1. จงอธิบายหลักการต่อไปนี้
   1. Encapsulate what vary พร้อมยกตัวอย่าง

คือ การซ่อนรายละเอียดภายใน และรักษาไม่ให้เกี่ยวข้องกัน ซึ่งจะแยกอกจากกัน เนื่องจากเวลาที่เรามีข้อมูลส่วนตัวภายในคลาส (หรือระบบ) ซึ่งไม่ต้องการเปิดเผยให้กับลูกค้ารู้และเราต้องซ่อนการทำงานจริงไว้เพื่อให้ปฏิสัมพันธ์รหว่างลูกค้าและบริการผ่านอินเตอร์เฟส ประโยชน์คือ เราสามารถเปลี่ยนแปลงได้โดยไม่รบกวนลูกค้า

ตัวอย่าง

class Message {  
 Long messageId;  
 MessageType messageType;  
 String senderId;  
 String receivedId;  
 MessageData messageData; //text, or URL  
}

abstract class MessageData {

//จะมีการทำงานของ messageData   
}

class TextMessageData extends MessageData {  
 String textMessage;  
}

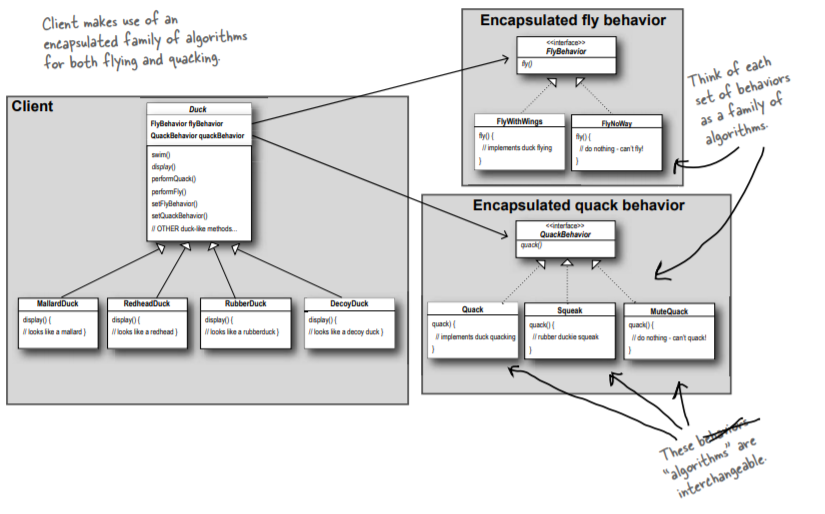
class PhotoMessageData extends MessageData {  
 String imageDownloadUrl;  
 Integer width;  
 Integer height;  
 String imageCaption;  
 //add more params as needed  
}

class VideoMessageData extends MessageData {  
 String videoDownloadUrl;  
 Long videoStartOffsetMillis;  
 //more such params  
}

ดังนั้น เราจะได้ message object ที่อาจจะมีบางฟิวลด์ที่เหมือนกัน และข้อมูลที่แตกต่างกันไปตามประเภทของข้อความจะถูกห่อหุ้มใน MessageData object (หรือ interface) ซึ่งสามารถขยายออกภายหลังได้

* 1. Program to interfaces, not implementations พร้อมยกตัวอย่าง

หลักการนี้เกี่ยวกับความสัมพันธ์ในการพึ่งพาซึ่งจะต้องมีการจัดการอย่างรอบคอบในแอพขนาดใหญ่ ง่ายต่อการอ้างอิงคลาส โดยเราจะทำการขึ้นกับอินเตอร์เฟสของเราเท่านั้น จะถูกแยกออกจากการใช้งาน นั้นหมายถึงการใช้งานอาจจะแตกต่างกันไป ซึ่งวิธีการนี้จะให้ความยืดหยุ่นแก่เรา แต่ยังมีการแยกส่วนของค่าจริงออกจากการใช้งานซึ่งช่วยให้ลูกค้าแยกจากการใช้งานได้ ดังนั้นเราสามารถเพิ่มพฤติกรรมใหม่ๆได้โดยไม่รบกวนลูกค้า



