Final Exam, Faculty of Engineering, Chulalongkorn University
Course ID: 2110215 Course Name: Programming Methodology I
Second Semester, Date: 11 May 2020 Time: 13.00-16.00 (Online Exam)

Instructions

1. Your projects must be submitted as an assignment 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 can be listed below*

+ (public)

# (protected)

- (private)

Underline (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 zip file called yourID.zip (for example, 6233001121.zip).
- Submit the zip file as an assignment 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 30 points, will be scaled to 25 points)

- Part1 = 10 points
- Part2 = 9 points
- Part3 = 11 points

Part 1: Interface (Guildmaster’s Revenge)

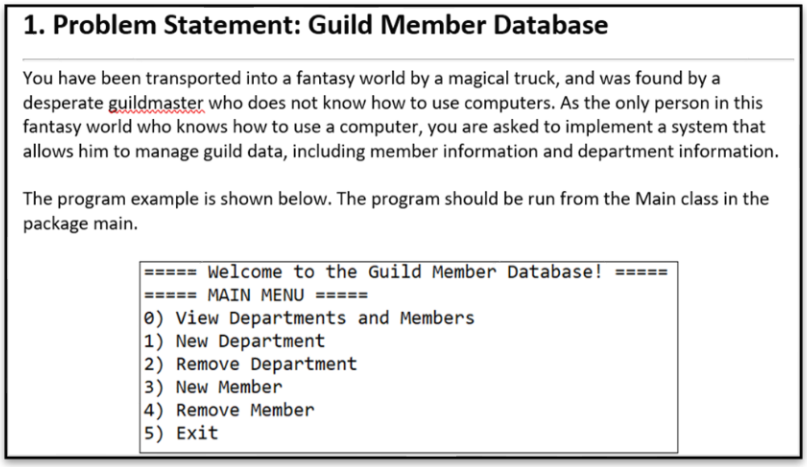
1. Objectives

1) Students are able to implement interfaces and use them with classes.

2. Instructions

1) Create Java Project named “**2110215_2_Final_Part1**” in your workspace.
2) Copy all folders in “**toStudent/Part1**” to your project directory src folder.
3) You are to implement the following classes (details for each class are given in section 4.)
    a) Tank (package fighters.derived)
    b) Wizard (package fighters.derived)
    c) Guildmaster (package fighters.derived)
4) After creating all the required classes, you are required to draw a UML Diagram using ObjectAid. Save it as .png in the root of the project’s folder. **(The UML Diagram is worth 2 points.)**

3. Problem Statement: Foe Emblem: Guildmaster's Revenge



1. Problem Statement: Guild Member Database

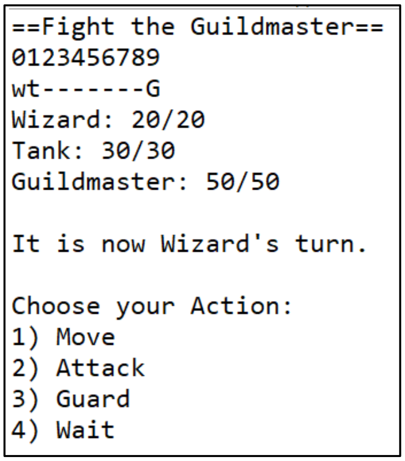
You have been transported into a fantasy world by a magical truck, and was found by a desperate guildmaster who does not know how to use computers. As the only person in this fantasy world who knows how to use a computer, you are asked to implement a system that allows him to manage guild data, including member information and department information.

The program example is shown below. The program should be run from the Main class in the package main.

```
===== Welcome to the Guild Member Database! =====
===== MAIN MENU =====
0) View Departments and Members
1) New Department
2) Remove Department
3) New Member
4) Remove Member
5) Exit
```

The guildmaster from Lab1 has betrayed you and claimed to have written the code by himself. You decide to teach him a lesson. Nobody gets away with taking credit for a software you wrote with your own sweat and blood. However, the Guildmaster has become too powerful to deal with on your own. You need help from a few old friends you met along the way.

1) The wizard: The wizard can shoot piercing magic bolts that pierces through defense from a distance but cannot raise up any shields.
2) The tank: He does not attack, but he has a very strong shield that can block powerful attacks.



```
==Fight the Guildmaster==
0123456789
wt-------G
Wizard: 20/20
Tank: 30/30
Guildmaster: 50/50

It is now Wizard's turn.

Choose your Action:
1) Move
2) Attack
3) Guard
4) Wait
```

You are to implement a battle strategy planner. Your program will be able to do the following:

1) Allow each character to take their turns. For each of your allies, you will be able to choose their actions. However, not all characters are able to do everything. For example, Tank cannot attack and Wizard cannot guard. The actions are as follow:
    a. Move: Move to the right (positive, +) or the left (negative, -).
    b. Attack: Deal damage to a character within this unit's range.
    c. Guard: Raise a shield. Units who can guard have a certain amount of defense that already deducts incoming damage. Raising a shield doubles this defense. A unit who attacks or moves stops guarding.
    d. Wait: Do nothing. If the unit was previously on guard, then the unit continues to be on guard until they move or attack.
2) Allow Guildmaster to automatically fight as an enemy. The Guildmaster's strategy is very simple.
    a. If there is a space directly to the left, the Guildmaster moves left.
    b. If he cannot move to the left, the Guildmaster assumes that there is an enemy to the left and attacks it.
    c. If Guildmaster arrives at cell 0, the battle is lost, and the game terminates.
    d. If Guildmaster's HP reaches 0, the battle is won, and the game terminates.
3) Display every character's position on the map.
    a. The map is displayed as a one-dimensional line with 10 cells.
    b. Each empty cell is represented as a "-".
    c. A cell occupied by a character will be represented as an alphabet. Lowercase for allies and uppercase for enemies.
    d. At the start of the game, the map will look like this:

```
0123456789
wt-------G
```

Meaning that at the start of the game, the wizard is at cell 0, the tank is at cell 1, and the guildmaster is at cell 9.

