Exam 2, Faculty of Engineering, Chulalongkorn University
Course ID: 2110215 Course Name: Programming Methodology I
First Semester, Date: 06 November 2020 Time: 9.30-11.30AM

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

Instructions

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

Please sign and submit

Signature (.....)

Important Rules

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

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

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

Set-Up Instruction

- Set workspace to "**C:\temp\progmeth2020_1\Exam2_2110215_(your id)_(FirstName)**" (if not exist, you must create it)
  - For example, "C:\temp\progmeth2020_1\Exam2_2110215_631234521_John"
- All your files must be in the workspace.
- **The code outside of the workspace will not be collected, or graded**

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

- Part1 = 10 points
- Part2 = 10 points

Part 1: Abstract Classes

1. Objective

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

2. Instruction

1) Create Java Project named "**2110215_Exam2_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) **Piece** (package logic)
    b) **Fighter** (package logic)
    c) **GameSystem** (package logic)

4) Make UML class diagram for classes in package logic, using ObjectAid. You must save your diagram as a .png file in the root of the project's folder **(2 marks).**
5) **JUnit for testing is in package test.**

3. Problem Statement: Space chess

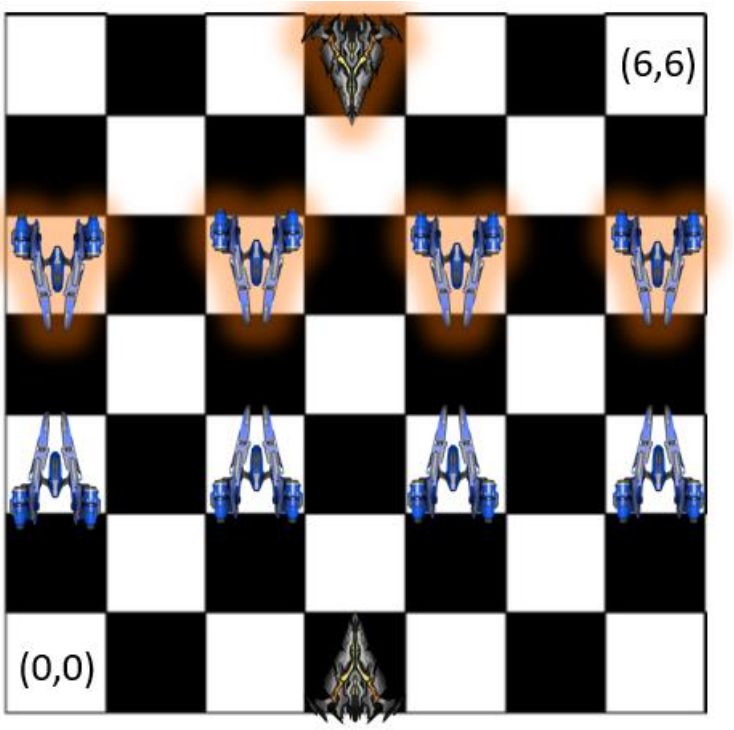

Figure 1 Space chess

Space chess is a fairy chess, chess with the special rules, which is popular among Prog-meth TAs. We have implemented this chess in text game version, but we have no time to finish it, so we want you to help us implement the game.

The rules are less complex than traditional chess. Win condition is capturing opponent's all pieces. The board is borderless. If a piece is on (0,0) and move left for 1 tile, it will be on (6,0). Also, if a piece is on (0,0) and it move back for 1 tile, it will be at (0,6). There are 2 types of pieces, Mothership and Fighter.

