Midterm Exam, Faculty of Engineering, Chulalongkorn University
Course ID: 2110215 Course Name: Programming Methodology I
Second Semester, Date: 6 March 2020 Time: 8.30-11.30AM

Name .... Student ID. .... No. in CR ....

Instructions

1. Write your Student ID, full name, and your number in CR58 in the space provided on this page.
2. Your answer must be on the computer center's machine in front of you.
3. Documents and files are allowed inside the exam room; however, internet and flash drive are prohibited. Borrowing is not allowed unless it is supervised by the proctor.
4. You must not carry mobile phone and flash drive during the exam.
5. *** You must not bring any part of this exam paper outside. The exam paper is a government's property. Violators will be prosecuted under a criminal court and will receive an F in the subject. ***
6. Students who wish to leave the exam room before the end of the exam period, must raise their hands and ask for permission before leaving the room. Students must leave the room in the orderly manner.
7. Once the time expires, student must stop typing and must remain seated quietly until the proctors collect all the exam papers or given exam booklets. Only then, the students will be allowed to leave the room in an orderly manner.
8. Any student who does not obey the regulations listed above will receive punishment under the Faculty of Engineering Official Announcement on January 6, 2003 regarding the exam regulations.
    a. With implicit evidence or showing intention for cheating, student will receive an F in that subject and will receive an academic suspension for 1 semester.
    b. With explicit evidence for cheating, student will receive an F in that subject and will receive an academic suspension for 1 year.

Please sign and submit

Signature (...)

Important Rules

- You must not bring any part of this exam paper outside. The exam paper is a government's property. Violators will be prosecuted under a criminal court and will receive an F in the subject.
- It is a student's responsibility to check the file. If it is corrupted or cannot be open, there is no score.
- For each question, a table is given, showing (color-coded) whether or not you have to modify or create each method or variable.

** Noted that Access Modifier Notations can be listed below*

+ (public)
# (protected)
- (private)
Underline (static)
Italic (abstract)

Set-Up Instruction

- Set workspace to "C:\temp\progmeth2019_2\Midterm_2110215_(your id)_(FirstName)" (if not exist, you must create it)
    - For example, "C:\temp\progmeth2019_2\Midterm_2110215_6170156021_Boss"
- All your files must be in the workspace.
- The code outside of the workspace will not be collected, or graded

Scoring (Total 30 points, will be scaled to 40 points)

- Part1 = 10 points
- Part2 = 10 points
- Part3 = 10 points

Part 1: OOP Concept

Objective

1) Be able to implement Objects and Classes.

Instruction

1) Create Java Project named “**2110215_Midterm_Part1**”.
2) Copy all folders in “**toStudent/Part1**” to your project directory src folder.
3) You are to implement the following classes (detail for each class is given in section 3 and 4)

a) **CPShop**	(package application)
b) **Item**	(package logic)
c) **OrderItem**	(package logic)
d) **Order**	(package logic)

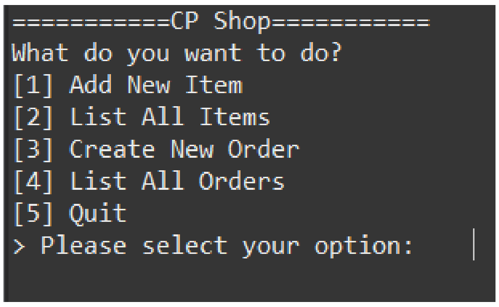
4) **JUnit for testing is in package test.grader**

Problem Statement: CP Shop

CP Shop is a JAVA application for the shop manager to add items and create orders. The order contains item orders that tell which item the order has and how much the amount is. However, the current system's implementation is still incomplete. You are tasked to implement the remaining Item, Order and OrderItem classes.

Here's is how the Shop should work, part of this is already provided.

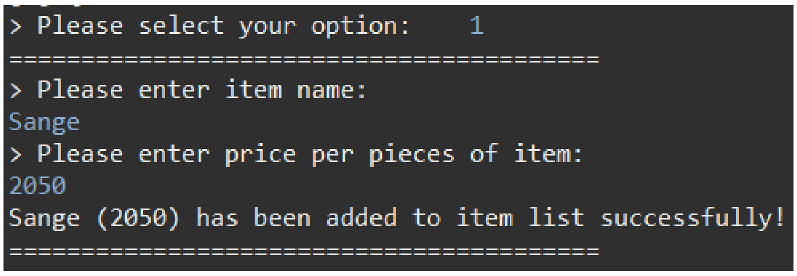
When the application is open, there will be a welcome message with all the available commands for the user.



```
===========CP Shop===========
What do you want to do?
[1] Add New Item
[2] List All Items
[3] Create New Order
[4] List All Orders
[5] Quit
> Please select your option:
```

