

The Employee Class Program

by Sophia



WHAT'S COVERED

In this lesson, you will explore the creation of a completed class that contains attributes and methods. Specifically, this lesson covers:

Table of Contents

- [1. The Employee Class](#)

1. The Employee Class

Since Java is one of the computer languages that follows the object-oriented programming (OOP) model, you've basically been working with classes and objects since the beginning of all of the tutorials (every program has a declared class and all library methods involve classes). When it comes to classes and objects, it's important to note that an object should have everything about itself encapsulated in a class.

Let's first start by looking at an employee of an organization. For an employee, think about what attributes may be needed to describe them. You would have their first name, last name, title, salary, hire date and employee ID as their basic attributes. The start of the Employee class and its attributes should look like this (declared in a file named Employee.java). Note that this code is not a complete class:

➞ EXAMPLE

```
import java.time.LocalDate;

public class Employee {
    private String firstName;
    private String lastName;
    private int empId;
    private String jobTitle;
    private double salary;
    private LocalDate hireDate;
```

Let's continue to define our class with an appropriate constructor. You will need to have each of those

instance variables (first name, last name, etc.) set as part of the parameters of the constructor. The only item that won't be passed in will be the hire date, which will use the current day's date for when the object representing the employee is created. You would need to import `java.util.LocalDate` for that:

➞ EXAMPLE

```
public Employee(String firstName, String lastName, int emplId, String jobTitle,
    double salary) {
    this.firstName = firstName;
    this.lastName = lastName;
    this.emplId = emplId;
    this.jobTitle = jobTitle;
    this.salary = salary;
    this.hireDate = LocalDate.now();
}
```



TRY IT

Directions: Create our `Employee` class with the attributes and a parameterized constructor.

Remember the needed import for `java.time.LocalDate`, as shown above. This gives us a starting point with the employee's details. Once the attributes are set, you typically won't be accessing these variables directly. Rather, you will use accessor methods to get the data from these variables. You will also want to set up mutator (or "setter") methods so you can set these values after a bit of error checking.

For example, for the first name, returning it may not have anything special or unique to check, but when you try to set the first name, you will make sure that the length of the value that's being passed in isn't empty. So, let's add a method called `getFirstName()` that returns the `firstName` variable when it is called by the new instance. And you will create another method called `setFirstName()` with an additional parameter of `firstName` to check that the argument passed is not an empty string. In order to use it, you must call it directly to set the `firstName` attribute. This could be done after the object has been defined, and the constructor has set the values already. Using this method, you will be able to update the attribute value afterwards.

➞ EXAMPLE

```
// Returns the first name
public String getFirstName() {
    return firstName;
}

// Sets the value of attribute firstName to value passed as parameter firstName
public void setFirstName(String firstName) {
    if(firstName.length() > 0) {
        this.firstName = firstName;
    }
}
```

**TRY IT**

Directions: Go ahead and add these two new methods. This of course doesn't check if the value passed into the parameter for the first name was an empty String (""). You would need to handle that separately. The last name and job title can use the same format for their respective methods to set and get those values.

↪ EXAMPLE

```
public String getLastName() {
    return lastName;
}

public void setLastName(String lastName) {
    if(lastName.length() > 0) {
        this.lastName = lastName;
    }
}
```

↪ EXAMPLE

```
public String getJobTitle() {
    return jobTitle;
}

public void setJobTitle(String jobTitle) {
    if(jobTitle.length() > 0) {
        this.jobTitle = jobTitle;
    }
}
```

**TRY IT**

Directions: Go ahead and add the four new methods for reading and setting the lastName and jobTitle. Let's see if you have what is currently created so far:

```
import java.time.LocalDate;
```

```
public class Employee {
    private String firstName;
    private String lastName;
    private int emplId;
    private String jobTitle;
    private double salary;
    private LocalDate hireDate;

    // Parameterized constructor
    public Employee(String firstName, String lastName, int emplId, String jobTitle,
```

```

        double salary) {
    this.firstName = firstName;
    this.lastName = lastName;
    this.emplId = emplId;
    this.jobTitle = jobTitle;
    this.salary = salary;
    this.hireDate = LocalDate.now();
}

// Returns the first name
public String getFirstName() {
    return firstName;
}

// Sets the value of attribute firstName to value passed as parameter firstName
public void setFirstName(String firstName) {
    if(firstName.length() > 0) {
        this.firstName = firstName;
    }
}

public String getLastName() {
    return lastName;
}

public void setLastName(String lastName) {
    if(lastName.length() > 0) {
        this.lastName = lastName;
    }
}

public String getJobTitle() {
    return jobTitle;
}

public void setJobTitle(String jobTitle) {
    if(jobTitle.length() > 0) {
        this.jobTitle = jobTitle;
    }
}
}

```

If your class looks different than what is displayed above, check line by line to see what may be different. Make sure it looks identical before moving on.



