

ใบงานการทดลองที่ 11

เรื่อง การใช้งาน Abstract และ Interface

1. จุดประสงค์ทั่วไป

- 1.1. รู้และเข้าใจการกำหนดวัตถุ การใช้วัตถุ การซ่อนวัตถุ และการสืบทอดประเภทของวัตถุ
- 1.2. รู้และเข้าใจโครงสร้างของโปรแกรมเชิงวัตถุ

2. เครื่องมือและอุปกรณ์

เครื่องคอมพิวเตอร์ 1 เครื่อง ที่ติดตั้งโปรแกรม Eclipse

3. ทฤษฎีการทดลอง

- 3.1. Abstract Class คืออะไร? มีลักษณะการทำงานอย่างไร? อธิบายพร้อมยกตัวอย่างประกอบ

Abstract Class คือคลาสที่ถูกออกแบบมาให้เป็นแม่แบบในการสืบทอดคุณสมบัติให้กับคลาสอื่น โดย **Abstract Class** ไม่สามารถถูกสร้างเป็น **Object** ได้ แต่สามารถสร้างเป็นคลาสลูก ๆ และใช้เป็นต้นแบบในการสร้าง **Object** ได้

- 3.2. Interfaces คืออะไร? มีลักษณะการทำงานอย่างไร? อธิบายพร้อมยกตัวอย่างประกอบ

Interfaces เป็นตัวกลางในการสื่อสารระหว่างคลาสที่ต่างกัน โดย **Interface** จะกำหนดชุดของ **methods** ที่ต้องมีในคลาสที่ใช้งาน **Interface** นั้น ๆ ซึ่งคลาสที่ใช้งาน **Interface** จะต้องประกาศว่าจะ **implement** หรือใช้งาน **Interface** นี้ เพื่อให้คลาสดังกล่าวต้องทำการสร้าง **method** ที่ถูกกำหนดไว้ใน **Interface** และเติม **implementation** ให้เรียบร้อยตามที่ต้องการ

- 3.3. คำสั่ง **extends** และ **implements** มีการใช้งานที่แตกต่างกันอย่างไร?

คำสั่ง **extends** และ **implements** เป็นคำสั่งที่ใช้ในการสืบทอดคุณสมบัติ (**inheritance**) ของคลาสหรืออินเทอร์เฟซในภาษา **Java** โดยทั้งสองคำสั่งนี้มีการใช้งานที่แตกต่าง

- 3.4. ภายใน **Abstract Class** มี **Constructor** หรือไม่? เพราะเหตุใด?

ใน **Abstract Class** สามารถมี **Constructor** ได้เหมือนกับ **Class** ทั่วไป แต่จะต้องระบุเป็น **protected** เพื่อให้ **Constructor** นั้นสามารถเรียกใช้งานได้เฉพาะจาก **Subclass** ของ **Abstract Class** เท่านั้น โดยที่ไม่สามารถเรียกใช้งาน **Constructor** ของ **Abstract Class** โดยตรงได้ นอกจากนี้ยังสามารถสร้าง **Constructor** ที่ไม่มีพารามิเตอร์หรือพารามิเตอร์มีค่าเริ่มต้นเองได้เช่นกัน

- 3.5. ภายใน **Interface** มี **Constructor** หรือไม่? เพราะเหตุใด?

ภายใน **Interface** ไม่มี **Constructor** เพราะ **Interface** เป็นแค่แบบฟอร์มที่บอกว่า **Class** จะต้อง มี **method** อะไรบ้าง แต่ไม่ได้ระบุว่าให้สร้าง **Constructor** หรือโครงสร้างอื่นใด ๆ ดังนั้นไม่จำเป็นต้องมี **Constructor** ใน **Interface** อย่างเดียวกันกับ **Abstract Class** โดยทั่วไป **Constructor** จะถูกนิยามใน **Class** ที่สืบทอด **Abstract Class** หรือ **implement Interface** นั้นเอง

4. ลำดับขั้นการปฏิบัติการ

- 4.1. ให้ผู้เรียนสร้าง **Abstract Class** ของรถถัง(**ClassicTank**) โดยจะต้องมีรายละเอียดดังต่อไปนี้
- 4.1.1. Properties : **HP** เพื่อกำหนดค่าพลังให้กับรถถัง

- 4.1.2. Properties : Str เพื่อกำหนดค่าความแรงในการยิงของรถถัง
- 4.1.3. Properties : Vit เพื่อกำหนดค่าพลังป้องกันของรถถัง
- 4.1.4. Properties : BaseDamage เพื่อการกำหนดค่าพลังการโจมตีพื้นฐาน
- 4.1.5. Method : SetHP() ; เพื่อทำการกำหนดค่าพลังเริ่มต้น
- 4.1.6. Method : GetHP() ; เพื่อตรวจสอบค่าพลัง ณ เวลาปัจจุบัน
- 4.1.7. Method : Attack(Tank Enemy) ; เพื่อทำการยิงปืนใหญ่โจมตีศัตรู โดยการโจมตี จะเป็นการลดค่าพลังของรถถังฝั่งตรงกันข้าม (Enemy คือรถถังของศัตรู, Points คือค่าพลังโจมตีของเรา)
- 4.2. ให้ผู้เรียนสร้างคลาส NormalTank เพื่อสืบทอด ClassicTank เพื่อเขียนรายละเอียดของ Method ทั้งหมดอันได้แก่ SetHP() , GetHP() , Attack(Tank Enemy)
- 4.3. ในคลาสหลัก ให้สร้าง Instance จาก NormalTank อยู่จำนวน 2 คัน เพื่อทำการต่อสู้กัน โดยควรต้องมีบทบาทดังนี้
- 4.3.1. สร้างรถถัง A และ B ให้มีค่าพลังเบื้องต้นดังต่อไปนี้
- | ค่าสถานะ | รถถัง A | รถถัง B |
|------------|---------|---------|
| HP | 200 | 250 |
| Str | 12 | 8 |
| Vit | 9 | 10 |
| BaseDamage | 11 | 10 |
- 4.3.2. รถถังทั้ง A และ B ผลัดกันโจมตีซึ่งกันและกัน เพื่อมุ่งหวังให้ค่าพลังของฝั่งตรงกันข้ามลดลงจนค่า HP = 0
- 4.3.3. รายละเอียดของพลังการโจมตีสามารถคำนวณได้ตามสมการดังต่อไปนี้