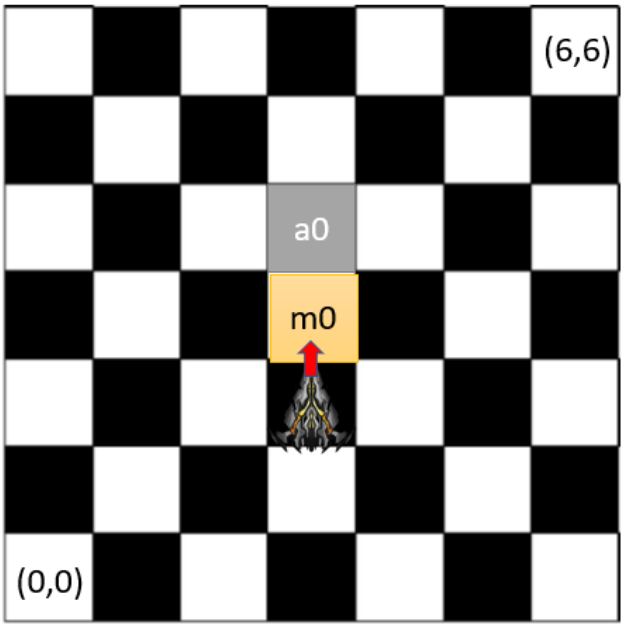

Figure 2 Mothership move patterns

Mothership can move in just 1 pattern like in the m0 tile (0,1) in Figure 2 (the move pattern and attack pattern are relative to the ship's position). However, Mothership move action will not capture any piece and this game allows us to have pieces in the same tile.

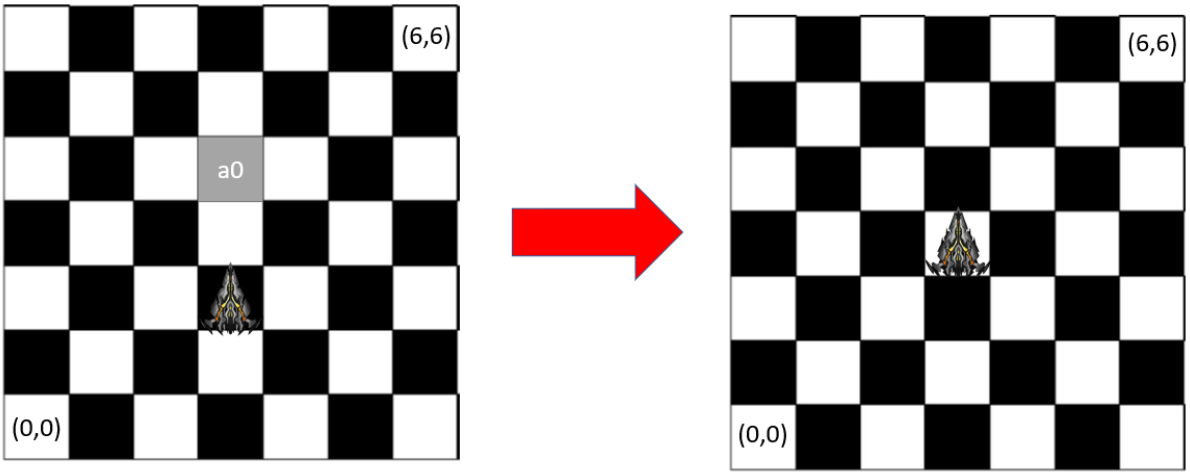

Figure 3 Mothership attack move

Mothership also can attack in just 1 pattern like in the grey tiles (0,2) in Figure 3. After Mothership attacks, it will move in pattern (0,1). In addition, this game allows friendly fire. All pieces which are in the attacked tiles will be removed from the game.

Mothership cannot be promoted.