Now you will have to set up the salary and employee ID.

For the employee ID (attribute named `emplId`), you will not permit the value to be updated (once an employee has an employee ID, it is typically set), so you only need to create the get method to return the value. You will not need a set method for this variable:

➤ EXAMPLE

```
// emplId cannot be changed, so there is only accessor, no mutator method public int  
getEmplId() { return emplId; }
```



TRY IT

Directions: Add the `getEmplId()` method to your class. For the salary, the accessor method will return the annual salary as a double. You need a method that returns the salary as a numeric value that can be used in calculations. You can also declare a method that returns the salary as a formatted String, but as you will see, that method can't be called `getSalary()`.

➤ EXAMPLE

```
public double getSalary() {  
    return salary;  
}
```

For the mutator or setter method, you will permit any value that's larger than 0:

➤ EXAMPLE

```
public void setSalary(double salary) {  
    if(salary > 0.00) {  
        this.salary = salary;  
    }  
}
```

To get the salary in a format that represents an amount as dollars and cents, you will use one of Java's format classes. You do not want to get into a lot of detail about formatting, but the Java `DecimalFormat` class makes it fairly easy to format a numeric value as an amount of money. Before the start of the `Employee` class, add this import:

➤ EXAMPLE

```
import java.text.DecimalFormat;
```

A `DecimalFormat` object is a pattern or template for displaying numeric values. This line of code (in the body of `main()`) defines a format that has a leading dollar sign, an initial 0 if there is no value to the left of the decimal point, and 2 decimal places (with 0's added to the right to fill out the pattern:

➤ EXAMPLE

```
DecimalFormat salaryFormat = new DecimalFormat("$0.00");
```

Here, the DecimalFormat formatting pattern is named salaryFormat. A DecimalFormat object includes a format() method that converts the numeric value into a String in the specified format. Here is the complete getSalaryAsString() method that returns the salary as a String:

```
public String getSalaryAsString() {  
    // Format salary with leading dollar sign and 2 decimal places  
    DecimalFormat salaryFormat = new DecimalFormat("$0.00");  
    // Use getSalary to get numeric value and then format  
    return salaryFormat.format(getSalary());  
}
```

Note that while this may look somewhat like an override of the getSalary() method, it really isn't because only the return type varies. Since the return type is not part of a method signature in Java, the compiler won't let us just call this a version of getSalary(), so the distinct name getSalaryAsString() is needed.



Directions: Now add the three new methods for salary, the get method (for returning), and the set method to make sure the value is greater than 0.

Altogether, our code should look like the following:

```
import java.text.DecimalFormat;  
import java.time.LocalDate;  
  
public class Employee {  
    private String firstName;  
    private String lastName;  
    private int emplId;  
    private String jobTitle;  
    private double salary;  
    private LocalDate hireDate;  
  
    // Parameterized constructor  
    public Employee(String firstName, String lastName, int emplId, String jobTitle,  
        double salary) {  
        this.firstName = firstName;  
        this.lastName = lastName;  
        this.emplId = emplId;  
        this.jobTitle = jobTitle;  
        this.salary = salary;  
        this.hireDate = LocalDate.now();  
    }  
}
```

```

// Returns the first name
public String getFirstName() {
    return firstName;
}

// Sets the value of attribute firstName to value passed as parameter firstName
public void setFirstName(String firstName) {
    if(firstName.length() > 0) {
        this.firstName = firstName;
    }
}

public String getLastName() {
    return lastName;
}

public void setLastName(String lastName) {
    if(lastName.length() > 0) {
        this.lastName = lastName;
    }
}

public String getJobTitle() {
    return jobTitle;
}

public void setJobTitle(String jobTitle) {
    if(jobTitle.length() > 0) {
        this.jobTitle = jobTitle;
    }
}

public double getSalary() {
    return salary;
}

public void setSalary(double salary) {
    if(salary > 0.00) {
        this.salary = salary;
    }
}

public String getSalaryAsString() {
    // Format salary with leading dollar sign and 2 decimal places
    DecimalFormat salaryFormat = new DecimalFormat("$0.00");
    // Use getSalary to get numeric value and then format

```

```

        return salaryFormat.format(getSalary());
    }

    // emplId cannot be changed, so there is only accessor, no mutator method
    public int getEmplId() {
        return emplId;
    }
}

```

Now that our class is set up, it's time to create instances (objects) of the `Employee` class.



