Final Exam, Faculty of Engineering, Chulalongkorn University
Course ID: 2110215 Course Name: Programming Methodology I
Second Semester, Date: 13 May 2021 Time: 13.00-16.00 (Online exam)

Instructions

1. Your projects must be submitted on MyCourseVille (please see "How to Submit").
2. Documents and files are allowed. Internet search is allowed (^_^).
3. No communication with another person is allowed.
4. Student must submit files before the time expires.
5. Any student who does not obey the regulations listed below 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. **(หมายความว่า แค่ "สงสัย จากตัวโค้ดที่ทำ" ว่าลอกกัน อาจารย์มีสิทธิให้ F และพักการเรียนได้เลย)**
    b. With explicit evidence for cheating, student will receive an F in that subject and will receive an academic suspension for 1 year. (**หมายความว่า ถ้าเห็นว่ากำลังลอกเพื่อนอยู่ตอนนั้น ก็ F และพักการเรียน 1 ปี**)

Important Rules

- It is a student's responsibility to check the file. If it is corrupted or cannot be open, there is no score.

** Noted that Access Modifier Notations are listed below*

+ (public)
# (protected)
- (private)
<u>Underline</u> (static)
Italic (abstract)

Set-Up Instruction

- Set workspace to your usual workspace.
- All your exam files for each question must be in the project folder for that question.

How to Submit

There are 3 questions (hence 3 projects) in this exam.

- In your workspace directory, zip all 3 projects' folders into a single zip file called yourID.zip (for example, 6333001121.zip).
- Submit the zip file on MyCourseville.
- Make sure you do not include other files apart from files for each question. A project that is too big in size may result in your submission NOT getting saved.

Scoring (Total 15 points)

- Part1 = 5 points
- Part2 = 5 points
- Part3 = 5 points

Part 1: Interfaces

1. Objective

1) Be able to use Interfaces.

2. Instruction

1) Create Java Project named "**2110215_Final_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) **Mafia** (package role.derived)
   b) **Detective** (package role.derived)
   c) **Mayor** (package role.derived)
   d) **GameLogicUtility** (package logic)
4) Make UML class diagram for classes in package role.base and role.derived, using ObjectAid.
5) **JUnit for testing is in package test.**

3. Problem Statement: Silence Mafia

Figure 1: werewolf game

During quarantine, it is hard to group and play board game with friends, so we create Werewolf-like simulation game called Silence Mafia. This time, we have re-balanced some rules and re-structured the code. There are 2 sides in Silence Mafia like in werewolf: The Mafia and the others. The Mafia's win condition is to remains as last 2 players, the others' one is to vote the Mafia out.

The game has 2 phases daytime and nighttime. During daytime, all players have to vote each other, a player who has been the most voted will be removed at the end of that daytime. No player will be removed if the most voted players are more than one player.

During nighttime, players whose role can take action will use their abilities.

There are 3 roles in the game: Mafia, Detective and Mayor. Mafia is the only role which is in Mafia side.

During nighttime, when Mafia take action, he/she can kill a player (even him/herself). In the daytime, Mafia can vote too but he/she does not have any special action during daytime.

For Mayor, Mayor cannot do anything at nighttime, but during daytime Mayor can choose a player and that player cannot vote for that round.

For Detective, at nighttime, Detective can check a player whether he/she is the Mafia or not. During daytime, he can check a player whether he/she is the Mayor or not.

When the game starts, you will be assigned as a character in the preset player list which contains a Mafia, a Mayor and 6 Detectives (8 players). The game always starts at daytime.

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 role.base

4.1.1 class Player /*This class is already implemented for you*/

This class is a base class for all roles of Player. It contains all common elements which a player must have.

Field

Name	Description
- String playerName	Name of the player. It is used to identify the player.
- int votedScore	Score that the player has been voted.
- boolean votable	If true, the player can vote in the current round. If false, the player cannot vote in the current round.

Method

Name	Description
+ Player(String playerName)	This is the Constructor. Call resetVotedScore method to set the vote scored to 0 and set votable as true
+ void vote(Player targetPlayer)	If this Player is votable, targetPlayer votedScore will be plus by 1. If the player is not votable, print out the reminder text that the player cannot vote this round.
+ void resetVotedScore()	Reset the votedScore by setting votedScore as 0. Set votable to true.
+ getter-setter for all fields.	

4.1.2. Interface DayActable /*This is already implemented for you*/

Method

Name	Description
+ abstract void dayAction(Player targetPlayer)	Method to take an action on a target player in the daytime.

4.1.3. Interface NightActable/*This is already implemented for you*/

Method

Name	Description
+ abstract void nightAction(Player targetPlayer)	Method to take an action on a target player at nighttime.

4.2 Package role.derived /* You must implement this entire package from scratch */

4.2.1. class Mafia /* You must implement this class from scratch */

This class is a role of Player which can act during nighttime. Mafia can kill a player (even him/herself) at nighttime.

Method

Name	Description
+ Mafia(String playerName)	/* FILL CODES */ This is the Constructor. Set the information of the super class.
+ void nightAction(Player targetPlayer)	/* FILL CODES */ Use method GameLogic.getInstance().removeVictimPlayer() to remove targetPlayer from the player list and get the removed player (MUST see code in GameLogic). Then, print out to confirm to all players - **removed playerName + " will be a dead body when day comes"**. (Reminder sentence is available in

	final_part1_text.txt)

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

This class is a role of Player which can act during daytime. Mayor cannot do anything at nighttime, but during daytime Mayor can choose a player that cannot vote that round.

