Exam 1, Faculty of Engineering, Chulalongkorn University
Course ID: 2110215 Course Name: Programming Methodology I
First Semester, Date: 18 September 2020 Time: 9.30-11.30AM

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\progmeth2020_1\Exam1_2110215_(your id)_(FirstName)**" (if not exist, you must create it)
  - For example, "C:\temp\progmeth2020_1\Exam1_2110215_631234521_John"
- All your files must be in the workspace.
- **The code outside of the workspace will not be collected, or graded**

Scoring (Total 20 points, will be scaled to 25 points)

- Part1 = 10 points
- Part2 = 10 points

Part 1: OOP Concept

Objective

1) Be able to implement Objects and Classes.

Instruction

1) Create Java Project named "**2110215_Exam1_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

   a) **Person** (package logic)
   b) **EnterProfile** (package logic)
   c) **Building** (package logic)
4) **JUnit for testing is in package test.grader**

Problem Statement: CV Checker

CV Checker is a JAVA application for Chulalongkorn university to keep track of people who enter building inside university for purpose of preventing COVID from spreading. The profile contains person identification and their body temperature. However, the current system's implementation is still incomplete. You are tasked to implement the remaining **Person**, **Building** and **EnterProfile** classes

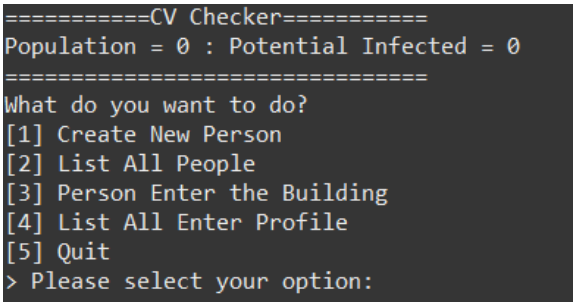


Figure 1. Welcome screen when the application starts.

1) The first option is adding a new person to the system. Please note that the new person's ID has to be different from existing people. Otherwise, the system will raise an error. If ID entered is less than 1, the system will set the ID to 1 (this is set if there is no one with ID 1).

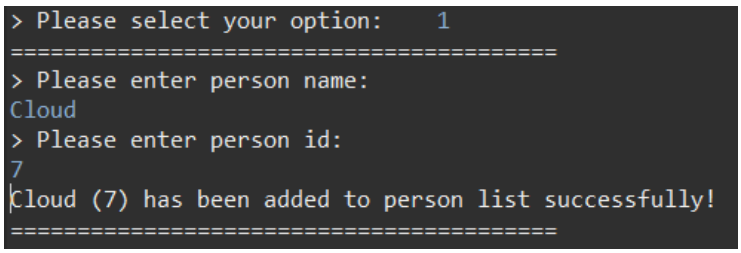


Figure 2. Successfully adding a new person.

2) The second option is to list all people in the system. Each person information displays the name and his/her ID. When the program starts, there will already be 4 initial people in the system.

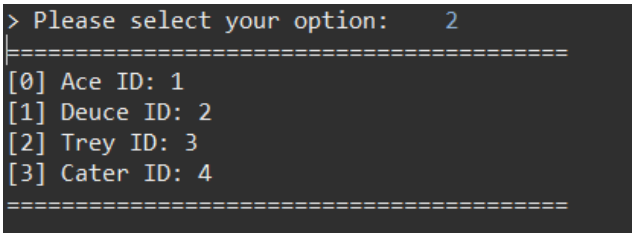


Figure 3. Initial Person Listing.

3) The third option is used to record people entering a building. People queue to enter buildings (the recording is carried out one after another). For each person entering the building, the program user will be asked to enter his/her ID and then enter his/her body temperature. The user can finish entering information by entering blank.