The program can be run from the class Application.java in the package app.

4. Implementation Detail

The class package is summarized below. Note that only important method that you need to complete AND provided methods that might be useful in completing this application are listed.

- private

# protected

+ public

Static variables and methods are <u>underlined</u>.

4.1 Package fighters.base

ALREADY GIVEN

Everything in this package is already written for you. Only important variables and methods will be listed for you to use.

4.1.1. public abstract class Unit

Variables

Name	Description
# String name	What this unit is called.
# String symbol	The symbol used to represent this unit in the map.
# int maxHealth	The amount of health this unit is spawned with.
# int health	The current amount of health. When it reaches 0, the unit is defeated and is removed from the map.
# int speed	How many steps this unit can move per turn.
# int power	The amount of damage dealt when attacking.
# int range	How far this unit can attack.
# int defense	How much damage this unit can deduct.
# boolean onGuard	Whether or not this unit is on guard.
# int location	Where the unit is currently standing on the map.

# boolean playerControlled	If true, then the player chooses the unit's actions. If false, the unit will be controlled by a simple AI.

Constructor

Name	Description
+ Unit(String name, String symbol, int maxHealth, int speed, int location, boolean playerControlled)	Constructor method. All the following variables are set: Health is set to Max Health. Power, range, and defense are set to 0. onGuard is set to false.

Methods

Name	Description
+ boolean move (int spaces)	Attempts to move to the right (if positive) or the left (if negative). Cannot move into a space occupied by another unit. Cannot move into a space blocked by a unit controlled by an opposing team. Cannot move out of the map. Cannot move 0 spaces. If the move is successful, return true. Otherwise return false.
+ boolean sameTeam (Unit unit)	Returns true if this unit and the given unit's playerControlled are the same value. Returns false otherwise.
+ getter/setter for all variables	Note that by default getters for all ints and Strings start with get… and getters for Booleans start with is…

	For example: getName(), getMaxHealth(), isPlayerControlled()

4.1.2. public interface Attackable

Variables

None.

Methods

Name	Description
+ abstract int attack (Unit e)	Attacks the given unit. Returns the damage dealt, depending on each unit’s damage calculation formula.

4.1.3. public interface Guardable

Variables

None

Methods

Name	Description
+ abstract void guard()	Calls an action to guard. Depending on the unit, guarding for each unit may be different.

4.2 package fighters.derived

TO BE IMPLEMENTED

This entire package must be written from scratch.

4.2.1 public class **Tank**

This class is a unit that can guard, but not attack.

Variables

None.

Constructor

Name	Description
+ Tank(int maxHealth, int speed, int defense, int location)	Initializes the values for the tank. For the values not given in the constructor, they are fixed as follow: Tank's name is always "Tank". Tank's symbol is always "t" (lowercase). Tank's playerControlled is always true.

Methods

Name	Description
+ void guard()	The tank gets on guard and onGuard is set as true.
+ boolean move(int spaces)	When the tank moves, he is no longer on guard. Set onGuard as false, then call the move function from Unit.

4.2.2 public class **Wizard**

This class is a unit that can perform ranged attacks that ignore the target's defense, but cannot guard.

Variables

None.

Constructor

Name	**Description**
+ Wizard(int maxHealth, int speed, int power, int location)	Initializes the values for the wizard. For the values not given in the constructor, they are fixed as follow: Wizard's name is always "Wizard". Wizard's symbol is always "w" (lowercase). Wizard's playerControlled is always true. Wizard's range is always 2.

Methods

Name	**Description**
+ int attack(Unit e)	If the target is in the same team as the wizard or the target's location is out of range, return -1 to signify a failed attack. Otherwise, return the wizard's power directly. Hint: BattleUtils class in the logic package should contain useful static methods for this.

4.2.3 public class **Guildmaster**

This class is a unit that can do both melee attacks and can raise a guard (but never does so), and is controlled by a simple AI.

Variables

None.

Constructor

Name	**Description**
+ Guildmaster(int maxHealth, int speed, int power, int defense, int location)	Initializes the values for the guildmaster. For the values not given in the constructor, they are fixed as follow: Guildmaster’s name is always “Guildmaster”. Guildmaster’s symbol is always “G” (uppercase). Guildmaster’s playerControlled is always false. Guildmaster’s range is always 1.

Methods

Name	**Description**
+ boolean move(int spaces)	No matter what number is inputted, the Guildmaster always attempts to move -1 spaces. (One to the left)

+ void guard()	Since the Guildmaster's AI never guards, this function does nothing.
+ int attack(Unit e)	If the target is in the same team as the guildmaster or the target's location is out of range, return -1 to signify a failed attack. Otherwise, return the damage after having deducted the target's defense (if they have any). Hint: BattleUtils class in the logic package should contain useful static methods for this.

4.3 package logic

ALREADY GIVEN

Everything in this package is already written for you. Only important variables and methods will be listed for you to use.

4.3.1 public class BattleUtils

Variables

None.

Methods

Name	Description
+ int calculateDamage(int power, Unit e)	Static method. If unit e is an instance of Guardable, then e's defense is saved as an int to deduct from the attacker's power. Then, if e is on guard, the deduction is doubled.