Method

Name	Description
+ Mayor (String playerName)	/* FILL CODES */ This is the Constructor. Set the information of the super class.
+ void dayAction(Player targetPlayer)	/* FILL CODES */ Mayor can choose a player that cannot vote this round. Set votable of targetPlayer as false.

4.2.3. class Detective/* You must implement this class from scratch */

This class is a role of Player which can play an action during both day and nighttime. Detective can check a player whether he/she is the Mafia or not at nighttime. During daytime, Detective can check a player whether he/she is the Mayor or not.

Method

Name	Description
+ Detective (String playerName)	/* FILL CODES */ This is the Constructor. Set the information of the super class.
+ void nightAction(Player targetPlayer)	/* FILL CODES */ During nighttime, Detective can check a player if he/she is the Mafia. If he/she is the Mafia print his/her playerName + " is the

	Mafia boss" (Reminder sentence is available in final_part1_text.txt) If he/she is not the Mafia print his/her playerName + " is not the Mafia boss" (Reminder sentence is available in final_part1_text.txt)
+ void dayAction(Player targetPlayer)	/* FILL CODES */ During daytime, Detective can check a player if he/she is the Mayor. If he/she is the Mayor print **his/her playerName + " is the Mayor"** (Reminder sentence is available in final_part1_text.txt) If he/she is not the Mayor print **his/her playerName + " is not the Mayor"** (Reminder sentence is available in final_part1_text.txt)

4.3 Package logic

4.3.1. class GameLogic/*This class is already implemented for you*/ Provided methods and variables MUST NOT be changed!

This class is the main game system. This class provide player list and game rules. The game will be run via this class.

Fields

Name	Description
- <u>GameLogic instance</u>	Instance that represents GameLogic class. This implementation confirms that we have only one GameLogic.
- ArrayList<Player> playerList	ArrayList containing all players in the game.
- boolean isGameEnd	True if the game ended.
- boolean isNightTime	True when it is nighttime. False when it is daytime.

Method

Name	Description
+ GameLogic()	This is the Constructor. Initialize the game by initializeGame method
+ void initializeGame()	Initialize playerList and add players to it. Set isGameEnd and isNightTime as false.
+ Player removeVictimPlayer(String targetName)	Remove and return player whose playerName is same as targetName from playerList. If there is no player whose is same as targetName return Detective that name "No one"
+ Player removeVotedPlayer(String targetName)	Remove and return the only player whose voted score points the most. If there is more than one player has the most point in voted score, return Detective that name "No one"
+ void resetAllVotes()	Reset all players' votedScore
+ void checkGameEnd()	This method is for checking if the game has ended If number of Mafia is greater than or equal to Non-Mafia players, set isGameEnd as true, and print out "Mafia win the game" If the Mafia is not in playerList, set isGameEnd as true, and print out "Detective and Mayor win the game"
+ void voting(ArrayList<Player> targetPlayers)	Players in playerList will vote to a player who is in the same position in targetPlayers arraylist
+ void doNightAction(ArrayList<Player> targetPlayers)	Call doNightAction from utility class.

+ void doDayAction(ArrayList<Player> targetPlayers)	Call doDayAction from utility class.
getter/setter for all fields	

4.3.2. class GameLogicUtility /* You must implement this class from scratch */

This class contains utility methods that will be used in GameLogic.

Name	Description
+ void doNightAction(ArrayList<Player> targetPlayers)	/* FILL CODES */ For all players in GameLogic.getInstance().getPlayerList() if he/she can act at nighttime, take a night action against target player in the same position in targetPlayers. For example, if getPlayerList() is [A,B,C] targetPlayers is [D,E,F] A will take a night action (if he can) on D. B will take a night action (if he can) on E. C will take a night action (if he can) on F. (Even if any player cannot take night action, targetPlayers MUST be the same length as GameLogic.getInstance().getPlayerList()).
+ void doDayAction(ArrayList<Player> targetPlayers)	/* FILL CODES */ For all players in GameLogic.getInstance().getPlayerList(), if he/she can act at daytime, take a day action to target player in the same position in targetPlayers. For example, if getPlayerList() is [A,B,C] targetPlayers is [D,E,F] A will take a night action (if he can) on D.

	B will take a night action (if he can) on E. C will take a night action (if he can) on F. (Even if any player cannot take day action, targetPlayers MUST be the same length as GameLogic.getInstance().getPlayerList()).

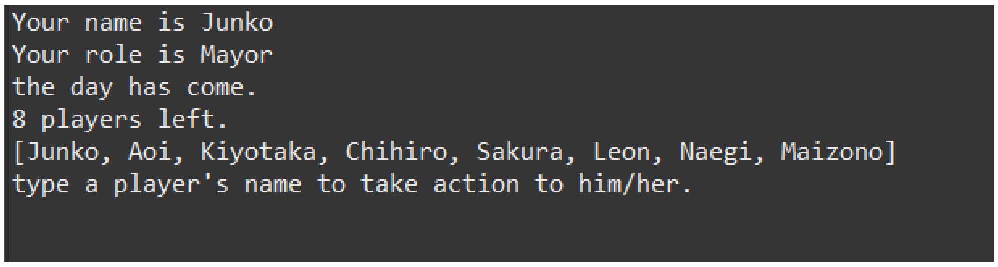
4.4 Package main

4.4.1 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.

