Part 2: Inheritance

1. Objective

1) Be able to understand and utilize a concept of inheritance and polymorphism in Object-Oriented Programming (OOP).

2. Instruction

1) Create a Java Project named “2110215_Midterm_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) BasicMember

b) FundamemtalMintMember

c) PhaiThongCasanovaMember

d) StarvingStudentMember

e) AppControllers (2 Methods)

4) JUnit for testing is in package test.grader

3. Problem Statement: PhaiThong Convenience-Store’s Membership

You are the store manager of a billion-dollar company called PhaiThong CO., LTD. You want to implement a 'membership system' for the customers so that it would be easier to keep all the purchase data. Each type of member has different privileges and abilities. For example, they can pay with digital money, get random GachaPon rewards, collect points, and borrow money from the store.

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 application (already given)

4.1.1 Class Main

repeatedly calling the home() method from an IO instance responsible for handling user interactions and managing the store's functions.

4.2 package logic.app (already given)

4.2.1 Class AppController

Central logic controller for the application, handling actions like item management, member registration, shopping cart operations, and payments.

4.3 package store (already given)

4.3.1 Class Item

Represent items with attributes like name, price, and quantity. It includes methods for creating, displaying, and comparing items based on their names.

Important Fields

Name	Description
- String name	The name of the Item
- int price	The price for 1 Unit of that item
- int amount	The quantity of that item

Important Methods

Name	Description
+ String toString()	Prints the object as a string. Using this format: `getName() + ": x" + getAmount()`

+ int getAmount()	Get the amount of that item.
+ int setAmount(int amount)	Set the amount of an item.
+ boolean equals(Object o)	Check whether two items are equal. Two items are considered equal when their names are the same.

4.3.2 Class Store (already given)

handles the store's inventory and it's members

Important Fields

Name	Description
- final Array<Item> stock	Array of Items in the Store that is available for purchese.
- final Array<Item> members	Array containing all members in the Store.
- int storeMoney	Total amount of money the Store have.
<u>+ Store instance</u>	An instance of the Store Class

Important Methods

Name	Description
<u>+ Store getInstance()</u>	Get an Instance of Store Class.
+ Item takeRandomItemFromStock()	Take one piece of an Item from the store's stock and return that Item. If the stock is empty, it returns null.
+ getStoreMoney()	Returns the current amount of money available in the store.
+ setStoreMoney(int	Set the current amount of money

storeMoney)	available in the store.

4.4 package utils (already given)

4.4.1 Class IO

manages user input/output for the app, offering functions like adding items to stock, checking stock, signing up new members, and member management.

4.4.2 Class ItemUtils

Important Methods

Name	Description
+ int calculateTotalPrice(ArrayList<Item> shoppingCart)	calculates the total price of items in a shopping cart “without” applying any discount.
+ int calculateTotalPrice(ArrayList<Item> shoppingCart , double discountPercent)	calculates the total price of items in a shopping cart “with” a specified discount percentage applied.

4.5 Package logic.member

4.5.1 Class BasicMember

BasicMember is a class representing store members with a name, ID, purchase history, and shopping cart. It handles cart calculations, checkout, and member management. /* You must implement this class from scratch */

Field

Name	Description
- final ArrayList<Item> purchaseHistory	Keeps a record of the **Items** the members have checked out and purchased.
- final ArrayList<Item> shoppingCart	The shopping cart of the member contains **Items** from the Store's stock that haven't been checked out yet.

- String name	Name of the member (If the name is blank it will be renamed to “UnknownMember”)
- int memberID	The member's ID number. Any members with the same ID are considered the same member.

Constructor

Name	Description
+ BasicMember(String name, int memberID)	Initialize the member’s **name** and **memberID** -if the **name** isBlank. Rename the member to "UnknownMember" -if the **memberID** is less than 0, set it to 0

Method

Name	Description
+ int totalCartPrice()	Return the totalPrice of the **shoppingCart**. See **ItemUtils Class** for a method to calculate the totalPrice of the shoppingCart
+ void addToPurchaseHistory(Item item)	Add an **Item** into **purchaseHistory**. -If an **Item** already exists in the **purchaseHistory**, increase its amount. -If it doesn't, append it to the end of the array.
+ void checkout()	Add all Items from the **shoppingCart** into **purchaseHistory**, then clear all the Items from the **shoppingCart**.
+ String toString()	Prints the object as a string. Use this format: "(" + getTierName() + ")" + " " + getMemberID() + "-" + getName()