	Finally, subtract the final deduction from the power, and return the damage. If it is less than 0, then return 0.
+ boolean validRange(int range, int myLocation, int targetLocation)	Static method. If the absolute of the sum between myLocation and targetLocation is lesser than or equal to range, then it returns true. Otherwise return false.

5. Scoring Criteria

Your work will be run through JUnit test cases similar to the ones you have been provided, and your scores will be based on how many test cases you passed, as well as whether or not you have submitted a correct UML Diagram.

This part is worth a total of 31 points, rounded to 10.

5.1 TankTest (Worth 9 points)

- testTankConstructor (2)
- testTankTakeDamage (1)
- testTankGuard (1)
- testTankGuardBreakFromMoving (1)
- testTankNoAttack (2)
- testTankGuardable (2)

5.2 WizardTest (Worth 9 points)

- testWizardConstructor (2)
- testWizardDealDamage (1)

- testWizardNoFriendlyFire (1)
- testWizardRange (1)
- testWizardNoGuard (2)
- testWizardAttackable (2)

5.3 GuildmasterTest (Worth 11 points)

- testGuildmasterConstructor (2)
- testGuildmasterTakeDamage (1)
- testGuildmasterGuard (1)
- testGuildmasterNoFriendlyFire (1)
- testGuildmasterRange (1)
- testGuildmasterMove (1)
- testGuildmasterAttackable (2)
- testGuildmasterGuardable (2)

5.4 Correct UML Diagram (Worth 2 points)

Part 2: JavaFX GUI

1. Objective

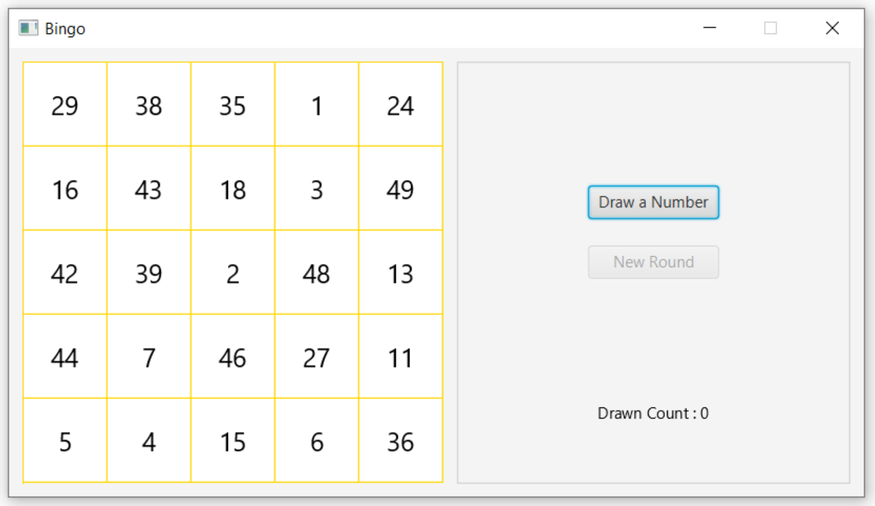
1) Students are able to implement GUI using JavaFx.

2. Instruction

1) Create Java Project named “**2110215_2_Final_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)	NumberSquare	(package gui)
b)	ControlPane	(package gui)

3. Problem Statement: Bingo



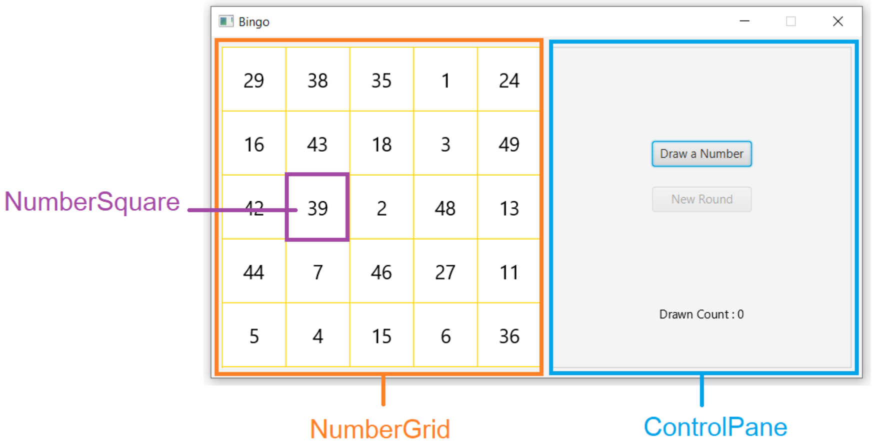
29	38	35	1	24
16	43	18	3	49
42	39	2	48	13
44	7	46	27	11
5	4	15	6	36

Drawn Count : 0

The initial GUI of the application.

You have seen a bingo game with alluring prize. With that alluring prize in mind, you develop a Bingo game to play and count how many times the number have to be drawn for you to bingo in a round. For analysis purpose, you have specified a random seed to random number generator to lock the bingo game results to be the same for each new application run. Possible numbers are 0-50.

4. Implementation Detail



29	38	35	1	24
16	43	18	3	49
42	39	2	48	13
44	7	46	27	11
5	4	15	6	36

Detailed GUI of the Application

The class package is summarized below.

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

4.1 Package gui

4.1.1. public class **NumberSquare**: This class represents a number square.

Field

Name	Description
- int number	The number to display of this square.
- boolean isDrawn	This indicate whether the number of this square is already drawn.
- Text numberText	The Text object to display the number.