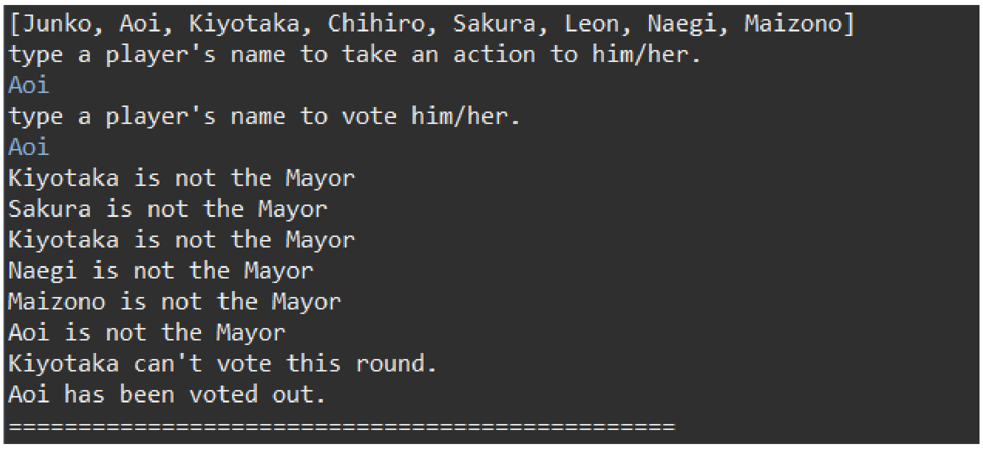
5. Finished Code Run Example (this is one of possible runs)

5.1. Start the game



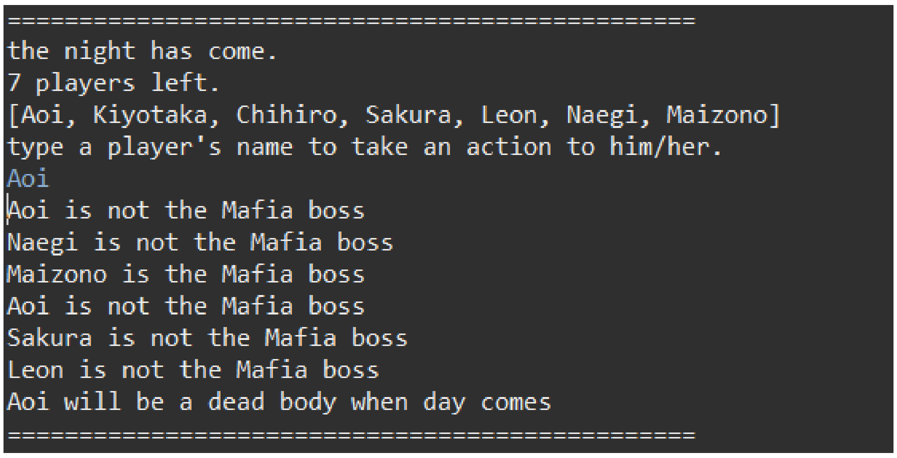
```
Your name is Junko
Your role is Mayor
the day has come.
8 players left.
[Junko, Aoi, Kiyotaka, Chihiro, Sakura, Leon, Naegi, Maizono]
type a player's name to take action to him/her.
```

5.2. Day time interface: typing a player name to take action and vote him/her. All Detectives and Mayor actions can be publicly seen.



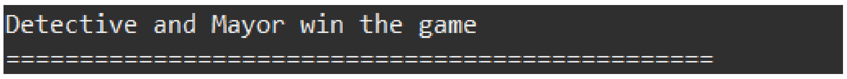
```
[Junko, Aoi, Kiyotaka, Chihiro, Sakura, Leon, Naegi, Maizono]
type a player's name to take an action to him/her.
Aoi
type a player's name to vote him/her.
Aoi
Kiyotaka is not the Mayor
Sakura is not the Mayor
Kiyotaka is not the Mayor
Naegi is not the Mayor
Maizono is not the Mayor
Aoi is not the Mayor
Kiyotaka can't vote this round.
Aoi has been voted out.
================================================
```

5.3. Night time interface: typing a player name to take action to him/her. All Detectives and Mafia actions can be publicly seen.



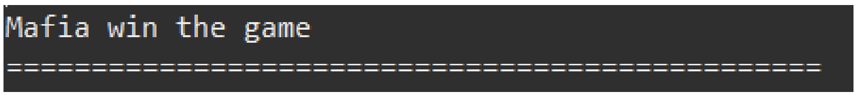
```
================================================
the night has come.
7 players left.
[Aoi, Kiyotaka, Chihiro, Sakura, Leon, Naegi, Maizono]
type a player's name to take an action to him/her.
Aoi
Aoi is not the Mafia boss
Naegi is not the Mafia boss
Maizono is the Mafia boss
Aoi is not the Mafia boss
Sakura is not the Mafia boss
Leon is not the Mafia boss
Aoi will be a dead body when day comes
================================================
```

5.4. Game ending interface (Detective and Mayor win)



```
Detective and Mayor win the game
================================================
```

5.5. Game ending interface (Mafia win)



```
Mafia win the game
================================================
```

6. Score Criteria

The maximum score for the problem is 20 and will be scaled down to 5.