ซึ่งจาก diagram เราสามารถทำการเพิ่มพฤติกรรมของลักษณะการบินของเป็ดได้ โดยเมื่อเราทำการเพิ่มแล้ว จะไม่เกิดผลกระทบกับ class Duck ซึ่ง class Duck จะเก็บค่าต่างของเป็ดไว้ และจำทำการแยกลักษณะพฤติกรรมของตัวเองออกมาเป็น 2 class interface คือ FlyBehavior และ QuackBehavior

* 1. Favor composition over inheritance พร้อมยกตัวอย่าง

เพราะถ้าเราออกแบบการสืบทอด (inheritance) นั้น หมายความว่า เรานิยามและออกแบบคลาสของเราให้ผูกพันธ์กันเองตั้งแต่แรก พอลูกค้าต้องการที่จะต้องมีการแก้ไข จะทำให้เราแก้ไขได้ยากเนื่องจากคลาสได้มีการผูกติดกัน ในทางกลับกัน composition เป็นความสัมพันธ์ที่ยืดหยุ่นกว่าเพราะเรานิยามความสามารถของคลาสแยกออกมา เมื่อคลาสต้องการมีความสามารถใหม่เราก็แค่เลือกความสามารถใหม่ผูกติดเข้าไป เมื่อไม่ต้องการก็สามารถถอดออกได้

ตัวอย่าง โจทย์ของเราคือต้องการสร้าง Dog และ Cat โดยมีคุณสมบัติดังนี้

* Dog: เห่าได้ พิมพ์ชื่อออกมาได้และเดินได้
* Cat: เห่าไม่ได้ พิมพ์ชื่อออกมาได้และเดินได้

// ใช้เดิน

const Walkable = {

walk() {

console.log("I'm walking.")

}

}

// ใช้พิมพ์ชื่อ

const Printable = {

print() {

console.log(this.name)

}

}

// ใช้เห่า

const Barkable = {

bark() {

console.log("ฮ่งๆ")

}

}

// หมาพิมพ์ชื่อได้ เดินได้และเห่าได้

const Dog = (name) => ({

name,

...Printable,

...Walkable,

...Barkable

})

// แมวพิมพ์ชื่อได้ เดินได้แต่เห่าไม่ได้

const Cat = (name) => ({

name,

...Printable,

...Walkable

})

const dog = new Dog('Boo')

dog.print() // Boo

dog.bark() // ฮ่งๆ

const cat = new Cat('Meaw')

cat.print() // Meaw

cat.walk() // I'm walking

* 1. กำหนดให้  
     class Controller{

Private Driver driver;

Public Controller(String type){

If (type==”Printer”)

this.driver = new PrinterDriver();

else if(type==”Monitor”)

this.driver = new MonitorDriver();

}

}

จงออกแบบคลาสข้างต้นใหม่เพื่อทำให้เกิด Strive for loose coupling และให้เหตุผลว่าทำไมจึงเป็นเช่นนั้น

ที่เราทำให้เกิด Strive for loose coupling คือการที่พยายามออกแบบให้มีการประสานกันระหว่างวัตถุที่มีความสัมพันธ์กันและกันแบบหลวม ก็เพื่อที่จะลดข้อผูกมัดของคลาสลง เพราะว่าเรามีการเพิ่ม driver หรือเพิ่มฟังก์ชันในอนาคตนั้น จะทำให้เราแก้ไขได้ยาก

