Revisiting the Employee Class Program

by Sophia

WHAT'S COVERED

In this lesson, you will be extending on the Employee class to create subclasses. Specifically, this lesson covers:

Table of Contents

1. Creating the Base Class Person

In a prior tutorial, we had created a basic Employee class that looks 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;
  }

  // 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);
    }
  }
}
```

TRY IT

Directions: If you don't already have this in Replit, enter it into a file named Employee.java since we will be modifying this example with an updated base class and new subclasses.

Although you have Employee as the prior class, you will want to consider other aspects about an employee. For example, you can have different types of employees. You could have full-time and part-time employees that get vacation hours and an annual salary, as we currently have in our “Employee” class. You could also have contractors that get an hourly wage but don’t accumulate vacation time or have an annual salary. Contractors could also have a contractor ID rather than an employee ID.

CONCEPT TO KNOW

In order to build a correctly defined base class, you need to pull in only the key information that would be consistent across both the contractor and employee classes. In our next example, we will define the base class as Person and only place in what is common. The Person class needs to be entered into a file named Person.java.

By removing all items related to salary and employee ID, you will have the following result:

```
import java.time.LocalDate;

public class Person {
  private String firstName;
  private String lastName;
  private String jobTitle;
  private LocalDate hireDate;
```

```
 // Parameterized constructor
 public Person(String firstName, String lastName, int emplId, String jobTitle) {
  this.firstName = firstName;
  this.lastName = lastName;
  this.jobTitle = jobTitle;
  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;
  }
 }
}
```

TRY IT

Directions: Remove all aspects of salary and employee ID information from this updated base class called Person now. Make sure your program looks like the code above. Compile the Person class using this command:

↪ EXAMPLE

```
javac Person.java
```

Now, with the Person base class, we are set up to create our unique subclasses.

REFLECT

Before we move on to classes that inherit from Person, it is important to note how the Person class (which will be our base or parent class) encapsulates data and functionality that will be common to all people working for a given company. This will help us avoid redundancy in the subclasses, and a good design avoids redundancy whenever possible.

2. Creating the Subclass Employee

In the Person base class, we have the common information defined as firstName, lastName, jobTitle, and the hireDate attributes. Now that you have that content in place, you can create the Employee subclass to extend the Person base class and have the custom content that makes it unique. You will be using most of the elements that you had in the prior base class that we initially set up.

The declaration of the Employee class indicates that it inherits from the Person class (using the extends keyword):

```
import java.text.DecimalFormat;

public class Employee extends Person {
  private int emplId;
  private double salary;

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

  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);
   }
  }
}
```

Note how the `Employee()` constructor has parameters that include the values needed to be passed to the constructor for the `Person` base class. The constructor for the base class (also called the "superclass") is called using `super()`. The call to `super()` passes the parameters needed by the superclass's constructor—in this case, the String values for first name, last name, employee ID, and salary. It's important that the first statement in the subclass's constructor is the call to `super()`. The `Employee()` constructor then sets the values for the emplId and salary attributes. These are the attributes that are specific to the Employee subclass.

TRY IT

Directions: Now type in the code for the Employee subclass in a file named Employee.java. Compile the Employee class by running this command:

EXAMPLE

```
javac Employee.java
```

Attributes `vacationDaysPerYear/` and `vacationDaysRemaining`

We did say that employees should receive vacation days. Let’s say by default for this organization, all employees have 14 days of vacation. Now it is helpful to have two different attributes for vacations—one for the yearly total (14) and one for the actual days remaining for the specific employee. You will add the field called `vacationDaysPerYear` and set it to 14. Next, we will create the field called vacationDaysRemaining which is set to `vacationDaysPerYear` as the default. We will update the constructor to include these attributes as follows:

```
public class Employee extends Person {
  private int emplId;
  private double salary;
  private int vacationDaysPerYear = 14;
  private int vacationDaysRemaining;

  // Parameterized constructor
  public Employee(String firstName, String lastName, int emplId, String jobTitle, double salary) {
    super(firstName, lastName, emplId, jobTitle);
    this.salary = salary;
    vacationDaysRemaining = vacationDaysPeryear;
}
```

TRY IT

Directions: Go ahead and add these additional attributes and update the constructor in the Employee subclass:

↱ EXAMPLE

method: increaseVacationDaysPerYear()

There will be a few methods that will be specific to vacations. One method will be to increase the vacation days per year. Typically, this could be increased by negotiation or based on how long the employee has been at the company.