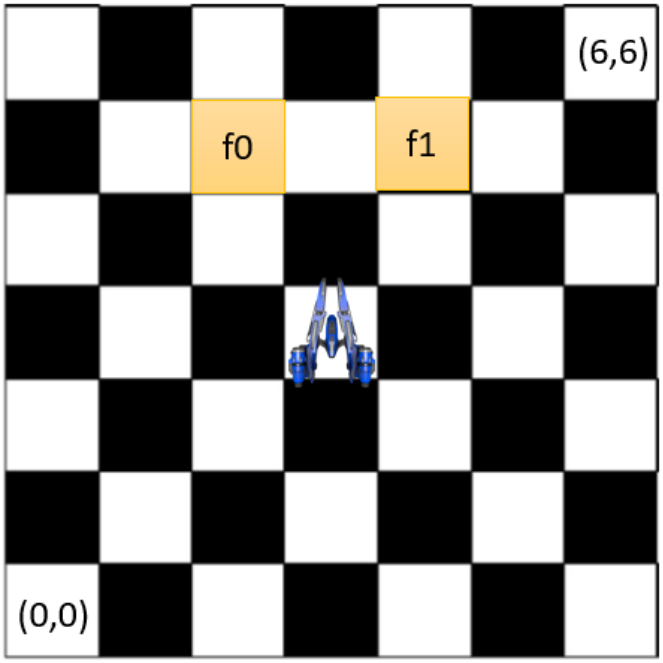

Figure 4 Fighter move patterns

For Fighter, before promotion, Fighter can move in 2 possible patterns like in f0 and f1 tiles ((-1,2), (1,2)). Unlike Mothership, Fighter behavior is like traditional chess pieces. It captures any pieces in the tile it moves to. Fighter must promote if it captures any piece (even its ally piece).

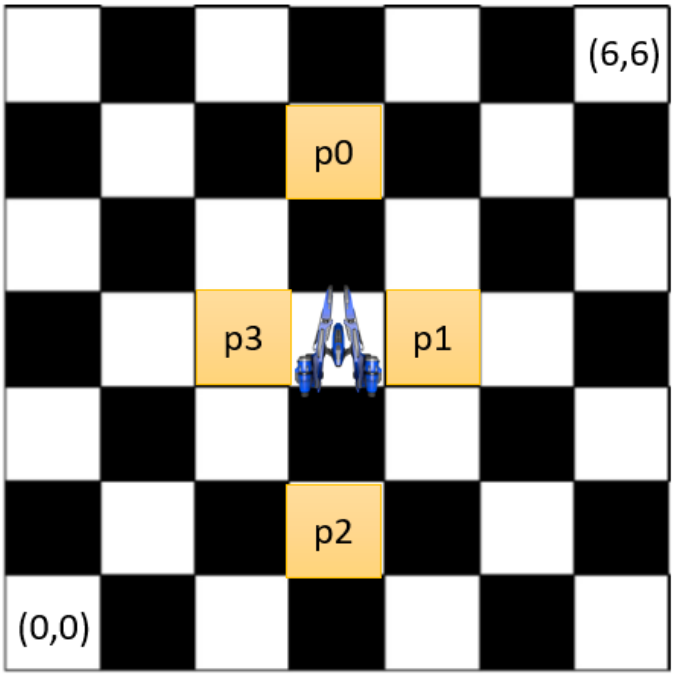

Figure 5 Promoted Fighter move patterns

Promoted Fighter's possible move patterns will change to the positions like in Figure 5 ((0,2), (1,0), (0,-2), (-1,0)). Like normal Fighter, when promoted Fighter moves, it will capture any pieces in the tile it moves to.

4. Implementation Detail

The class package is summarized below.

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

4.1 Package logic

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

This class provided position system of the board of the game.

Field

Name	Description
- int x	Value of X-axis of the position
- int y	Value of Y-axis of the position

Method

Name	Description
+ Position(int x, int y)	Set X and Y of the position
+ boolean equal(Position comparedPosition)	Check that if comparedPosition is equal to or not. return true if they are equal and false for not
+ Position addPositionValue(Position addedPosition)	Add X and Y value from addedPosition to this position and normalized the result before returning
+ void reverse()	Multiply X and Y value by -1
+ void normalizedPosition()	Check that the position is beyond the border of the board or not. if it beyond change the position to its right position if x or y < 0 plus 7 to it if x or y > 0 mod 7 from it
+ getter and setter of all fields	

4.1.2 class PositionList extends ArrayList<Position>

/*This class is already implemented for you*/

This class extends from ArrayList<Position> to provide easier way to add Position to ArrayList.

Method

Name	Description
+ PositionList()	This is the Constructor.
+ void add(int x, int y)	Add a new Position(x,y) to this PositionList.