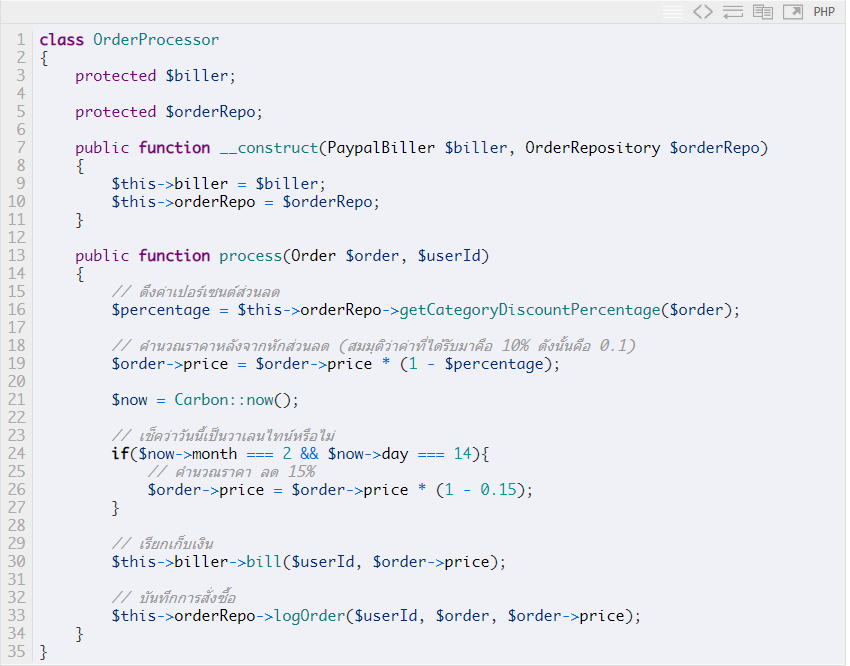
1. จงยกตัวอย่างคลาสที่ทำเกิดการระเมิดกฎ Single Responsibility และแก้ไขให้สอดคล้องกับกฎดังกล่าว

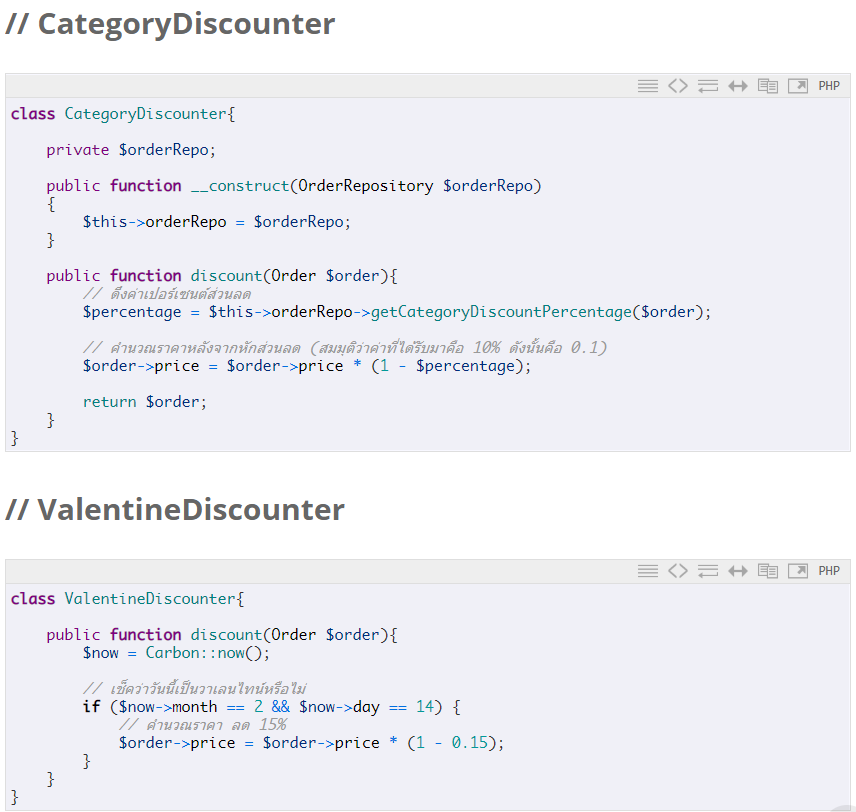
เช่น class programmer ที่มีการทำงานอยู่ 2 ตัว คือ เขียนโค้ด (Code) และ ออกแบบ (designUI) ซึ่งเมื่อเรามีการพัฒนาสกิลทางด้าน programming และ designing เราก็สามารถเพิ่มเข้าไปในการทำงานของเราได้ ดังนั้นเราจะเห็นได้เลยว่า programmer คนนี้ทำงานหนักมากเกินไป เพราะต้องทำทั้งเขียนโค้ด และออกแบบพร้อมๆกัน ซึ่งมันผิดกฎของ Single Responsibility ที่ได้บอกไว้ว่า คลาส 1 คลาสควรมีแค่การทำงานเดียวเท่านั้น วิธีการแก้ไขคือ เราควรที่จะมีอีก คลาสหนึ่งที่มีหน้าที่ในการออกแบบ และอีกคลาสทำหน้าที่ในการเขียนโค้ด และให้มี programmer ใช้สำหรับเขียนโค้ด และให้ designer ทำการออกแบบ แล้วทั้งสองคนสามารถที่จะสื่อสาร หรือติดต่อข้อมูลหากันได้