```
// Increase vacation days per year
public void increaseVacationDaysPerYear(int days) {
  if(days > 0) {
    this.vacationDaysPerYear += days;
  }
}
```

We defined a method called increaseVacationDaysPerYear() with a parameter for the number of days to add. That way, you can pass in an integer to change the default vacation. Next, we have an if() statement that checks if the number passed is larger than 0. If it is, you add that value to the vacationDaysPerYear attribute.

TRY IT

Directions: Go ahead and add the increaseVacationDaysPerYear() method to the Employee subclass:

↱ EXAMPLE

method: increaseVacationDaysRemaining()

The next method you will add will be one that will increase the actual vacation days remaining if the added days were granted. You will again check if days is greater than 0, and if so, you can add to the existing attribute vacationDaysRemaining:

```
// Increase remaining vacation days
public void increaseVacationDaysRemaining(int days) {
  if(days > 0) {
    this.vacationDaysRemaining += days;
  }
}
```

TRY IT

Directions: Go ahead and add the `increaseVacationDaysRemaining()` method to the Employee subclass. We will need to have a method that we will use when an employee wants to take some days off. This method will accept the number of requested days off. As long as the value is greater than 0 and the employee still has days left that's greater than the days requested, it will be permitted. Otherwise, if the days requested is less than or equal to 0, meaning the employee entered 0, we will inform the employee that their request must be greater than 0.

```
 // Use vacation days
 public void takeVacationDays(int days) {
   if(days > 0 && vacationDaysRemaining >= days) {
     this.vacationDaysRemaining -= days;
    }
    else if(days <= 0) {
      System.out.println("Requested vacation days must > 0");
    }
    else {
      System.out.println("Employee doesn't have sufficient vacation to take " +
                     days + " days off.");
    }
 }
```

Here, we have defined a method called `takeVacationDays()` with the parameter days that will accept the requested days off.

TRY IT

Directions: Go ahead and add this `takeVacationDays()` method to the Employee subclass:

↪ EXAMPLE

method: getVacationDaysRemaining()

We'll also have a simple accessor method to return the number of vacation days.

```
// Return number vacation days remaining
public int getVacationDaysRemaining() {
  return vacationDaysRemaining;
}
```

This getVacationDaysRemaining() method will return the value of vacationDaysRemaining.

TRY IT

Directions: Add the getVacationDaysRemaining() method to the Employee subclass. Before we add some instance calls to test this subclass, make sure your program looks like the following:

```
import java.text.DecimalFormat;

public class Employee extends Person {
  private int emplId;
  private double salary;
  private int vacationDaysPerYear = 14;
  private int vacationDaysRemaining;

  // Parameterized constructor
  public Employee(String firstName, String lastName, int emplId, String jobTitle, double salary) {
    super(firstName, lastName, emplId, jobTitle);
    this.salary = salary;
    this.emplId = emplId;
    vacationDaysRemaining = vacationDaysPerYear;
  }

  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);
    }
  }

  // Increase vacation days per year
  public void increaseVacationDaysPerYear(int days) {
    if(days > 0) {
      this.vacationDaysPerYear += days;
    }
  }

  // Increase remaining vacation days
  public void increaseVacationDaysRemaining(int days) {
    if(days > 0) {
      this.vacationDaysRemaining += days;
    }
  }
  // Use vacation days
  public void takeVacationDays(int days) {
    if(days > 0 && vacationDaysRemaining >= days) {
      this.vacationDaysRemaining -= days;
    }
    else if(days <= 0) {
      System.out.println("Requested vacation days must > 0");
    }
    else {
      System.out.println("Employee doesn't have sufficient vacation to take " +
                         days + " days off.");
    }
  }

