Midterm Exam, Faculty of Engineering, Chulalongkorn University
Course ID: 2110215 Course Name: Programming Methodology I

Second Semester, Date: 11 March 2020 Time: 8.30-11.30AM

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

Instructions for on-site students (online students - just ignore what you cannot do)

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. Online students must use mobile phone as a Zoom camera.
5. No communication is allowed.
6. *** 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. ***
7. 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.
8. Online students are not allowed to leave the online exam until the exam finishes.
9. 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.
10. 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.
- Each question shows 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\progmeth2021_2\midterm_2110215_(your id)_(FirstName)**" (if not exist, you must create it)
  - For example, "C:\temp\progmeth2021_2\midterm_2110215_6470156021_Boss"
- All your files must be in the workspace.
- The code outside of the workspace will not be collected, or graded

To submit

For on-site students:

Just leave your IDE on. The TA will save the file from Computer Center for you.

For online students:

1.1. go to the workspace folder that you actually do the coding for the 3 questions of this exam.

1.2. Zip the workspace (there must be 3 folders, 1 for each question) to .zip and name it using your ID (for example, 6431300021.zip)

1.3. Submit the zipped file as an assignment on MyCourseville.

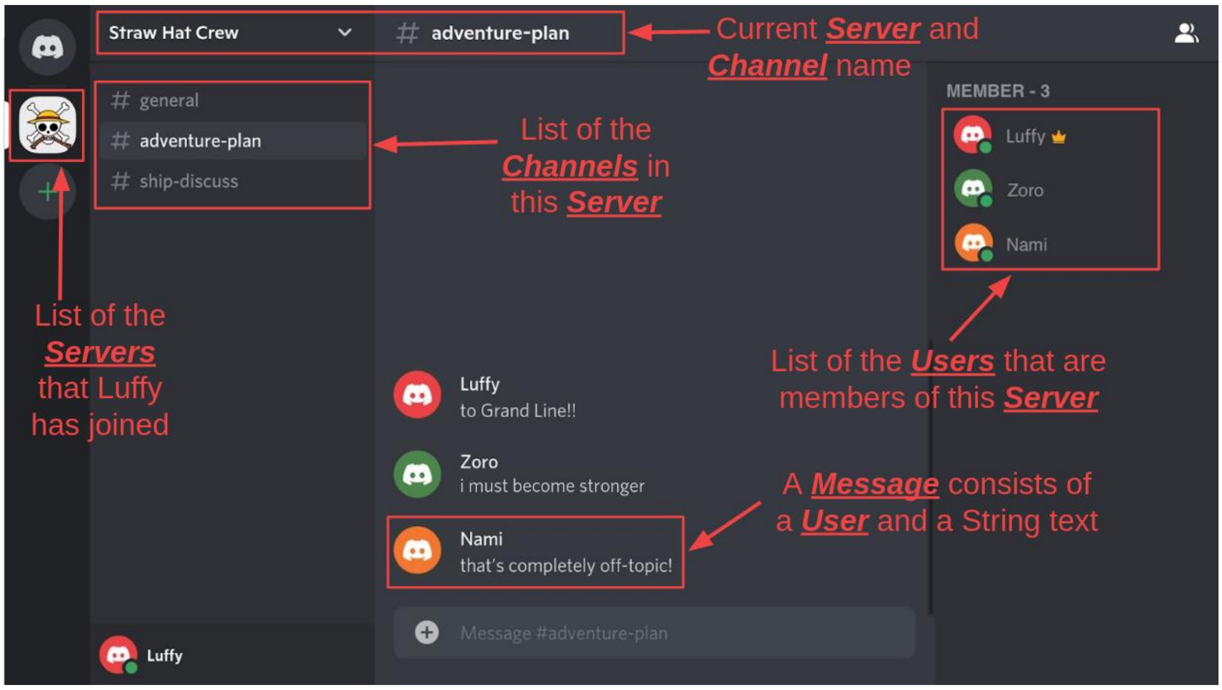
Question 1: (60 minutes, 08.30-09.30)

1. Instruction

1) Create a Java Project named “2110215_Midterm_Q1”.
2) Copy all folders in “Q1” to your project directory src folder.
3) You are to implement the following classes (the details for each class are given in section 4)
   3.1. **Server (package logic)**
   3.2. **Channel (package logic)**
4) Make UML class diagram for the two classes above. Save the picture file in your logic folder.
5) JUnit for testing is in package test.student

2. Problem Statement: Simple Discord

Socializing and classroom learning are not easy now. But thanks to the wonderful Discord platform, we can still connect and learn together even in this hard time. Wouldn’t life be a bit miserable without Discord? Discord is cool. Let’s make a simple text-based Discord.


Straw Hat Crew
# adventure-plan
Current Server and Channel name
# general
# adventure-plan
# ship-discuss
List of the Channels in this Server
MEMBER - 3
Luffy
Zoro
Nami
List of the Servers that Luffy has joined
List of the Users that are members of this Server
A Message consists of a User and a String text
Message #adventure-plan
Luffy

Luffy
to Grand Line!!

Zoro
i must become stronger

Nami
that's completely off-topic!

Figure 1: Parts of Discord with their names (<u>Note that the program for this task is 100% text-based. This figure is only to help you visualize and confirm the terminology.</u>)

Below are the Discord terms for this problem.

1) **Server**: A place in Discord for **Users** with the same interest to hang out. For Example, ProgMeth 2021 (Term2) and CP CHULA HOMEROOM are **Servers** that you join in real life. In Figure 1, the current **Server** being shown is Straw Hat Crew. The **User** that created the **Server** is the **owner** of that **Server**.
2) **Channel**: **Servers** are organized into **Channels**. This is where **Users** send **Messages**. For Example, in Figure 1 general, adventure-plan and ship-discuss are **Channels**. Only the **owner** can add a new **Channel**.
3) **User**: Each **User** represents a person. **Users** can join other **Users**' **Server** or create their own **Servers**. Once they are a member of a **Server**, they can send **Messages** in a **Channel** in that **Server**. The **owner** is also a member of the **Server**.
4) **Message**: Represents a message. It has two components which are 1) the text itself (a String), and 2) the **User** who sends it. Each **Message** belongs to a **Channel**.
5) **TemplateType**: Determines the starting **Channel** at the creation of a **Server**. For this task, there are 3 **TemplateTypes**. (See part 3 for the complete implementation details and part5 figure 8 for a demonstration.) For example, if a new **Server** called "Our Little School" is being created with the **TemplateType** STUDY, then a **Channel** called "homework-help" will be generated in "Our Little School" **Server**.

Your task is to implement 2 classes from scratch: **Server** and **Channel**. The **User, Message** and **TemplateType** classes are already provided.

3. Implementation Detail

The two classes are summarized below.