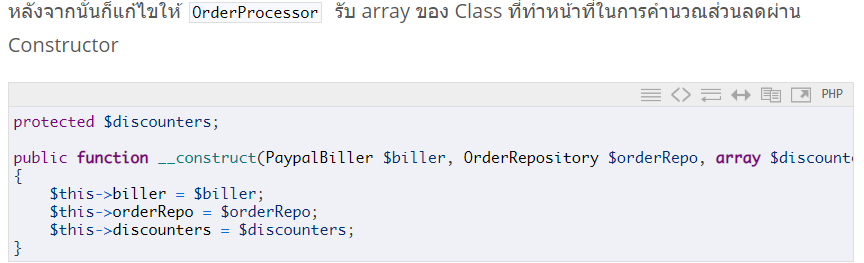
1. จงยกตัวอย่างคลาสที่ทำเกิดการระเมิดกฎ **O**pen/Closed และแก้ไขให้สอดคล้องกับกฎดังกล่าว

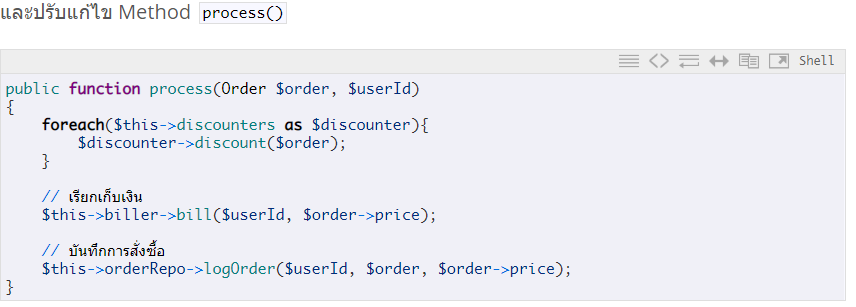
เช่น 

ถ้าให้มีการเพิ่มการทำงาน โดยให้มีส่วนลด 15 เปอร์เซ็นเป็นเวลา 1 วันสำหรับวันวาเลนไทน์



แต่ถ้าในอนาคต เรามีการเพิ่มส่วนลดอีกล่ะ มันจะยากมากต่อการแก้ และการทำเช่นนี้ ทำให้ผิดต่อกฎของ **O**pen/Closed คือ ยอมให้มีการเพิ่มต่อส่วนขยายได้ แต่ปิดสำหรับการแก้ไข ดังนั้นวิธีการแก้คือ จะแยกคลาสส่วนลดออกเป็นสองคลาส ได้แก่