  // Return number vacation days remaining
  public int getVacationDaysRemaining() {
    return vacationDaysRemaining;
  }
}
```

REFLECT

Since the Employee subclass inherits from the Person class, it will inherit the public methods from the Person class. Think about which methods these are that are inherited from the Person class and how they provide functionality to the Employee class "for free." Note how the attributes and methods in the Employee class add to what is provided by the Person base class.

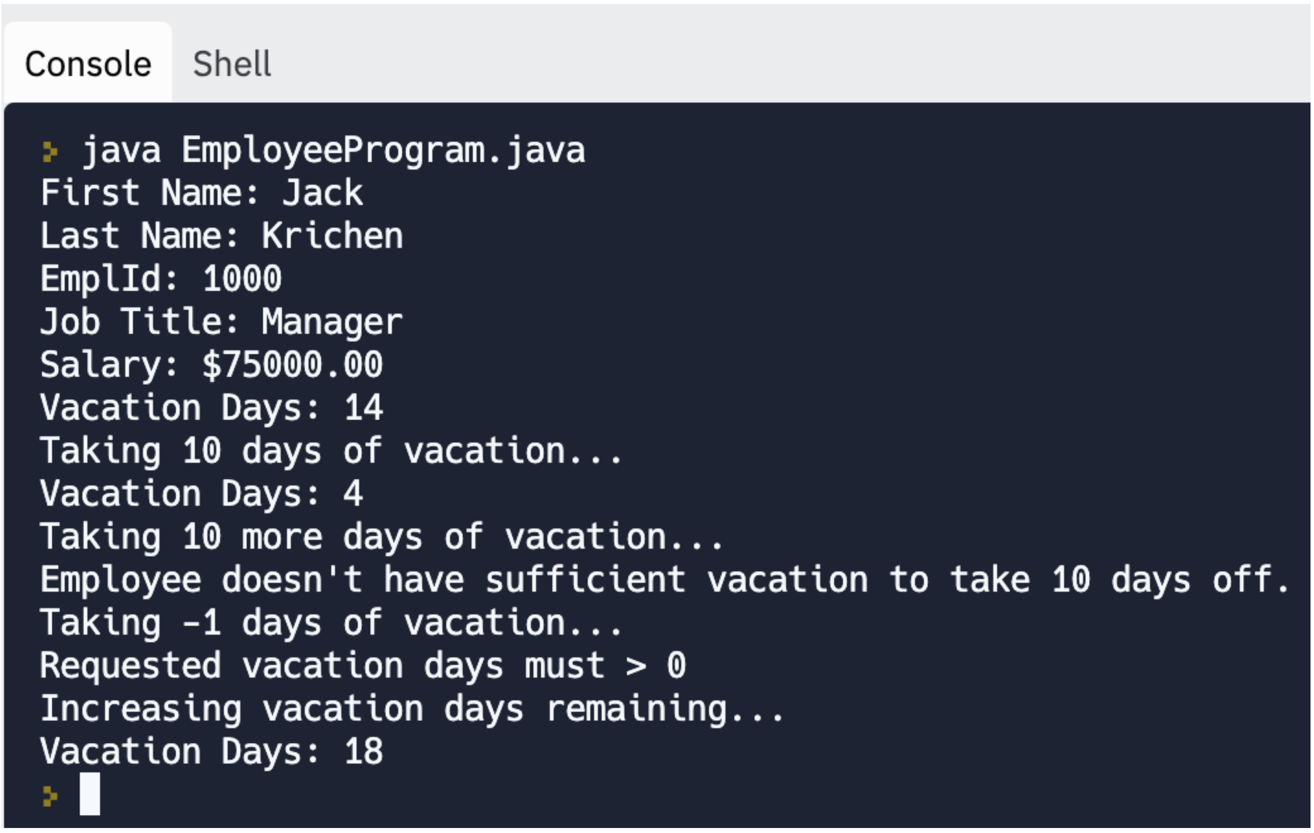
Let’s test out the code, especially around the vacation days methods in the “Employee” subclass, to ensure all is working as expected. First, you will create an instance of the subclass called empl. You will pass some arguments for first name, last name, title, salary, and employee ID. Then, we will create some println() calls, so we can see output to the screen.

