

Week 2 review - TEams (Project Management Software)

You've been tasked with building an internal project management system for teams to track their projects. You don't need to build a UI for this application. Your job is to build the foundational classes that drive the application's features.

Classes

The core of this application consists of three classes, which you'll create in the main package `com.techelevator`. Make sure to read through the requirements for each class before writing any code.

Note: All dates in this application are strings using the format mm/dd/yyyy.

Step One: Create the `Department` class

Create a new class called `Department.java` with the following requirements.

Instance variables

Name	Type	Getter	Setter
departmentId	int	x	x
name	String	x	x

Constructors

`Department` must have one constructor that accepts two parameters: `departmentID` and `name`.

Step Two: Create the `Employee` class

Create a new class called `Employee.java` with the following requirements.

Instance variables

Name	Type	Getter	Setter
employeeId	long	x	x
firstName	String	x	x
lastName	String	x	x
email	String	x	x
salary	double	x	x
department	Department	x	x

Name	Type	Getter	Setter
hireDate	String	x	x

Static constants

The default starting salary for all employees is \$60,000 and is stored in a static constant variable of type `double`.

Constructors

`Employee` needs two constructors.

The first one accepts all the arguments needed to create a new `Employee`: `employeeID`, `firstName`, `lastName`, `email`, `department`, and `hireDate`.

Note: The first constructor doesn't include a `double` argument for the salary. Make sure to initialize each employees' salary to the static constant you created.

The second constructor is a no-argument constructor. This constructor allows you to create your `Employee` objects in multiple ways.

Methods

Method Name	Description
<code>getFullName()</code>	A derived property that returns the employee's full name in the following format: "Last, First"
<code>raiseSalary(double percent)</code>	A method that raises the employee's salary by x percent

Step Three: Create the `Project` class

Create a new class called `Project.java` with the following requirements.

Instance variables

Name	Type	Getter	Setter
name	String	x	x
description	String	x	x
startDate	String	x	x
dueDate	String	x	x
teamMembers	List<Employee>	x	x

Note: Make sure to set `teamMembers` to an empty list in your implementation.

Constructors

Project must have one constructor that accepts four parameters: **name**, **description**, **startDate** and **dueDate**.

Application

Now that you've created the core classes for this application, you'll write some code to test them. The logic for this application is in `/src/main/java/com/techelevator/Application.java`.

Step One: Create and print departments

Create an instance variable in the **Application** class called **departments** to hold a `List<Department>`.

Next, in `createDepartments()`, create these three departments and add them to the list you created:

departmentId	name
1	Marketing
2	Sales
3	Engineering

Then, in the `printDepartments()` method, iterate over each element in **departments** and print them out. The final output in the console looks like this:

```
----- DEPARTMENTS -----  
Marketing  
Sales  
Engineering
```

Step Two: Create and print employees

Create an instance variable in the **Application** class called **employees** to hold a `List<Employee>`.

Next, in `createEmployees()`, create three employees and add them to the list:

1. Dean Johnson: Create this employee using the no-argument constructor and call setter methods to set each instance variable.
2. Angie Smith: Create this employee using the all-argument constructor.
3. Margaret Thompson: Create this employee using the all-argument constructor.

Tip: use the **Departments** from the **departments** list to assign each employee's department. Retrieve the two departments you need by using the `get()` method.

employeeId	firstName	lastName	email	salary	department	hireDate
1	Dean	Johnson	djohnson@teams.com	60000	Engineering	08/21/2020

employeeId	firstName	lastName	email	salary	department	hireDate
2	Angie	Smith	asmith@teams.com	60000	Engineering	08/21/2020
3	Margaret	Thompson	mthompson@teams.com	60000	Marketing	08/21/2020

Before printing the list of `Employees`, give Angie a 10% raise.