เพียงเท่านั้นเราก็สามารถแก้ไขได้แล้ว หากเรามีประเภทส่วนลดเพิ่มขึ้นอีก เราก็สามารถทำการสร้าง class ใหม่ได้ และสามารถทำการเปลี่ยนแปลงได้โดยไม่กระทบต่อโค้ดเดิม

1. จงยกตัวอย่างคลาสที่ระเมินกฎ **D**ependency Inversion และแก้ไขให้ถูกต้อง

เช่น High-level Code นั้นไม่ควรที่จะผูกติดอยู่กับ Low-level code แต่ทั้งสองนั้นควรจะผูกติดและขึ้นอยู่กับ Abstraction และ Abstraction นั้นก็ไม่ควรผูกติดอยู่กับรายละเอียดการทำงาน แต่ในทางกลับกันรายละเอียดการทำงานควรจะผูกติดอยู่กับ Abstraction

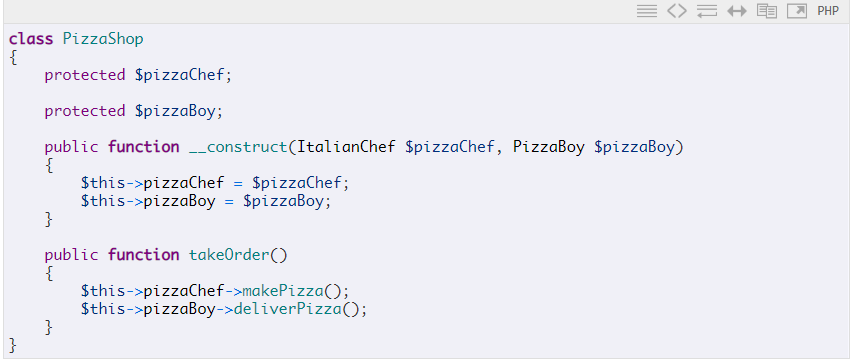
High-level code คือ โค้ดที่มีความซับซ้อน อาจจะเป็น โค้ดที่รวมการทำงานของ low-level code

Low-level code คือ โค้ดที่เกี่ยวกับการทำงานพื้นฐานต่างๆ เช่น การอ่าน / เขียนไฟล์

(นิยามของ DIP นั้นก็คือ ทั้ง High-level และ Low-level code ควรจะต้องผูกติดอยู่กับ Abstract หรือ Interface นั่นเอง)

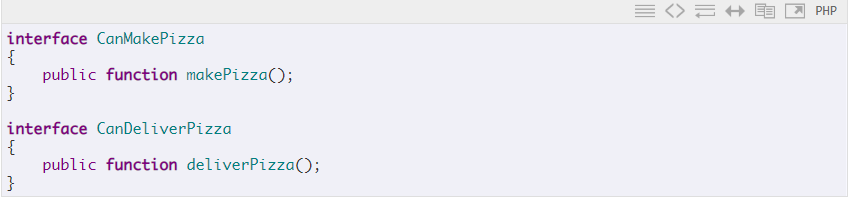
ร้านพิซซ่า เมื่อมี order เข้ามาที่ร้านนั้นต้องอาศัย พ่อครัว (PizzaChef) ในการทำพิซซ่า และ เด็กส่งพิซซ่า (PizzaBoy) เพื่อส่งพิซซ่าให้ลูกค้า เราจะมอง ให้ PizzaChef และ PizzaBoy เป็น low-level code คือคลาสที่ทำงานทั่วๆไป และจะมอง PizzaShop เป็น high-level code ที่จะควบคุมการทำงานที่ซับซ้อนกว่า