Constructor

Name	Description
+ NumberSquare ()	Constructor method. **This method is already provided.**

	Initializes with the following specifications: - Sets the alignment to **CENTER**. - Sets minimum and maximum width and height to 64. - Sets border to **GOLD** color stroke style **SOLID**, corner radii **EMPTY**, right and bottom widths to 1 and top and left widths to 0. - Initializes *numberText*, and set font with size 20.

Method

Name	**Description**
+ void setupNumber (int number)	/* FILL CODE */ This method is called when the square is assigned a new number. Does the following: - Sets the *number* field to the number parameter. - Sets *isDrawn* to false. - Sets text of *numberText* to string of the number parameter. - Sets the **Background** to be **filled** with **WHITE** color. Hint: class **Background** and **BackgroundFill** will be useful.
+ void highlight ()	/* FILL CODE */ Sets the **Background** to be **filled** with **ORANGE** color and sets *isDrawn* to true.
+ getter/setter for each field.	//only necessary ones are already provided.

4.1.2. public class **NumberGrid**: This class represents the number grid with NumberSquares.

This class is already provided.

Field

Name	Description
- ObservableList<NumberSquare> numberSquares	List that contains NumberSquare objects in the grid.

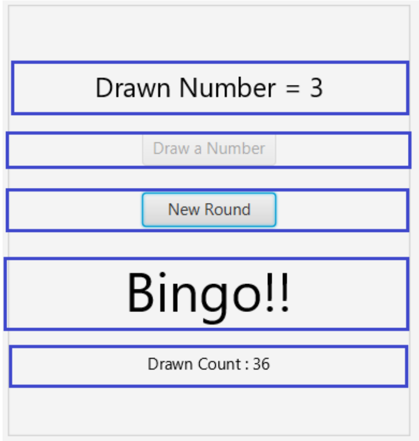
Constructor

Name	Description
+ NumberGrid ()	Constructor method. Initializes with the following specifications: - Sets border to **GOLD** color, stroke style **SOLID**, corner radii **EMPTY**, right and bottom widths to 0 and top and left widths to 1. - Initializes *numberSquares* by using **BingoUtil**.

Method

Name	Description
+ boolean highlightNumber (int drawnNumber)	This method highlights the number square that has the same number as the drawn number. Returns true if the grid is bingo after highlighting a number.
+ ObservableList<NumberSquare> getNumberSquares ()	Getter method for *numberSquares*.

4.1.3. public class **ControlPane**: This class is the pane that contains buttons and information texts. Items in the pane is arranged vertically.



Field

Name	Description
- Text drawnNumberText	The Text for displaying the number drawn.
- Text bingoText	The Text for displaying bingo text when the grid is bingo.
- Text drawCountText	The Text for displaying number drawn count.
- Button drawButton	The button for drawing a number.
- Button newRoundButton	The button for beginning a new round.
- NumberGrid numberGrid	A numberGrid that is updated by this ControlPane.

Constructor

Name	Description
+ ControlPane (NumberGrid numberGrid)	/* FILL CODE */ Constructor method. Sets *numberGrid* field to match the parameter. Then, initializes with the following specifications: - Sets the alignment to **CENTER**.

	- Sets preferred width to 300. - Sets spacing to 20. - Sets border to a border with border stroke **LIGHTGRAY** color, stroke style **SOLID**, corner radii **EMPTY**, border widths **DEFAULT**. - Initializes *drawnNumberText* and set its font with size 20. - Initializes *drawCountText*. - Initializes *bingoText* with initializeBingoText(). (See below) - Initializes *drawButton* with initializeDrawButton(). (See below) - Initializes *newRoundButton* with initializeNewRoundButton(). (See below) - Adds all Text and Button fields to this pane's children in correct order. - Sets *drawnNumberText*'s text and *drawCountText*'s text to beginning of a round texts by using **BingoUtil**.

Method

Name	Description
- void initializeBingoText()	**/* FILL CODE */** - Initializes *bingoText* with text "Bingo!!" - Sets font with size 40 and set visible to false.
- void initializeDrawButton()	**/* FILL CODE */** - Initializes *drawButton* with text "Draw a number". - Sets the button preferred width to 100. - Sets onAction to handle with drawButtonHandler() method. (See below)

- void initializeNewRoundButton()	/* FILL CODE */ - Initializes *newRoundButton* with text "New Round". - Sets the button preferred width to 100. - Sets disable to true. - Sets onAction to handle with newRoundButtonHandler() method. (See below)
- void drawButtonHandler ()	/* FILL CODE */ This method is the handler method for *drawButton*. Does the following: - Draws a random int number by using **BingoUtil** and highlights the drawn number in numberGrid. - If the numberGrid is bingo, set *bingoText* to visible, disable *drawButton* and enable *newRoundButton*. - Updates *drawnNumberText*'s text and *drawCountText*'s text by using **BingoUtil.**
- void newRoundButtonHandler ()	/* FILL CODE */ This method is the handler method for *newRoundButton*. Does the following: - Assigns random numbers to number squares by using **BingoUtil** - Sets *bingoText* to invisible, enable *drawButton* and disable *newRoundButton*. - Sets *drawnNumberText*'s text and *drawCountText*'s text to beginning of new round texts by using **BingoUtil**.
+ getter for each field.	/* FILL CODE */

4.1.4. public class **BingoUtil**: **This class is already provided.** Only useful public methods are shown here.