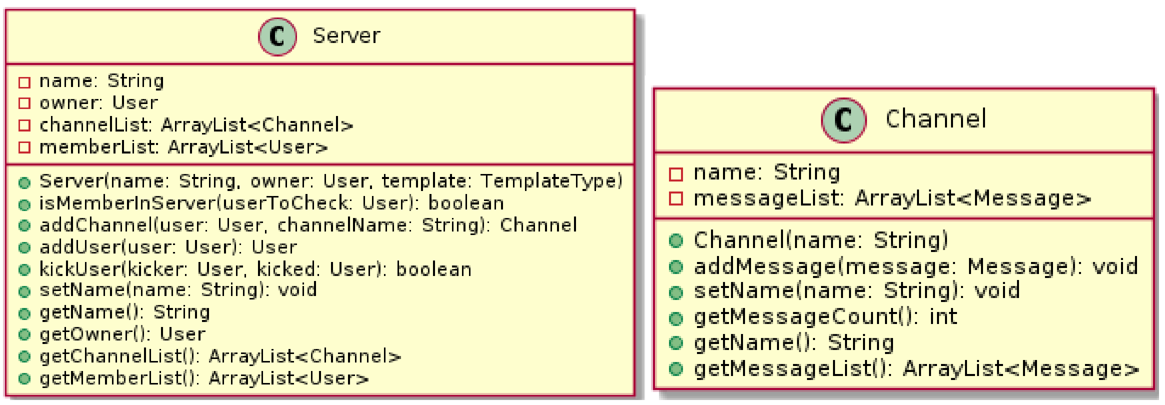

C Server
name: String
owner: User
channelList: ArrayList<Channel>
memberList: ArrayList<User>
Server(name: String, owner: User, template: TemplateType)
isMemberInServer(userToCheck: User): boolean
addChannel(user: User, channelName: String): Channel
addUser(user: User): User
kickUser(kicker: User, kicked: User): boolean
setName(name: String): void
getName(): String
getOwner(): User
getChannelList(): ArrayList<Channel>
getMemberList(): ArrayList<User>
C Channel
name: String
messageList: ArrayList<Message>
Channel(name: String)
addMessage(message: Message): void
setName(name: String): void
getMessageCount(): int
getName(): String
getMessageList(): ArrayList<Message>

Figure 2. Class Diagram

You must write java classes using the UML diagram specified above.

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

3.1 Package logic

3.1.1 Enum TemplateType /*ALREADY PROVIDED*/

Determines the starting Channel. It is an enumeration (a list of constants) that specifies a Server's starting Channel at the creation of a Server. (See Server's constructor)

Value

Name	Description
BASIC	Servers instantiated with BASIC TemplateType have one starting Channel called "general".
GAMING	Servers instantiated with GAMING TemplateType have one starting Channel called "gaming".
STUDY	Servers instantiated with STUDY TemplateType have one starting Channel called "homework-help".

3.1.2 Class Message /*ALREADY PROVIDED*/

This class represents messages that are sent in Channel.

Fields

Name	Description
- String text	The text to be sent.
- User sender	The sender of this message.

Constructor

Name	Description
+ Message(String text, User sender)	Create a new Message with the specified information.

Methods

Name	Description
+ getters/setters for each field	

3.1.3 **Class User** /*ALREADY PROVIDED*/

This class represents users in our Discord system.

Fields

Name	Description
- String name	The name of the user.
- String status	The status of the user.
- ArrayList<Server> joinedServerList	An ArrayList containing Servers that this user has joined, including those that this user is the owner of.

Constructor

Name	Description
+ User(String name)	Create a new User with the specified name. Set the status to "hi there, I'm new here". Initialize the joinedServerList.

Methods

Name	Description
+ boolean equals(Object other)	If the User other is not null, check whether that User is equal to this User by comparing the name of the Users. Otherwise return false.
+ void addJoinedServersList(Server server)	If server is null, do nothing. Otherwise, add the server to this User's joinedServersList.

+ void setStatus(String status)	If the specified status is not blank, set this User's status to that status. Otherwise do nothing (the status is unchanged).
+ void setName(String name)	If the specified name is not blank, set this User's name to that name. Otherwise set the User's name to "Nobody".
+ getters for name, status, joinedServersList	

3.1.4 **Class Channel** /* You must create this class from scratch. */

This class represents channels in a Discord Server. It contains the channel's name and all the Messages that are sent in the channel.

Fields

Name	Description
- String name	The name of the channel.
- ArrayList<Message> messageList	An ArrayList containing all the Messages sent in the channel.

Constructor

Name	Description
+ Channel(String name)	Create a Channel with the specified name. Initialize the messageList.

Methods

Name	Description
+ void addMessage(Message message)	Add the message to the messageList.
+ void setName(String name)	Set the name of this Channel. If the specified name is a blank string, set this Channel's name to "off-topic". Otherwise, set it to that name. Hint: use name.isBlank().
+ int getMessageCount()	Return the size of the messageList.
+ remaining getters	

3.1.5 **Class Server** /* You must create this class from scratch. */

This class represents the servers in Discord. It contains data related to the Server, including a list of channels within that server and a list of Users that are members.

Fields

Name	Description
- String name	The name of the Server.
- User owner	The owner of the Server.
- ArrayList<Channel> channelList	An ArrayList containing all the Channels in the Server.
- ArrayList<User> memberList	An ArrayList containing all the members of the Server, including the owner.

Constructor

Name	Description
+ Server(String name, User owner, TemplateType template)	(Note: The order of initializations is important!!) Set the owner of the Server to the specified owner. Initialize the memberList. Initialize the channelList. Add the owner to the memberList and add this Server to the owner's joinedServersList. (Hint: use the addUser method of this class) Set the name of the server. Then create and add 1 Channel to the channelList depending on the TemplateType specified. If the template is 1) BASIC: create and add a Channel called "general" 2) GAMING: create and add a Channel called "gaming" 3) STUDY: create and add a Channel called "homework-help" (Hint: Use the addChannel method (see below), and pass the owner of the Server as one of the arguments.)

Methods

Name	Description
+ Channel addChannel(User user, String channelName)	If the user in the argument is the owner of the Server, create a new Channel with the specified channelName. Then add it to the channelList and return the added Channel. Otherwise, if the user is not the owner of the server, return null and do nothing else.
+ User addUser(User user)	There are 2 cases. 1) If the user is not already in the memberList, add the user to the memberList. (Hint: use the *contains* method of ArrayList) Also, add this server to the user's joinedServersList (See 4.1.3 Class User for details), then return the newly added user. 2) If the user is already in the memberList, return null and do nothing else.
+ boolean kickUser(User kicker, User kicked) throws Exception	There are three cases. 1) If the kicker is not the Server's owner: Throw an Exception and do nothing else. 2) Else If the kicker is the owner but the user to be kicked is either **i)** not in the memberList or **ii)** is the Server's owner himself/herself: Return false and do nothing else. (Owner can't be removed): 3) Else: Remove this Server from kicked's joinedServersList. Then, remove the User kicked from the Server's memberList and

	return true. (Note: there are 2 variants of the ArrayList method *remove*. Make sure you're using the one that returns boolean and not the polymorphic one)
+ void setName(String name)	If the name is blank, set the Server's name to <owner's name> + " home". (For example, "Citron home") Otherwise, set it to the specified name
+ remaining getter for each field	

3.2 Package main /*ALREADY PROVIDED */

3.2.1 Class Main: This class is the main application. You do not need to know any details about this class to be able to complete the problem. Its purpose is for the program demonstration (see section 5).