4.1.3 class Piece /* You must implement some important things in this class */

This class is a base class for all types of Piece. It contains all common elements which a piece should have.

Field

Name	Description
# String name	Name of the piece, use to identify the piece.
# boolean reverse	true if the pieces are in red side, false if they are in white. When this variable is true, this piece's move pattern will be reversed.
# PositionList movePositions	List of possible relative move positions (relative to current position). It stores positions like (0,1), (0,-1), etc.. (-1,2) means moving -1 square in the x-axis and +2 squares up in the y-axis.
# Position currentPosition	Position that represents current position of this piece on the board.

Method

Name	Description

+ Piece(int initialPositionX, int initialPositionY, boolean reverse, String name)	This is the Constructor. Set movePositions to new empty PositionList and set currentPosition to the initial position then check and make sure that current position is in range: (0 – 6, 0 - 6). Set the rest of the fields with their respective values.
+ void move(int movePattern)	/* FILL CODES */ Move the piece to another tile with the pattern specified by a given number. Moving rules are different for each type of pieces.
+ Position attackTargetPosition(int actionPattern)	/* FILL CODES */ Attack and return attack position. Attack rules are different for each type of pieces.
+ setCurrentPosition(int x, int y)	Special setter for currentPosition, Then, normalize current position.
+ getter-setter for all fields.	For currentPosition, there is another setter that uses argument (Position position).

4.1.4. class MotherShip /*This class is already implemented for you*/

This class is a type of Piece. Mothership can be selected to move or attack. Attacking will force it to move in the up-position. Mothership cannot be promoted.

Field

Name	Description
- PositionList attackPositions	PositionList that store attackable positions relative to the current position. Although mothership only has 1 relative attackable position, we use PositionList so that many attackable positions are allowed in the future.

Method

Name	Description
+ MotherShip(int initialPositionX, int initialPositionY,boolean reverse,String name)	This is the Constructor. Set the information of the super class. This also initializes attackPositions and adds Positions to movePositions and attackPositions. • For movePositions, add (0,1). • For attackPositions, add (0,2). If this piece is reversed, use method reverse from class Position to multiply (-1) to all values in attackPositions and movePositions.
+ void move(int movePattern)	Move currentPosition by the given move pattern number. Use method .get(movePattern) to get the relative position in movePositions. Use method .add(Position position) for adding Position values. Then, normalize the current position.
+ Position attackTargetPosition(int actionPattern)	Attack the position that is selected by actionPattern number. Use method .get(actionPattern) to get the relative position to attack from attackPositions. Then, add the relative position with current position to get the absolute target position. Then, normalize target position. Then move the mother ship in pattern number 0. Return the target position.
getter for attackPositions	

4.1.5. class Fighter/* You must implement this class from scratch */

This class is a type of Piece. Fighter will capture any pieces in the tile it moves to. If it captures any piece, even its ally, it must be promoted.

Field

Name	Description
- boolean promoted	true if Fighter has been promoted
- PositionList promotedMovePositions	PositionList that store move sets after promotion

Method

Name	Description
+ Fighter (int initialPositionX, int initialPositionY, boolean reverse,String name)	This is the Constructor. Set the information of the super class. This also initialize promotedMovePositions and add Positions to movePositions and promotedMovePositions. • For movePositions, add (-1,2), (1,2). • For promotedMovePositions add (0,2), (1,0), (0,-2), (-1,0). All positions must be added in the order specified above. If this piece is reversed, use method reverse from class Position to multiply (-1) to all values in promotedMovePositions and movePositions. Also, set promoted to false.
+ void move(int movePattern)	Set currentPosition to the position updated from adding the current position with the position from the given move pattern number. Use method .get(movePattern) to get the position in movePositions. If this Fighter has already been promoted, use

	promotedMovePositions instead of movePositions. Do not forget to normalize current position.
+ Position attackTargetPosition(int actionPattern)	Attack the relative position that is selected by actionPattern number. According to the rules, this will return the absolute position that the Fighter will move to. You can use method addPositionValue from class Position to help. **Do not forget to normalize target position and move this Fighter to the target position before return.**
getter/setter for promoted and promotedMovePositions	You do not need a setter for promotedMovePositions.