Figure 1. Welcome screen when the application starts

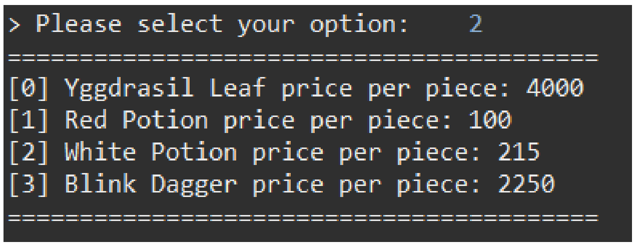
1) The first option is adding a new item to the system. Please note that the new item name has to be different from existed items and not blank. Otherwise, the system will raise an error. If price per piece entered is less than 1, the system will set the item price to 1.



```
> Please select your option:    1
==========================================
> Please enter item name:
Sange
> Please enter price per pieces of item:
2050
Sange (2050) has been added to item list successfully!
==========================================
```

Figure 2. Successfully adding a new item

2) The user can choose to list all current items in the system. Each item information displays the item name and its price per piece. When the program starts, there will already have 4 initial items in the system.

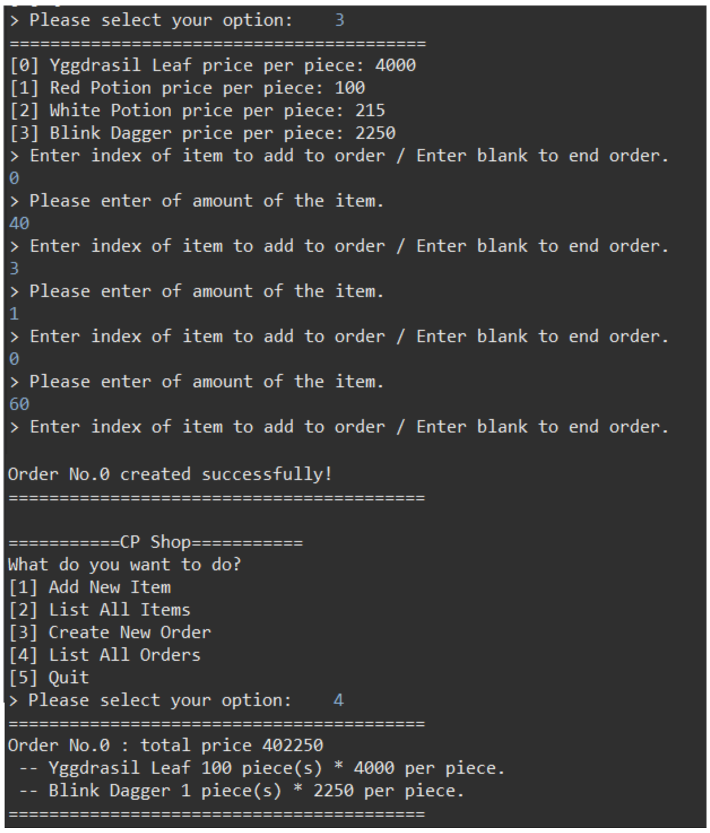


```
> Please select your option:    2
==========================================
[0] Yggdrasil Leaf price per piece: 4000
[1] Red Potion price per piece: 100
[2] White Potion price per piece: 215
[3] Blink Dagger price per piece: 2250
==========================================
```

Figure 3. Initial Item Listing

3) Third option is creating a new order. The user will then be asked to select item by enter its index and then enter the amount to add to the order. The user can finish creating order by pressing Enter key.

If the user select to add item that already in the order, the amount of that item in the order will be increased.



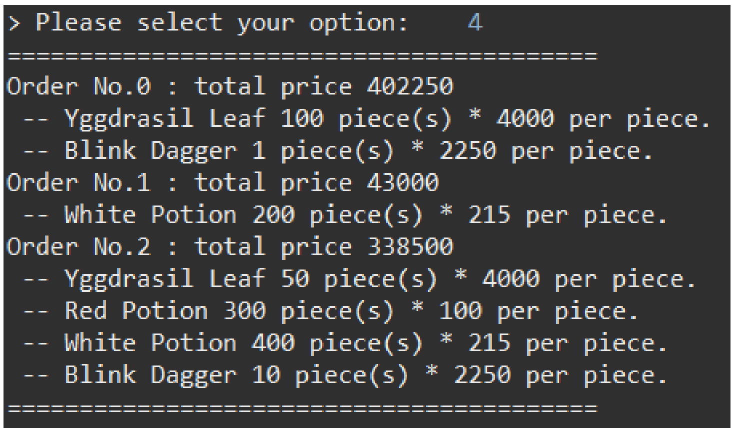
```
> Please select your option:    3
========================================
[0] Yggdrasil Leaf price per piece: 4000
[1] Red Potion price per piece: 100
[2] White Potion price per piece: 215
[3] Blink Dagger price per piece: 2250
> Enter index of item to add to order / Enter blank to end order.
0
> Please enter of amount of the item.
40
> Enter index of item to add to order / Enter blank to end order.
3
> Please enter of amount of the item.
1
> Enter index of item to add to order / Enter blank to end order.
0
> Please enter of amount of the item.
60
> Enter index of item to add to order / Enter blank to end order.

Order No.0 created successfully!
========================================

===========CP Shop===========
What do you want to do?
[1] Add New Item
[2] List All Items
[3] Create New Order
[4] List All Orders
[5] Quit
> Please select your option:    4
========================================
Order No.0 : total price 402250
 -- Yggdrasil Leaf 100 piece(s) * 4000 per piece.
 -- Blink Dagger 1 piece(s) * 2250 per piece.
========================================
```

Figure 4. Creating a new order.

