Practices for Lesson 3: Encapsulation and Subclassing

Chapter 3

Practices for Lesson 3: Overview

Practices Overview

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

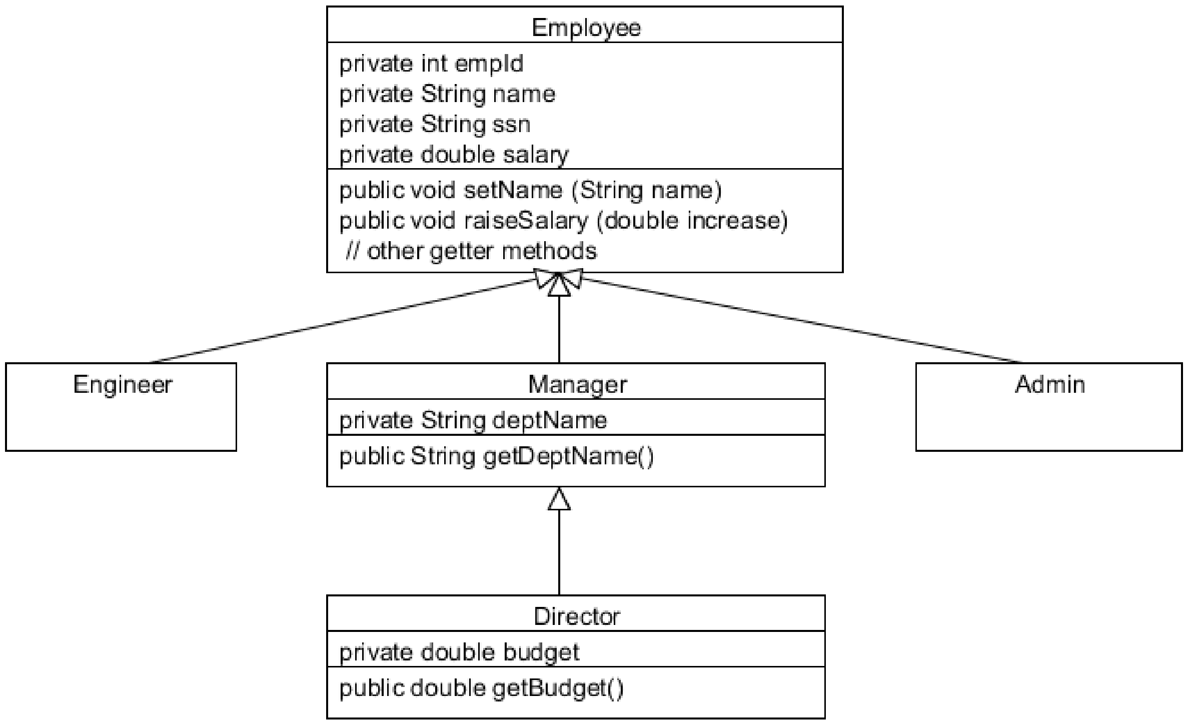
Practice 3-1: Summary Level: Creating Subclasses

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 `Employee03-01Prac` in the `practices/practice1` 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. Add a method named `printEmployee` to print the `Employee` object details.
   f. 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

   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 Munroe
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. Use the `printEmployee` method to print out information about each of your Employee objects.

g. (Optional) Use the `raiseSalary` and `setName` methods on some of your objects to make sure that those methods work.

