I. Dependency

- ขึ้นอยู่กับอีก Class หนึ่ง


```
public class TestStudent{
    public static void main(String[] args){
        Student std = new Student("Mr.X");

    }
}
```

```
public class Student{
    private String name;

    public Student(String name){
    this.name = name;
    }
    public String getName(){
    return name;
    }
}
```

II. Aggregation

- แบ่งเป็น Container และ Component

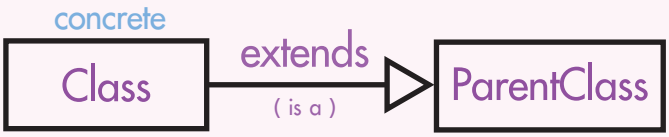

```
public class Student{
    private String name;
    private int stdId;

    public Student(String name,int stdId){
    this.name = name;
    this.stdId = stdId;
    }
    public String getName(){
    return name;
    }
}
```

```
public class Room{
    private int roomNum;
    private Student[] std;

    public Student(int roomNum,Student[] std){
    this.roomNum = roomNum;
    this.std = std;
    }
    public int getRoomNum(){
    return roomNum;
    }
}
```

III. Inheritance



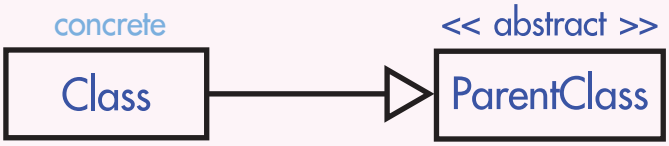

```
public class Graduate extends Student {
    private String project;

    public Student(String name,int stdId, String project){
    super(name, stdId);
    this.project = project;
    }
    public String getNum(){              //Override
    return "Mr."+super.getName();        //เรียกผ่าน Class พ่อ
    }
}
```

```
public class Student{
    private String name;
    private int stdId;

    public Student(String name,int stdId){
    this.name = name;
    this.stdId = stdId;
    }
    public String getName(){
    return name;
    }
}
```

III. Abstract



- มี อย่างน้อย 1 Method ที่ไม่สมบูรณ์ทำให้ต้อง Implement หรือ สมบูรณ์แล้วแต่ไม่อยากให้ new object

```
public abstract class Circle extends Shape {
    private double radius;

    public double area(){
    return Math.PI*Math.pow(radius,2);
    }
    public double getRadius(){return radius;}
}
```

```
public abstract class Shape{
    protected int shaped;

    public abstract double area();
    public abstract double calculate();
}
```

```
public class Circle extends Circle {
    public double calculate(){
    return 2*Math.PI*getRadius;
    }
}
```

IV. Interface

- เป็นการ Lock โครงสร้าง (เหมือนเป็นกฎข้อบังคับว่าต้องมี)
- Attributes เป็น Constant + Method ที่ไม่สมบูรณ์เท่านั้น
- Interface supports multiple inheritance

```
public abstract class Circle implements Shape {
    private double radius;

    public double area(){
    return Math.PI*Math.pow(radius,2);
    }
    public double getRadius(){return radius;}
}
```

```
public interface Shape{
    int COLOR=256;

    public abstract double area();
    public abstract double calculate();
}
```

```
public class Circle extends Circle {
    public double calculate(){
    return 2*Math.PI*getRadius;
    }
}
```

- Comparable Interface

```
public class Object implements Comparable<Object> {
   public int compareTo (Object obj){
   return method() - o.method() ;
   }
}
```

V. Polymorphism

- เป็นการ Downgrade
- เรียกใช้ได้แค่บรรพบุรุษเดียวกัน
- Parent Class ชี้ Class ลูก

```
Shape s1;                //Reference Type
s1 = new Square();       //Object Type

Square s2;               //Reference Type
s2 = new Square2();      //Object Type
```