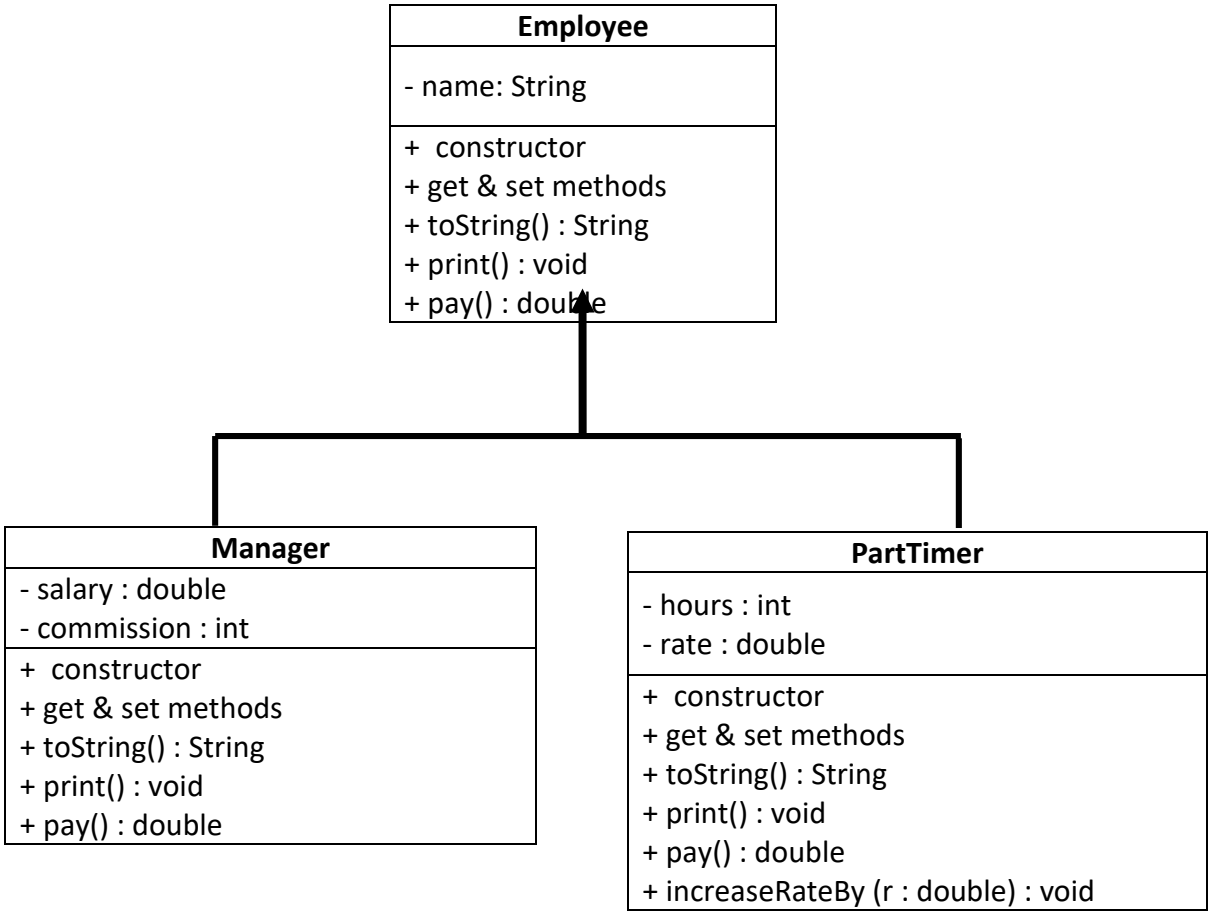
Java Exercises – Set 4
Inheritance and Composition

Exercise 1

Consider the inheritance hierarchy shown below in the UML class diagram (the class description is next):

UML : Unified Modeling Language



Write an inheritance hierarchy for the above classes.