If the user selects to add a person that he/she already add, that person's profile will be replaced with the new one.

```
> Please select your option:    3
========================================
[ID: 1] Ace
[ID: 2] Deuce
[ID: 3] Trey
[ID: 4] Cater
> Enter person's ID to enter the building / Enter blank to go back to menu.
2
> Please enter of this person temperature
35
A Person has enter this buidling
> Enter person's ID to enter the building / Enter blank to go back to menu.
3
> Please enter of this person temperature
36
A Person has enter this buidling
> Enter person's ID to enter the building / Enter blank to go back to menu.
4
> Please enter of this person temperature
37
A Person has enter this buidling
> Enter person's ID to enter the building / Enter blank to go back to menu.
5
Error: Invalid ID!
> Enter person's ID to enter the building / Enter blank to go back to menu.

========================================
```

Figure 4. Creating a new enter profile.

4) The fourth option is to list all people that entered buildings. It will show the person name, and the result of his/her body temperature. If the body temperature is 36 or lower, the result will be "negative", otherwise it will be "positive".

```
> Please select your option:    4
========================================
 -- Deuce --- Negative
 -- Trey --- Negative
 -- Cater --- Positive
========================================
```

Figure 5. Listing all enter profiles.

The total number of people in the building as well as potential infected will be shown above the menu.

```
===========CV Checker===========
Population = 4 : Potential Infected = 2
================================
What do you want to do?
[1] Create New Person
[2] List All People
[3] Person Enter the Building
[4] List All Enter Profile
[5] Quit
> Please select your option:
```

Figure 6. Menu screen after adding some enter profiles.

Implementation Detail

The class package is summarized below.

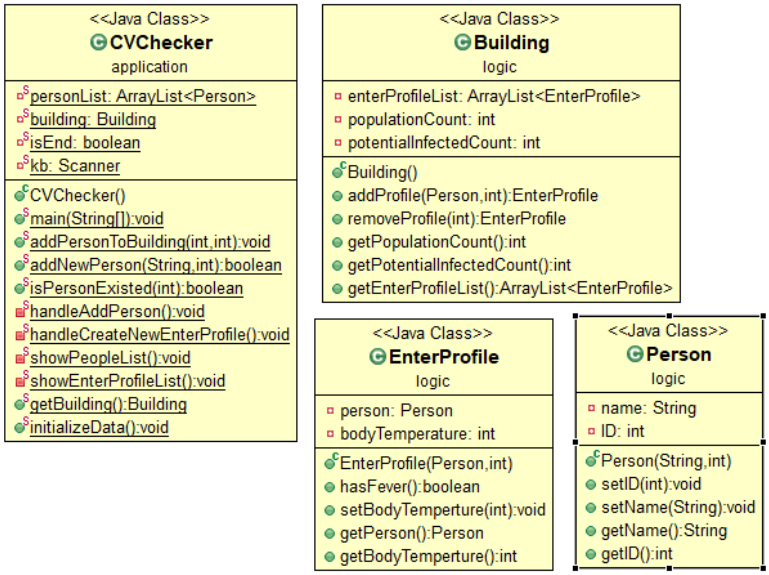


Figure 6. Class Diagram

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 Person /* You must implement this class from scratch */

This class represents the Person.

Field

Name	Description
- String name	The name of the person.
- int ID	The ID of the person, it can only be positive number.

Constructor

Name	Description
+ Person(String name, int ID)	Create a new person with specified name and ID. Set related fields with the given parameters; ID value must be greater than or equal to 1, if not, set it to 1.

Method

Name	Description
+ void setID(int ID)	Set ID of the item. If the given value is less than 1, set it to 1.
+ getter/setter for other variables	

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

This class is used for storing a person and his/her body temperature when he/she enters the building.

Field

Name	Description
- Person person	Person who enters the building in this profile.
- int bodyTemperature	Body temperature of person when he/she enters this building.

Constructor

Name	Description
+ EnterProfile(Person person, int bodyTemperature)	Create a new EnterProfile object with specified person and bodyTemperature. Set related fields with the given parameters.