4.1.6. class GameSystem**/* You must implement 1 method in this class */**

This class is the main game system. This class provide board status and game rules. Game will play via this class.

Field

Name	Description
- ArrayList<Piece> allWhitePieces	ArrayList containing all white pieces in the game
- ArrayList<Piece> allRedPieces	ArrayList containing all red pieces in the game
- boolean gameEnd	True if the game ended

Method

Name	Description
+ GameSystem()	This is the Constructor. Initialize the Piece ArrayLists and all pieces. Add all the pieces to its ArrayList. Set gameEnd to false.
+ void printBoardStatus()	Print all pieces' names and position

+ boolean removeOtherPieces(Piece user, Position position)	Remove all pieces in the position that is not user. return true there is any pieces remove by this method and return false if there is not.

+ void action(Piece piece, boolean attack, int actionPattern)	/* FILL CODES */
	This method is for moving or attacking a piece on the board. Select relative position to move or attack by using actionPattern.
	
	First, this method must check what type the piece is.
	
	If it is a MotherShip,
	• If attack is true,
	  ○ use method .attackTargetPosition(action) to get the attack target position.
	  ○ Remove all pieces in the same position as the target position by using .removeOtherPiece() method in this class.
	• If attack is false, move piece to the position indicated by (actionPattern) by using .move(actionPattern) method.
	
	If it is a Fighter, you do not need to consider boolean attack.
	• Use method .attackTargetPosition(actionPattern) to get the attack target position.
	• Remove all pieces in the same position as attack target position by using .removeOtherPiece() method in this class.
	• If there are any pieces removed by this method, promote the Fighter.

getter/setter for gameEnd, allWhitePieces and allRedPieces	(Setters for allWhitePieces and allRedPieces are not needed.)

4.2 Package main

4.2.1 class Main

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

5. Finished Code Run Example

5.1. Start the game (white always starts first!)

```
Welcome to space chess!
wm1: (3, 0)
wf1: (0, 2)
wf2: (2, 2)
wf3: (4, 2)
wf4: (6, 2)
rm1: (3, 6)
rf1: (0, 4)
rf2: (2, 4)
rf3: (4, 4)
rf4: (6, 4)
this is white turn.
Avaiable Pieces:
<0> wm1
<1> wf1
<2> wf2
<3> wf3
<4> wf4
```

5.2. Choosing mothership has 2 choices (attack or move: user selects 1 or 2).

```
this is white turn.
Avaiable Pieces:
<0> wm1
<1> wf1
<2> wf2
<3> wf3
<4> wf4
0
<1> attack/<2> move
```

5.3. Fighter that is not promoted can move in pattern 0 and 1 only.

```
Avaiable Pieces:
<0> wm1
<1> wf1
<2> wf2
<3> wf3
<4> wf4
1
select the number of move pattern.
3
please select 0 - 1
0
```

5.4. Game end when a move/attack destroys the opponent's last unit.

```
Avaiable Pieces:
<0> wf2
0
select the number of move pattern.
0
Game end.
White win!
```

6. Score Criteria

The maximum score for the problem is 32 (UML = 2 marks) and will be rounded down to 10.

6.1 Implement Piece correctly = 3 points

6.2 Class Fighter: = 19.5 points

- Constructor = 2 points (1 for each test case)
- Setter = 1 point
- AttackTargetPosition = 3 points (1.5 for each test case)
- Move = 13.5 points (1.5 for each test case)

6.3 Class Game System: = 7.5 points

- All test cases = 7.5 points (2.5 for each test case)

Part 2: Interface

1. Objectives

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

2. Instructions

1) Create Java Project named **"2110215_Exam2_Part2"** in your workspace.
2) Copy all folders in **"toStudent/Part2"** to your project directory src folder.
3) You are to implement the following classes (details for each class are given in section 4.)
    a) BaseProduct (package products)
    b) FreeDiscountProduct (package products)
    c) PercentDiscountProduct (package products)
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: Smoke Sales

You work for a grocery store called Smoke. Smoke is about to have its 1-year anniversary celebration, and is planning to do so by giving away discounts for its products in a "Smoke Sale". Your task as one of the programmers is to work on a discount calculation program to be used at the cash register. The program can be run from the class **Main.java** in the **app** package.

