      person.getName();

      Student student = new Student ();
      student.getName();
      student.getStudentID();
      student.getClassName();
   }
}
```

This is a perfect example of Inheritance where the child class receives members from a parent class. Next, we will take a look at Polymorphism.