+ boolean equals(Object o)	Determine if two members are equal. If any two members have the same **memberID** “and” belong to the **same class or an inherited class**, then they are considered equal.
+ String getTierName()	Return the name of the Member’s tier Which is: "Basic"
+ void setName(String name)	Set the member’s **name** -If the member’s **name** is Blank change it to "UnknownMember"
+ void setMemberID(int memberID)	Set the member’s **memberID** -If the **memberID** is less than 0 Set it to 0.
+ getter…(for all fields)	getter methods for all of the Basic Members fields.

4.5.2 Class FundamentalMintMember

A FundamentalMint Member can perform all the functions of a Basic Member with slight variations and possess additional abilities and variables as described below. /* You must implement this class from scratch */

Additional Fields

Name	Description
- int point	Tracks loyalty points earned by the member, which can be exchanged for rewards or benefits in the store's membership program.
# double discountPercent	The percentage discount applied to a member's purchases.
- int digitalMoney	Digital currency that members can use to buy Items.

Constructor

Name	Description
+ FundamentalMintMember (String name, int memberID, int point, int digitalMoney)	Initialize the member's **name** , **memberID**, **point**, **digitalMoney**, and **discountPercent** -set **discountPercent** to 0.05 -if the **name** isBlank. Rename the member to "UnknownMember" -if the **memberID** <0 set it to 0 -if the **point** <0 set it to 0 -if the **digitalMoney** <0 set it to 0

Additional Method

Name	Description
+ void convertPoint()	Converts accumulated **points** into **digitalMoney**. It works as follows: -For every "100" **points**, convert it into 1 unit of **digitalMoney**. -Convert as many **points** into **digitalMoney** as possible and leave the remainder of the **points** untouched. Example: -pointBefore = 1637 (member's **points** before conversion). -digitalMoneyGained = totalPoints/100 = 16 Baht (**digitalMoney** gained will be according to the floor function of the conversion unit). -pointAfter = (1637%100) = 37 **points** (member's **points** after conversion will be the remainder of the operation).

+ String toString()	Prints the object as a string. Use this format: "(" + getTierName() + ")" + " " + getMemberID() + "-" + getName() + " DMoney: " + getDigitalMoney() + " Pts: " + getPoint()
+ String getTierName()	Return the name of the Member's tier Which is: "FundamentalMint"
+ int totalCartPrice()	Return the totalPrice of the **shoppingCart**. See **ItemUtils Class** for a method to calculate the totalPrice with discount of the shoppingCart
+ void checkout()	-**First,** add the member's **points** equal to the calculated **totalCartPrice**. -**Then,** add all items from the **shoppingCart** into the **purchaseHistory** and clear the shoppingCart.
+ void setPoint(int point)	Set the member's **point** -If the member's **point** is less than 0 set it to 0
+ void setDigitalMoney(int digitalMoney)	Set the member's **digitalMoney** -If the **digitalMoney** is less than 0 Set it to 0.
+ getters for all additional fields	getter methods for all of the additional fields.

4.5.3 Class PhaithongCasanovaMember

A Phaithong Casanova Member can perform all the functions of a FundamentalMint Member with slight variations and possess additional abilities as described below. /* You must implement this class from scratch */

Constructor

Name	Description

+ PhaiThongCasanovaMember (String name, int memberID, int point, int digitalMoney)	Initialize the member's **name** , **memberID**, **point**, **digitalMoney**, and **discountPercent.** -set **discountPercent** to 0.10 -if the **name** isBlank. Rename the member to "UnknownMember" -if the **memberID** <0 set it to 0 -if the **point** <0 set it to 0 -if the **digitalMoney** <0 set it to 0