Method

Name	Description
+ boolean hasFever()	Return boolean value, if bodyTemperature is 37 or higher, return true. Otherwise, return false.
+ void setBodyTemperature (int bodyTemperature)	Set bodyTemperature of the EnterProfile by the given value. If the given value is lower than 35, set bodyTemperature to 35. If the given value is higher than 42, set bodyTemperature to 42.
+ getter for each variable	

4.1.3 Class Building /* PARTIALLY PROVIDED */

This class represents a building that this application is used. It saves the list of enterProfile. It is partially provided. Please fill necessary code.

Field

Name	Description
- ArrayList<EnterProfile> enterProfileList	ArrayList of EnterProfile.
- int populationCount	Number of people in the building, should be equal to number of enterProfile in enterProfileList.
- int potentialInfectedCount	Number of this people in the building that are potentially infected, should be equal to number of enterProfile in enterProfileList that has fever.

Constructor

Name	Description
+ Building()	Create a new building object. Initialize all variables.

Method

Name	Description
+ EnterProfile addProfile(Person person, int temperature)	/* FILL CODE */ Create enterProfile with a given person and temperature, and add the created enterProfile to this building. Check if this building already has this **person** in enterProfileList. If it has, remove the old one (Hint: Look at method below this one). Increase populationCount by 1 and if enterProfile hasFever is true, Increase potentialInfectedCount by 1. This method returns the added EnterProfile object.
+ EnterProfile removeProfile(int index)	/* FILL CODE */ Remove enterProfile from enterProfileList according to the index number.

	Decrease populationCount by 1 and if that enterProfile hasFever is true, decrease potentialInfectedCount by 1. This method returns the removed EnterProfile object. Hint : Lists cannot be modified while it's in a list loop.
+ other remaining getters	

4.2 Package application

4.2.1 Class CVChecker /*ALREADY PROVIDED */

This class represents the CO-VID checking system. It also contains main method. This class is provided. Please run its main method to test your program.

5. Scoring Criteria

The maximum score for the problem is 30 and will be rounded down to 10

5.1 Class Person (PersonTest): = 8 points

testConstructor	= 3 points
testConstructorMinusID	= 1 points
testGetName	= 1 points
testSetName	= 1 points
testSetID	= 1 points
testSetIDZero	= 1 points

5.2 Class EnterProfile (EnterProfileTest): = 10 points

testConstructor	= 3 points
testConstructorBelow35Temp	= 1 points
testConstructorAbove42Temp	= 1 points
testHasFever	= 2 points
testSetBodyTemperatureNormal	= 1 points
testSetBodyTemperatureTooLow	= 1 points
testSetBodyTemperatureTooHigh	= 1 points

5.3 Class Building (BuildingTest): = 12 points

testAddProfileNormal	= 2 points
testAddProfileFever	= 2 points
testAddProfileExistedNormal	= 1 points
testAddProfileExistedFever	= 1 points
testRemoveProfileNoFever	= 2 points
testRemoveProfileWithFever	= 2 points
testGetPopulationCount	= 1 points
testGetPotentialInfectedCount	= 1 points

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_Exam1_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) **Attack** (package logic.attack)
   b) **SPAttack** (package logic.attack)
   c) **Monster** (package logic.monster)
   d) **Leader** (package logic.monster)

4) Make UML class diagram for classes in package logic, using ObjectAid.
5) **JUnit for testing is in the package test.grader.**

3. Problem Statement: Creatures of ProgMeth

Indie Games are one of the main driving forces behind the game industry nowadays. You are currently one of the members in the upcoming indie game "Creatures of ProgMeth".