Directions: Under the `Employee` class, start to create the first object. Call it “sophia” or any employee name you would like as the first object. Start with the name and the equal sign (=), then our class `Employee`, then the first parenthesis:

➞ **EXAMPLE** As a reminder, here is the signature for the `Employee()` constructor:

```
Employee(String firstName, String lastName, int emplId, String jobTitle,
        double salary)
```

➞ **EXAMPLE** A sample call to the constructor looks like this:

```
Employee empl = new Employee("Jack", "Krichen", 1000, "Manager", 75000);
```



Directions: Let's write the code that creates an `Employee` object. You can use the same arguments as seen in the code below, or create your own employee. After entering the arguments, make sure to add the `print()` functions with each variable, so you can see that the instance created, “sophia” in this case, did include our arguments:

```

public class EmployeeProgram {
    public static void main(String[] args) {
        Employee empl = new Employee("Jack", "Krichen", 1000, "Manager", 75000);
        System.out.println("First Name: " + empl.getFirstName());
        System.out.println("Last Name: " + empl.getLastName());
        System.out.println("EmplId: " + empl.getEmplId());
        System.out.println("Job Title: " + empl.getJobTitle());
        System.out.println("Salary: " + empl.getSalaryAsString());
    }
}

```



Directions: Go ahead and run the program to see if you get the same output, or the expected output if you used different employee values. Remember to compile the `Employee` class first, using this command:

➞ **EXAMPLE**


```
javac Employee.java
```



TRY IT

Directions: Then, run the program using this command:

➞ EXAMPLE

```
java EmployeeProgram.java
```

The results should look like this:

Console Shell

```
> javac Employee.java
> java EmployeeProgram.java
First Name: Jack
Last Name: Krichen
EmplId: 1000
Job Title: Manager
Salary: $75000.00
> 
```



REFLECT

This would be a good foundation for our Employee class. However, you could add some more functionality, such as allowing for increases in the employee's salary. You could pass in a percentage and the salary would automatically be increased by that amount:

➞ EXAMPLE

```
// Method to increase salary by percent as decimal. 0.02 is a 2% raise
public void increaseSalary(double percentAsDecimal) {
    if(percentAsDecimal > 0.0) {
        salary *= (1 + percentAsDecimal);
    }
}
```



REFLECT

Notice that as part of the calculation of this new `increaseSalary()` method, you check if the percentage is greater than 0 or not. If it is, you will add 1 to the decimal representing the rate of increase and multiply the current salary by the sum. Using the **compound assignment operator for multiplication** (`*=`) takes the current

value of the salary attribute and multiplies it by the factor for the raise and assigns the product back to the salary attribute.



Directions: Let's give it a shot by adding a 2% increase in salary. First, add this new method to the Employee class and then add the increase of salary code below to the program. Finally, run the program.

Here is the code for the Employee class (Employee.java):

```
import java.text.DecimalFormat;
import java.time.LocalDate;

public class Employee {
    private String firstName;
    private String lastName;
    private int emplId;
    private String jobTitle;
    private double salary;
    private LocalDate hireDate;

    // Parameterized constructor
    public Employee(String firstName, String lastName, int emplId, String jobTitle,
        double salary) {
        this.firstName = firstName;
        this.lastName = lastName;
        this.emplId = emplId;
        this.jobTitle = jobTitle;
        this.salary = salary;
        this.hireDate = LocalDate.now();
    }

    // Returns the first name
    public String getFirstName() {
        return firstName;
    }

    // Sets the value of attribute firstName to value passed as parameter firstName
    public void setFirstName(String firstName) {
        if(firstName.length() > 0) {
            this.firstName = firstName;
        }
    }

    public String getLastName() {
        return lastName;
    }
}
```

```

public void setLastName(String lastName) {
    if(lastName.length() > 0) {
        this.lastName = lastName;
    }
}