4) The user can list out all order in the system.



```
> Please select your option:    4
=========================================
Order No.0 : total price 402250
 -- Yggdrasil Leaf 100 piece(s) * 4000 per piece.
 -- Blink Dagger 1 piece(s) * 2250 per piece.
Order No.1 : total price 43000
 -- White Potion 200 piece(s) * 215 per piece.
Order No.2 : total price 338500
 -- Yggdrasil Leaf 50 piece(s) * 4000 per piece.
 -- Red Potion 300 piece(s) * 100 per piece.
 -- White Potion 400 piece(s) * 215 per piece.
 -- Blink Dagger 10 piece(s) * 2250 per piece.
=========================================
```

Figure 5. Listing all orders.

Implementation Detail

The class package is summarized below.

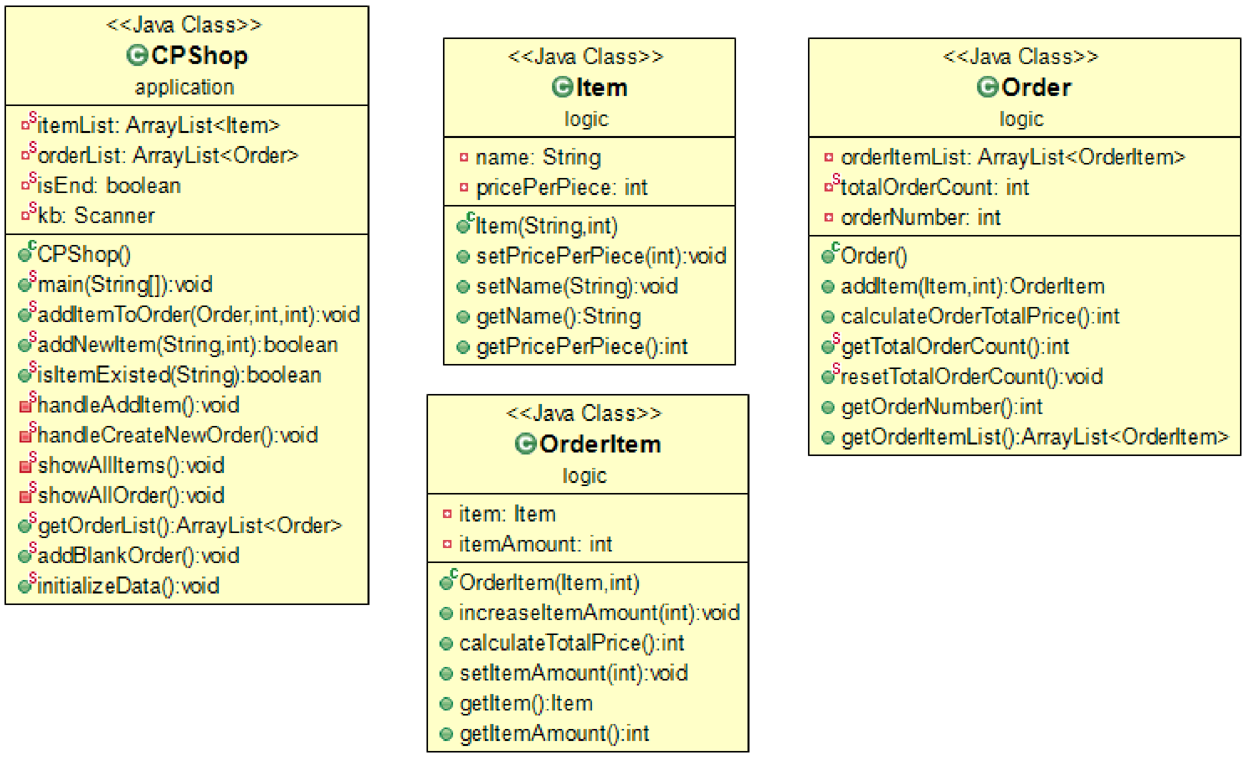


Figure 6. Class Diagram

You must write java classes using UML diagram specified above.

* In the following class description, only details of IMPORTANT fields and methods are given. *

4.1 Package logic

4.1.1 Class Item: This class represents the item.

You must implement this class from scratch.