4. Score Criteria (35 marks: will be scaled down to 5)

ChannelTest

- testConstructor (1 mark)
- testConstructorEmptyName (1 mark)
- testSetName (1 mark)
- testSetNameEmpty (1 mark)
- testAddMessage (1 mark)
- testGetMessageCount (1 mark)

ServerTest

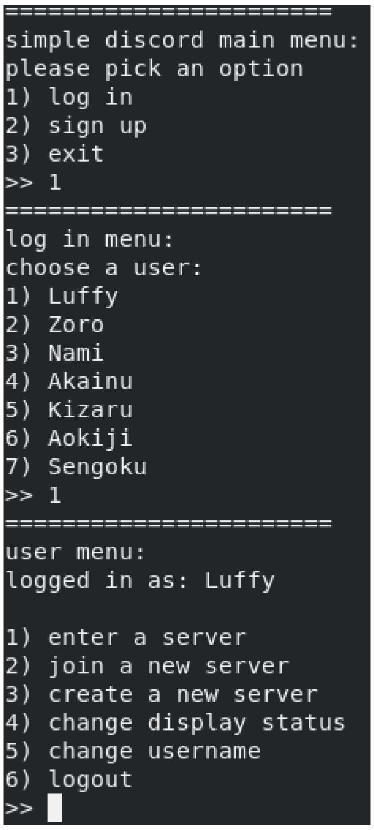
- testConstructorBasicTemplate (2 mark)
- testConstructorGamingTemplate (2 mark)
- testConstructorStudyTemplate (2 mark)
- testConstructorEmptyServerName (2 mark)

- testSetName (1 mark)
- testSetNameEmpty (1 mark)
- testAddUser (2 mark)
- testAddDuplicateUser (2 mark)
- testKickUser (2 mark)
- testKickUserNotOwner (2 mark)
- testKickUserNonExistingUserOrKickingOwner (2 mark)
- testAddChannelOwner (2 mark)
- testAddChannelNotOwner (2 mark)

UML

- 2.5 mark for each class

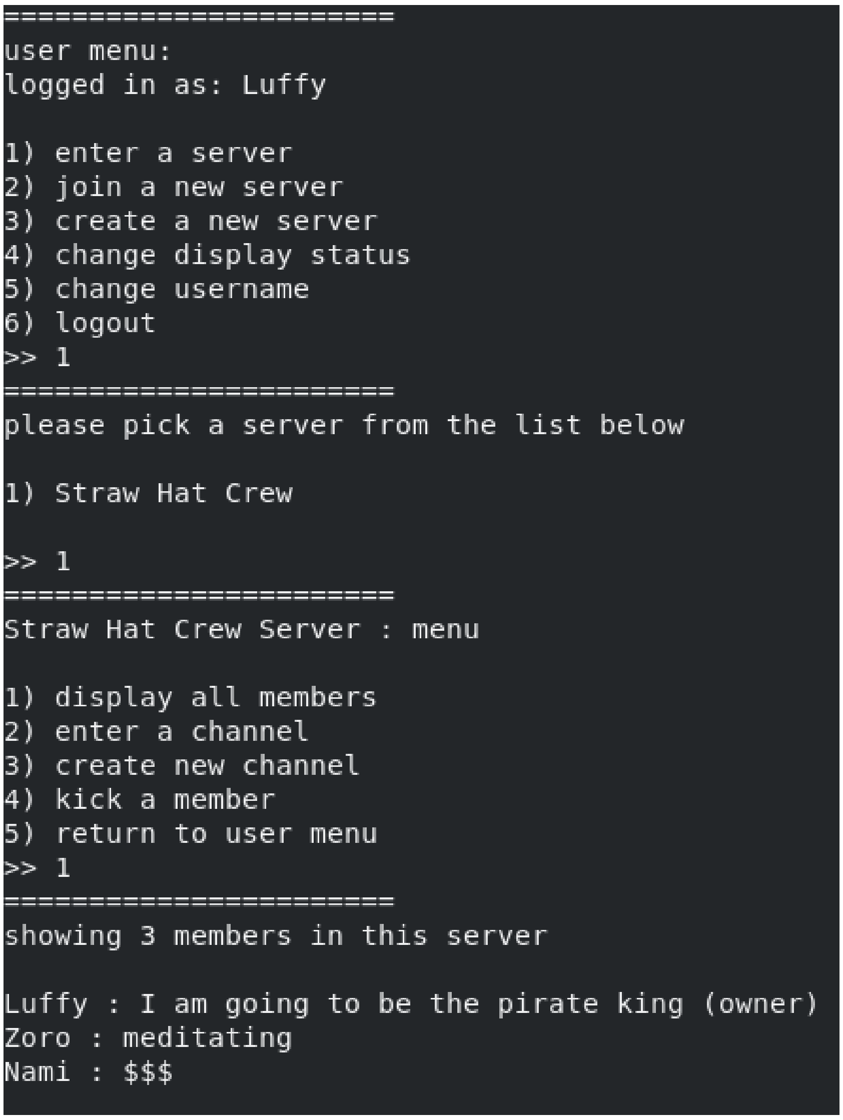
5. Program Demonstration



```
=======================
simple discord main menu:
please pick an option
1) log in
2) sign up
3) exit
>> 1
=======================
log in menu:
choose a user:
1) Luffy
2) Zoro
3) Nami
4) Akainu
5) Kizaru
6) Aokiji
7) Sengoku
>> 1
=======================
user menu:
logged in as: Luffy

1) enter a server
2) join a new server
3) create a new server
4) change display status
5) change username
6) logout
>> 
```