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

8. (Optional) Improve the look of the salary print output using the `NumberFormat` class.
   a. In the `printEmployee()` method of `Employee.java`, 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((double)
getSalary()));
```

9. (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.

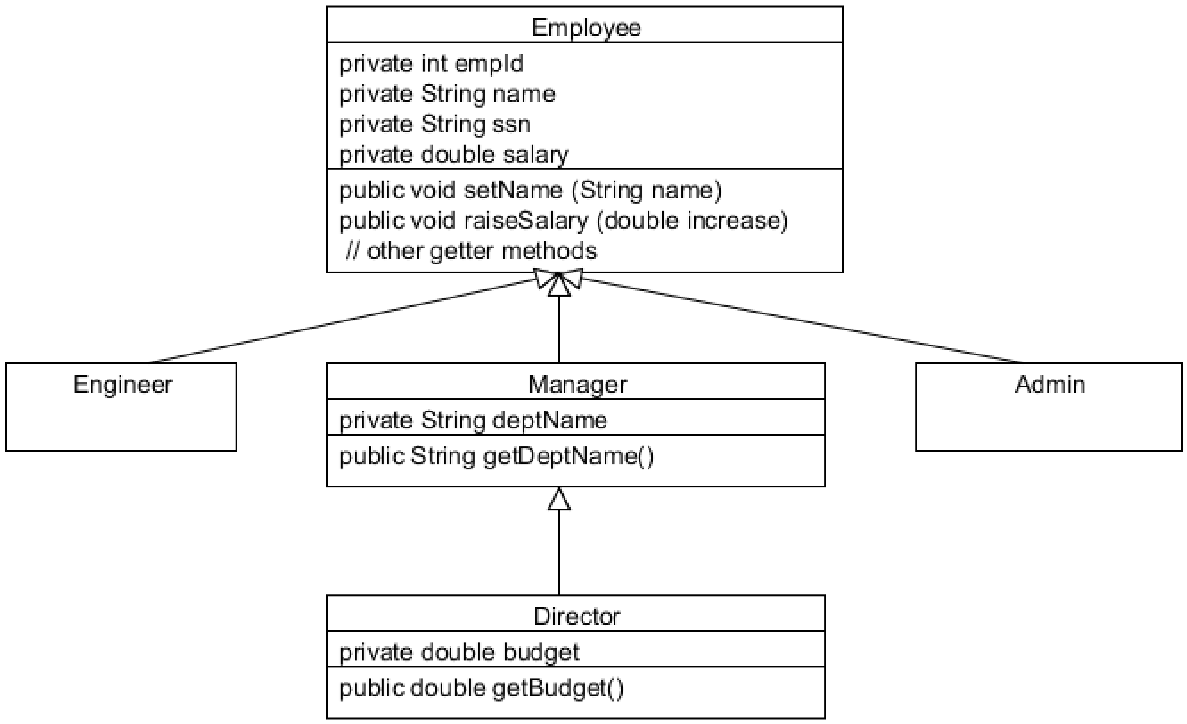
Practice 3-1: Detailed Level: Creating Subclasses

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. In NetBeans, open the project `Employee03-01Prac` from the `practices` directory.
   a. Select **File > Open Project**.
   b. Browse to `/home/oracle/labs/03-Encapsulation/practices/practice1`.
   c. Select `Employee03-01Prac`.
   d. Click Open Project.
2. Apply encapsulation to the `Employee` class.
   a. Open `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. Add a method named `printEmployee`.

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

g. Resolve any missing import statements.

h. 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. In the New Java Class window, perform the following steps:

1) Enter the class name as `Manager`.

2) Click Finish.

c. Modify the `Manager` 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.

1) Add a private String field called `deptName` to store the department name.

2) 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;
}
```

3) Add a getter method for `deptName`.

d. 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

   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 Munroe
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. Delete the `System.out.println` statements used to display the details of the `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());
```

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

```
eng.printEmployee();
adm.printEmployee();
mgr.printEmployee();
dir.printEmployee();
```

h. (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);
mgr.printEmployee();
```

8. Save the `EmployeeTest` class.
9. Test your work, run the `EmployeeTest` class.
10. (Optional) Improve the look of the salary print output by using the `NumberFormat` class.
    a. In the `printEmployee()` method of `Employee.java`, 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.

b. Replace `emp.getSalary()` by

```
NumberFormat.getCurrencyInstance().format((double)
getSalary()));
```

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