Programming Fundamentals II

Lap6: - Interface
- Multiple Inheritance

1.1 Interface

ให้นิสิตสร้าง Interface ดังนี้

MyInterface.java เขียนโคดดังนี้

```
public interface MyInterface
{
	int MAX = 100;
	void m1();
}
```

และให้นิสิตสร้าง MyInterfaceTest เพื่อใช้ในการทดสอบการทำงาน

ตัวอย่างโค้ด MyInterfaceTest.java ไว้ทดลองรันการทดลอง

```
public class MyInterfaceTest
{
	public static void main(String[] args)
	{
		// Input your Statement
		// Input your Statement
	}
}
```

Lab 6.1 ให้นิสิตสร้าง Class ชื่อ Lab61 และเรียกใช้งาน MyInterface

Lab61.java เขียนโคดดังนี้

```
class Lab61 implements MyInterface{
	public void m1() {
		System.out.println("Call from A");
	}
}
```

ผลการเรียกใช้ Method m1() (ถ้าใช้ไม่ถูกต้องให้บันทึกสาเหตุการใช้ที่ไม่ถูกต้องด้วย)

Lab 6.2 ให้นิสิตสร้าง Class ชื่อ Lab62 และเรียกใช้งาน MyInterface

Lab62.java เขียนโคดดังนี้

```
class Lab62 implements MyInterface{
	public void m1() {
		System.out.println("Call from B");
	}
}
```

ผลการเรียกใช้ Method m1() (ถ้าใช้ไม่ถูกต้องให้บันทึกสาเหตุการใช้ที่ไม่ถูกต้องด้วย)

Lab 6.3 ให้นิสิตสร้าง Class ชื่อ Lab63 และเรียกใช้งาน MyInterface

Lab63.java เขียนโคดดังนี้

```
abstract class Lab63 implements MyInterface{
	void m1(){
		System.out.println("Call from C");
	}
}
```

ผลการเรียกใช้ Method m1() (ถ้าใช้ไม่ถูกต้องให้บันทึกสาเหตุการใช้ที่ไม่ถูกต้องด้วย)

Lab 6.4 ให้นิสิตสร้าง Class ชื่อ Lab64 และเรียกใช้งาน MyInterface

Lab64.java เขียนโคดดังนี้

```
abstract class Lab64 implements MyInterface{
	public void m1() {
		System.out.println("call from D1");
	}
	public void m2() {
		System.out.println("call from D2");
	}
}
```

ผลการเรียกใช้ Method m1() (ถ้าใช้ไม่ถูกต้องให้บันทึกสาเหตุการใช้ที่ไม่ถูกต้องด้วย)

Lab 6.5 ให้นิสิตสร้าง Class ชื่อ Lab65 และเรียกใช้งาน MyInterface

Lab65.java เขียนโคดดังนี้

```
abstract class Lab65 implements MyInterface{
	public void m2() {
		System.out.println("call from E");
	}
}
```

ผลการเรียกใช้ Method m1() (ถ้าใช้ไม่ถูกต้องให้บันทึกสาเหตุการใช้ที่ไม่ถูกต้องด้วย)

Lab 6.6 ให้นิสิตสร้าง Class ชื่อ Lab66 และเรียกใช้งาน MyInterface

Lab66.java เขียนโคดดังนี้

```
abstract class Lab66 implements MyInterface{
	public void m1() {
		System.out.println(MyInterface.MAX);
		System.out.println(MAX);
	}
}
```

ผลการเรียกใช้ Method m1() (ถ้าใช้ไม่ถูกต้องให้บันทึกสาเหตุการใช้ที่ไม่ถูกต้องด้วย)

ให้นิสิตขียนโคดใน Class MyInterfaceTest ที่เรียกใช้งาน Lab1-Lab6 โดยให้ผลลัพท์ดังนี้
(ถ้ามีการสร้าง Class ใหม่ ให้สร้าง Class ใน MyInterfaceTest)

```
Call from A
Call from B
Call from D2
call from D1
call from E
100
100
```

เขียนโคด MyInterfaceTest ในกล่องข้อความด้านล่าง

```
public class MyInterfaceTest
```

1.2 Multiple Inheritance

นิสิตสามารถที่จะเขียนโปรแกรมให้มีการรับการถ่ายทอดคุณสมบัติจากหลายๆ ที่ได้โดยใช้ Interface ให้นิสิตสร้าง package ใหม่ และสร้าง Interface ดังนี้