1.1)

The **Employee** class is a superclass that has the following members:
Attributes (declared as private)*:*

- name: a String to store the name of the Employee.

Methods:

- Constructor
- get & set methods
- toString( ): a method that returns a description of the Employee.
- *pay*( ): a method that returns a double (it returns simply a Zero).

1.2) Inheritance

The **Manager** is a class derived from the Employee. A **Manager** is paid a fixed monthly salary plus a commission. It has the following:
Attributes (declared as private)*:*

- salary: a double containing the salary of the Manager.
- commission: an integer containing the *commission* granted to a Manager.

Methods:

- Constructor.
- get & set methods
- toString( ): a method that returns a description of the Manager.
- pay( ): a method that returns the salary as a double **plus** the *commission***.**

1.3) Inheritance

The **PartTimer** is a class derived from the Employee. A **PartTimer** is paid on an hourly basis. It has the following:
Attributes (declared as private)*:*

- hours: an integer containing the number of hours worked.
- rate: a double containing the hourly rate of the PartTimer.

Methods:

- Constructor.
- get & set methods
- toString( ): a method that returns a description of the PartTimer.
- pay( ): a method that returns the salary as a double
- increaseRateBy(): a method that takes a double and adds it to the hourly rate.

1.4)

Now, create a driver class called **Staff** whose **`main`** method should create the following:

a) Name: Maya Hours: 100 Rate: $20	b) Name: Tony Hours: 50 Rate: $30

The main method in the class **Staff** should perform the followings tasks:

a) Create an array of 2 **`PartTimer`** called **`PTList`**
b) Instantiate the array according to the above information
c) Print the information of all PartTimers and invoke (call) their **`pay`** methods to determine how much each PartTimers should be paid.
d) Increase the rate per hour for Maya to be $35 instead of $20.
e) Print the information of all PartTimers and invoke (call) their **`pay`** methods to determine how much each PartTimers should be paid.

1.5) Composition

A Company has a manager and a group of Part-timers. Create the class **Company** as the following:

Attributes (declared as private)*:*

- name: a string that represents the name of the company
- manager: an object of the class **Manager**
- partTimerList: an **`ArrayList`** of PartTimer's Objects

Methods:

- Constructors: default and with arguments (name and manager)
- get & set methods for name and manager
- toString( ): a method that returns a description of a company.
- print() : a method to print a description of a company
- AddPartTimer: a method that adds a new PartTimer to the partTimerList

Company
- name : String - manager : an object of the class **Manager** - partTimerList : an **`ArrayList`** of PartTimer's Objects
+ constructor + get & set methods + toString() : String + print() : void + AddPartTimer (p : PartTimer): void

1.6)

Now, create a second driver class called **CompanyTest** whose **`main`** method should do the following:

a) Create the company (C1) using the default constructor and print its information
b) Create the company (C2) using the 2nd constructor and print its information
c) Create the company (C3) using the 3rd constructor, add to it 2 part-timers and print its information
d) Create the company (C4) using the 2nd constructor, use the method **`AddPartTimer`** to add 2 part-timers and print its information
e) Replace part-timers in Company 4 by part-timers in Company 3
f) Print the information of part-timers in Company 4
g) Assign a name and a manager, and add a new PartTimer to Company 1 and print its information
h) Move part-timers in Company 4 to Company 1
i) Print the information of Company 1