It is a turn-based game featuring a battle between a party of four monsters. Each Party has a single "Leader" monster, which has more life and can attacks with more power than other monster on the field at the cost of needing charging turns.

```
Choose which monster to make a move.
1) Zetterburn(20/20), DEF: 4, SPDEF: 3, CHARGE(1/4)
2) Porcuber(6/6), DEF: 2, SPDEF: 1
3) Infernacal(8/10), DEF: 2, SPDEF: 3
4) Blazeen(12/12), DEF: 5, SPDEF: 3
>> 3
============================
Choose which monster to attack.
1) Maypul(15/15), DEF: 4, SPDEF: 3, CHARGE(1/2)
2) Leffox(8/8), DEF: 4, SPDEF: 3
3) Folet(3/6), DEF: 1, SPDEF: 3
4) Boulderoar(14/14), DEF: 5, SPDEF: 3
>> 3
Infernacal attacks Folet!
Folet takes 3 damage!
Folet is defeated!
------------
Enemy's Turn
------------
Boulderoar attacks Zetterburn!
Zetterburn takes 1 damage!
------------
```

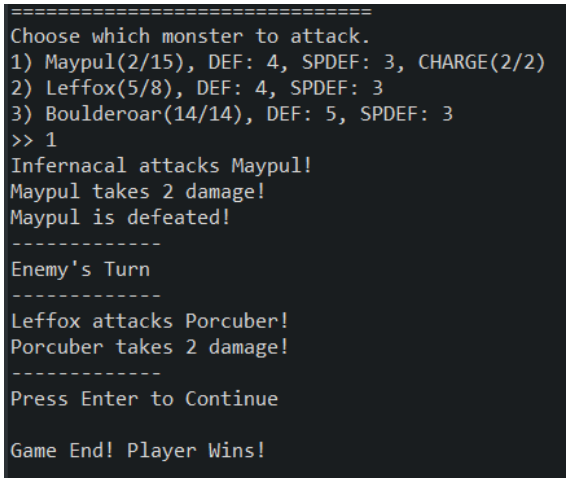
Each side pick a single monster to attack their opponent. If any monster's HP got reduced to 0, then it is defeated.

```
Choose which monster to make a move.
1) Zetterburn(18/20), DEF: 4, SPDEF: 3, CHARGE(3/4)
2) Porcuber(6/6), DEF: 2, SPDEF: 1
3) Infernacal(8/10), DEF: 2, SPDEF: 3
4) Blazeen(12/12), DEF: 5, SPDEF: 3
>> 1
============================
Choose which monster to attack.
1) Maypul(15/15), DEF: 4, SPDEF: 3, CHARGE(2/2)
2) Leffox(5/8), DEF: 4, SPDEF: 3
3) Boulderoar(14/14), DEF: 5, SPDEF: 3
>> 2
Zetterburn cannot attack!
It needed 1 turn more to charge!
------------
Enemy's Turn
------------
Boulderoar attacks Zetterburn!
Zetterburn takes 1 damage!
------------
```

If the leader monster is not the one that made the move that turn, it's charge will increase at the end of the turn. Leader Monster can only make a move if it is fully charged.

```
==============================
What are you going to do?
1) Make a Move
2) Guard
3) Skip Turn
>> 2
2
Zetterburn raise up the guard!
-------------
Enemy's Turn
-------------
Leffox attacks Zetterburn!
Zetterburn takes no damage!
-------------
```

Leader Monster can also guard itself as well. If it received any attack that turn, then it will take no damage. Do note that by guarding, you cannot make any attack move with any other monster. And your charge turns will not increase by the end of the turn.



If the leader monster got defeated, then that side will lose the battle.

During the final crunch before the release of the game, one of the programmers accidentally delete some part of the code. Luckily, most of the battle logics remained intact. You are tasked to re-implement the missing classes according to the Implementation Detail.

This problem is partly inspired by the game “Creatures of Aether” by Dan Fornace, as well as other games in the “monster battling” genres.

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.attack

4.1.1 class Attack **/* Partially Provided */**

This class represents monster’s attack. It contains all the Attack’s information.

Field

Name	Description
# int power	Power of the Attack
# String name	Name of the Attack
# boolean isLeader	Is the attack belong to Leader Monster or not

Constructor

Name	Description
+ Attack (power, name, isLeader)	This sets all the fields with their respective value.