CanBark.java

```
public interface CanBark {
	void bark();
}
```

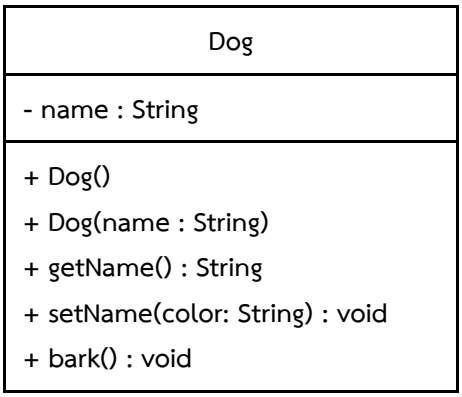
CanFetch.java

```
public interface CanFetch {
	void fetch();
}
```

CanSwim.java

```
public interface CanSwim {
	void swim();
}
```

จากนั้นให้นิสิตสร้าง Class Dog จาก UML Diagram ด้านล่าง

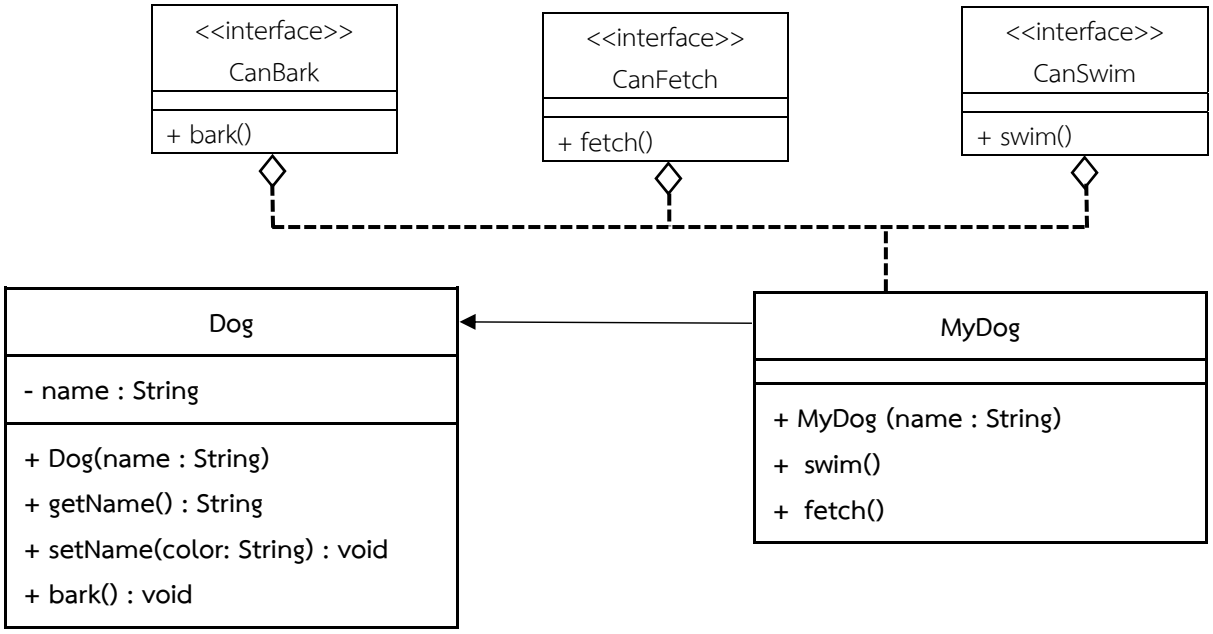


Class Dog มีรายละเอียดดังนี้

- มี Attribute คือ ชื่อ (name)
- Default Attribute null
- Methods คือ getter และ setter
- bark() มีการทำงานดังนี้

```
public void bark() {
	System.out.println("Woof Woof");
}
```

จากนั้นให้นิสิตสร้าง Class MyDog รายละเอียดจาก UML Diagram ด้านล่าง



Class MyDog มีรายละเอียดดังนี้

- ไม่มี Attribute
- Override Methods ที่ Implement มา
- swim() มีการทำงานดังนี้

```
public void swim() {
	System.out.printf("%s is swiming.\n", super.getName());
}
```

- fetch() มีการทำงานดังนี้

```
public void fetch() {
	System.out.printf("%s is fetching.\n", super.getName());
}
```

Lab 6.7 ให้นิสิตสร้าง Class ชื่อ Lab67MyDogTest และเขียนโปรแกรมให้ได้ผลลัพท์ดังนี้

```
Woof Woof
Deang is fetching.
Deang is swiming.
Deang is Woof Woof
```