Figure 3: Start of the program ⟶ log in ⟶ user menu

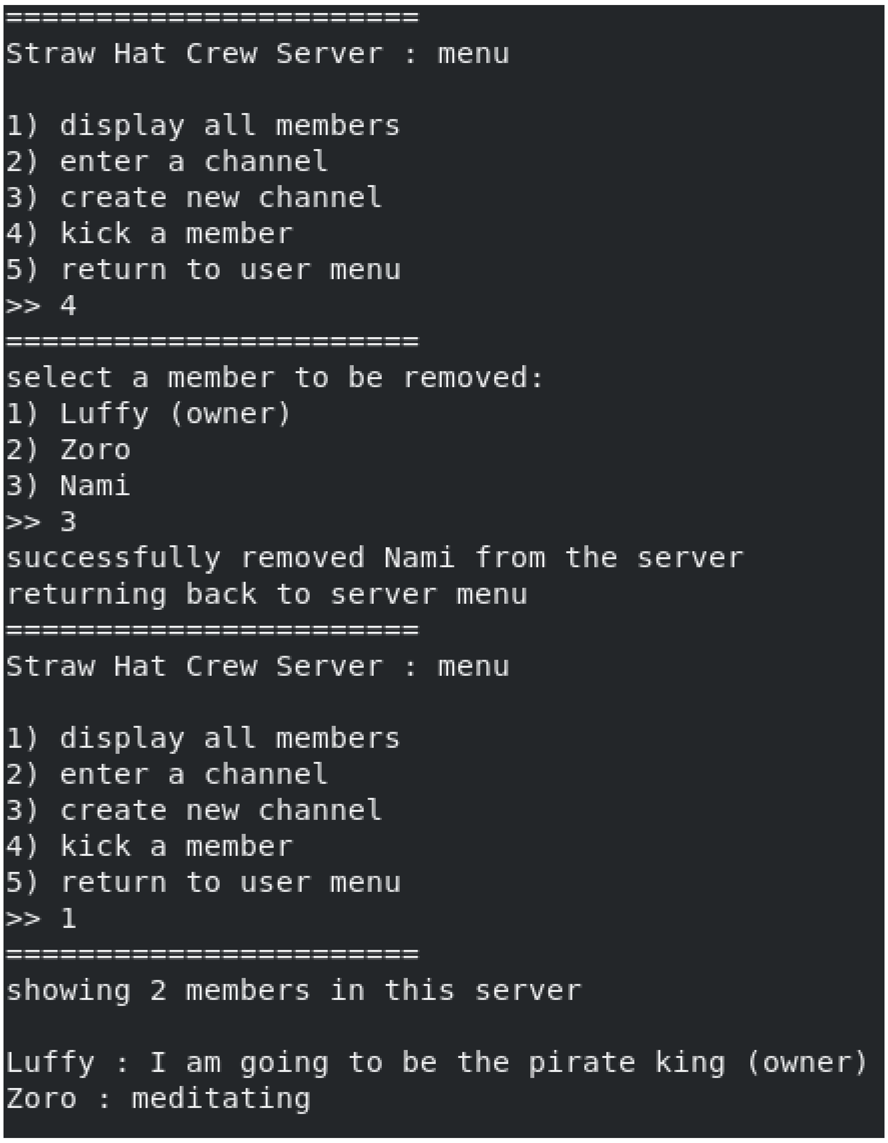
1. Start of the program.
2. Choose to log In.
3. Log in as Luffy (The program is populated with 7 Users from the beginning).



```
=======================
user menu:
logged in as: Luffy

1) enter a server
2) join a new server
3) create a new server
4) change display status
5) change username
6) logout
>> 1
=======================
please pick a server from the list below

1) Straw Hat Crew

>> 1
=======================
Straw Hat Crew Server : menu

1) display all members
2) enter a channel
3) create new channel
4) kick a member
5) return to user menu
>> 1
=======================
showing 3 members in this server

Luffy : I am going to be the pirate king (owner)
Zoro : meditating
Nami : $$$
```

Figure 4: user menu ⟶ server menu ⟶ display all members

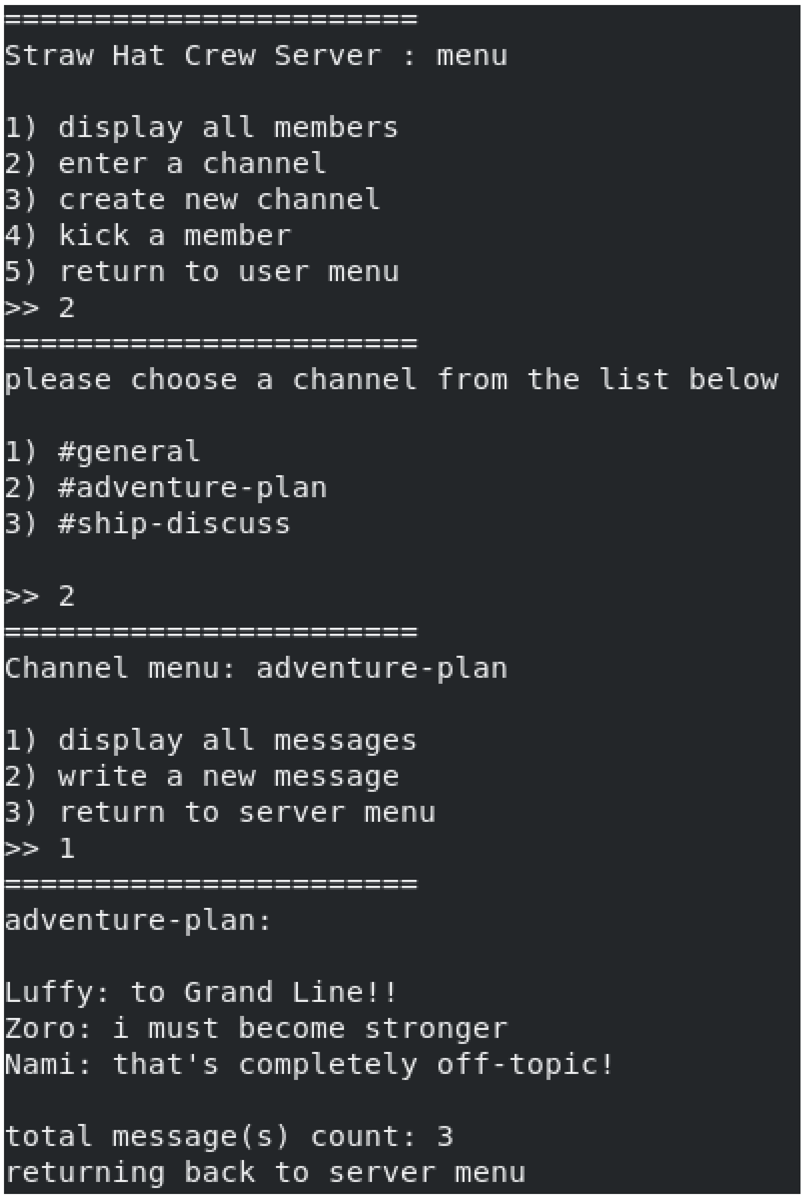
1. Logged in as Luffy, currently in the user menu.
2. Choose to enter a server that Luffy has joined.
3. A list of Servers that Luffy has joined is shown (there is only 1 Server).
4. Choose to enter “Straw Hat Crew” Server.
5. Currently in the “Straw Hat Crew” Server menu.
6. Choose to display all members in this Server.
7. A list of all the User’s name and their status is shown. (total of 3 Users). Also note that the (owner) is marked.



```
=======================
Straw Hat Crew Server : menu

1) display all members
2) enter a channel
3) create new channel
4) kick a member
5) return to user menu
>> 4
=======================
select a member to be removed:
1) Luffy (owner)
2) Zoro
3) Nami
>> 3
successfully removed Nami from the server
returning back to server menu
=======================
Straw Hat Crew Server : menu

1) display all members
2) enter a channel
3) create new channel
4) kick a member
5) return to user menu
>> 1
=======================
showing 2 members in this server

Luffy : I am going to be the pirate king (owner)
Zoro : meditating
```

