Practices for Lesson 2

Practices Overview

In these practices, you will extend your existing Employee class to create new classes for Engineers, Admins, Managers, and Directors.

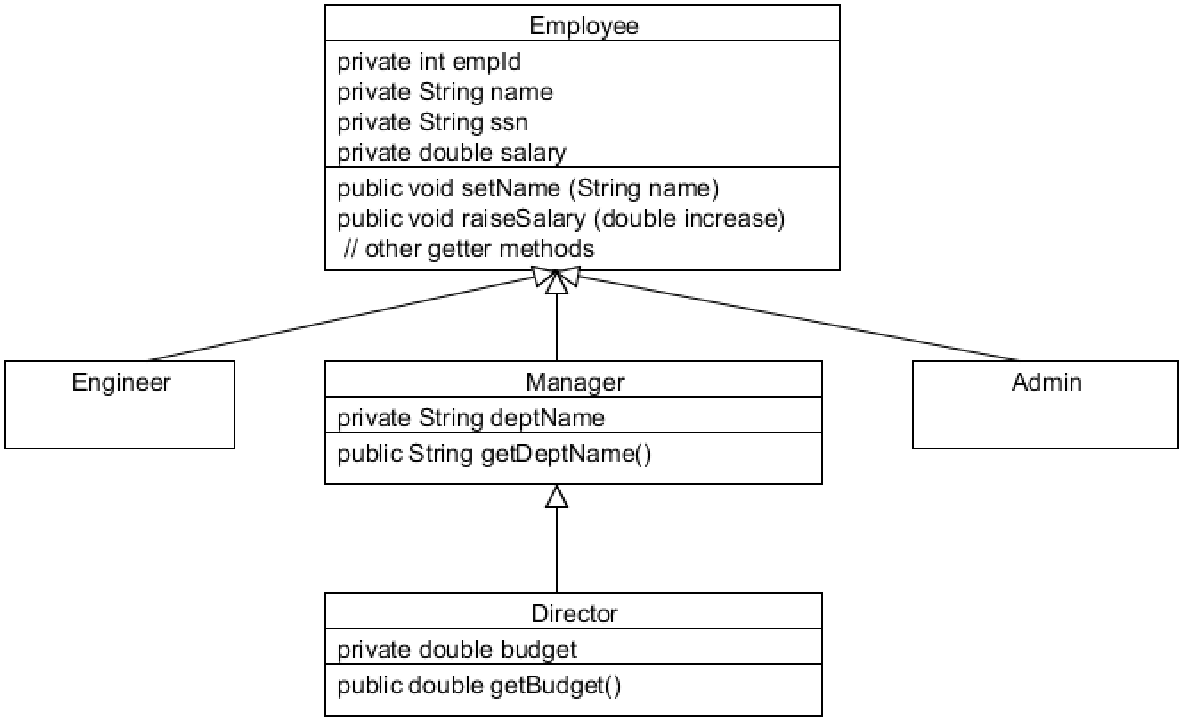
Practice 2-1: Summary Level: Creating Subclasses

Overview

In this practice, you will create subclasses of Employee, including Manager, Engineer, and Administrative assistant (Admin). You will create a subclass of Manager called Director, and create a test class with a `main` method to test your new classes.

Assumptions

Use this Java class diagram to help guide this practice.



Tasks

1. Open the project `EmployeePractice` in the practices directory.
2. Apply encapsulation to the `Employee` class.
    a. Make the fields of the `Employee` class private.
    b. Replace the no-arg constructor in Employee with a constructor that takes `empId`, `name`, `ssn`, and `salary`.
    c. Remove all the setter methods except `setName`.
    d. Add a method named `raiseSalary` with a parameter of type `double` called `increase` to increment the salary.
    e. Save `Employee.java`.

3. Create a subclass of `Employee` called `Manager` in the same package.
    a. Add a private String field to store the department name in a field called `deptName`.
    b. Create a constructor that includes all the parameters needed for Employee and `deptName`.
    c. Add a getter method for `deptName`.
4. Create subclasses of `Employee`: `Engineer` and `Admin` in the `com.example.domain` package. These do not need fields or methods at this time.
5. Create a subclass of `Manager` called `Director` in the `com.example.domain` package.
    a. Add a private field to store a double value `budget`.
    b. Create a constructor for Director that includes the parameters needed for Manager and the `budget` parameter.
    c. Create a getter method for this field.
6. Save all the classes.
7. Test your subclasses by modifying the `EmployeeTest` class. Have your code do the following:
    a. Remove the code that creates an instance of the “Jane Smith” Employee.
    b. Create an instance of an `Engineer` with the following information:

Field	Choices or Values
ID	101
Name	Jane Smith
SSN	012-34-5678
Salary	120_345.27

You will likely see an error beside the line that you added to create an Engineer. This is because NetBeans cannot resolve Engineer using the existing import statements in the class. The quick way to fix import statements is to allow NetBeans to fill them in for you. Right-click in the class and select Fix Imports, or press the Ctrl + Shift + I key combination. NetBeans will automatically add the import statement for Engineer in the appropriate place in the class and the error will disappear.

    c. Create an instance of a `Manager` with the following information:

Field	Choices or Values
ID	207
Name	Barbara Johnson
SSN	054-12-2367
Salary	109_501.36
Department	US Marketing

    d.

Create an instance of an `Admin` with the following information:

Field	Choices or Values
ID	304
Name	Bill Monroe
SSN	108-23-6509
Salary	75_002.34

e. Create an instance of a `Director`:

Field	Choices or Values
ID	12
Name	Susan Wheeler
SSN	099-45-2340
Salary	120_567.36
Department	Global Marketing
Budget	1_000_000.00

f. Save `EmployeeTest` and correct any syntax errors.

8. Add a `printEmployee` method to `EmployeeTest` to print out a formatted Employee object (its data fields). The `printEmployee` method should take an Employee object as a parameter.
9. Use the `printEmployee` method to print out information about each of your Employee objects.
10. (Optional) Use the `raiseSalary` and `setName` methods on some of your objects to make sure that those methods work.
11. Save the `EmployeeTest` class and test your work.
12. (Optional) Improve the look of the salary print output using the `NumberFormat` class.
    a. Use the following code to get an instance of a static `java.text.NumberFormat` class that you can use to format the salary to look like a standard US dollar currency:

```
NumberFormat.getCurrencyInstance().format(emp.getSalary())
```

In the lesson on abstract classes, you will see how to use an abstract factory, such as `NumberFormat.getCurrencyInstance()`.

13. (Optional) Add additional business logic (data validation) to your `Employee` class.
    a. Prevent a negative value for the `raiseSalary` method.
    b. Prevent a null or empty value for the `setName` method.

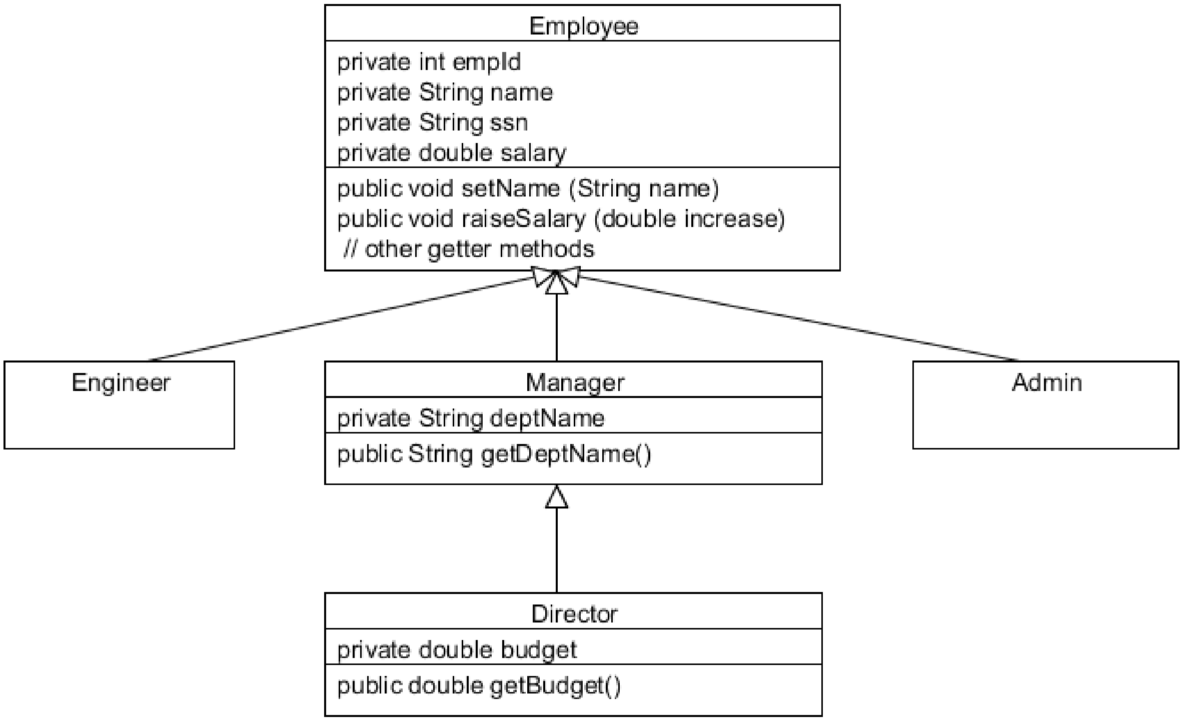
Practice 3-1: Detailed Level: Creating Subclasses