Method

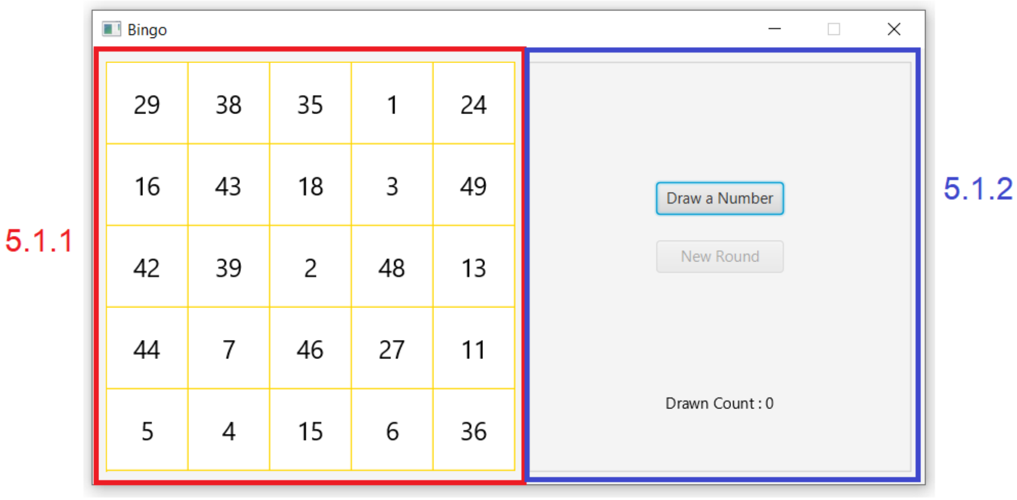
Name	**Description**
+ static void assignRandomNumbers (ObservableList<NumberSquare> numberSquares)	This method assigns random numbers to numberSquares passed in the parameter.
+ static void drawNumber ()	This method returns a random number that has not already drawn in the round.
+ static void updateTextsAfterDrawn (int drawnNumber, Text drawnNumberText, Text drawCountText)	This method update drawnNumberText's text according to drawnNumber and update drawCountText's text according to current drawn count
+ static void setTextsBeginning (Text drawnNumberText, Text drawCountText)	This method set drawnNumberText's text and drawCountText's text at the beginning of a round.

4.2 Package main

4.2.1. public class **Main**: This class contains main method and is entry point of the application. **This class is already provided.**

5. Scoring Criteria (9 points)

5.1 Initial



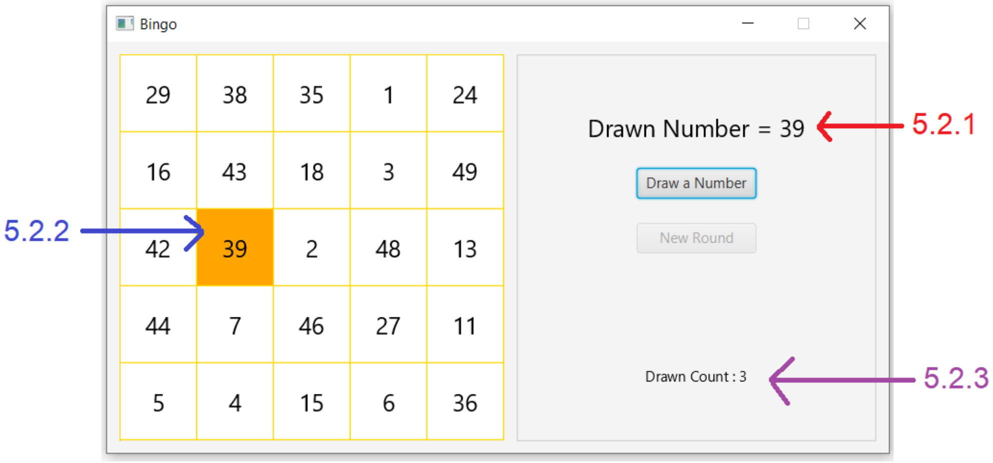
29	38	35	1	24
16	43	18	3	49
42	39	2	48	13
44	7	46	27	11
5	4	15	6	36