Figure 5: server menu ⟶ kick member

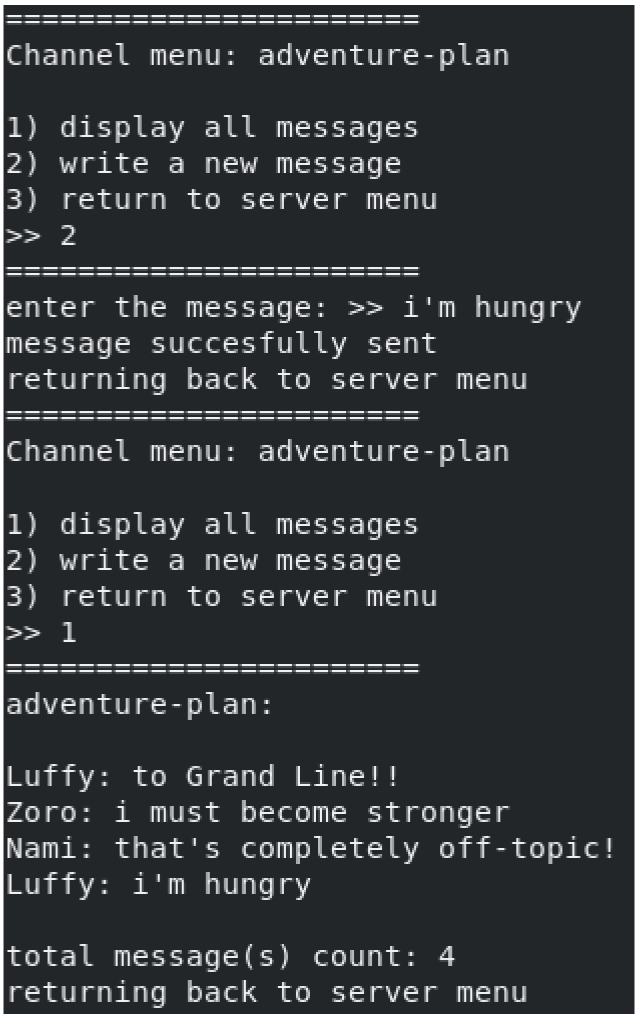
1. Currently in the “Straw Hat Crew” Server menu.
2. Kick a member.
3. Kicking the 3rd member Nami.
4. The kick is successful, because the kicker is Luffy (the Server’s owner).
5. There are 2 members left.



```
=======================
Straw Hat Crew Server : menu

1) display all members
2) enter a channel
3) create new channel
4) kick a member
5) return to user menu
>> 2
=======================
please choose a channel from the list below

1) #general
2) #adventure-plan
3) #ship-discuss

>> 2
=======================
Channel menu: adventure-plan

1) display all messages
2) write a new message
3) return to server menu
>> 1
=======================
adventure-plan:

Luffy: to Grand Line!!
Zoro: i must become stronger
Nami: that's completely off-topic!

total message(s) count: 3
returning back to server menu
```

Figure 6: server menu ⟶ channel menu ⟶ show all Messages

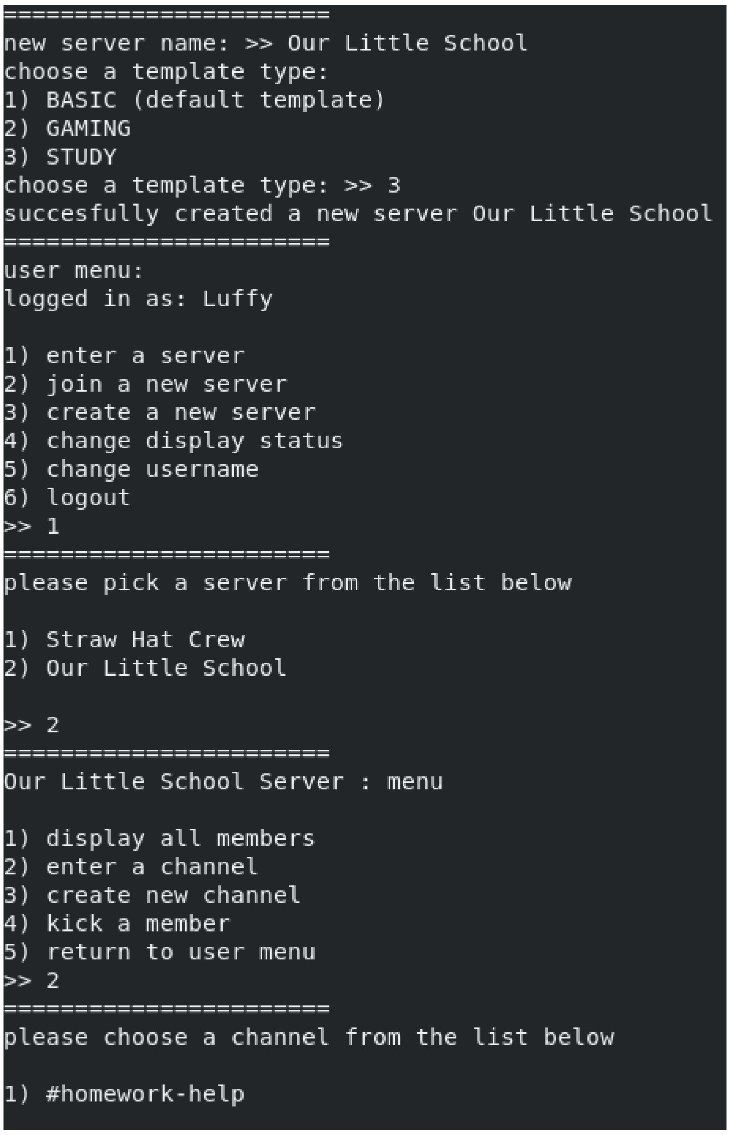
1. Currently in the “Straw Hat Crew” Server menu.
2. Choose to enter a Channel.
3. A list of Channels that is in this Server is shown (total of 3 channels).
4. Choose to enter the “adventure-plan” Channel.
5. Currently in the “adventure-plan” channel menu.
6. Choose to display all Messages sent in this Channel.
7. A list of all Messages that is sent in this Channel is shown (currently there are 3 Messages).



```
=======================
Channel menu: adventure-plan

1) display all messages
2) write a new message
3) return to server menu
>> 2
=======================
enter the message: >> i'm hungry
message succesfully sent
returning back to server menu
=======================
Channel menu: adventure-plan

1) display all messages
2) write a new message
3) return to server menu
>> 1
=======================
adventure-plan:

Luffy: to Grand Line!!
Zoro: i must become stronger
Nami: that's completely off-topic!
Luffy: i'm hungry

total message(s) count: 4
returning back to server menu
```

Figure 7: writing a new Message to the channel

1. Currently in the “adventure-plan” Channel menu.
2. Chooses to write a new message (“i’m hungry”).
3. Now there are a total of 4 Messages in this Channel.



```
=======================
new server name: >> Our Little School
choose a template type:
1) BASIC (default template)
2) GAMING
3) STUDY
choose a template type: >> 3
succesfully created a new server Our Little School
=======================
user menu:
logged in as: Luffy

1) enter a server
2) join a new server
3) create a new server
4) change display status
5) change username
6) logout
>> 1
=======================
please pick a server from the list below

1) Straw Hat Crew
2) Our Little School

>> 2
=======================
Our Little School Server : menu

1) display all members
2) enter a channel
3) create new channel
4) kick a member
5) return to user menu
>> 2
=======================
please choose a channel from the list below

1) #homework-help
```

Figure 8: creating a new Server with TemplateType STUDY

1. Create a new Server called “Our Little School”.
2. Choose the TemplateType as STUDY.
3. There are now 2 Servers that the User Luffy is a member of.
4. In Our Little School Server, there is a #homework-help channel, because the chosen TemplateType was STUDY.