Field

Name	Description
- String name	The name of the item.
- int pricePerPiece	Price per piece of the item.

Constructor

Name	Description
+ Item(String name, int pricePerPiece)	Create a new Item object with specified name and price per piece. Set related fields with the given parameters; Price per piece must be equals or more than 1.

Method

Name	Description
+ void setPricePerPiece(int pricePerPiece)	Set price per piece of the item. If given price per piece is less than 1, set price per piece to 1.
+ getter/setter for other variables	

4.1.2 Class OrderItem: This class is used for storing ordered Item and amount of the ordered item in the Order made through the application menu.

You must implement this class from scratch.

Field

Name	Description
- Item item	Ordered item of this OrderItem.
- int itemAmount	Amount of ordered item

Constructor

Name	Description
+ OrderItem(Item item, int itemAmount)	Create a new OrderItem object with specified item and itemAmount. Set related fields with the given parameters.

Method

Name	Description
+ void increaseItemAmount(int amount)	Increase the itemAmount by given amount. If given amount is negative values, the itemAmount should not be changed.
+ int calculateTotalPrice()	Calculate the total price of this OrderItem and return it.
+ void setItemAmount(int itemAmount)	Set itemAmount of the OrderItem by given itemAmount. If given itemAmount is negative values, set itemAmount to 0.
+ getter for each variable	

4.1.3 Class Order: This class represents a single order that is made from the application menu. An order can contain more than one item (each item has the amount to order). It is partially provided. Please fill necessary code.

/* PARTIALLY PROVIDED */

Field

Name	Description
- ArrayList<OrderItem> orderItemList	ArrayList of OrderItem.
<u>- int totalOrderCount</u>	Total number of orders created in the application so far. This will be used to assign order number to a new Order.
- int orderNumber	Number of this order.

Constructor

Name	Description
+ Order()	/* FILL CODE */ Create a new order object. Initialize variables, assign the orderNumber to totalOrderCount and then increase the totalOrderCount by 1.

Method

Name	Description
+ OrderItem addItem(Item item, int amount)	/* FILL CODE */ Adding item to this order by creating new OrderItem with given amount. If an OrderItem with the same item already existed, <u>increase the ItemAmount in the OrderItem</u> by a given amount instead.
+ int calculateOrderTotalPrice()	/* FILL CODE */ Calculate total price of the order by summing total price of each orderItem in orderItemList
+ <u>void resetTotalOrderCount()</u>	Reset totalOrderCount to 0.

+ <u>int getTotalOrderCount()</u>	Return totalOrderCount
+ other remaining getters	

4.2 Package application

4.2.1 Class CPShop: This class represents the shop. It also contains main method. This class is partially provided. You only need to fill the code where necessary.

/* PARTIALLY PROVIDED */

Field

Name	Description
- ArrayList<Item> itemList	ArrayList of item containing all items
- ArrayList<Order> orderList	ArrayList of order containing all orders

Method

Name	Description
+ void main()	Handle CPShop process.
+ addItemToOrder(Order order, int itemIndex, int amount)	/* FILL CODE */ Add item at given itemIndex from itemList to the order with given amount.

Part 2: Inheritance

1. Objective

1) Be able to understand and utilize a concept of inheritance and polymorphism in Object-Oriented Programming (OOP).

2. Instruction

1) Create Java Project named **"2110215_Midterm_Part2"**.
2) Copy all folders in **"toStudent/Part2"** to your project directory src folder.
3) You are to implement the following classes (detail for each class is given in section 3 and 4)

a)	**NormalRock**	(package logic.rocks)
b)	**PoisonRock**	(package logic.rocks)
c)	**GameManager**	(package logic.rocks)

4) **JUnit for testing is in package test.grader**

3. Problem Statement: Terry VS. Zombies

You are developing a game called Terry VS. Zombies, a fangame based on Terraria and Plants VS. Zombies. You play an apocalyptic survivor named Terry who can pull rocks from his pouch of infinite rocks and throw them at the zombies. The rocks have different properties, such as being able to pierce through armor or being able to deal poison damage.

4. Implementation Detail

The class package is summarized below.

* In the following class description, only details of IMPORTANT fields and methods are given. *

4.1 Package logic.rocks

4.1.1 Class NormalRock /* You must implement this class from scratch */

Field

Name	Description
# int damage	How much damage this rock will deal.

Constructor

Name	Description
+ NormalRock(int damage)	Constructor method. Set the rock's damage here.

Method

Name	Description
+ void setDamage(int damage)	Set the damage. If damage is less than 0, it is set to 0.
+ int getDamage()	Returns the damage.
+ int dealDamage(Zombie zombie)	If the zombie's defense is greater than or equal to the rock's damage, decrease 0 health from the given zombie. Otherwise, get the zombie's defense, subtract it from this rock's damage, and decrease that much health from the given zombie. Finally, return the damage dealt after accounting for the zombie's defense.
+ String toString()	Prints the object as a string. Use this format: `"Normal Rock (" + getDamage() + ")"`

4.1.2 Class PoisonRock /* You must implement this class from scratch */

Note that this class is a type of rock.