Drawn Count : 0

5.1.1. Initial numbers in squares grid are displayed properly. (1 point)

5.1.2. Initial buttons and texts are displayed properly. (1 point)

5.2 Draw



29	38	35	1	24
16	43	18	3	49
42	39	2	48	13
44	7	46	27	11
5	4	15	6	36

Drawn Number = 39

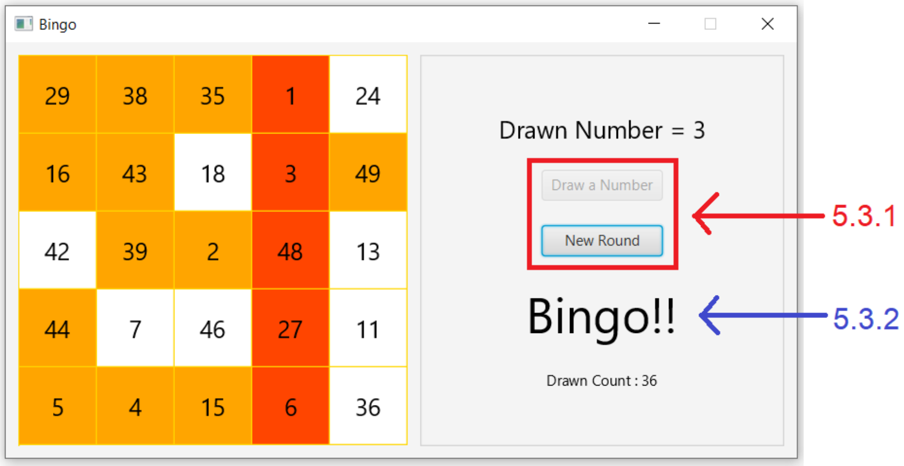
Drawn Count : 3

5.2.1 The drawn number text is displayed properly. (1 point)

5.2.2 The number square is highlighted properly when its number is drawn. (1 point)

5.2.3 The drawn count text is displayed properly. (1 point)

5.3 Bingo



29	38	35	1	24
16	43	18	3	49
42	39	2	48	13
44	7	46	27	11
5	4	15	6	36

Drawn Number = 3

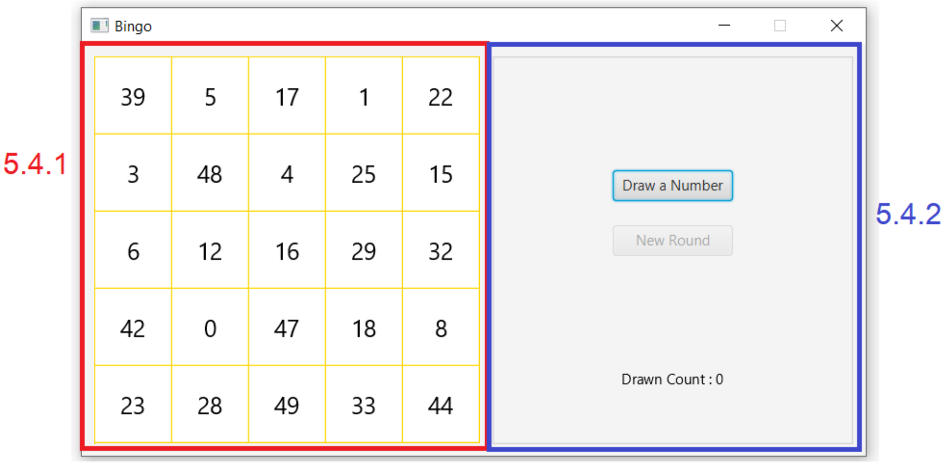
Bingo!!

Drawn Count : 36

5.3.1 Draw button is disabled and new round button is enabled when bingo. (1 point)

5.3.2 Bingo text is displayed when bingo. (1 point)

5.4 New round



39	5	17	1	22
3	48	4	25	15
6	12	16	29	32
42	0	47	18	8
23	28	49	33	44

Drawn Count : 0

5.4.1 New round button assigns new random numbers to squares. (1 point)

5.4.2 New round button enables draw button, disables itself, hides and shows text properly. (1 point)

Part 3: Java Thread

1. Objective

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

2. Instruction

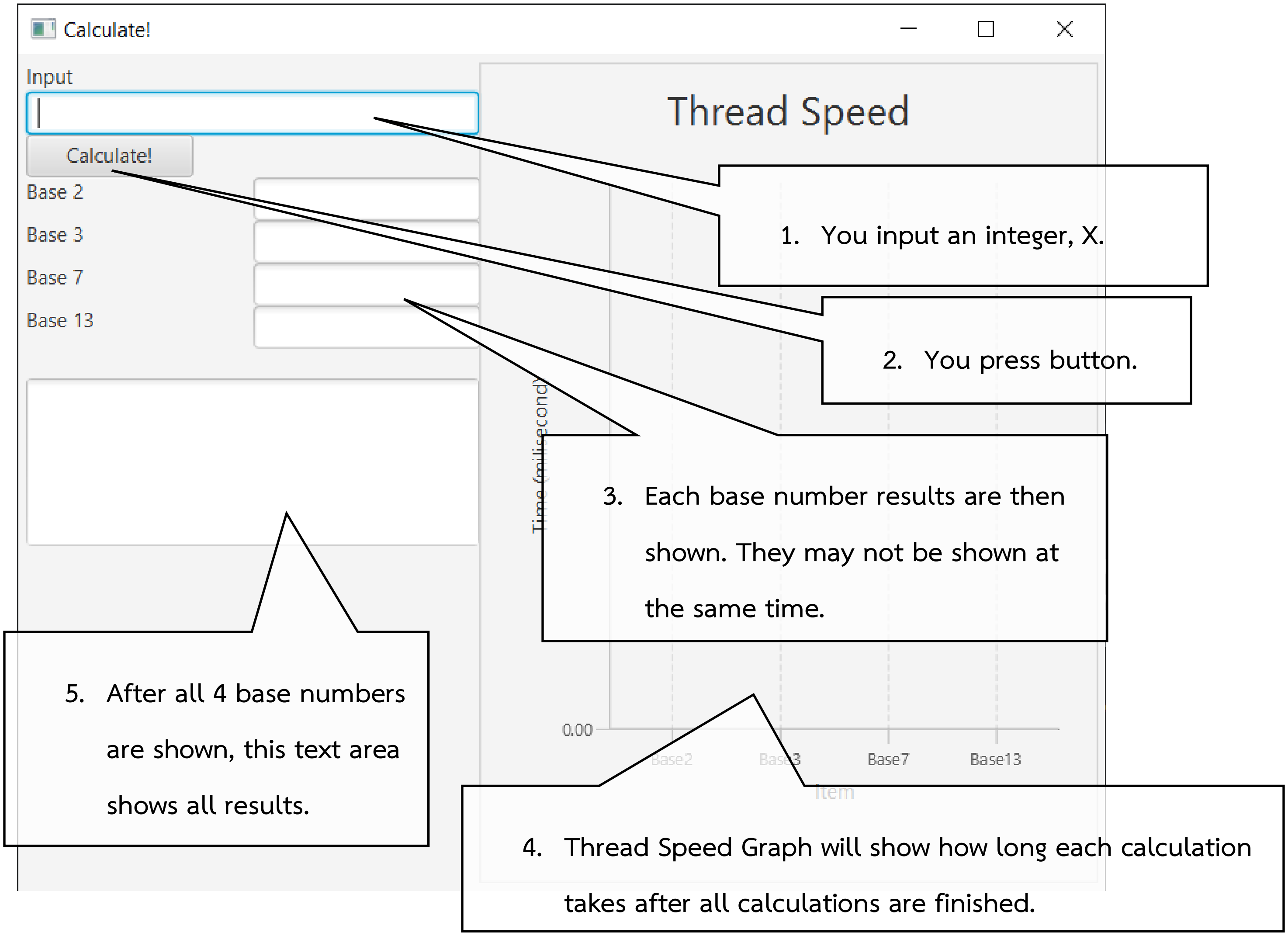
1) Create a new Java Project named “**2110215_2_Final_Part3**”.