Method

Name	Description
+ boolean equals(Object o)	Checks if the current object is equals to another. This will be used during JUnit testing.
+ int calculateDamage (Monster target)	/* FILL CODE */ Calculate the damage by subtracting the power with the target's **Defense** value. If the result is less than 0, returns 0
+ getter/setter of all fields	

4.1.2 class SPAttack /* You must implement this class from scratch */

This class is another type of Attack, which have different damage calculation.

Method

Name	Description
+ int calculateDamage (Monster target)	Calculate the damage by subtracting the power with the target's **Special Defense** value. If the result is less than 0, returns 0

4.2 Package logic.monster

4.2.1. class Monster /* Partially Provided */

This class represents regular Monster in the game. It contains basic property for each monster.

Fields

Name	Description
# String name	Name of the Monster
# int hp	Health Point (HP) of the Monster. If its HP is depleted, then it will be defeated.
# int maxhp	Maxmimum HP of the Monster
# int def	Defense Value of the Monster, used for calculating Attack's damage.
# int sp_def	Special Defense Value of the Monster, used for calculating Special Attack's damage.
# Attack attack	The Monster's Attack. Which will be used if performing an attack.
# boolean isDead	Check if the Monster has been defeated or not.
# boolean attackStatus	Keeps track if the Monster has attacked in the current turn or not.

Constructor

Name	Description
+ Monster(name, hp, def, sp_def, attack)	This sets all the fields with their respective value.

Method

Name	Description
+ boolean isReady()	For Monster, this always returns true.
+ int takeDamage(Attack attack)	/* FILL CODE */ Reduce the HP with the damage amount calculated from Attack.

	If the final HP is less than or equal zero, set the **isDead** value to true as well. This method returns the amount of damage.
+ getter/setter of all fields	

4.2.2. class Leader /* You must implement this class from scratch */

This class represents leader Monster in the game. It is a type of monster.

This class contains properties that are exclusive to Leader monster, as well as different damage calculation.

Fields

Name	Description
- int maxChargeTurns	A maximum amount of charge turns needed before attacking.
- int currentChargeTurns	A current amount of charge turns needed before attacking.
- boolean isGuard	Keeps track if the Leader is guarding in the current turn or not

Method

Name	Description
+ Leader(name, hp, def, sp_def, attack, chargeTurns)	This sets all the fields with their respective value. **Important:** 1. Set **maxChargeTurns** to chargeTurns and **currentChargeTurns** to 0 2. You need to set **maxChargeTurns** before **currentChargeTurns**.

	3. Always use the setter instead of setting value manually
+ setCurrentChargeTurns(int currentChargeTurns)	Set currentChargeTurns. **If the value is below 0, set it to 0.** **If the value is above maxChargeTurns, set it to maxChargeTurns as well.**
+ boolean isReady()	Returns true if currentChargeTurns is greater or equal to maxChargeTurns. Returns false otherwise.
+ int takeDamage(Attack attack)	Check if **isGuard** is true or not. **If it is, then this method returns 0.** Else, check if the attack comes from Leader monster or not using **attack.isLeader().** If the attack comes from Leader, reduce the HP with the damage amount calculated from Attack. Otherwise, only reduce the HP with the half of the damage instead **using the following formula.** Math.ceil(damage*0.5) If the final HP is less than or equal zero, set the **isDead** value to true as well. This method returns the amount of damage.
Other remaining getter/setters	

4.3 Package main

4.3.1 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. Scoring Criteria

The maximum score for the problem is 27 and will be rounded down to 10

5.1 Class Attack: = 3 points (1 point for each case)

5.2 Class SPAttack: = 7.5 points

Field & Constructor = 1.5 points

Inheritance = 3 points

Calculate Damage = 3 points (1 point for each case)

5.3 Class Monster: = 3 points (1 point for each case)

5.4 Class Leader: = 13.5 points

Field & Constructor = 1.5 points

Inheritance = 2 points

Getter & Setter = 5 points (1 point for each case)

Take Damage = 5 points (1 point for each case)