```
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());
    // Now display vacation information
    System.out.println("Vacation Days: " + empl.getVacationDaysRemaining());
    System.out.println("Taking 10 days of vacation...");
    empl.takeVacationDays(10);
    System.out.println("Vacation Days: " + empl.getVacationDaysRemaining());
    System.out.println("Taking 10 more days of vacation...");
    empl.takeVacationDays(10);
    System.out.println("Taking -1 days of vacation...");
    empl.takeVacationDays(-1);
    System.out.println("Increasing vacation days remaining...");
    empl.increaseVacationDaysRemaining(14);
    System.out.println("Vacation Days: " + empl.getVacationDaysRemaining());
  }
}
```

TRY IT

Directions: Add the code above to your EmployeeProgram.java file. Give the employee a first name, last name, title, salary, and employee ID. To keep consistent with this example, test with the vacation days indicated. Once entered, run the program.

In the output, you should see the employee’s first name, last name, employee ID, title, and salary for the first five System.out.println() calls.



```
> java EmployeeProgram.java
First Name: Jack
Last Name: Krichen
EmplId: 1000
Job Title: Manager
Salary: $75000.00
Vacation Days: 14
Taking 10 days of vacation...
Vacation Days: 4
Taking 10 more days of vacation...
Employee doesn't have sufficient vacation to take 10 days off.
Taking -1 days of vacation...
Requested vacation days must > 0
Increasing vacation days remaining...
Vacation Days: 18
>
```

REFLECT

Notice that the first output of the current vacation days is 14, which is correct since the attribute vacationDaysRemaining was initially set to the attribute vacationDaysPerYear, which has 14 as the default value. Then, we pass 10 vacation days as a request (argument) to the takeVacationDays() method and print out the value of vacationDaysRemaining once the subtraction is done, so 4 days left is also correct. We then try to take another 10 days of vacation; however, we get an error message since there aren't enough vacation days left (we only had 4 days left after the first request). Next, we try to take a negative number of vacation days, which also returns an error that the argument (request for days off) needs to be greater than 0. Lastly, we increase the vacation days based on the yearly increase and accurately see 18 days, as there were 4 days left and the yearly increase was 14 days.

BRAINSTORM

Directions: Now that you have a working “Employee” subclass, try changing a few arguments to see if you can change what is output to the screen.

3. Creating the Subclass Contractor

Our next step will be to create the “Contractor” subclass. The framework of this class will be the same structure as the “Employee” subclass with some small differences. In particular, you will have a contractorid instead of the employeeid. There will also be an hourly wage instead of a salary, and no vacation.

```
public class Contractor extends Person {
  int contractorId;
  double hourlyWage;
  double totalWage;
  public Contractor(String firstName, String lastName, int contractorId, String jobTitle, double hourlyWage) {
    super(firstName, lastName, jobTitle);
    this.contractorId = contractorId;
    this.hourlyWage = hourlyWage;
  }

  public int getConstractorId() {
    return contractorId;
  }

  public double getHourlyWage() {
    return hourlyWage;
  }
  public void setHourlyWage(double hourlyWage) {
    if(hourlyWage > 0) {
      this.hourlyWage = hourlyWage;
    }
  }
}
```

Most of this should be quite familiar, as the coding structure is the same in this subclass, with some slight differences from the names of the attributes in the Employee subclass.

The getContractorId() method was modeled on the getEmplId() method. The setHourlyWage()method is based on the setSalary() method and getHourlyWage() is a version of the getSalary() method.

TRY IT

Directions: Enter the Contractor subclass in Replit in a file named Contractor.java. Compile the subclass using this command:

EXAMPLE