จงเติมโคดให้ได้ผลลัพท์ดังกล่าวกล่องคำตอบ

```
public class Lab67MyDogTest
{

	public static void main(String[] args)
	{
		MyDog dog = new MyDog("Deang");

		act1(dog);	//treat as CanBark
		act2(dog);	//treat as CanFetch
		act3(dog);	//treat as CanSwim
		act4(dog);	//treat as MyDog
	}

	private static void act1(_____________________)
	{




	}

	private static void act2(_____________________)
	{




	}

	private static void act3(_____________________)
	{




	}

	private static void act4(_____________________)
	{




	}
}
```

แบบทดสอบ Lab6

All Those Things with Four Sides

Complete the followings:

1. Write an inheritance hierarchy for classes Quadrilateral (สี่เหลี่ยม), Trapezoid (สี่เหลี่ยมคางหมู), Parallelogram (สี่เหลี่ยมด้านขนาน), Rectangle (สี่เหลี่ยมผืนผ้า) and Square (สี่เหลี่ยมจัตุรัส).

2. Create and use a Point class in each shape to represent the points representing an x-y coordinate pair.

   Attributes (should be private):

   x, y as double

   Methods:

   Point(double x, double y)

   double getX()

   double getY()

3. Use Quadrilateral as the superclass of the hierarchy. It must contain the following methods:

   Attributes (should be private):

   endpoints[] as Point

   Methods:

   Quadrilateral(Point p0, Point p1, Point p2, Point p3)

   double getArea() (which, for simplicity, returns 0.0 forQuadrilateral)

   Point getEndpoint(int index) (returns one of the four endpoints as specified by index)

4. Make the hierarchy as deep (i.e., as many levels) as possible. (ตรวจสอบความสัมพันธ์ has-a และ is-a ให้ดี Point ควรเป็นอะไรกับสี่เหลี่ยม และสี่เหลี่ยมแต่ละประเภทมีความสัมพันธ์กันอย่างไร)

5. Specify a proper constructor for each class. For Rectangle, it may need only two endpoints to define it, and for Square, it may need only one endpoint at a predefined (say, top-left) corner, and the length of each side. The constructor should also utilize its superclass' constructor. (อ่านหมายเหตุเพิ่มเติมด้านล่าง)

6. All classes, except Point and Quadrilateral, must override double getArea() and return the correct value for the area of each type of quadrilateral. Also, use the @Override annotation to prevent mistakes.

7. Write a program as a TestQuadrilateral class that instantiates objects of your classes and outputs each object's area (except Quadrilateral).

หมายเหตุ1

การเรียกใช้ constructor ของ superclass ด้วย super(...) ต้องทำเป็นคำสั่งแรกเสมอ ไม่สามารถทำคำสั่งอื่นก่อนได้ แต่ในโจทย์ข้อนี้ constructor ของสี่เหลี่ยมบางประเภทอาจต้องคำนวณเพื่อหาพิกัดของจุดบางจุดก่อนจะเรียก constructor ของ superclass เพราะฉะนั้นทางเลือกหนึ่งที่น่าจะเป็นไปได้คือการแยกส่วนการคำนวณล่วงหน้าไปเป็น method ย่อยแล้วเรียกใช้เป็น argument เช่น

```
super(p0, computeDiagPoint(p0, r));
```

ในตัวอย่างนี้ computeDiagPoint() เป็น method ที่เราสร้างขึ้นเพื่อคำนวณค่า argument ที่ยังขาดไปอีกตัวของ constructor ของ superclass

แต่ก็จะมีปัญหาอีกอย่างตามมา คือ ก่อนเรียก super() โปรแกรมจะถือว่าตัวอ็อบเจกต์ยังไม่ถูกสร้างขึ้น ทำให้ไม่สามารถเรียก method ปกติได้ การเรียก computeDiagPoint()อย่างในตัวอย่างนี้จึงทำให้เกิด error ขึ้นได้ ทางแก้ไขก็คือ ต้องประกาศให้computeDiagPoint() เป็น static method เพื่อให้เรียกใช้ได้เลยโดยไม่ต้องมีตัวอ็อบเจกต์ก่อน

หมายเหตุ2

สำหรับแบบฝึกหัด ให้ทำการ comment ในโปรแกรม ในส่วนของ statement หลักๆของโค้ดที่นิสิตเขียน ถ้าไม่มีการ comment จะถือว่าโค้ดไม่ครบสมบูรณ์