```

```

public String getJobTitle() {
    return jobTitle;
}

```

```

public void setJobTitle(String jobTitle) {
    if(jobTitle.length() > 0) {
        this.jobTitle = jobTitle;
    }
}

```

```

public double getSalary() {
    return salary;
}

```

```

public void setSalary(double salary) {
    if(salary > 0.00) {
        this.salary = salary;
    }
}

```

```

public String getSalaryAsString() {
    // Format salary with leading dollar sign and 2 decimal places
    DecimalFormat salaryFormat = new DecimalFormat("$0.00");
    // Use getSalary to get numeric value and then format
    return salaryFormat.format(getSalary());
}

```

```

// emplId cannot be changed, so there is only accessor, no mutator method
public int getEmplId() {
    return emplId;
}

```

```

// Method to increase salary by percent as decimal. 0.02 is a 2% raise
public void increaseSalary(double percentAsDecimal) {
    if(percentAsDecimal > 0.0) {
        salary *= (1 + percentAsDecimal);
    }
}
}

```

After entering this code in a file named Employee.java, remember to compile the class using this command:

➔ EXAMPLE

```
javac Employee.java
```

Here is the code for the application's driver class (EmployeeProgram) that uses the Employee class. This code needs to be entered in a file named EmployeeProgram.java:

```
import java.text.DecimalFormat; // Needed for DecimalFormat
```

```
public class EmployeeProgram {  
    public static void main(String[] args) {  
        Employee empl = new Employee("Jack", "Krichen", 1000, "Manager", 75000);  
        System.out.println("First Name: " + empl.getFirstName());  
        System.out.println("Last Name: " + empl.getLastName());  
        System.out.println("EmplId: " + empl.getEmplId());  
        System.out.println("Job Title: " + empl.getJobTitle());  
        System.out.println("Salary: " + empl.getSalaryAsString());  
    }  
}
```



REFLECT

Did you get the following as expected? Now our employee will get a bump of \$1000.00.

Console

Shell

```
> javac Employee.java  
> java EmployeeProgram.java  
First Name: Jack  
Last Name: Krichen  
EmplId: 1000  
Job Title: Manager  
Salary: $75000.00  
> 
```



THINK ABOUT IT

What happens in the case when the salary increase is less than 0?

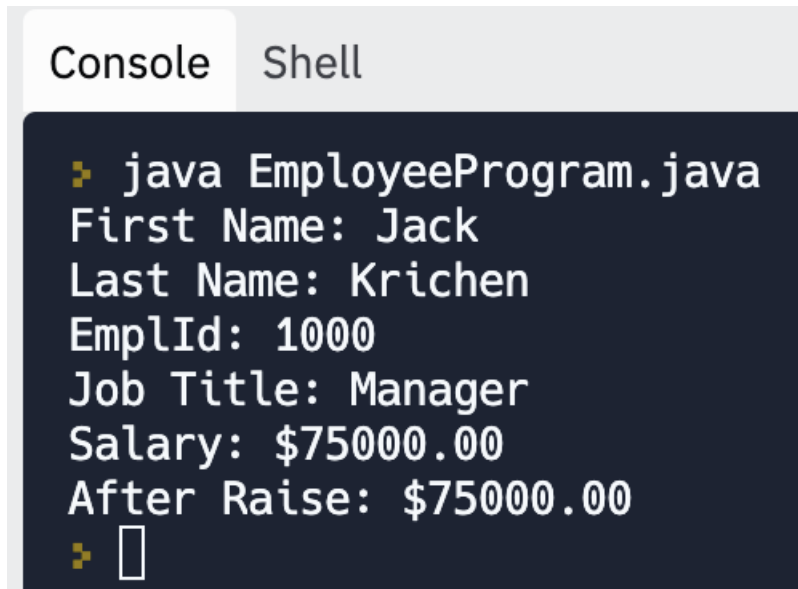


TRY IT

Directions: Add the following two lines after the line that prints out the salary:

```
empl.increaseSalary(-0.02);  
System.out.println("After Raise: " + salaryFormat.format(empl.getSalary()));
```

The result should look like this:



```
> java EmployeeProgram.java  
First Name: Jack  
Last Name: Krichen  
EmpId: 1000  
Job Title: Manager  
Salary: $75000.00  
After Raise: $75000.00  
> 
```

 REFLECT

In this case, there is no change. But this was probably an error on the part of the user entering the values.