In the `printEmployees()` method, iterate over each element in `employees` and print out their name, salary, and department. Use the derived property `getFullName` for the employee's name. The final output in the console looks like this:

```
----- EMPLOYEES -----  
Johnson, Dean (60000.0) Engineering  
Smith, Angie (66000.0) Engineering  
Thompson, Margaret (60000.0) Marketing
```

Step Three: Create and print projects

Create an instance variable in the `Application` class called `projects` to hold a collection of projects. The variable must be of type `Map<String, Project>` where the key is the name of the project.

In `createTeamsProject()`, create the following project:

- name: TEams
- description: Project Management Software
- startDate: 10/10/2020
- dueDate: 11/10/2020

After you create the project, follow these steps:

1. Add all the employees from the engineering department to this project.
2. Add the project to the `projects` map.

Then, in `createLandingPageProject()`, create the following project:

- name: Marketing Landing Page
- description: Lead Capture Landing Page for Marketing
- startDate: 10/10/2020
- dueDate: 10/17/2020

After you create this project, follow these steps:

1. Add all the employees from the marketing department to this project.
2. Add the project to the `projects` map.

Finally, in `printProjectsReport()`, print out the project's name with the total number of employees on the project. The final output in the console looks like this:

```
----- PROJECTS -----  
TEams: 2  
Marketing Landing Page: 1
```

Final output

```
----- DEPARTMENTS -----  
Marketing  
Sales  
Engineering  
  
----- EMPLOYEES -----  
Johnson, Dean (60000.0) Engineering  
Smith, Angie (66000.0) Engineering  
Thompson, Margaret (60000.0) Marketing  
  
----- PROJECTS -----  
TEams: 2  
Marketing Landing Page: 1
```

Bonus challenges

If you finish early, here are three challenge projects you can work on.

Employee salary formatting

Right now, an employee's salary is of type `double`. When it prints to the console, it looks like this: `60000.0`. It'd be better to display this number in currency format, so it looks like this: `$60,000.00`.

In the Java Standard Library, there's a class called `NumberFormat`. The `NumberFormat` class has a static method called `getCurrencyInstance()` that formats currency given your `Locale`:

```
NumberFormat currency = NumberFormat.getCurrencyInstance();  
currency.format(number)
```

Given this information, you can refactor the `printEmployees()` method to display this output:

```
----- EMPLOYEES -----  
Johnson, Dean ($60,000.00) Engineering  
Smith, Angie ($66,000.00) Engineering  
Thompson, Margaret ($60,000.00) Marketing
```

Dates

In the `Project` and `Employee` classes, you used the type `String` for the dates. In a real-world application, if you needed to perform calculations on dates, you wouldn't use a `String`.

In the Java Standard Library, there's a class called `LocalDate` that you can use with dates.

Project

Right now, you create a new project by hard-coding a start date. You can use the `LocalDate` API to get today's date and set that as the start date instead.

There's also a way to set the end date to `x` amount of days after the start date. Update both `startDate` and `dueDate` to type `LocalDate` using the following requirements:

- Project TEams
 - start date: today
 - due date 30 days after today
- Marketing Landing Page
 - start date: today + 31 days
 - due date: start date + 7 days

Employee

Update the `hireDate` to use the `LocalDate` type. In the `createEmployees()` method, create a variable called `today` and use the `LocalDate` API to get today's date. You can now use that variable when creating each of your employees.

Find department by name

Earlier, you wrote code to retrieve a specific department from the `departments` list by its index, but what if you didn't know its index, or if it was an unordered collection? You could instead loop through the collection, test for a particular value, and return the item that matches. This is a common technique to use when a language or library doesn't provide a built-in method, or you're managing complex data or conditions.

Create another `private` method in the `Application` class. Have the method accept a `String` and return a `Department`. Give the method a descriptive name for what it does—for example, `getDepartmentByName`.

In the method, iterate through `departments`, and test if each department's `name` matches the `String` passed into the method. If it matches, return that `Department`. If there's no match, return `null`.

Now, go back to the `createEmployees()` method. Locate the code you wrote for getting the Engineering and Marketing departments. Instead of retrieving it from `departments` by index, use the new method you wrote, passing in the department name.

You'll know your new method was successful if you still have the same output from the `printEmployees()` and `printProjectsReport()` methods.