2) Copy folder “**BarChart**” and “**main**” (in “toStudent/Part3”) into your project src folder.

3) Fix the code (detail shown below) inside.

3. Problem Statement: Calculating various base numbers.

main.java is an application that calculates the value of different base number (Using **base_X(double x,int base)** method) including base 2, 3, 7, 13. The application works as follows:



1. You input an integer, X.

2. You press button.

3. Each base number results are then shown. They may not be shown at the same time.

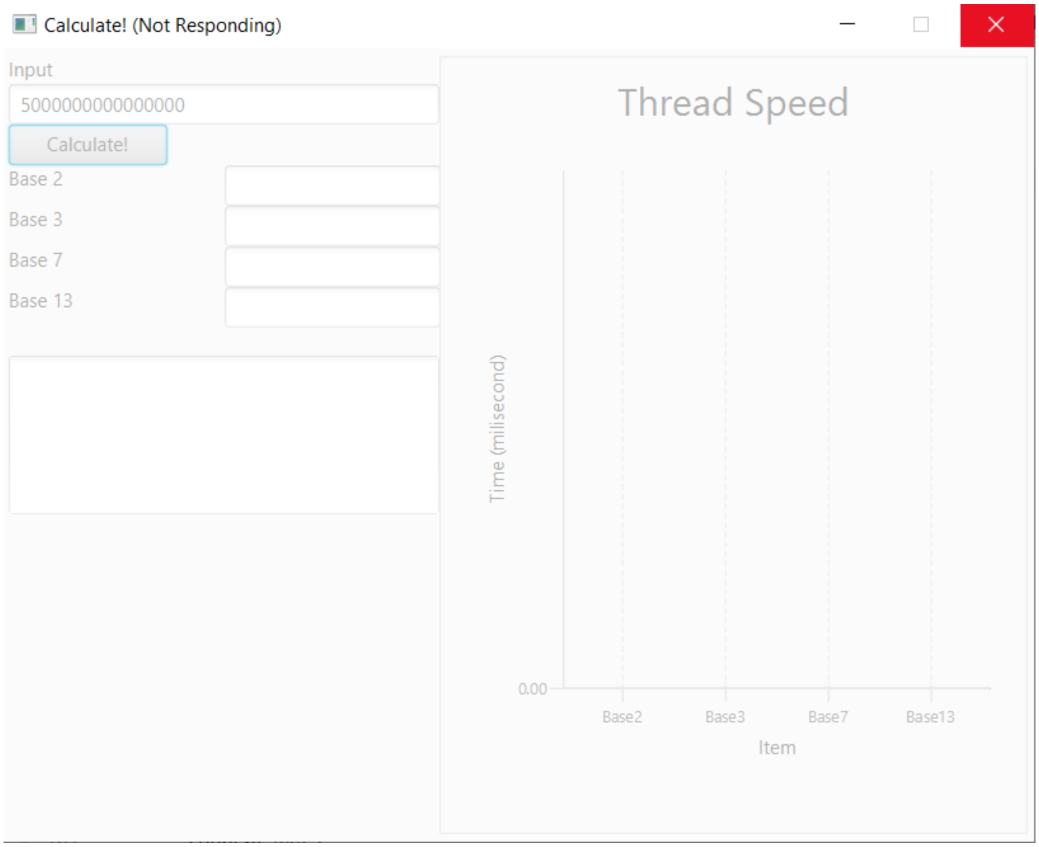
4. Thread Speed Graph will show how long each calculation takes after all calculations are finished.

5. After all 4 base numbers are shown, this text area shows all results.

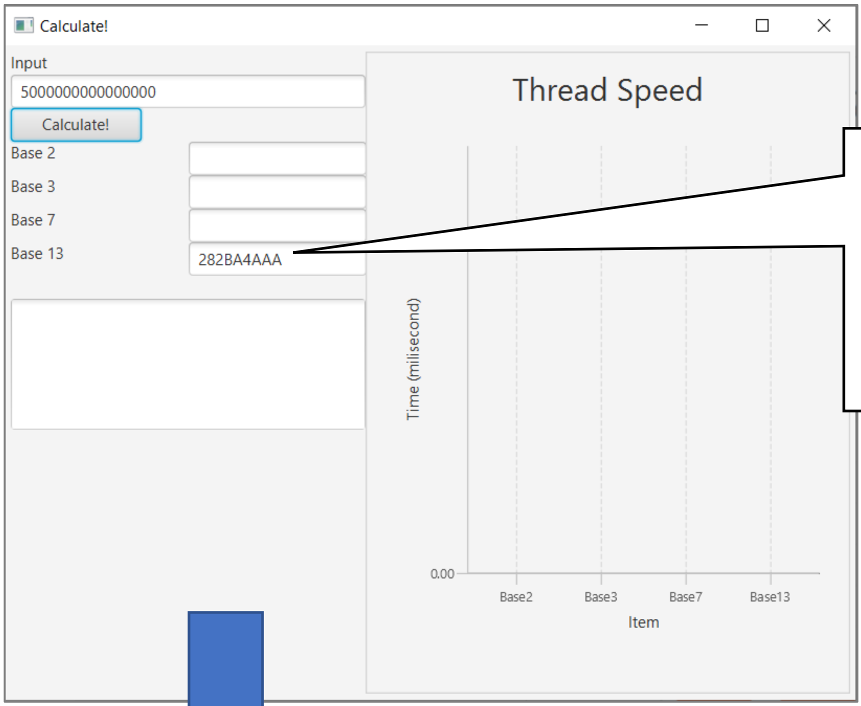
(Chart will show approximately how long each base number calculation should take individually. You don’t have to change any code related to chart or run time. Chart should work the same way in a finished code as in the initial code)

