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

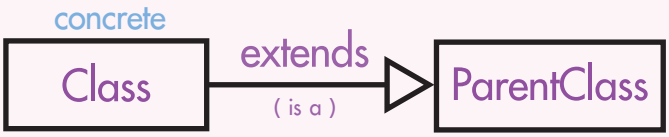

```
public class Student{
    private String name;
    private int stdId;

    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



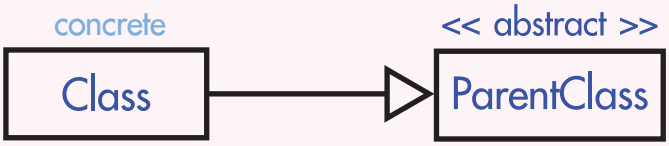

```
public class Graduate extends Student {
    private String project;

    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 Circle2 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 Circle2 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() ;
   }
}
```

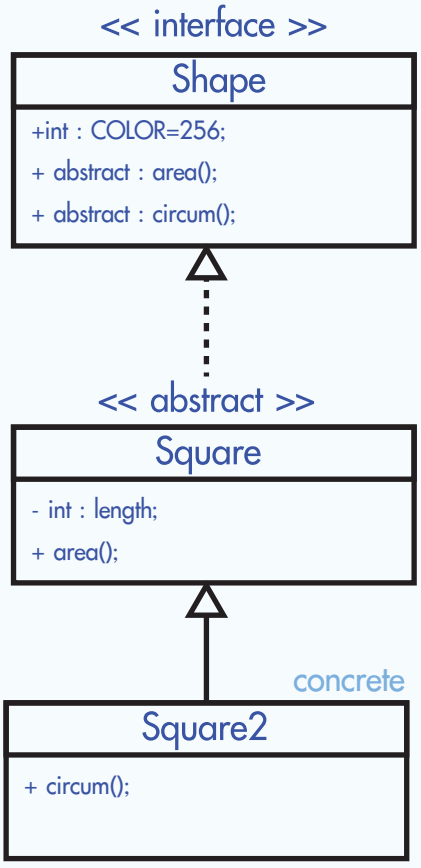
**เรียกใช้เหมือนปกติเช่น rectangle1.compareTo(regtangle2)

V. Polymorphism

- เป็นการ Downgrade เรียกใช้ได้แค่บรรพบุรุษเดียวกัน
- Parent Class ชี้ Class ลูก
- เกิดจาก 3 องค์ประกอบ
  1. Inheritance
  2. Dynamic Binding ขึ้นอยู่กับสิ่งที่ไปผูก ณ ตอนนั้น
     - ผูกค่าตอน runtime
  3. Method Ridding
- ทำให้เกิดการยืดหยุ่น ทำงานได้หลากหลายรูปแบบ

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

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



VI. ARRAY

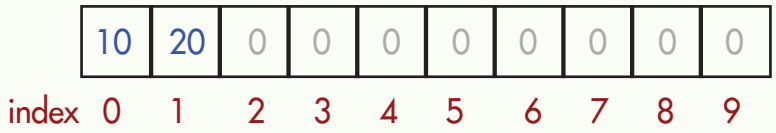
- Array เป็น Object เก็บได้หลายค่าภายใต้ชื่อเดียวกัน
- มี index ไว้ใช้กำกับตำแหน่งค่าใน Array เริ่มต้น index ที่ 0 ถึง num.length-1

การเขียน

```
1.   int [ ] nums = new int[10];          //ค่าพื้นฐานที่เก็บไว้คือ 0, false, null
หรือ int [ ] nums;                        //array reference(null)-แค่ประกาศ
     nums = new int[10];                  //จองพื้นที่
2. int [ ] nums = {10,25,50}
3. int [ ] nums = new int[] {10,25,50}    //Syntax แบบ 1+2
```

**การเขียนประเภท Object Array เขียนได้ดังนี้ Student [ ] std = new Student [10]

ลักษณะ




```
int [ ] nums = new int[10];   //ประกาศ
nums[0] = 10;                 //ใส่ค่า
nums[1] = 20;                 //ใส่ค่า
```

ความยาว = nums.length = 10
ช่องแรก = 0
ช่องสุดท้าย = nums.length-1 = 9

การนำไปใช้

- ใช้ Loop ช่วยในการใช้ เช่น นับจำนวน, บวกลบ
- แสดงผลใช้ for หรือ แบบย่อดังนี้ for (ชนิด ชื่อตัวแปรใหม่ : ชื่อ Array ที่ต้องการแสดง)

  เช่น for (int show : nums){ system.out.println(show); } **ถ้าไม่กฎบัดให้ใช้ for ปกติ
- หากมีจำนวน Array ที่ไม่เท่ากัน ให้จัดการส่วนที่ที่เท่าไปก่อน แล้วค่อยจัดการส่วนที่เกิน (อันที่ยาวกว่า)

for 1 : เริ่มที่ 0 ทำจนถึง < num2.length
for 2 : เริ่มที่ num2.length ทำจนถึง < num1.length

- ถ้าเกิด Error : OutOfBound คือขนาดเกินให้ไปตรวจสอบที่มีการเช็ค/เพิ่มค่า Array

VII. Another

1. Object.equals(obj)

```
public boolean equals(Object obj) {
        Book temp = null;
        if (obj != null && obj instanceof Book) {      //Object ที่รับเข้ามาไม่ว่างและเป็นลูกหลานของ Book
            temp = (Book) obj;                         //Casting ให้ Object เป็น book
            if (isbn == temp.isbn) {                   //ตัวแปลที่ต้องการเช็ค
                return true;                           //ถ้าตรงกันให้ return true
        } }
        return false;                                  //ถ้าไม่ตรงกันให้ return false
}
```

2. Wrapper Class

แปลง Primitive ----> String : Integer.toString (int_variable) เช่น String id = Integer.toString(123456);
แปลง String ----> Primitive : Integer.parseInt (String_variable) เช่น int stuYear = Integer.parseInt(id);