Question 2: (60 minutes, 09.30-10.30)

1. Instructions

a) Create Java Project named **“2110215_Midterm_Q2”** in your workspace.

b) Copy all folders in folder **“Q2”** to your project directory src folder.

c) You are to implement the following classes (details for each class are given in section 4.)

a)	Hospital	(package disease)
b)	Covid19	(package disease)
c)	Delta	(package disease)

d) You also need to create a UML Diagram including the classes Hospital, Covid19 and Delta from the package disease (UML has points too). Save it as picture in the project folder.

e) JUnit for testing is in package test. **This program does not have a main meth-od**. You must run JUnit to test the program.

2. Problem Statement: COVID-19 pandemic

You tasked are to implement Hospital admit and report system where you can model a virus Covid19 and Delta from data provided below:

Table 1. Comparison of variants of concern (VoC) of SARS-CoV-2 as of January 25, 2022

Name	country of first appearance	basic reproduction number (R0) min-max value	Number of mutations on the spike protein
Covid19	China	2-3	-
Delta	India	5-7	10

Details: Worldwide; Al Jazeera; WHO; Expert(s) (F. Campbell et al.); CDC; as of January 25, 2022

Table 2. Severe level data of vaccinated and unvaccinated subject after **infected** by virus

Subject	Severe Level	
	Covid19	Delta
unvaccinated	severe illness	severe illness
vaccinated	less	mild or less

3. Implementation Details

The class packages are summarized below. Note that only important methods that you either need to write and methods that you need to complete this question are listed below.

3.1 package util

3.1.1. **public class R0** /*ALREADY PROVIDED*/

This class represent basic reproduction number R_0, pronounced "R naught," is the expected number of cases directly generated by one case in a population where all individuals are susceptible to infection.

a.) Variables

Variable	Description
-int min	Min expected number of cases
-int max	Max expected number of cases

b.) Methods

Method	Description
+R0 (int min, int max)	Constructor. Set variables to the given parameter values. Example: Covid-19 (R_0 = 2-3) means min = 2 and max = 3

+ getter/setter for everything	

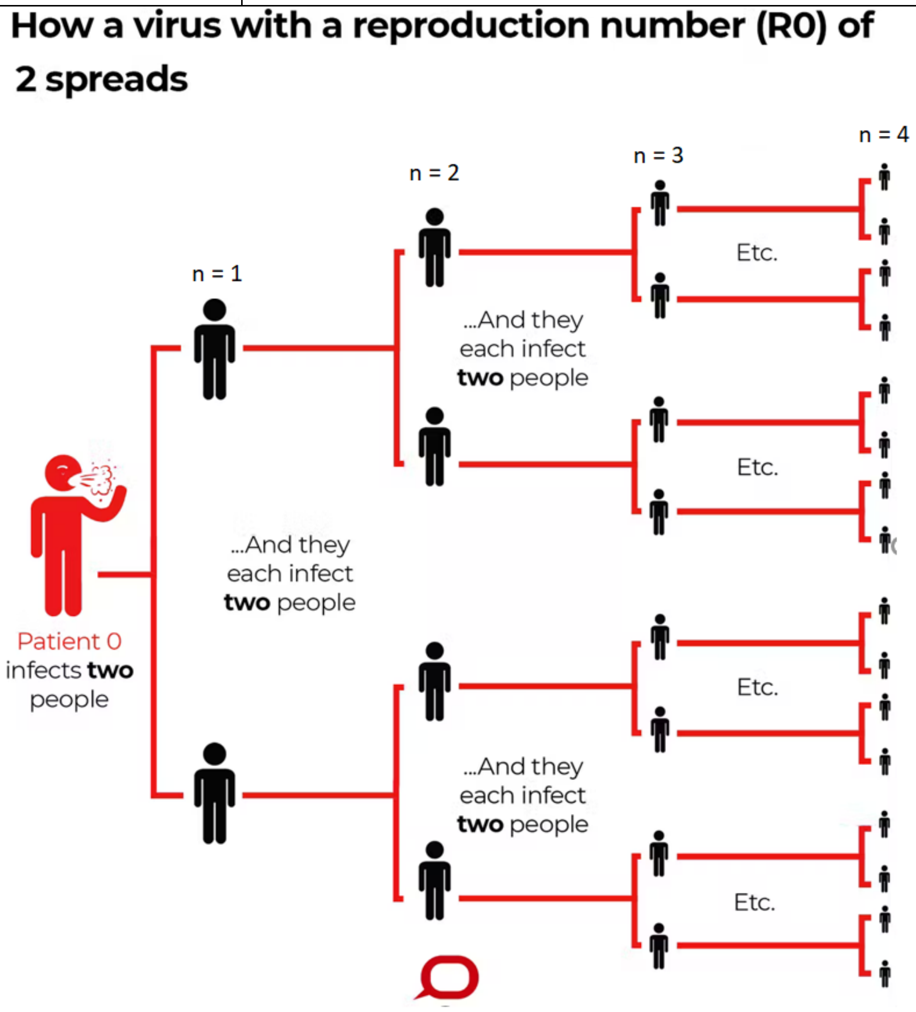

How a virus with a reproduction number (R0) of 2 spreads
n = 1
n = 2
n = 3
n = 4
Patient 0 infects two people
...And they each infect two people
...And they each infect two people
...And they each infect two people
Etc.
Etc.
Etc.
Etc.

Figure 1. Virus with a reproduction number R_0 = 2 spreads

3.1.2. **public enum SevereLevel**/*ALREADY PROVIDED*/

This enum contains Severe level after infected.

3.1.3. **public class Patient**/*ALREADY PROVIDED*/

This class represents data of patient.

a.) Variables

Variable	Description
-String firstname	Patient’s first name

-String lastname	Patient's last name
-String id	Patient's identification number
-Disease disease	Patient's disease
-boolean vaccinated	True if patient is vaccinated, otherwise false

b.) Methods

Method	Description
+Patient (String firstname, String lastname, String id,Disease disease, boolean vac)	Constructor. Set variables to the given parameter values.
+SevereLevel getSevereLevel ()	Return Severe Level
+String toString	Print patient status
+ getter/setter for everything	

3.2 package disease

3.2.1. **public class Disease** /*ALREADY PROVIDED*/

This class is base class for all diseases.

a) Variables

This class has no variables.

b) Methods

Method	Description
+SevereLevel getSevereLevel (boolean isVaccinated)	Determine severe level after infected by virus. **Return** value depends on each type of disease.