Additional Method

Name	Description
+ void convertPoint()	Converts accumulated **points** into **digitalMoney**. It works as follows: -For every "50" **points**, convert it into 1 unit of **digitalMoney**. -Convert as many **points** into **digitalMoney** as possible and leave the remainder of the **points** untouched. Example: -pointBefore = 1637 (member's **points** before conversion). -digitalMoneyGained = totalPoints/50 = 32 Baht (**digitalMoney** gained will be according to the floor function of the conversion unit). -pointAfter = (1637%50) = 37 **points** (member's **points** after conversion will be the remainder of the operation).
+ String getTierName()	Return the name of the Member's tier Which is: "PhaiThongCasanova"
+ Item	Exchange 1000 **points** and receive a random

giveRandomItemFromStore()	**Item** from the store's **stock**.This method works as follows. -checks if "the member has at least 1000 **points**". -If they do, 1.Selects a random **Item** from the store, 2.deducts 1000 **points** from the member, 3.adds the selected **Item** to their **purchaseHistory**, and returns the added **Item**. -If "the member doesn't have enough **points**" **or** "if there are no **Items** available in the store's(stock array is empty)", it returns **null**. See methods from **Store Class** for **getting a random Item** from the store's stock

4.5.4 Class StarvingStudentMember

A Starving Student Member can perform all the functions of a FundamentalMint Member with slight variations and possess additional abilities and variables as described below. /* You must implement this class from scratch */

Additional Fields

Name	Description
+ final int MAX_LOAN	Represents the most money a StarvingStudentMember can borrow as a loan, ensuring they don't borrow beyond this limit.
- int loan	The loan variable keeps track of how much money the member has borrowed and needs to repay.

Constructor

Name	Description
+ StarvingStudentMember(String name, int memberID, int point, int digitalMoney)	Initialize the member's **name** , **memberID**, **point**, **digitalMoney** , **loan** , and **discountPercent** -set **discountPercent** to 0.20 -set **loan** to 0 -if the **name** isBlank. Rename the member to "UnknownMember" -if the **memberID** <0 set it to 0 -if the **point** <0 set it to 0 -if the **digitalMoney** <0 set it to 0

Additional Method

Name	Description
+ void convertPoint()	Converts accumulated **points** into **digitalMoney**. It works as follows: -For every "75" points, convert it into 1 unit of digital money. -Convert as many **points** into **digitalMoney** as possible and leave the remainder of the points untouched. Example: -pointBefore = 1637 (member's **points** before conversion). -digitalMoneyGained = totalPoints/75 = 21 Baht (**digitalMoney** gained will be according to the floor function of the conversion unit). -pointAfter = (1637%75) = 62 **points** (member's **points** after conversion will be the remainder of the operation).

+ String toString()	Prints the object as a string. Use this format: "(" + getTierName() + ")" + " " + getMemberID() + "-" + getName() + " DMoney: " + getDigitalMoney() + " Pts: " + getPoint() " Loans: " + getLoan()
+ String getTierName()	Return the name of the Member's tier Which is: "StarvingStudent"
+ void setLoan(int loan)	Set the member's **loan** -If the member's **loan** is less than 0 set it to 0
+ boolean loanMoney(int amount)	Allows the member to loan money from the **storeMoney**. It works as follows: -check if **a.** requested **amount** + member's **loan**, must **not exceed** the **MAX_LOAN** limit **b.** Total **storeMoney** is more than the requested **amount**. -If **a. And b. conditions are satisfied**, 1.reduce the **storeMoney** by **amount**, 2.increases the member's **digitalMoney** by the **amount.** 3.increases the member's **loan** balance by **amount**. 4.returns **true** (to indicate a successful loan transaction.) -If **a or b condition is not met**, it returns **false**. (to indicate a failed loan transaction.) See **Store Class** for a method to **get/set storeMoney**
+ boolean returnLoan(int amount)	Repay a portion or the entire **loan** amount, It works as follows:

	Firstly, check if the given **amount** is **more than** the member’s **loan**. If so, reduce the return **amount** to be equal to the **loan** of the member. Secondly, check if the member has enough **digitalMoney** to cover the repay **amount**. -If the member **has enough digitalMoney**: 1. Increase the **storeMoney** by the specified **amount**. 2. Reduce the member's **digitalMoney** by the specified **amount**. 3. Decrease the member's **loan** balance by the specified **amount**. 4. Return **true (**to indicate a successful transaction.) -If the member **has less digitalMoney** than the repayment **amount**. return **false** (to indicate a failed transaction.) See **Store Class** for a method to **get/set storeMoney**