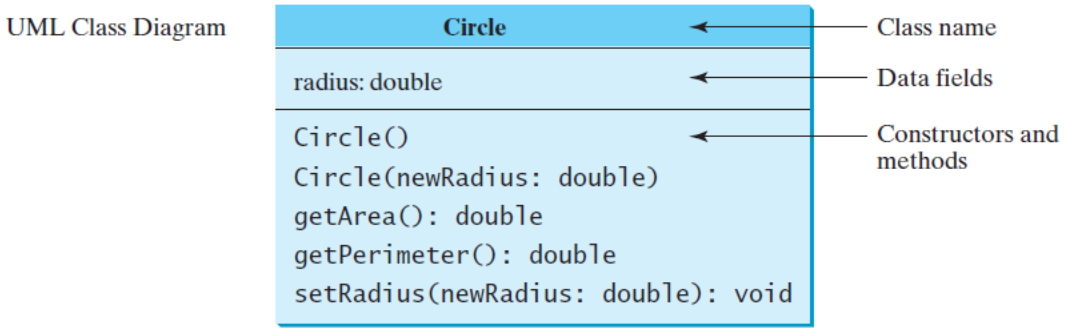
UML Notation

UML (Unified Modeling Language) is a standard language for specifying, visualizing, constructing, and documenting the artifacts of software systems

UML Class diagram

The UML Class diagram is a graphical notation used to construct and visualize object oriented systems. A class diagram in the Unified Modeling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's:

- classes,
- their attributes,
- operations (or methods),
- and the relationships among objects.



Class Visibility

The +, - and # symbols before an attribute and operation name in a class denote the visibility of the attribute and operation.

- `+ denotes public attributes or operations`
- `- denotes private attributes or operations`
- `# denotes protected attributes or operations`

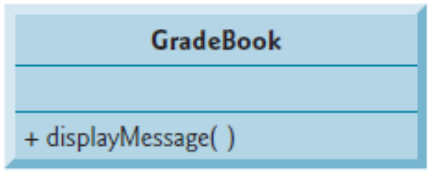


Fig. 3.3 | UML class diagram indicating that class `GradeBook` has a `public` `displayMessage` operation.

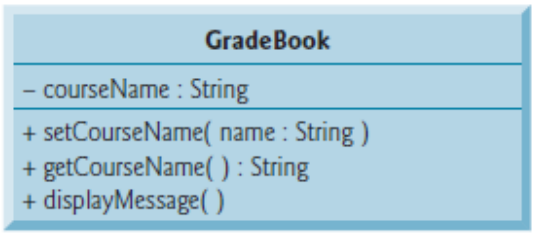


Fig. 3.9 | UML class diagram indicating that class `GradeBook` has a private `courseName` attribute of UML type `String` and three public operations—`setCourseName` (with a `name` parameter of UML type `String`), `getCourseName` (which returns UML type `String`) and `displayMessage`.

public	+
private	-
protected	#

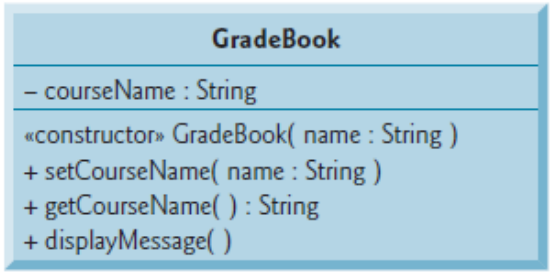


Fig. 3.12 | UML class diagram indicating that class `GradeBook` has a constructor that has a `name` parameter of UML type `String`.

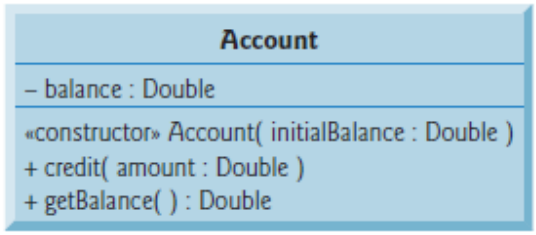


Fig. 3.15 | UML class diagram indicating that class `Account` has a `private balance` attribute of UML type `Double`, a constructor (with a parameter of UML type `Double`) and two `public` operations—`credit` (with an `amount` parameter of UML type `Double`) and `getBalance` (returns UML type `Double`).

For more information visit :

https://www.visual-paradigm.com/guide/uml-unified-modeling-language/uml-class-diagram-tutorial/