Additional Fields

Name	Description
- int damageOverTime	How much decay will be added to the targeted zombie when this rock hits. (Decay will deal damage (ignoring defense) whenever a zombie takes its turn. If a zombie dies by decay, it will not take an action.)

Constructor

Name	Description
+ PoisonRock(int damage, int, damageOverTime)	Constructor method. Set the rock's damage and damageOverTime here.

Method

Name	Description
+ void setDamageOverTime(int)	Set the damageOverTime. If damageOverTime is less than 0, set it to 0.
+ int getDamageOverTime()	Returns the damageOverTime.
+ int dealDamage(Zombie)	Adds this rock's damageOverTime to the zombie's decay, then deal the damage and return the damage value like in NormalRock. **(Note: Do not use damageOverTime to reduce the zombie's health here.)**
+ String toString()	Prints the object as a string. Use this format: `"Poison Rock (" + getDamage() +", DoT = " +getDamageOverTime() +")"`

4.1.3 Class RockManager

This class is given. It creates random rock used in the game.

Method

Name	Description
<u>+ NormalRock randomRock()</u>	Returns a random rock with random parameters. The rock returned here can either be a NormalRock, a PierceRock, or a PoisonRock.

4.2. Package logic.game

4.2.1. Class GameManager /* PARTIALLY PROVIDED */

Additional Fields

Name	Description
<u>+ Player player</u>	The player object.
<u>+ ArrayList<Zombie> zombies</u>	The list containing the zombies the player is currently fighting.
<u>+ NormalRock currentRock</u>	The rock that will be used to attack when attackZombie(Zombie zombie) is called.
<u>+ NormalRock nextRock</u>	The rock that will replace currentRock once currentRock is used.

Method

Name	Description
<u>+ int attackZombie (Zombie zombie)</u>	/* FILL CODE */ Attacks the given zombie with the current rock.

4.3. Package logic. zombies (Do not modify any class in this package.)

4.3.1. Class Zombie

Fields

Name	Description
- int health	The zombie's current health. If it reaches 0, the zombie is dead and cannot take action.

- int maxHealth	The zombie's max health.
- int power	The zombie's power. When the zombie takes action, the zombie will deal this much damage to the player.
- int defense	The zombie's defense. When attacked, the zombie takes this much less damage unless attacked by a PierceRock, which ignores this defense.
- int decay	The zombie's decay. When the zombie takes action, the zombie will lose this much health (ignoring defense). If the zombie's health reaches 0 this way, they will not take action.

Method

Name	Description
Public setters for all above	Note that if a value less than 0 is given, the value is set as 0.
Public getters for all variables above	

Part 3: Abstract Classes

1. Objective

1) Be able to implement abstract classes and use abstract methods.

2. Instruction

1) Create Java Project named "**2110215_Midterm_Part3**".
2) Copy all folders in "**toStudent/Part3**" to your project directory src folder.
3) You are to implement the following classes (detail for each class is given in section 3 and 4)

   a) **Employee** (package logic)
   b) **OfficeWorker** (package logic)
   c) **Janitor** (package logic)
   d) **Database** (package logic)
4) Make UML class diagram for classes in package logic, using ObjectAid.
5) **JUnit for testing is in package test.grader**

3. Problem Statement: Macro Cosmos

Macro Cosmos

You are a part of World-Leader Mega-Corporate Company, Macro Cosmos. The company is developing its new Employee Management System. Unfortunately, the previous programmer went to Wuhan for vacation then unable to comeback until a few weeks later. However, the system needs to be finished by this week. So, you are tasked to complete the system.

4. Implementation Detail

The class package is summarized below.

* In the following class description, only details of IMPORTANT fields and methods are given. *

4.1 Package logic

4.1.1 class BackEndAPI

This class is the back-end API that the previous programmer has completed. You don't have to implement anything in this class. However, you might need to use the provided methods later.

Method

Name	Description
+ int calculateMonthlySalary(int baseSalary,int bonus, int day)	Calculate monthly salary of the employee.
+ getOfficeWorkerDescription(int id,String name,String department,int bonus)	Format the Description String of the Office Worker.
+ getJanitorDescription(int id,String name,String area,int bonus)	Format the Description String of the Janitor.

4.1.2 abstract class Employee /* You must implement this class from scratch */

This abstract class is a base class for all types of Employee. It contains the basic information and methods that all the Employee should have.

Field

Name	Description
# String name	Name of the employee
# int id	ID of the employee
# int baseSalary	Base salary of the employee per working hour
# int bonus	Bonus of the employee

Method

Name	Description
+ Employee(String name, int id,int baseSalary)	This is the Constructor. This sets all the fields with their respective value. (For bonus, set it to 0)
+ *int computeSalary()*	Compute monthly salary of the respective Employee. Exact implementations are different between each type of employees.
+ *String getDescription()*	Obtain the description of the respective Employee Exact details are different between each type of employees.
+ getter-setter for all fields.	For setBaseSalary and setBonus, if the value is below 0, set it to 0.