6.1 Class Mafia:	**= 4 points**
Constructor	= 1 points
nightAction	= 3 points
6.2 Class Detective:	**= 7 points**
Constructor	= 1 points
dayAction	= 3 points (1.5 for each test case)
nightAction	= 3 points (1.5 for each test case)
6.3 Class Mayor:	**= 4 points**
Constructor	= 1 points
dayAction	= 3 points
6.3 Class GameLogicUtility:	**= 4 points**
All test cases	= 4 points (2 for each test case)
6.4 UML:	**= 1 points**

Part 2: JavaFX GUI

1. Objective

1) Students are able to implement GUI using JavaFx.

2. Instruction

1) Create a Java Project named “**2110215_Final_Part2**”.
2) Copy folders inside “**toStudent/Part2**” to your project directory src folder (except **vmArgument.txt**).
3) Example command line parameter is given in file “**vmArgument.txt**”
4) You are to implement the following classes (detail for each class is given in section 3 and 4)
   a) TopControlPane (package gui)
   b) BottomControlPane (package gui)
   c) Main (package application)

3. Problem Statement: Tanjiro Kamado's Two Years of Training

In this problem, you are assigned to create a Java application to control Tanjiro's action to train (with the lowest number of clicks).

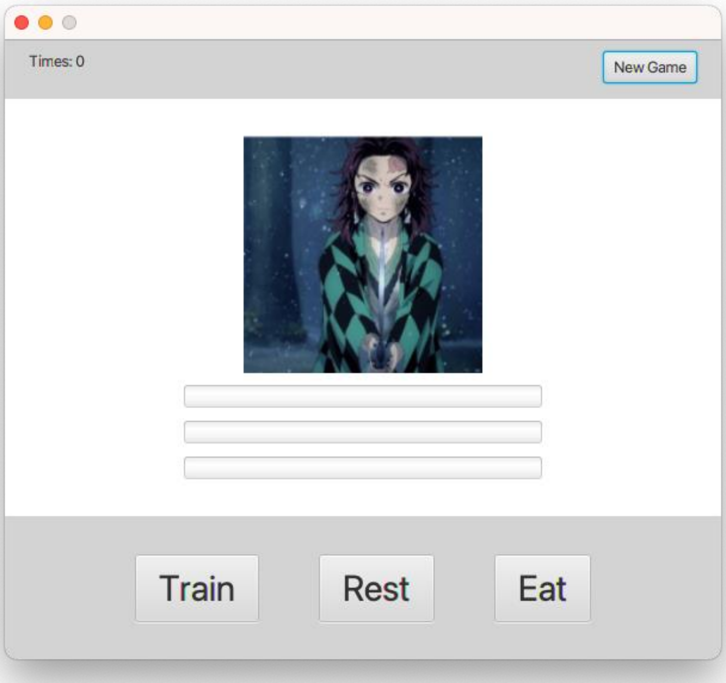


Figure 1: Sample screenshot of the application.

Actions in the Java application

Action 1: Rest button using REST action

- The Rest progress bar (second bar) will be increased by 0.1.
- The statusImage will change to the rest image (shown in Figure 2).

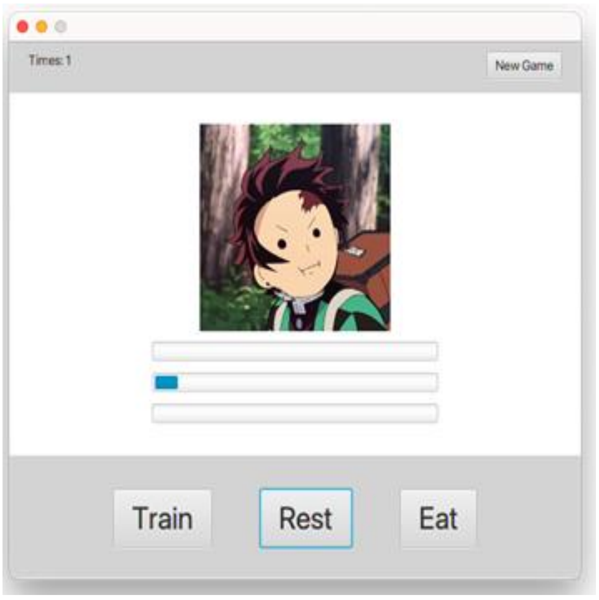


Figure 2: Pressing Rest Button.

Action 2: Eat button using EAT action

- The Eat progress bar (third bar) will be increased by 0.1.
- The statusImage will change to the eat image (shown in Figure 3).