Try out main.java with small input such as 50 or 100. You will see each result shown quickly one after the other.

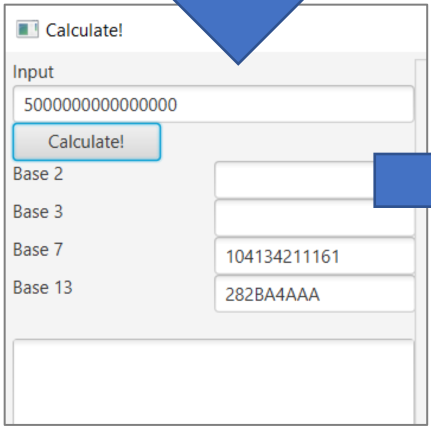
The problem is, when we try a larger number, such as 5000000000000000 (five thousand billon, 5 x 10^{15}), the user interface will stop responding for a while (as shown in the picture below). The Calculate button won’t bounce back.

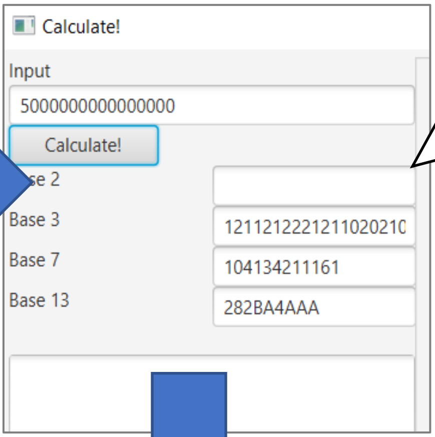

Input
5000000000000000

Your task is to use threads to allow the user interface to respond when Calculate button is clicked. After your fix, pressing Calculate when your input is 5000000000000000 will result in the following:

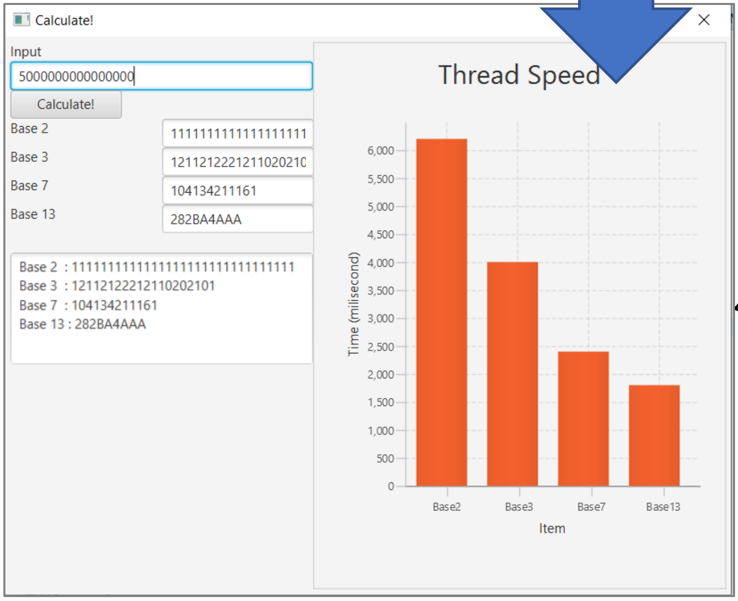


1. Base 13 will be the first to respond. Meanwhile, you can play around with user interface.





2. Then Base 7 will follow, then Base 3. Meanwhile, you can play around with user interface.



```
Base 2  : 1111111111111111111111111111111
Base 3  : 12112122212110202101
Base 7  : 104134211161
Base 13 : 282BA4AAA
```

3. Base 2 results, text area results, and graph results take a while to show. Meanwhile, you can play around with user interface.

4. Scoring Criteria: (11 points)

4.1. When the Calculate button is pressed (when input is large, such as 5000000000000000).

4.1.1. Calculate button bounces back immediately. **(2 point)**

4.1.2.User can click in all text box and text area while waiting for the results. **(2 points)**

4.1.3. After a short time, result is shown on Base 13 text box first. **(1 points)**

4.1.4.Next, result is shown on Base 7 text box after Base 13. **(1 points)**

4.1.5.Next, result is shown on Base 3 text box after Base 7 and 13. **(1 points)**

4.1.6.After some time, result is shown on Base 2 text box after other base numbers. **(2 points)**

4.1.7.When all results are shown, the text area and graph immediately show their results. **(2 points)**

5. Implementation Detail

The code is given in **main.java**. You do not need to know the details of the user interface, just the components that get updated from the calculation are needed.

- You can add your own methods and variables.
- You **MUST NOT change the code of base_X(double x,int base)**. If you do, you get **0 point**.
- If no threads are used to solve this question, you get **0 point**.

BarChartPane.java class is given, **You MUST NOT change anything in BarChartPane.java.** If you do, you get **0 point**.