จะเห็นว่าตอนนี้ PizzaShop นั้นผูกติดกับ PizzaChef และ PizzaBoy ซึ่งไม่เป็นไปตามหลักของ DIP สมมติว่าร้านพิซซ่าของเรามีการเพิ่มพ่อครัวคนใหม่เป็นอตาลี (ItalianChef) มาแทนพ่อครัวคนเดิม (PizzaChef) จะทำให้เกิด error เนื่องจาก PizzaShop นั้นถูกผูกติดกับเฉพาะ PizzaChef เท่านั้น



วิธีการแก้ไข

1. เราจะสร้าง Abstraction (Abstraction คือคลาสนามธรรม ในทางโปรแกรมมิ่งคือ abstract class และ interface) หลักการคิดคือ สังเกตที่คลาส high-level code ว่าสนใจอะไรบ้าง นั้นก็คือ ทำพิซซ่า กับ ส่งพิซซ่า



1. ให้ high-level code ทำการผูกติดกับ abstraction



1. ให้ low-level code ทำหารผูกติดกับ abstraction เช่นกัน



1. บทตัวละคร (Character) มีหลายชนิดได้แก่ King มี Queen มี Knight ทุกตัวละครมีลักษณะต่อสู้ได้ (flight) แต่ละตัวละคุณมีพฤติกรรมการใช้อาวุธ (use weapon) แตกต่างกัน พฤติกรรมการใช้อาวุธแบ่งเป็นหลายแบบได้แก่ ดาบ (Sword Behavior) มีด (Knife Behavior)

5.1 จงเขียนออกแบบคลาสโดยใช้ Strategy patterns

Character k = new King();

Character q = new Queen();

Character knight= new Knight();

k.setWeapon(new Sword());

q.setWeapon(new Knife());

knight.setWeapon(new Sword());

k.fight(); // I’m a king and I can use my sword.

q.fight(); // I’m a queen and I can use my knife.

knight.fight(); // I’m a knight and I can use my sword

//ในโปรแกรม

* 1. delegation คืออะไร ใช้อย่างไรใน 5.1

delegation คือ การมอบหมายจะใช้คลาสอื่นเป็นตัวจัดการแทนตัวมัน โดยคลาสโฮสต์จะทำการสร้างอินสแตนท์ส่วนตัวของคลาสผู้ช่วย และส่งความถึงคลาสโฮสต์ตามความเหมาะสม

นำมาใช้ในการอ้างอึงถึงพฤติกรรมของการใช้อาวุธ โดยที่คลาสโฮสต์ จะสร้าง class interface weaponBehavior ขึ้นมาเรียกใช้แทน พฤติกรรมการใช้อาวุธต่างๆ (Sword , Knife)

* 1. จงเพิ่ม GunBehavior เพื่อให้

Character s = new Soldier();

s.setWeapon(new Gun());

s.fight(); //I’m a soldier and I can use my gun.

//ในโปรแกรม

1. ขนมเค๊ก (Cake) แบ่งเป็นชนิดย่อยได้แก่ CupCake, butter cake เป็นต้น ซึ่งมีหน้า (Topping) ได้หลายหน้าได้แก่ small candies, chocolate ship, White chocolate โดยกำหนดราคาให้ CupCake ราคาชิ้นละ 15 และ butter cake ราคา 20 บาท ส่วนหน้าของ small candies, chocolate ship, White chocolate หน้าละ 1, 2, 3 บาทตามลำดับ
   1. จงเขียนโปรแกรมสร้างคลาสโดยใช้ Decorator pattern

//ในโปรแกรม

* 1. จงอธิบายการใช้ delegation ใน 1.1

เราได้ใช้หลักการ delegation โดยเราทำการสร้าง abstract class (DecoratorCake) เป็นตัวแทนในการมอบหมายงานให้ทำการเรียกค่าของส่วนประกอบของเค้กแต่ละชนิดแทนการเรียกใช้ส่วนนั้นโดยตรง

* 1. จงวาด UML

เมื่อรันโปรแกรมดังนี้

Cake c = **new** CupCake();

DecoratorCake d = **new** SmallCandiesDecorator(c);