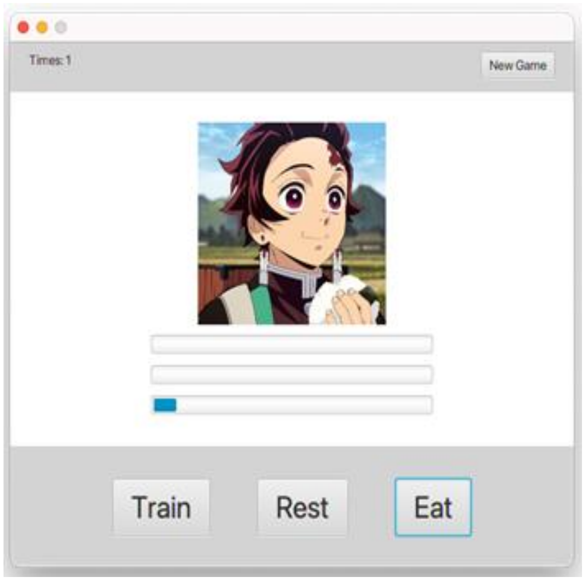


Figure 3: Pressing Eat Button.

Action 3: Train button using TRAIN action

- If the rest value >= 0.2 and the eat value >= 0.2, the Train progress bar (first bar) will be increased by 0.1.
- If train progress bar is increased, the rest and eat progress each gets consumed by 0.2.
- If train progress bar is increased, the statusImage will change to the train image (shown in Figure 4).

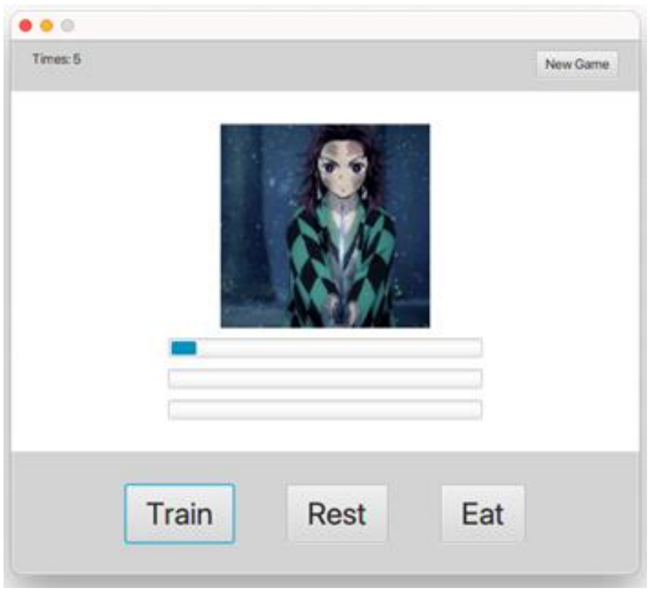


Figure 4: Pressing Train Button.

Additional suggestion:

- The value of each progress bar is a float between 0.0 and 1.0
- Every click will increase “Times” label by 1.

4. Implementation Detail

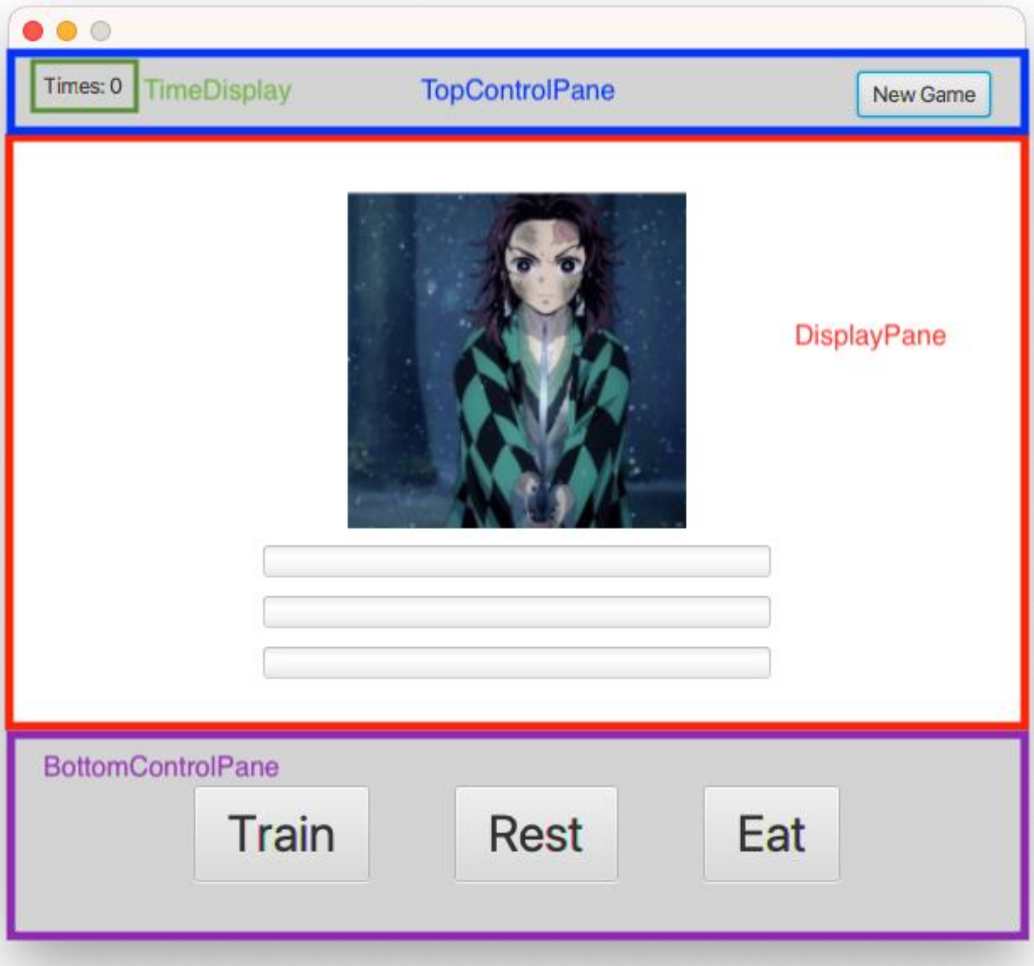


Figure 5: Detail of the GUI.