4. Implementation Detail

The class package is summarized below. Note that only important/useful methods are listed.

- private

protected

+ public

Static variables and methods are underlined.

4.1 package discount

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 interface Sellable

a. Variables

None.

b. Methods

Name	Description
+ abstract int calculateDiscount(int quantity)	Method to calculate the total discount value for the given quantity of Sellable objects.

4.1.2. public interface FreeDiscountable

a. Variables

None.

b. Methods

Name	**Description**
+ abstract int calcFreeDiscountPieces(int totalQuantityBought)	Method to calculate the number of pieces that will get a discount given the total amount of items bought. (Buy X get Y free – given the value of (X+Y), the method returns the value of Y)

4.1.3. public interface PercentDiscountable

a. Variables

None

b. Methods

Name	**Description**

+ abstract int calcDiscountPerPiece()	Method to calculate the discount value for 1 piece of the product.

4.2 package products

****PARTIALLY PROVIDED****

This entire package, except for BaseProduct, must be written from scratch.

4.2.1 public class BaseProduct

A base Sellable object in the store with no discounts. This class is almost complete, but is **missing one very important component**. **To run the program, you must add that component to this class.**

a. Variables

Name	Description
# String productName	The name of the product.
# int price	The default price of the product.

b. Methods

Name	Description
+ BaseProduct(String name, int price)	Initializes the values for the BaseProduct. Make sure to use the appropriate setters.
+ int calculateDiscount(int quantity)	Method from Sellable. Since this product does not offer promotions, the discount is always 0.
+ String toString()	Returns the object as a string.
+ getters and setters for all variables	For setPrice: If the input is less than 0, set price as 0. For setProductName: If the input is a blank string, set the name as “PlaceholderProduct”.

4.2.2 public class FreeDiscountProduct extends BaseProduct

A product which can be discounted by the FreeDiscount promotion, and is Sellable.

a. Variables

Name	Description
- int promoQuantity	The promotion quantity of this product type. (The X in Buy X Get Y Free)
- int freeQuantity	The quantity of the product you get for free after buying promoQuantity amount of product. (The Y in Buy X Get Y Free)

b. Methods

Name	Description
+ FreeDiscountProduct(String name, int price, int promoQ, int freeQ)	Initializes the values for the FreeDiscountProduct. Make sure to use the appropriate setters.
+ int calcFreeDiscountPieces(int totalQuantityBought)	Calculates the number of free pieces that will be given at a certain quantity. Use ShopUtil's calculateFreeDiscountPieces for this.
+ int calculateDiscount(int quantity)	Returns the discount. For this product, for the given quantity, the discount is equal to the number of free pieces multiplied by the default price.
+ String toString()	Returns this object as a string. The code for this method is provided in Exam2Part2ProvidedCodes.txt.
+ getters and setters for all variables.	For setPromoQuantity and setFreeQuantity: If the given value is less than 1, set it as 1.

4.2.3 public class PercentDiscountProduct extends BaseProduct

A product which can be discounted with a simple percent discount, and is Sellable.

a. Variables

Name	Description

- double percent	The discount percentage applied to this product.

b. Methods

Name	Description
+ PercentDiscountProduct(String name, int price, double percent)	Initializes the values for the PercentDiscountProduct. Make sure to use the appropriate setters.
+ int calcDiscountPerPiece()	Calculates the discount that will be given per piece. Use ShopUtil's calculateDiscountUsingPercent for this.
+ int calculateDiscount(int quantity)	Returns the discount. For PercentDiscountProduct, the discount is equal to the quantity purchased multiplied by the discount per piece.
+ String toString()	Returns this object as a string. The code for this method is provided in Exam2Part2ProvidedCodes.txt.
+ getter/setter for all variables.	For setPercent: If the input is less than 0, set it as 0. If the input is greater than 100, set it as 100.