4.1.3. class OfficeWorker /* You must implement this class from scratch */

This class is a type of Employee who works in the office of each department of the company. Each employee starts with base salary of 30$ per hour and works 20 days each month.

Fields

Name	Description
- String department	Department of the OfficeWorker

Method

Name	Description
+ OfficeWorker(String name, int id,String department)	This is the Constructor. In addition to setting the information of the super class, it also sets the department, too. The salary of the Office Worker is 30$ per hour.

+ int computeSalary()	This method returns monthly salary of the Office Worker. You need to obtain the calculated value from a method in BackEndAPI. OfficeWorker works 20 days each month.
+ String getDescription()	This method returns the description of the Office Worker. The String format is provided in BackEndAPI.
getter/setter for department	

4.1.4. class Janitor /* You must implement this class from scratch */

This class is a type of Employee who cleans the floor of the respective area in the company. Each Janitor starts with base salary of 15$ per hour and works 30 days each month.

Fields

Name	Description
- String area	The Area where Janitor works at

Method

Name	Description
+ Janitor(String name, int id,String area)	This is the Constructor. In addition to setting the information of the super class, it also sets the area, too. The salary of the Janitor is 15$ per hour.
+ int computeSalary()	This method returns monthly salary of the Janitor. You need to obtain the calculated value from a method in BackEndAPI. Janitor works 30 days each month.

+ String getDescription()	This method returns the description of the Janitor. The String format is provided in BackEndAPI.
getter/setter for area	

4.1.5. class Database /* PARTIALLY PROVIDED */

This class is the employee database. It contains all employee information, as well as the all relevant methods.

Fields

Name	Description
- ArrayList<Employee> employees	ArrayList containing all employees of the company

Method

Name	Description
+ Database()	This is the Constructor. Initialize the employees Array List.
+ Employee getEmployeeById(int id)	This method returns an Employee from the ArrayList who has the matched id. If there is no match, returns null.
+ Employee getEmployeeByIndex(int index)	This method returns an Employee at the provided index.
+ boolean addEmployee(Employee e)	This method adds new Employee into the ArrayList. If the employee with the same ID already existed, then returns false (and do not add). Otherwise, add the employee and return true.
+ boolean removeEmployeeById(int id)	This method removes an Employee with the provided ID from the ArrayList. If the employee with the same ID does not exist, then do nothing and return false. Otherwise, remove the employee and return true.

+ ArrayList<String> getAllEmployeeDescriptions()	/* FILL CODES */ This method returns an ArrayList containing the description of all employees.
+ int calculateAllSalary()	/* FILL CODES */ This method returns the summation of the monthly salary from all employees.
+ int getTotalEmployeeCount()	This method returns the current employee counts in the database.

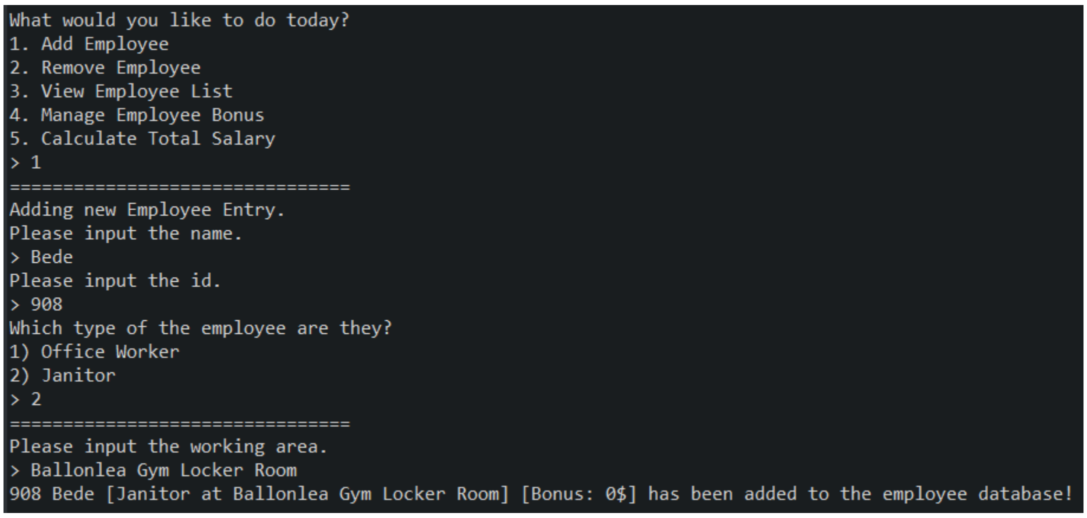
4.2 Package main

4.2.1 class Main

This class is the main program. You don't have to implement anything in this class. You can test the program by running this class.