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. public class **ButtonLogic**: This class contains a useful button method to complete some part in the application. This class is already provided.

Method

Name	Description
+ void *bottomControlPaneOnClick* (String c)	This method should be called by button in BottomControlPane, it will update TimeDisplay and call action in DisplayPane.
+ void *topControlPaneOnClick()*	This method should be called by button in TopControlPane, it will set TimeDisplay to 0 and reset DisplayPane.

4.2 Package gui

4.2.1. public class **TimeDisplay**: This class represents a Pane that can be shown time label. This class is already provided.

Field

Name	Description
- Label timeLabel	Representing the time label.
- int timeValue	Time value in this TimeDisplay.

Constructor

Name	Description

+ TimeDisplay(int number)	Constructor method, which is already provided.

Method

Name	Description
+ void addTime(int number)	This method is already provided.
+ getter/setter for each field.	These methods are already provided.

4.2.2. public class **TopControlPane**: This class represents a BorderPane that can group the essential components.

Field

Name	Description
- TimeDisplay timeDisplay	Representing the TimeDisplay.
- Button newGameButton	Used to reset game logic.

Constructor

Name	Description
+TopControlPane()	/* FILL CODES */ Constructor method. Initialize with the following specifications: - Initialize super class. - set preference height to 50.

	- set preference width to 600. - set padding with Insets(10,20,0,20). - initialize timeDisplay with 0. - initialize newGameButton and set mouse click event with proper method **(see class ButtonLogic)**. - set timeDisplay to the left of TopControlPane by using setLeft method in BorderPane class. - set newGameButton to the right of TopControlPane by using setRight method in BorderPane class. - set background with BackgroundFill(Color.LIGHTGRAY, null, null).

Method

Name	Description
+ getter/setter for each field.	/* FILL CODES */ You can implement any appropriate getter/setter.

4.2.3. public class **DisplayPane**: This class represents a VBox that can group the essential components. This class is already provided.

Field

Name	Description
- ImageView statusImage	Representing status image.

- String trainUrl	Url for the train image.
- String restUrl	Url for the rest image.
- String eatUrl	Url for the eat image.
- ProgressBar skillBar	Representing the progress bar for skill.
- ProgressBar staminaBar	Representing the progress bar for stamina.
- ProgressBar foodBar	Representing the progress bar for food.

Constructor

Name	Description
+DisplayPane()	This method is already provided.

Method

Name	Description
- void initImageUrl()	This method is already provided. Suggestion: you may need to change the path for your picture files in the code.
- void initProgressBar()	This method is already provided.
- void initStatusImage()	This method is already provided.
+ void act(String action)	This method is already provided.

+ void resetDisplayPane()	This method is already provided.
+ getter/setter for each field.	These methods are already provided.

4.2.4. public class **BottomControlPane**: This class represents a HBox that can group the essential components.

Field

Name	Description
- Button trainButton	Representing the train button.
- Button restButton	Representing the rest button.
- Button eatButton	Representing the eat button.

Constructor

Name	Description
+BottomControlPane()	/* FILL CODES */ Constructor method. Initialize with the following specifications: - Initialize super class. - Initialize all buttons by using initAllButton(). - set preference height to 120. - set preference width to 600.

	- set spacing by 50. - set alignment to Pos.CENTER. - set background with BackgroundFill(Color.LIGHTGRAY,null,null). - add all buttons to this pane.

Method

Name	Description
+ void initAllButton()	/* FILL CODES */ For each button, you must implement the following: (1) initialize buttons with "Train", "Rest", and "Eat", respectively for each button. (2) set text font on button with size 30 and "Arial", which is font name. (3) set mouse click event with proper method. HINT: ButtonLogic class in the logic package will be useful to this method.
+ getter/setter for each field.	/* FILL CODES */ You can implement appropriate getter/setter.

4.3 Package application

4.3.1. public class **Main**: This class contains main method. It is an entry point of the application.

Field

Name	Description
- TopControlPane tcp	Representing the top control pane.
- DisplayPane dp	Representing the display pane.
- ButtomControlPane bcp	Representing the bottom control pane.

Method

Name	Description
+ void start(Stage stage)	/* FILL CODES */ At the TODO section, you must fill the code to complete the start method with the following: (1) set scene with the appropriate scene object. (2) set resizable with false. (3) set title with “Tanjiro’s training”. (4) show the stage.
+ getter/setter for each field.	These methods are already provided.