Overview

In this practice, you will create subclasses of Employee, including Manager, Engineer, and Administrative assistant (Admin). You will create a subclass of Manager called Director, and create a test class with a `main` method to test your new classes

Assumptions

Use this Java class diagram to help guide this practice.



Tasks

1. Open the project `EmployeePractice` in the practices directory.
    a. Select File > Open Project
    b. Browse to `D:\labs\03-Encapsulation\practices.`
    c. Select `EmployeePractice`.
    d. Click Open Project.
2. Apply encapsulation to the `Employee` class.
    a. Open the `Employee` class in the editor.
    b. Make the fields of the `Employee` class private.

c. Replace the no-arg constructor in Employee with a constructor that takes `empId`, `name`, `ssn`, and `salary`.

```
public Employee(int empId, String name, String ssn, double
salary) {
    this.empId = empId;
    this.name = name;
    this.ssn = ssn;
    this.salary = salary;
}
```

d. Remove all the setter methods except `setName`.

e. Add a method named `raiseSalary` with a parameter of type `double` named `increase` to increment the salary

```
public void raiseSalary(double increase) {
    salary += increase;
}
```

f. Save `Employee.java`

3. Create a subclass of `Employee` called `Manager`.

a. Right-click the package `com.example.domain` and select New > Java Class.

b. Enter the class name `Manager` and click Finish.

c. Modify the class to subclass `Employee`.

Note that the class declaration now has an error mark on it from Netbeans. Recall that constructors are not inherited from the parent class, so you will need to add a constructor that sets the value of the fields inherited from the parent class. The easiest way to do this is to write a constructor that calls the parent constructor using the `super` keyword.

d. Add a private String field to store the department name in a field called `deptName`.

e. Add a constructor that takes `empId`, `name`, `ssn`, `salary`, and a `deptName` of type `String`. The `Manager` constructor should call the `Employee` constructor with the `super` keyword, and then set the value of `deptName`.

```
public Manager(int empId, String name, String ssn, double
salary, String deptName) {
    super (empId, name, ssn, salary);
    this.deptName = deptName;
}
```

f. Add a getter method for `deptName`.

g. Save the Manager class.

4. Create two subclasses of `Employee`: `Engineer` and `Admin` in the `com.example.domain` package. These do not need fields or methods at this time.

a. Because Engineers and Admins are Employees, add a constructor for each of these classes that will construct the class as an instance of an Employee.
Hint: Use the `super` keyword as you did in the Manager class.

b. Save the classes.

5. Create a subclass of `Manager` called `Director` in the `com.example.domain` package.
    a. Add a private field to store a `double` value `budget`.
    b. Add the appropriate constructors for `Director`. Use the `super` keyword to construct a `Manager` instance and set the value of `budget`.
    c. Create a getter method for `budget`.
6. Save the class.
7. Test your subclasses by modifying the `EmployeeTest` class. Have your code do the following:
    a. Remove the code that creates an instance of the “Jane Smith” Employee.
    b. Create an instance of an `Engineer` with the following information:

Field	Choices or Values
ID	101
Name	Jane Smith
SSN	012-34-5678
Salary	120_345.27

You will likely see an error beside the line that you added to create an Engineer. This is because NetBeans cannot resolve Engineer using the existing import statements in the class. The quick way to fix import statements is to allow NetBeans to fill them in for you. Right-click in the class and select Fix Imports, or press the Ctrl + Shift + I key combination. NetBeans will automatically add the import statement for Engineer in the appropriate place in the class and the error will disappear.

c. Create an instance of a `Manager` with the following information:

Field	Choices or Values
ID	207
Name	Barbara Johnson
SSN	054-12-2367
Salary	109_501.36
Department	US Marketing

d. Create an instance of an `Admin` with the following information:

Field	Choices or Values
ID	304
Name	Bill Monroe
SSN	108-23-6509
Salary	75_002.34

e.

Create an instance of a `Director`:

Field	Choices or Values
ID	12
Name	Susan Wheeler
SSN	099-45-2340
Salary	120_567.36
Department	Global Marketing
Budget	1_000_000.00

f. Save `EmployeeTest` and correct any syntax errors.