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 ShopUtil

a. Variables

None.

b. Methods

Name	Description
+ int calculateFreeDiscountPieces(int freeQ, int promoQ, int totalQ)	Static method. Calculates the number of free pieces of product given in a Buy X Get Y Free. Put the X in freeQ, Y in promoQ, and the total purchase amount in totalQ.

+ int calculateDiscountUsingPercent (int price, double percent)	Static method. Calculates the discount, given the original price and the percentage.

4.3.2. public class BillItem

This class will be referred to in the method you will have to write in Main. This class represents a single line in a bill or receipt.

a. Variables

Name	Description
Sellable itemInLine	The product in this line in the bill.
int quantity	The amount of the product purchased in the bill.

b. Methods

Name	Description
+ int getPrice (boolean withDiscount)	Returns the price of this line in the bill. If withDiscount is true, all available discounts on the product will be applied.
+ getter/setter for all variables	

4.3.3. public class Bill

This class will be referred to in the method you will have to write in Main. This class represents a bill you will receive when shopping at the store.

a. Variables

Name	Description
+ ArrayList<BillItem> itemsInBill	The list of BillItem lines in the bill. Each BillItem is one line in this bill.

b. Methods

None.

4.4 package app

PARTIALLY PROVIDED

You need to complete the addItemToBill method inside Main for the program to run properly.

4.4.1. public class Main

a. Variables

Name	Description
Bill bill	The bill containing all the items you are about to check out.
ArrayList<Sellable> shop	The quantity of the product you get for free after buying promoQuantity amount of product. (The Y in Buy X Get Y Free)
Scanner sc	A scanner that receives input from the user.

b. Methods

Name	Description
+ addItemToBill()	The description of how to write this method is provided within the Main file. You will need to write your codes inside the try brackets, divided into two parts: **Before the "Choose Quantity" line:** A1. Accept a string input with sc.nextLine() A2. Attempt to parse the string into an int (the try/catch will handle bad cases automatically) A3. Use the int from step A2 to get a Sellable object from the ArrayList called shop **After the "Choose Quantity" line:** B1. Accept a string input with sc.nextLine() B2. Attempt to parse the string into an int (the try/catch will handle bad cases automatically) B3. Use the Sellable from step A3 and the quantity from B2 to initialize a new BillItem B4. Add the BillItem in step B3 to bill.itemsInBill NOTE: There is no JUnit available for this function. If you can correctly add items to the bill in the application, then your function is correct.

5. Score Criteria (Marks will be scaled down to 10)

Your work will be checked against JUnit test cases, as well as program functionality. Scorable items are shown in the list below.

1) BaseProductTest
    a) sellableTest (3 marks)
2) FreeDiscountProductTest
    a) constructorTest (1 mark)
    b) setPromoQuantityTest (1 mark)
    c) setPromoQuantityLessThanOneTest (1 mark)
    d) setFreeQuantityTest (1 mark)
    e) setFreeQuantityLessThanOneTest (1 mark)
    f) calculateDiscountTest (1 mark)
    g) freeDiscountableTest (2 marks)
    h) notPercentDiscountableTest (2 marks)
    i) toStringTest (0.5 mark)
3) PercentDiscountProductTest
    a) constructorTest (1 mark)
    b) setPercentTest (1 mark)
    c) setPercentLessThanZeroTest (1 mark)
    d) setPercentGreaterThanHundredTest(1 mark)
    e) calculateDiscountTest (1 mark)
    f) percentDiscountableTest (2 marks)
    g) notFreeDiscountableTest (2 marks)
    h) toStringTest (0.5 mark)
4) Main Functionality
    a) The method addItemToBill must be completed and products is able to be added to Bill in the program. (5 marks)
5) UML Diagram
    a) A correct UML Diagram must be put in the project directory. Please include all classes inside the packages discount and products. (Other classes are optional.) (2 marks)