Week 3 Pairs Exercise

Congratulations—you've been hired by a pet setting and grooming company called Pet Elevator. They're building their own software to use in-house.

Your job is to create classes for the Customer Relationship Management (CRM) system. A CRM system helps to manage customer information and other related data.

Since Pet Elevator is a pet sitting and grooming company, you'll need to create a class to represent the human customers and a class to represent their pets.

Getting started

Step One: Open the project

1. In Eclipse, select **File > Import...**.
2. In the import window, expand **Maven**, select **Existing Maven Projects**, and click **Next >**.
3. Click **Browse...** and select the directory that contains the `pom.xml` file of this project.
4. Click **Finish** to import the project.

Step Two: Review the starting code:

The software team has started building a system for the Human Resources (HR) department to manage employee, manager, and department information.

Take a moment to explore the starting code in the `com.techelevator` package and `com.techelevator.hr` package:

- `com.techelevator`
  - `Person.java`: base class representing a person
  - `Billable.java`: interface defining methods for objects that should be "billable"—meaning someone that can be billed for services
- `com.techelevator.hr`
  - `Department.java`: class to represent a department in the business
  - `Employee.java`: class to represent an employee of the company, inherits `Person` class
  - `Manager.java`: class to represent a manager of the company, inherits `Employee` class

Step Three: Review tests

Review and run the existing tests in the `/src/test/java/com/techelevator` package.

Notes for all classes

- If there's nothing in the set column, that means the property is a derived property.

CRM system requirements

- *All classes you create must be in the `com.techelevator.crm` package.*
  - By convention, these classes go in the `src\main\java\com\techelevator\crm` folder.

- The project must not have any build errors.
- All unit tests pass as expected.
- Appropriate variable names and data types are used.
- Code is presented in a clean, organized format.
- Code is appropriately encapsulated.
- The code meets the specifications defined below.

There are no provided unit tests. You'll also write tests for the methods you write.

Step Four: Create the `Pet` class

Properties

Property	Data Type	Get	Set	Description
name	String	X	X	Name of pet.
species	String	X	X	Species of pet, like dog or cat.
vaccinations	List	X	X	Vaccinations the pet has received.

> Note: Remember to set `vaccinations` to a new initialized `ArrayList`. You can do this in the property declaration or constructor.

Constructors

Create one constructor for `Pet` that accepts two `String`s to set `name` and `species`.

Methods

Method Name	Return Type	Parameters
listVaccinations	String	none

The `listVaccinations` method returns the elements of `vaccinations` as a comma-delimited string. For example, if the `List` contains `["Rabies", "Distemper", "Parvo"]`, the output must be `"Rabies, Distemper, Parvo"`.

Keep in mind the spaces between and not to have a trailing comma.

Unit tests

Create a `PetTests` class in the `com.techelevator.crm` package. Create a test for `listVaccinations`.

Step Five: Create the `Customer` class

Declare a `Customer` class that inherits the `Person` class.

Properties

Property	Data Type	Get	Set	Description

Property	Data Type	Get	Set	Description
phoneNumber	String	X	X	Customer's phone number.
pets	List	X	X	Collection of customer's pets.

> Note: Remember to set `Pets` to a new initialized `ArrayList`. You can do this in the property declaration or constructors.

Constructors

`Customer` needs two constructors:

- One that accepts three `String` parameters for first name, last name, and phone number.
  - This constructor must set the phone number property, and call the base class constructor for first and last name.
- One that accepts two `String` parameters for first name and last name.
  - This constructor must call the above constructor with an empty string for phone number.

Step Six: Implement the `Billable` interface

You received an additional requirement to implement the `Billable` interface on the `Customer` class and the `Employee` class because employees can also be customers.

The `Billable` interface defines a method with the signature `double getBalanceDue(Map<String, Double>)`. You need to implement this method in the `Customer` and `Employee` classes.

Methods

Method Name	Return Type	Parameters
getBalanceDue	double	Map<String, Double>

The `getBalanceDue` method returns the total amount the customer owes.

It accepts one parameter, a `Map` of services rendered:

- The key is a `String` representing the type of service—for example, "Grooming", "Walking", or "Sitting."
- The value is a `Double` representing the price for each service.

Employees receive a 50% discount on walking services, but the discount isn't applied in the `Map` provided. In the `Employee` implementation of the method, you'll have to calculate the discount.

Unit tests

Create a `CustomerTests` class in the `com.techelevator.crm` package. Create a test for `getBalanceDue`.

You'll also need to add a test for `getBalanceDue` in the `EmployeeTests` class. Keep in mind the discount.