$$\text{DamagePoint} = \text{MyTank_BaseDamage} * \text{Floor}(\text{MyTank_Str} / \text{Enemy_Vit}) * \text{Random}(0.7, 0.9)$$
- 4.3.4. แสดงผลการทำงานผ่าน Console เพื่อให้เห็นรายละเอียดค่าพลังปัจจุบันของรถถังแต่ละคัน พลังการโจมตี ณ ขณะนั้น จนกว่าจะมีรถถังคันใดคันหนึ่งมีค่า HP = 0

โค้ดโปรแกรมภายใน Abstract Class

```
package Project;
abstract class ClassicTank{
    int Str, Vit, BaseDamage;
    double HP , point_A , point_B ;
    public abstract void setHP();
    public abstract void getHP();
    public abstract void attack();
}
```

โค้ดโปรแกรมภายใน NormalTank

```
class normalTankB extends ClassicTank{
    public void setHP() {
        HP = 250;
        Str = 8;
        Vit = 10;
        BaseDamage = 10;
    }
    public void getHP() {
        System.out.println("|-- Tank B |--");
        System.out.println(" HP : "+HP);
        System.out.println(" Str : "+Str);
        System.out.println(" Vit : "+Vit);
        System.out.println(" BaseDamage : "+ BaseDamage);
    }
    public void attack() {
        float min = (float) 0.7 ;
        float max = (float) 0.9 ;
        float number = (float)(Math.random()*(max-min+0.1)+min);
        double DamagePoint = BaseDamage * (0.8) * number;
        System.out.println(" DamagePoint _tankB : "+ DamagePoint);
        point_B = DamagePoint;
    }
}
```

โค้ดโปรแกรมภายในฟังก์ชันการทำงานหลัก

```
}

normalTankA tankA = new normalTankA();
normalTankB tankB = new normalTankB();

tankA.setHP();
tankA.getHP();
System.out.println("-----");
tankB.setHP();
tankB.getHP();
System.out.println("----- ROUND 1 -----");
tankA.attank();
tankB.attank();
tankA.getHP();
tankB.getHP();
System.out.println("----- ROUND 2 -----");
tankA.attank();
tankB.attank();
tankA.getHP();
tankB.getHP();
}
```

ผลลัพธ์การทำงานของโปรแกรม

4.4. เปลี่ยน Abstract Class ให้กลายเป็น Interfaces และเปรียบเทียบผลลัพธ์การทำงานของโปรแกรม

หลังจากเปลี่ยน Abstract Class เป็น Interface แล้ว เกิดอะไรขึ้นอย่างไร? อธิบายพร้อมยกตัวอย่างประกอบให้ชัดเจน

--

5. สรุปผลการปฏิบัติการ

[illegible]

6. คำถามท้ายการทดลอง

6.1. เมื่อใดจึงควรเลือกใช้งาน Abstract Class

การเลือกใช้ **abstract class** นั้นขึ้นอยู่กับการออกแบบโปรแกรม หากเรามีคลาสที่มีความสามารถคล้ายคลึงกันและต้องการให้ **method** และ **properties**..... บางส่วนของทั้งหมดเหมือนกัน แต่การทำงานที่แตกต่างกัน เราสามารถสร้าง **abstract class** ขึ้นมาและกำหนด **method** ที่เหมือนกันใน **abstract class** นั้น ๆ ได้ และให้คลาสที่สืบทอดมาจาก **abstract class** นี้ต้อง **implement method** ที่เหมือนกันนั้นโดยเฉพาะ ซึ่งจะช่วยให้โค้ดมีความสมบูรณ์และใช้งานได้ง่ายขึ้น

.....

6.2. เมื่อใดจึงควรเลือกใช้งาน Interface

การเลือกใช้ **Interface** ขึ้นอยู่กับวิธีการออกแบบโปรแกรม หากเราต้องการให้คลาสสามารถใช้งาน ได้กับคลาสอื่นที่มีความแตกต่างกัน แต่การทำงานบางอย่างที่เหมือนกัน เราสามารถกำหนด **Interface** ขึ้นมาและให้คลาสที่ต้องการใช้งานต้อง **implement Interface** นี้ด้วย

ซึ่งจะช่วยให้โค้ดมีความยืดหยุ่นและสามารถเปลี่ยนแปลงโครงสร้างของโปรแกรมได้ง่ายขึ้น