Scoring Criteria (10 points, will be scaled down to 5)

- In the top control pane, the time label is on the very left (1 mark).
- In the top control pane, the new game button is on the very right (1 mark).
- When the new game button is clicked, the time label is reset to zero and all progress bars are reset to zero (1 mark).
- Bottom panel has the Train, Rest, and Eat button (1 mark).
- The Train, Rest, and Eat buttons spread out well (1 mark).
- Train button is clicked (1 mark):
  - The time label is increased.
  - If the Rest and Eat progress bar each does not reach 2 units, no training can happen.
  - Otherwise, the training happens. The Train Progressbar is increased by 1 unit. The Rest and Eat progress bar are reduced by 2 units. The picture changes to the Train image.
- Rest Button is clicked (1 mark):
  - The time label is increased.
  - The Rest progress bar value is increased by 1 unit.
  - The picture changes to the Rest image.
- Eat Button is clicked (1 mark):
  - The time label is increased.
  - The Eat progress bar value is increased by 1 unit.
  - The picture changes to the Eat image.
- Padding is correct (1 mark).
- Main window has the correct name and cannot be resized (1 mark).

Part 3: Java Thread

1. Objective

1) Be able to implement Java thread to make a program responsive.

2. Instruction

1) Create a new Java Project named “2110215_Final_Part3”.
2) Copy folder “**application**” and “**data**” (in “**toStudent/Part3**”) into your project src folder.
3) Copy folder “**res**” into your project and make it a resource folder.
4) Fix the code (detail shown below) inside.

3. Problem Statement: **Between Us**

To end this exam with a throwback to the very first question of midterm exam. Between Us (don’t ask, we are running out of idea on how to plagiarize detective games) is a game where crewmates are running around and eventually get killed while red alarm constantly goes off. In this program, we are going to simulate this game that allows multiple running crewmates and alarm to appear on the scene at the same time without slow down.

main.java is a javafx application that shows the room where crewmates can run, die, and get annoyed by red alarm.

Click on “Create Crewmate” button (Figure 1) will create a crewmate that will run around the room(pane) below and it will bounce-off pane wall (Color and speed of each crewmate will be random).

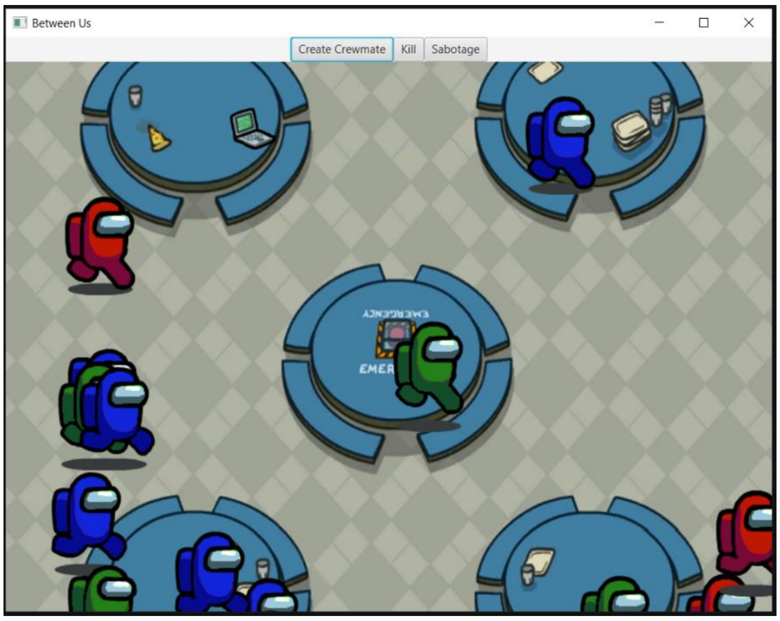


Figure 1: Click "Create Crewmate" screenshot.

Click on "Kill" button (Figure 2) will kill one of the crewmates (First Come First Die), and it will stop moving. If there is no crewmate alive, nothing will happen.

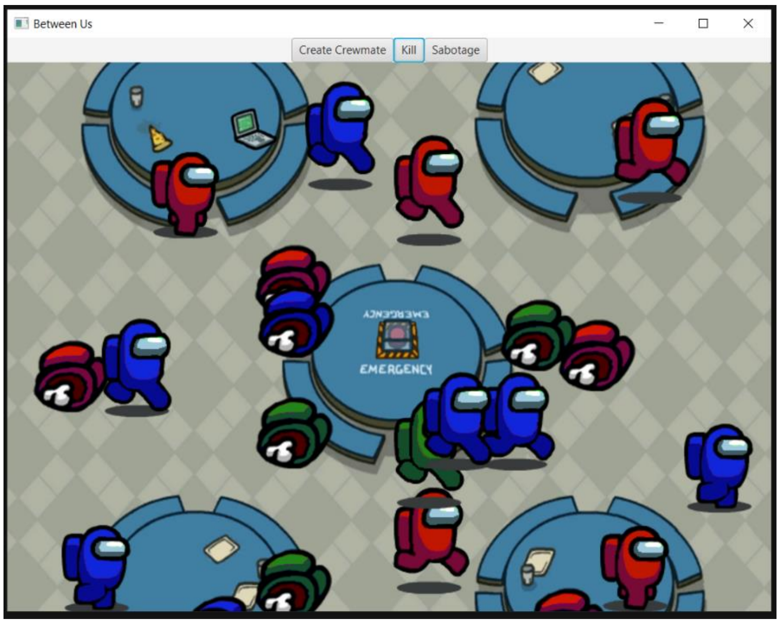


Figure 2: Click "Kill" screenshot.