8. Add a `printEmployee` method to `EmployeeTest`.

   a. Adding `System.out.println` methods after each of the instances you created is going to create a lot of redundant code. Instead, you will use a method that takes an `Employee` object as the parameter:

```
public static void printEmployee (Employee emp) {
    System.out.println(); // Print a blank line as a separator
    // Print out the data in this Employee object
    System.out.println ("Employee id:          " + emp.getEmpId());
    System.out.println ("Employee name:        " + emp.getName());
    System.out.println ("Employee Soc Sec #:   " + emp.getSsn());
    System.out.println ("Employee salary:      " + emp.getSalary());
}
```

   Note that all the object instances that you are creating are `Employee` objects, so regardless of which subclass you create, the `printEmployee` method will work. However, the `Employee` class cannot know about the specialization of its subclasses. You will see how to work around this in the next lesson.

9. Use the `printEmployee` method to print out information about your classes. For example:

```
printEmployee(eng);
printEmployee(man);
printEmployee(adm);
printEmployee(dir);
```

10. (Optional) Use the `raiseSalary` and `setName` methods on some of your objects to make sure those methods work. For example:

```
mgr.setName ("Barbara Johnson-Smythe");
mgr.raiseSalary(10_000.00);
printEmployee(mgr);
```

11. Save the `EmployeeTest` class and test your work.

12. (Optional) Improve the look of the salary print output using the `NumberFormat` class.
    a. Use the following code to get an instance of a static `java.text.NumberFormat` class that you can use to format the salary to look like a standard U.S. dollar currency. Replace the `emp.getSalary()` with the following:

       ```
       NumberFormat.getCurrencyInstance().format(emp.getSalary())
       ```

    In the lesson on abstract classes, you will see how to use an abstract factory, such as `NumberFormat.getCurrencyInstance()`.

13. (Optional) Add additional business logic (data validation) to your `Employee` class.
    a. Prevent a negative value for the `raiseSalary` method.
    b. Prevent a null or empty value for the `setName` method.

(Optional) Practice 2-2: Adding a Staff to a Manager

Overview

In this practice you modify the Manager class to add an array of Employee objects (a staff) to the manager class, and create methods to add and remove employees from the Manager. Finally, add a method to Manager to print the staff names and IDs.

Assumptions

Start with the completed project from Practice 2-1 (Summary or Detailed)

Tasks

1. Add fields to the `Manager` class to keep the employee objects.
   a. Declare a private field called `staff` that is declared as an array of `Employee` objects .
   b. You will need to keep track of the number of employees in `staff`, so create a private integer field `employeeCount` to keep a count of the number of employees. Initialize the employee count with 0.
   c. In the constructor, initialize the `staff` array with a maximum of 20 employees.
2. Add a method called `findEmployee`. This method scans the current staff Employee array to see whether there is a match between the any member of staff and the Employee passed in.
   a. Return `-1` if there is no match, and the index number of the Employee if there is a match.
3. Add a method called `addEmployee`. This method adds the `Employee` passed in as a parameter to the end of the array.
   a. This method should return a `boolean` value and take an `Employee` object as a parameter. The method should return true if the employee was successfully added and false if the employee already exists as a member of staff.
   b. Call the `findEmployee` method to determine whether the Employee is a member of staff already. Return false if there is match.
   c. Add the employee object to the staff array. (Hint: Use the `employeeCount` as the index of the array element to assign the employee parameter to.)
   d. Increment the `employeeCount` and return true.
4. Add a method called `removeEmployee`. This method is a bit more complicated. When you remove an element from the array, you must shift the other elements of the array so there are no empty elements. The easiest way to do this is to create a new array and assign a copy of each of the staff elements to it except for the match. This effectively removes the match from the array.
   a. Declare a local `boolean` variable initialized to false to return as the status for the method.
   b. Declare a temporary array of Employee objects to copy the revised staff array to.
   c. Declare an integer counter of the number of employees copied to the temporary array.

   d. Use a for loop to go through the staff array and attempt to match the employee ID of each element of the staff array with the employee ID of the Employee passed into the method as a parameter.
   e. If the employee ID’s do not match, copy the employee reference from the staff array to the temporary array from step b and increment the count of employees in the temporary array.
   f. If there is a match, “skip” this employee by continuing to the next element in the staff array, and set the local `boolean` variable from step a to true.
   g. If there was a match (the local `boolean` is true), replace the current staff array with the temporary array, and the count of employees with the temporary counter from step c.
   h. Return the local `boolean` variable.

5. Add a method called `printStaffDetails`. This method prints the name of the manager and then each of the elements of staff in turn.