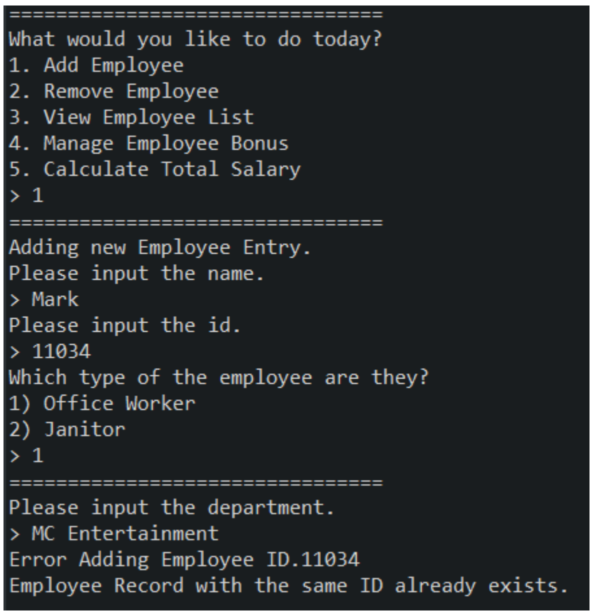
5. Finished Code Run Example

5.1.1. Adding an Employee



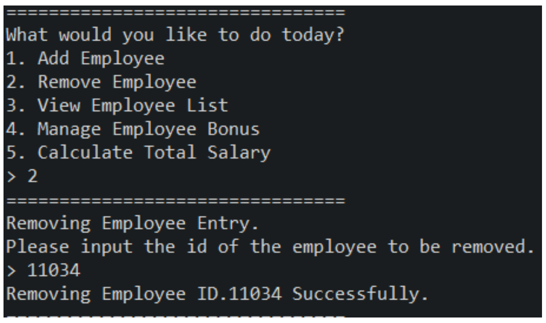
```
What would you like to do today?
1. Add Employee
2. Remove Employee
3. View Employee List
4. Manage Employee Bonus
5. Calculate Total Salary
> 1
================================
Adding new Employee Entry.
Please input the name.
> Bede
Please input the id.
> 908
Which type of the employee are they?
1) Office Worker
2) Janitor
> 2
================================
Please input the working area.
> Ballonlea Gym Locker Room
908 Bede [Janitor at Ballonlea Gym Locker Room] [Bonus: 0$] has been added to the employee database!
```

5.1.2. Adding an Employee with the same ID



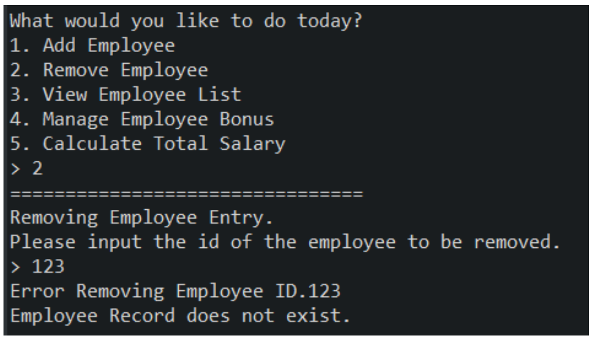
```
================================
What would you like to do today?
1. Add Employee
2. Remove Employee
3. View Employee List
4. Manage Employee Bonus
5. Calculate Total Salary
> 1
================================
Adding new Employee Entry.
Please input the name.
> Mark
Please input the id.
> 11034
Which type of the employee are they?
1) Office Worker
2) Janitor
> 1
================================
Please input the department.
> MC Entertainment
Error Adding Employee ID.11034
Employee Record with the same ID already exists.
```

5.2.1 Removing an Employee



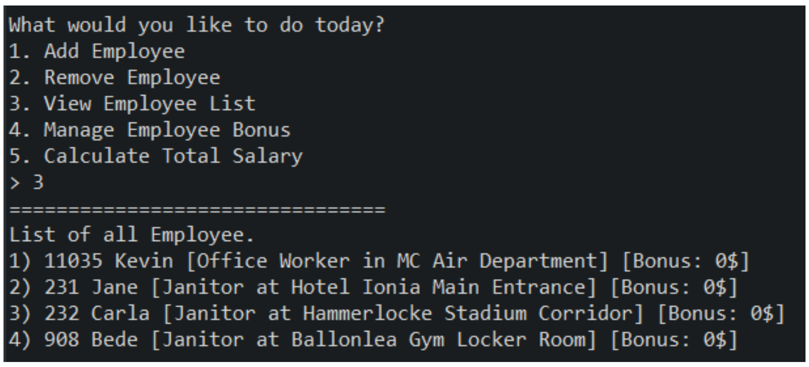
```
================================
What would you like to do today?
1. Add Employee
2. Remove Employee
3. View Employee List
4. Manage Employee Bonus
5. Calculate Total Salary
> 2
================================
Removing Employee Entry.
Please input the id of the employee to be removed.
> 11034
Removing Employee ID.11034 Successfully.
```