 TRY IT

Directions: Go ahead and try a negative salary increase. This is not an ideal result, as you most likely would want to inform the user that the value was not accepted. You can add that message to the user to inform them of the error. In the `Employee` class, revise the `increaseSalary()` method to look like this:

```
// Method to increase salary by percent as decimal. 0.02 is a 2% raise  
public void increaseSalary(double percentAsDecimal) {  
    if(percentAsDecimal > 0.0) {  
        this.salary *= (1 + percentAsDecimal);  
    }  
    else {  
        System.out.println("Salary increase must be greater than 0.");  
    }  
}
```

 REFLECT

Here you have added a selection statement to look to see if the value entered (passed as an argument) is greater than 0. If it is not, it will present an error.

 TRY IT

Directions: Add the conditional statement lines to our `increaseSalary()` method.

Your program should look like this:

```
import java.time.LocalDate;

public class Employee {
    private String firstName;
    private String lastName;
    private int emplId;
    private String jobTitle;
    private double salary;
    private LocalDate hireDate;

    // Parameterized constructor
    public Employee(String firstName, String lastName, int emplId, String jobTitle,
        double salary) {
        this.firstName = firstName;
        this.lastName = lastName;
        this.emplId = emplId;
        this.jobTitle = jobTitle;
        this.salary = salary;
        this.hireDate = LocalDate.now();
    }

    public String getFirstName() {
        return firstName;
    }

    public void setFirstName(String firstName) {
        if(firstName.length() > 0) {
            this.firstName = firstName;
        }
    }

    public String getLastName() {
        return lastName;
    }

    public void setLastName(String lastName) {
        if(lastName.length() > 0) {
            this.lastName = lastName;
        }
    }

    // EmplId cannot be changed, so there is only accessor, no mutator method
    public int getEmplId() {
        return emplId;
    }
}
```

```

}

public String getJobTitle() {
    return jobTitle;
}

public void setJobTitle(String jobTitle) {
    if(jobTitle.length() > 0) {
        this.jobTitle = jobTitle;
    }
}

public double getSalary() {
    return salary;
}

public void setSalary(double salary) {
    if(salary > 0.00) {
        this.salary = salary;
    }
}

public String getSalaryAsString() {
    // Format salary with leading dollar sign and 2 decimal places
    DecimalFormat salaryFormat = new DecimalFormat("$0.00");
    // Use getSalary to get numeric value and then format
    return salaryFormat.format(getSalary());
}

// Method to increase salary by percent as decimal. 0.02 is a 2% raise
public void increaseSalary(double percentAsDecimal) {
    if(percentAsDecimal > 0.0) {
        this.salary *= (1 + percentAsDecimal);
    }
    else {
        System.out.println("Salary increase must be greater than 0.");
    }
}
}

```



Directions: Compile the revised Employee class using this command:

➞ EXAMPLE

```
javac Employee.java
```

**TRY IT**

Directions: Try running the program again using the following in the shell:

➞ EXAMPLE

```
java EmployeeProgram.java
```

You should now see the error message appear:

Console**Shell**

```
> javac Employee.java
> java EmployeeProgram.java
First Name: Jack
Last Name: Krichen
EmplId: 1000
Job Title: Manager
Salary: $75000.00
> 
```

**BRAINSTORM**

Directions: This marks the end of this program for now. Try making some additional changes on your own. See if you can add any additional parameters to our Employee class.

**REFLECT**

We have built out the Employee class to the point where it is starting to resemble a class that would be useful in the real world. What other data and functionality might be appropriate parts of the Employee class? How could a programmer determine what information and related functionality would or would not belong in a class?

**TERM TO KNOW**

Compound Assignment Operator for Multiplication (*=)

This operator multiplies the variable to the left by the value or expression to the right and assigns the product back to the variable to the left.

**SUMMARY**

In this lesson, you went through a program that creates an **Employee class** with all of the attributes and methods associated with it. You have included validation and verification of the object attributes

to ensure that valid values are being set when they are being updated.

Source: This content and supplemental material has been adapted from Java, Java, Java: Object-Oriented Problem Solving. Source cs.trincoll.edu/~ram/jjj/jjj-os-20170625.pdf

It has also been adapted from “Python for Everybody” By Dr. Charles R. Severance. Source py4e.com/html3/



TERMS TO KNOW

Compound Assignment Operator for Multiplication (*=)

This operator multiplies the variable to the left by the value or expression to the right and assigns the product back to the variable to the left.