3.2.2. **public class Hypopnea** /*ALREADY PROVIDED*/

This class represents a disease in breathing, abnormally slow or shallow breathing.

a) Variables

This class has no variables.

b) Methods

Method	Description
+SevereLevel getSevereLevel(boolean isVaccinated)	Determine severe level. For this question, the value is just MildOrLess regardless of the argument.
+String toString()	Return "Hypopnea"

3.2.3. **public class Covid19** /* You must implement this class from scratch*/

This class represents the Covid-19 base virus. **You can add your own protected method(s) here if needed.**

a) Variables

Variable	Description
-R0 reproductionNumber	basic reproduction number R0
-String countryOfFirstAppearance	Country of first appearance

b) Methods

Method	Description
+Covid19 ()	Constructor. Set variables according to Table 1.
+SevereLevel severeLevel(boolean isVaccinated)	Determine severe level after infected by virus. **Return** Severe level. Consult Table 2. for possible return value.
+int minimumInfectionSpread (int n)	Return minimum number of people (in total) that will get infected by the virus if no one get vaccinate at n

	time(s) (see Figure 1). Example: Covid-19 base has R_0 = 2-3, only min will be selected, hence 2 because R_0 min = 2. Then minimumInfectionSpread (3) is 2+4+8 = 14 Note: ● to write a to the power of b in Java, use Math.pow(a.b). ● you can assume that n is never lower than 1.
+String toString()	Return "Covid19"
+ getter/setter for everything	

3.2.4. **public class Delta** /* You must implement this class from scratch*/

This class represents a mutate version of the base Covid19. <u>Your score will also depend on how much code you can re-use</u>.

a.) Variables

Variable	**Description**
-int spikeProtein	Number of mutations on the spike protein

b.) Methods

Method	**Description**
+ Delta ()	Constructor. Consult Table 1 for the values to be set.
+SevereLevel severeLevel(boolean isVaccinated)	Determine severe level after infected by virus. Return Severe level. Consult Table 2. for possible return value.
+int minimumInfectionSpread (int n)	Return minimum number of people (in total) that will get infected by the virus if no one get vaccinate at n

	time(s) (see Figure 1). Example: Delta has R_0 = 5-7, only min will be selected, hence 5 because R_0 min = 5. Then minimumInfec-tionSpread (2) is 5+25 = 30 Note: ● to write a to the power of b in Java, use Math.pow(a.b). ● you can assume that n is never lower than 1.
+String toString()	Return "Delta"
+getter/setter for everything	

3.2.5. **public class Hospital** /* You must implement this class from scratch*/

This class represents a hospital. Patients can be admitted, and report can be generated by this class. A more organized code will receive more marks.

Note: this class uses class Covid19 and Delta. You must finish those classes before you can test this class.

a.) Variables

Variable	**Description**
-ArrayList<Patient> patients	List of admitted patient.

b.) Methods

Method	**Description**
+Hospital()	Initialize patient Arraylist to empty.
+void admit (String firstname, String lastname, String id, String disease, boolean isVacinated)	Admit a patient, with details from the pa-rameters, to the hospital.

	Note: disease being a String here is intentional.
+ArrayList<Patient> getCovidPatients (SevereLevel s)	Return a list of Covid19 patients (including Delta) with a specified severe level. Data must be in the same order as in the "patients" list.
+getter for its variable.	

3.3 package test.grader

3.3.1. **public class Test** /*ALREADY PROVIDED*/

This class is for testing the objects you wrote. You can use this class to do whatever you want; it will not be checked.

3.4 package app

3.4.1. **public class Application** /*ALREADY PROVIDED*/

This class create an instance Hospital get data from mock up data, Call admit then print the report.

4. Score Criteria

- The scoring criteria is listed below. There are 3 JUnit files.
- The total score of this section is 30, will be scaled down to 5. **The failure to reuse code, as mentioned in the instruction for each class, will get mark deducted from these test results**.

4.1 HospitalTest (Total: 6)

a) testAdmit (3)

b) testGetCovidPatientsSevereIllness (1)

c) testGetCovidPatientsMildOrLess (1)

d) testGetCovidPatientsLess (1)

4.2 Covid19Test (Total: 9)

a) testConstructor (2)

b) testGetSevereLevel (2)

c) testMinimumInfectionSpread (4)

d) testToString (1)

4.3 DeltaTest (Total: 10)

a) testConstructor (3)

b) testGetSevereLevel (2)

c) testMinimumInfectionSpread (4)

d) testToString (1)

4.4 UML Diagram of Hospital, Covid19, and Delta. Save the UML Diagram as png or jpg in your project directory. (5)

Question 3: (60 minutes, 10.30-11.30)

1. Instruction

1) Create Java Project named "**2110215_Midterm_Q3**".
2) Copy all folders in **"Q3"** 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) **BaseCard** (package logic.card)
   b) **RobotCard** (package logic.card)
   c) **MageCard** (package logic.card)
4) Make UML class diagram for classes in package logic.card, using any tool you choose. Save its picture file in your logic.card folder.
5) JUnit for testing is in package test.

2. Problem Statement: Card Duelist

(This question is based on the game Inscryption by Daniel Mullins Games, released in October 2021.)

We create a card game again called Card Duelist. This game is very similar to Card Fighter but it's more like the original game. This time there is no special attack effect in different cards, but different ways to summon it.

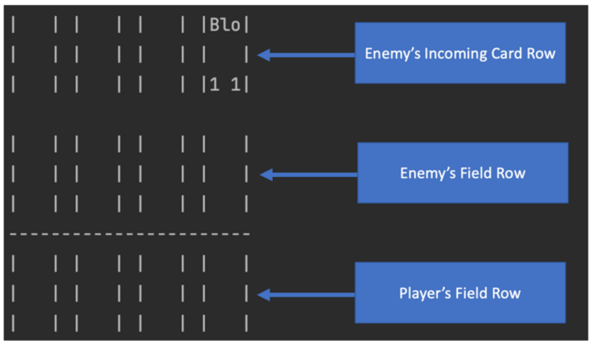
(This image of the game is not entirely accurate to the game of this question)

2.1 Game Component

1) Player

- At the start of the game, each player will have a damage point set to 0. If the player's damage point is equal to or more than opponent's damage point by 5 points, you win. You lose if the opposite happens.

2) Field



```
|   | |   | |   | |Blo|
|   | |   | |   | |   |
|   | |   | |   | |1 1|

|   | |   | |   | |   |
|   | |   | |   | |   |
|   | |   | |   | |   |
-----------------------
|   | |   | |   | |   |
|   | |   | |   | |   |
|   | |   | |   | |   |
```

- Player can summon a unit card from their hand to a field, each player has 4 spaces to place his/her units (Player use lower row, opponent use middle and upper row) middle row is current enemy cards and upper row is incoming enemy cards row. You **can't** replace a card on the board with different card from hand if the card does not say so.