Click on "Sabotage" button (Figure 3) will make alarm go off (a red light that beeps on and off). Click this button again to stop the alarm.

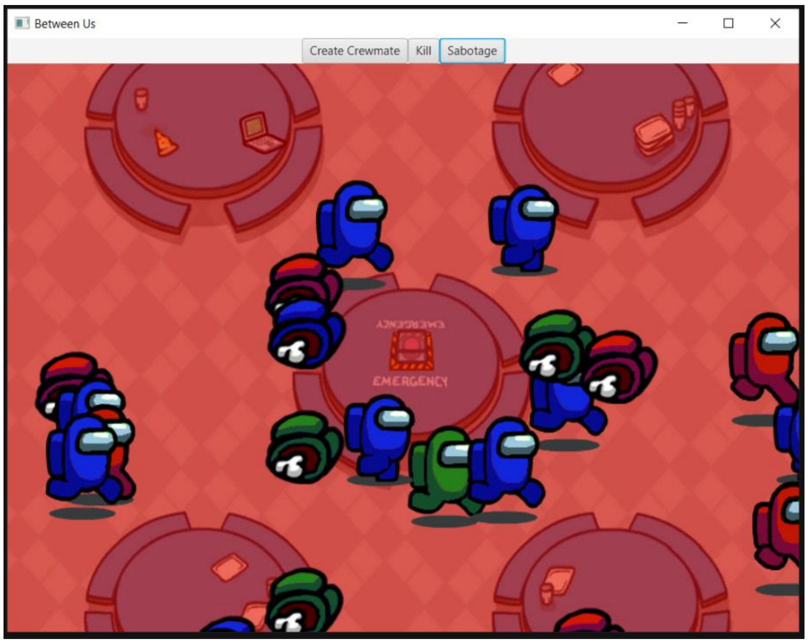


Figure 3: Click “Sabotage” screenshot.

A video file is given to you to show how a finished application should look like!!

Try out main.java and click on “Create Crewmate” or “Sabotage” button. You will see that application will not respond at all (Figure 4). This is because application tried to draw a crewmate running animation or red alarm indefinitely. But without thread implemented, other functions like button and pane don’t get a chance to run.

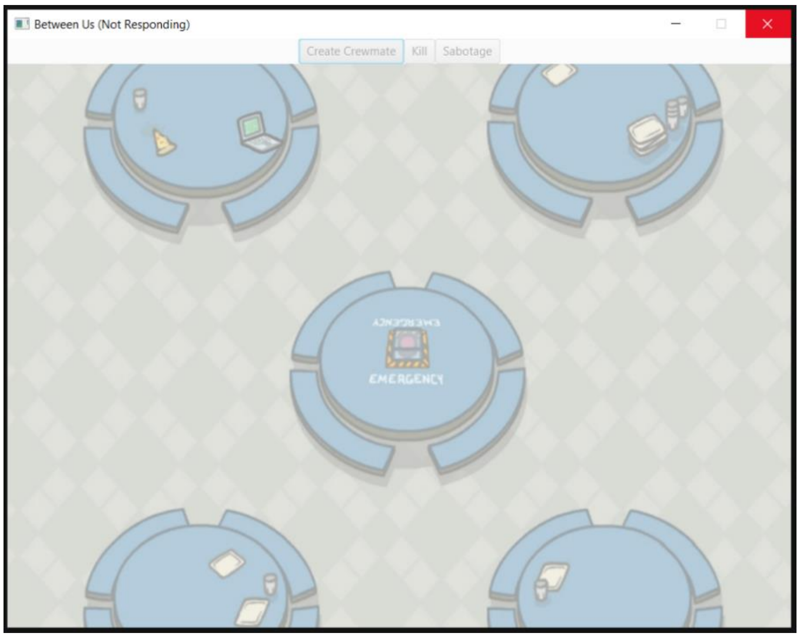


Figure 4: Non-respond screenshot.

Your task is to use threads to allow this crewmate animation and alarm to be shown on the pane. **For a crewmate, it always has the same speed no matter how many other lines are on the pane.** The application might be slower with high number of crewmates, we will only look at 1-20 crewmates cases.

4. Implementation Detail

The code is given in **ThreadMain.java**. You do not need to know the details of the user interface

There are a total of **3 methods you must implement in ThreadMain.java.**

1. **initalizeNewCrewmate**

2. **updateCrewmateMovement**

3. **initalizeAlarm**

Hint: Both **Crewmate and Alarm in Main.java use ImageView,** which is related to JavaFX.

- You can add your own methods and variables.
- You MUST NOT change the line "Thread.sleep(50);" If you do, you get 0 point.
- If no threads are used to solve this question, you get 0 point.

Main.java, Alarm.java, Crewmate.java, CrewmateField.java, and **SpriteAnimation.java** classes are given, You MUST NOT change anything in These 5 classes. If you do, you get 0 point.

5. Scoring Criteria: **(5 points)**

5.1. Click "Create Crewmate" button once - create a crewmate and show it moving on the pane while the application is still responding **(2 points).**

5.2. After creating more crewmates, the crewmates don't slow down and not causing any exception (from 2 to 20 crewmates) **(1.5 points).**

5.3. Click "Sabotage" button once – make alarm goes off and show it on the pane while the application is still responding and crewmates still running **(1.5 points).**