DecoratorCake s = **new** ChocolateShipDecorator(d);

DecoratorCake w = **new** WhiteChocolateDecorator(s);

System.***out***.println(" Name :"+ w.getText()+ " Price:"+ w.getCost());

Cake butterCake = **new** ButterCake();

DecoratorCake chocoButter = **new** SmallCandiesDecorator(butterCake);

DecoratorCake butterShip = **new** ChocolateShipDecorator(chocoButter);

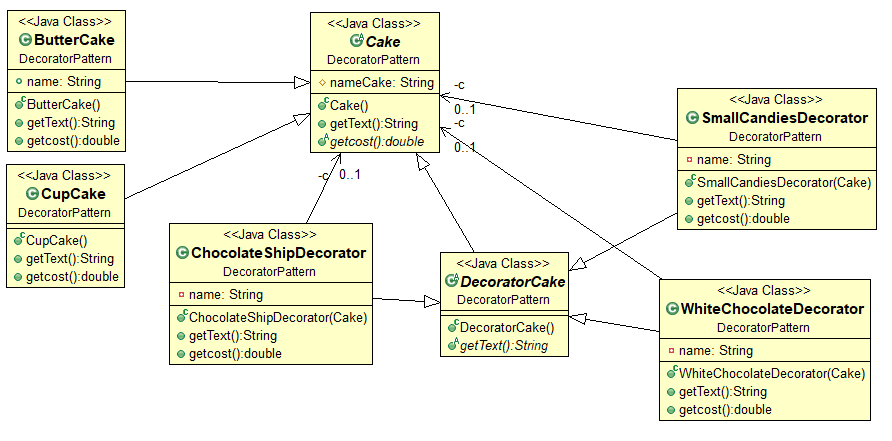
System.***out***.println(" Name :"+ butterShip.getText()+ " Price:"+

butterShip.getCost());

ผลลัพธ์:

Name : CupCake+ Small candies + Chocolate ship + White chocolate Price:21.0

Name :Butter Cake+ Small candies + Chocolate ship Price:23.0



1. กำหนดให้เจ้ามือ(Dealer) มีการออกเลขรางวัล(lottery results)โดยสุ่มจากตัวเลข 1-10 เมื่อได้ตัวเลขจะต้องแจ้งไปยังสมาชิกทุกคน สมาชิกแบ่งเป็น 2 ประเภทได้แก่ PremiumMember , PlatinumMember เมื่อสร้างวัตถุของแต่ละประเภทสมาชิกต้องแจ้งความประสงค์ว่าตัวเลขใดที่ต้องการ ตัวเลขหนึ่งตัวสมาชิกอาจใช้ซ้ำกันได้
   1. จงเขียนโปรแกรมสร้างคลาสโดยใช้ observer pattern
   2. วาดแผนภาพคลาส UML ของคลาสต่างๆ

IDealer dealer = new Dealer();

Observer g1 = new PremiumMember();

Observer g2 = new PremiumMember();

Observer d1 = new PlatinumMember();

dealer.addMember(g1);

dealer.addMember(g2);

dealer.addMember(g3);

g1.buyNumber(2);

g2.buyNumber(7);

d1.buyNumber(8);

dealer.doLotteryResult();

dealer.doLotteryResult();

dealer.removeMember(g2);

dealer.doLotteryResult();;

ตัวอย่างผลลัพธ์:

Lottery Result =7

I lost a golden prize, my number is 5

I win a golden prize, my number is 7

I lost a platinum prize, my number is 8

Lottery Result =6

I lost a golden prize, my number is 5

I lost a golden prize, my number is 7

I lost a platinum prize, my number is 8

Lottery Result =3

I lost a golden prize, my number is 5

I lost a platinum prize, my number is 8

หมายเหตุ: วิธีใช้ Random ให้สุ่มตัวเลข 0-10

Random random = new Random();

number = random.nextInt(10);