3) Deck

- Contains as many cards as you want, you can get more card by visiting some place.

4) Card

- All card has Name, Attack power, Health point.
- Player may play a unit card on to a field on any of 4 spaces, but the player has to follow the mechanic of the card.
- Unit card on the field will attack a unit in the same column once on player's turn, dealing damage equal to its attack power. If it overkills, it will deal the excess damage to the incoming enemy card. And if there is no unit on the same column, the unit will deal damage to opposite player.
- When a unit card takes damage from unit's attack or damage spell and health point reduces to 0, that unit card will be removed from the field.
- There are 3 different types of cards.

4.1) Orb Card

- A card used for the set-up before playing Mage card. It has no play condition.

4.2) Robot Card

- This card uses energy as its playing cost.

4.3) Mage Card

- This card can be played if there is an orb card with element type as this card.
- This card's attack power will be increase by 1 for each orb card with same element type.

2.2 Game Flow

- This game is rogue-like game. Player can pick a place to go. For example, to get more card, to get an upgrade for card, or encounter enemy.
- When player encounters the enemy a card game phase will commence.
- At the start of the game, the player shuffles his deck and draw 4 cards from deck. While opponent will place some card in incoming cards row.

1) Start with the player turn, the player draws a card.
2) The player can choose to play as many cards as he/she wants.
3) When the player finished, cards will attack from left to right.
4) Start the opponent's turn, the card in incoming cards row will slide down to the opponent's field. Then, execute attack from left to right.
5) Opponent may place card in incoming cards row.
6) Repeat until win condition is met by any player.

3. Implementation Detail

The class package is summarized below.

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

3.1 Package logic.card /* You must implement some class in this package */

3.1.1 class BaseCard /* You must implement this class from scratch*/

This class is a base class for all kind of Cards. It contains all common elements which a card should have.

Field

Name	Description
- String name	Name of the card.
- int power	Power or attack point of the card.
- int health	Heath of the card.

Method

Name	Description
+ BaseCard(String name, int power, int health)	This is the Constructor. Set the parameter using proper setter
+ void play(Player player)	This method is called when the card is played in the **position** for the **player**. This method will perform action depending on each type of Cards.
+ boolean canPlay(Player player)	This method returns true if the card can be played according to the rule and returns false when the card is illegal to play. The rules are different for each type of Cards. No need to check if the field is full.
+ int attack(BaseCard target)	This method is called when this card attacks target. It will decrease target's health by its power but not lower than 0. And **return excess damage.** If the attack does not deal excess damage, return 0. E.g., "Power 5 card" attack "Health 2 Card" will return 3
+ void setPower(int power)	Setter for power. If power is less than 0, set it to 0
+ void setHealth(int health)	Setter for health. If health is less than 0, set it to 0
+ getter-setter for all fields.	

3.1.2. class OrbCard /*ALREADY PROVIDED*/

This class is a type of Card. This card does not have condition to be played. Use to summon mage card.

Field

Name	Description
- Element orbType	Element type of the card.

Method

Name	Description
+ OrbCard(String name, int power, int health, Element orbType)	This is the Constructor. Set the information of the super class. Set orbType as one given in the parameter.
+ void play(Player player)	This card does not have any action when played.
+ boolean canPlay(Player player)	This card does not have special conditions and can always be played. So, this will return true.
+ getter-setter for all fields.	

3.1.3. class RobotCard /* You must implement this class from scratch */

This class is a type of Card. This card's play condition is player must have enough energy and use energy according to its cost.

Field

Name	Description
- int energyCost	Energy cost of the card.

Method

Name	Description

+ RobotCard(String name, int power, int health, int energyCost)	This is the Constructor. Set the information of the super class. Set energyCost as one given in the parameter.
+ void play(Player player)	Subtract **player** energy by this card's **energy cost. Player's energy cannot be negative.**
+ boolean canPlay(Player player)	Return true if **player** have enough energy to play this card. Otherwise, return false.
+ void setEnergyCost(int energyCost)	Setter for Energy cost. If Energy cost is less than 0, set it to 0
+ getter-setter for all fields.	

3.1.4. class MageCard /* You must implement this class from scratch*/

This class is a type of Card. This card can only be played if there is orb card with same element type with this card. When play this card, this card power increase by 1 for each orb card with same element type on player's field.

Field

Name	Description
- Element mageType	Element type of the card.

Method

Name	Description
+ MageCard(String name, int power, int health, Element mageType)	This is the Constructor. Set the information of the super class. Set mageType as one given in the parameter.
+ void play(Player player)	Increase this card's **attack power** by number of orb card with the same element on player's field.

+ boolean canPlay(Player player)	Return true if **player** have orb card with the same element on player's field. Otherwise, return false.
+ getter-setter for all fields.	

3.2 Package logic.game

3.2.1. enum Element /*ALREADY PROVIDED*/

This enum contains element type for orbs and mages.

3.3 Package logic.player

3.3.1 class Player /*ALREADY PROVIDED*/

This class contains a method which will be used for storing player's data.

Field

Name	Description
- int damage	Damage that player have taken.
- int energy	Amount of energy that player has.
- ArrayList<BaseCard> deck	Player's deck
- ArrayList<BaseCard> hand	Player's hand
- ArrayList<BaseCard> field	Player's field

Method

Name	Description
+ Player(ArrayList<BaseCard> deck)	This is the Constructor. Set deck as one given in the parameter.
+ boolean playCard(int indexHand, int position)	This method played the card that in player's hand at **indexHand** to the selected **position** on player's field. If it can be played, remove that card from player's hand then return true, otherwise do nothing, and return false.

+ void drawCard()	This method draws first card from deck to player's hands.
+ getter-setter for all fields.	

3.4 Package application

3.4.1 class Main /*ALREADY PROVIDED*/

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.

3.5 Package test.student /*ALREADY PROVIDED*/

This package provides JUnit test for each of your class.

4. Score Criteria

The maximum score for the problem is 35 and will be rounded down to 5.

4.1 Implement Class BaseCard correctly = 12 points.

4.2 Class OrbCard:	**= 4 points**
testConstructorLegalValue	= 1 point
testConstructorIllegalValue	= 1 point
testAttackNoExcess	= 1 point
testAttackWithExcess	= 1 point
4.3 Class RobotCard:	**= 8 points**
testConstructorLegalValue	= 1 point
testConstructorIllegalValue	= 1 point
testSetEnergyCost	= 1 point
testCanPlay	= 1 point
testPlay	= 1 point
testPlayNoEnergy	= 1 point
testAttackNoExcess	= 1 point
testAttackWithExcess	= 1 point
4.4 Class MageCard:	**= 6 points**
testConstructorLegalValue	= 1 point
testConstructorIllegalValue	= 1 point
testCanPlay	= 1 point

testPlay	= 1 point
testAttackNoExcess	= 1 point
testAttackWithExcess	= 1 point

4.5 UML = 5 points

All classes in package logic are present (correct variables and methods)	= 2 points
Inheritance and Abstract structures are correct	= 3 points