```
javac Contractor.java
```

Now, let's write some code to test the Contractor class and save it in a file named ContractorProgram.java:

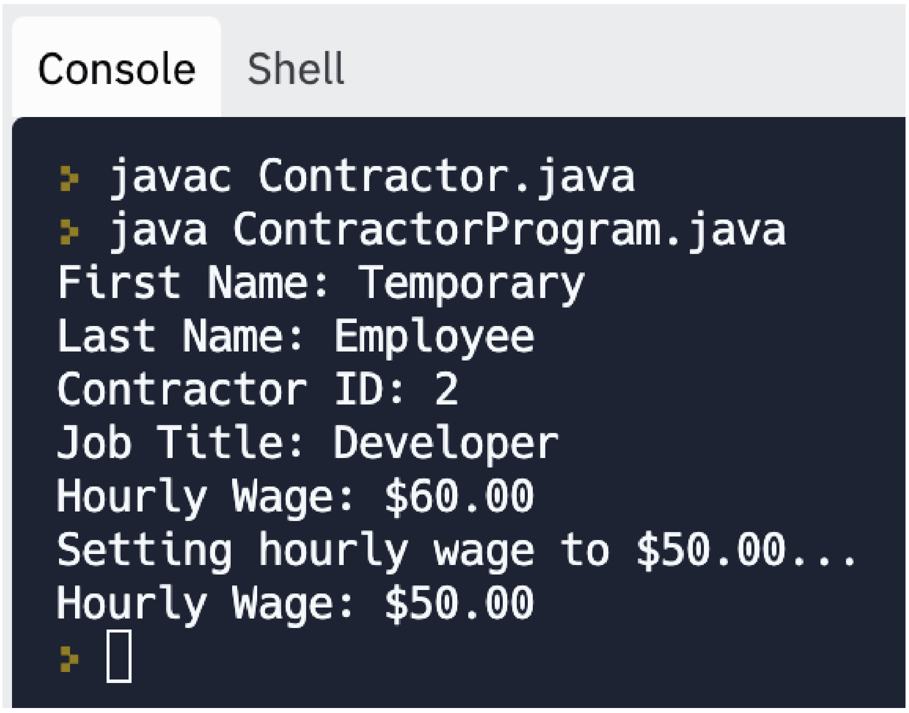
```
import java.text.DecimalFormat;

class ContractorProgram {
  public static void main(String[] args) {
    Contractor contractor = new Contractor("Temporary", "Employee", 2, "Developer", 60.00);
    System.out.println("First Name: " + contractor.getFirstName());
    System.out.println("Last Name: " + contractor.getLastName());
    System.out.println("Contractor ID: " + contractor.getConstractorId());
    System.out.println("Job Title: " + contractor.getJobTitle());
    DecimalFormat wageFormat = new DecimalFormat("$0.00");
    System.out.println("Hourly Wage: " + wageFormat.format(contractor.getHourlyWage()));
    System.out.println("Setting hourly wage to $50.00...");
    contractor.setHourlyWage(50.00);
    System.out.println("Hourly Wage: " + wageFormat.format(contractor.getHourlyWage()));
  }
}
```

TRY IT

Directions: Add the code needed to construct a Contractor object and display its information in a class named ContractorProgram (in a file named ContractorProgram.java) to your program. Give the contractor a first name, last name, title, hourly wage, and contractor ID. To keep consistent with this example, test with the hourly wage indicated. Once entered, run the program:

As we see, the output and contents are slightly different:



```
> javac Contractor.java
> java ContractorProgram.java
First Name: Temporary
Last Name: Employee
Contractor ID: 2
Job Title: Developer
Hourly Wage: $60.00
Setting hourly wage to $50.00...
Hourly Wage: $50.00
> 
```

In the output, we should see the contractor's first name, last name, contractor ID, title, and hourly wage for the first five System.out.println() calls.

Then, we changed the hourly wage to $50 an hour using thesetHourlyWage() method and reprinted the hourly wage again using the getHourlyWage() method.

THINK ABOUT IT

As you look at the code to test, think about how else you would test the code to ensure that it works correctly. What values would you try to set?

SUMMARY

In this lesson, you moved the standard attributes and methods to a **Person base class** that we wanted to exist globally. Then, we took the Employee specific attributes and methods and placed those into a new

Employee subclass. We added methods to the “Employee” subclass to account for salary and vacation days, and tested our program for vacation requests against what an employee has in their current vacation bank. Finally, we **created the Contractor subclass** and introduced an hourly wage as opposed to a salary. In both subclasses, we added a unique ID method only associated with those subclasses.

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/