5.2.2 Removing an Employee that does not exist



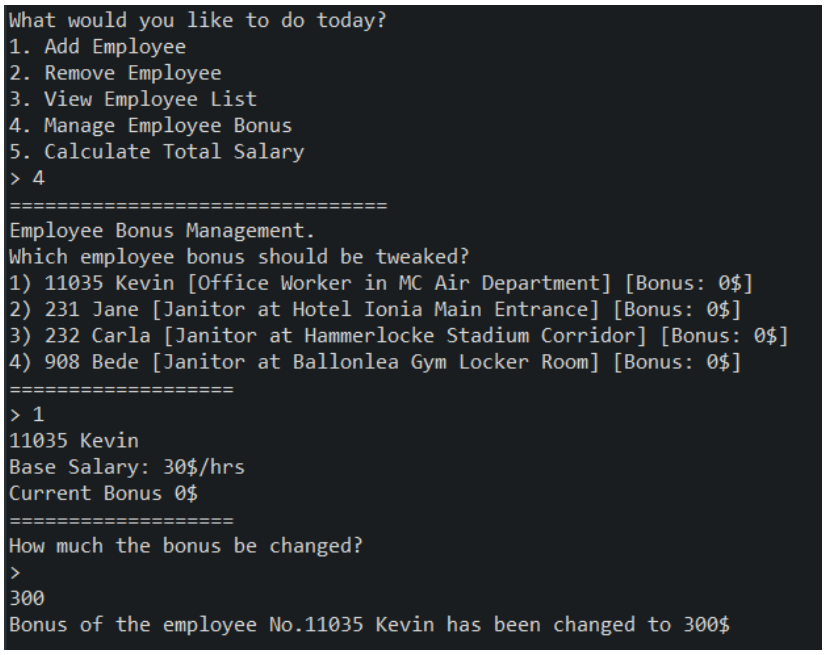
```
What would you like to do today?
1. Add Employee
2. Remove Employee
3. View Employee List
4. Manage Employee Bonus
5. Calculate Total Salary
> 2
================================
Removing Employee Entry.
Please input the id of the employee to be removed.
> 123
Error Removing Employee ID.123
Employee Record does not exist.
```

5.3 View Employee Lists



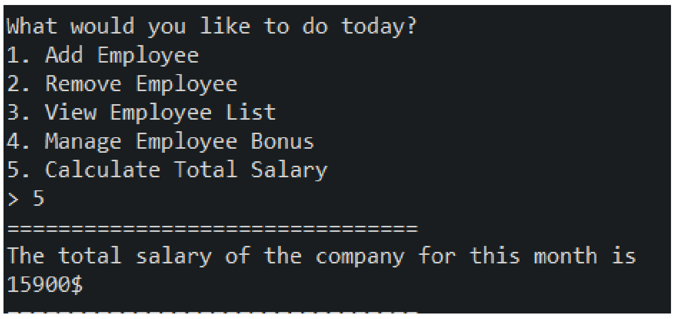
```
What would you like to do today?
1. Add Employee
2. Remove Employee
3. View Employee List
4. Manage Employee Bonus
5. Calculate Total Salary
> 3
================================
List of all Employee.
1) 11035 Kevin [Office Worker in MC Air Department] [Bonus: 0$]
2) 231 Jane [Janitor at Hotel Ionia Main Entrance] [Bonus: 0$]
3) 232 Carla [Janitor at Hammerlocke Stadium Corridor] [Bonus: 0$]
4) 908 Bede [Janitor at Ballonlea Gym Locker Room] [Bonus: 0$]
```

5.4 Manage Employee Bonus



```
What would you like to do today?
1. Add Employee
2. Remove Employee
3. View Employee List
4. Manage Employee Bonus
5. Calculate Total Salary
> 4
================================
Employee Bonus Management.
Which employee bonus should be tweaked?
1) 11035 Kevin [Office Worker in MC Air Department] [Bonus: 0$]
2) 231 Jane [Janitor at Hotel Ionia Main Entrance] [Bonus: 0$]
3) 232 Carla [Janitor at Hammerlocke Stadium Corridor] [Bonus: 0$]
4) 908 Bede [Janitor at Ballonlea Gym Locker Room] [Bonus: 0$]
===================
> 1
11035 Kevin
Base Salary: 30$/hrs
Current Bonus 0$
===================
How much the bonus be changed?
>
300
Bonus of the employee No.11035 Kevin has been changed to 300$
```

5.5 Calculate Total Salary



```
What would you like to do today?
1. Add Employee
2. Remove Employee
3. View Employee List
4. Manage Employee Bonus
5. Calculate Total Salary
> 5
================================
The total salary of the company for this month is
15900$
```