Software Construction
Midterm Scenario

Ed. 3

25 / 09 / 2016

Contents

Part I : Type Conversion - Primitive Data Type

Scenario 1 – assign big type to small type

```
int i = 10;
long l = 100;
float f = 0.2f;
double d = 0.1;

int j = l;     // Error
int k = d;     // Error
float g = d;   // Error
```

Summary – primitive type ไม่สามารถ assign type ที่ขนาดใหญ่กว่า ให้ type ที่ขนาดเล็กกว่าได้

Solution – สามารถแก้ปัญหาได้โดยการ cast

Scenario 2 – assign small type to big type

Summary – สามารถ assign จาก small ไป big ได้เลย ไม่จำเป็นต้อง cast

```
int i = 10;
long l = 100;
float f = 0.2f;
double d = 0.1;

long m = i;
double e = i;
```

Part II : Type Conversion – Reference Data Type

Base Code

```
public class A {

}
public class B extends A{

}
public class A {

}
A a1 = new A();
A a2 = new B();
B b1 = new B();
C c1 = new C();
```

Scenario 1

Summary

```
A a3 = b1;
B b2 = (B) a1;      // Run time error
B b3 = (B) a2;
C c2 = (C) b1;      // Compile error
```

- auto convers จากคลาสลูกไปหาคลาสแม่ได้ ไม่มีปัญหา
- cast จากคลาสแม่ไปคลาสลูกมี 2 แบบ
    - ตอนที่สร้าง สร้างด้วย constructor ของคลาสแม่ (a1)
        แบบนี้ถ้า cast จะไม่ขึ้นเส้นแดง แต่ตอนรันจะบึ้ม
    - ตอนที่สร้าง สร้างด้วย constructor ของคลาสลูก (a2)
        แบบนี้ cast ได้ ไม่มีปัญหา
- ไม่สามารถ cast จากคลาสที่ไม่มีความสัมพันธ์กันทาง Inheritance

Cause

- เมื่อตอนที่สร้าง obj. ตัว constructor จะทำการจองพื้นที่สำหรับ attribute และ method
- B เป็นคลาสลูกของ A จะได้ attribute และ method ทั้งหมดของ A มา แต่สามารถมีเพิ่มได้
- ดังนั้นเมื่อตอนที่สร้าง a1 จะจองพื้นที่ไว้แค่สำหรับ class A เมื่อนำมา cast เป็นคลาส B จึงทำไม่ได้เพราะข้อมูลที่เตรียมไว้ไม่มีส่วนที่คลาส B ต้องมี
- ในทางเดียวกัน a2 ถูกสร้างด้วย constructor B จะจองพื้นที่ไว้เท่าขนาดของคลาส B แม้จะเก็บไว้คลาส A ก็ตาม เมื่อนำมา cast จึงสามารถทำได้ เพราะมีข้อมูลทั้งหมดที่คลาส B ต้องมี

Part III : Overloading

Scenario 1 – access modify ต่างกัน

```
public class A {

	public void methodA(int a){

	}
	private int methodA(String s){
		return 0;
	}
}
```

Summary – สามารถใช้ access modify และ return type ต่างกันได้

Scenario 2 – ชื่อ parameter ต่างกัน

```
public class A {

	public void methodA(int x, int y){

	}
	private void methodA(int w, int h){

	}
}
// Error : Duplicate method
```

Summary – แม้ชื่อ parameter จะต่างกัน แต่ type ยังคงเหมือนกัน ไม่ถือว่าเป็นการ overload

- ถ้าอยู่ในคลาสเดียวกัน -> Complie error
- ถ้าทำให้คลาสลูก จะกลายเป็น override แทน

Scenario 3 – สลับ type

```
public class A {

     public void methodA(int x, String y){

     }
     public void methodA(String y, int x){

     }
}
```

Summary – แบบนี้ก็ยังถือว่า overload ไม่บึ้มด้วย

Scenario 4 – overload final method

```
public class A {

     public final void methodA(int x){

     }
}
public class B extends A{

     public void methodA(String x){

     }
}
```

Summary – final method แม้ว่าจะ override ไม่ได้ แต่ยัง overload ได้อยู่

Part IV : Constructor

Scenario 1

```
public class A {
     public A(){
          System.out.println("constructor A");
     }
}
public class B extends A{

     public B(){
          System.out.println("constructor B");
     }
}
```

```
A a1 = new A();
A a2 = new B();
B b1 = new B();
```

Summary

- constructor คลาสลูก จะเรียก constructor ของคลาสแม่เสมอ
- (a2) แม้ว่าตอนประกาศ type ของ obj. เป็นคลาสแม่ แต่สร้างด้วย constructor ของคลาสลูก การทำงานก็จะเข้าที่ constructor ของคลาสลูกเหมือนเดิม

Scenario 2

```
public class A {
	public A(){
		System.out.println("constructor A");
	}
	public A(int a){
		System.out.println("constructor with int a");
	}
}
```

```
public class B extends A{
	public B(){
		System.out.println("constructor B");
	}
}
```

```
B b1 = new B();
```

```
constructor A
constructor B
```

```
public class B extends A{
	public B(){
		super(3);
		System.out.println("constructor B");
	}
}
```

```
B b1 = new B();
```

Summary

- ถ้าคลาสลูกไม่ระบุว่าจะเรียก constructor ตัวไหนของแม่ ระบบจะเรียก default constructor ให้เอง
- คลาสลูกสามารถระบุ constructor ของแม่ได้ ผ่านคำสั่ง super

Scenario 3

```
public class A {
	public A(int a){
		System.out.println("constructor with int a");
	}
}
```

```
public class B extends A{
	public B(){
		System.out.println("constructor B");
	}
}
```

Compile Error

Summary

- เมื่อกี้บอกไปแล้วว่า ถ้าคลาสลูกไม่ระบุ constructor ของคลาสแม่ จะไปเรียก default constructor ของคลาสแม่ พอกลับขึ้นไปดู คลาสแม่ไม่มี default constructor มันก็จะ error

Scenario 4

```
public class C {
     public C(int i){
          System.out.println("create C"+i);
     }
}
public class A {
     public A(){
          System.out.println("constructor A");
     }
}
public class B extends A{
     public C c1 = new C(1);
     public final C c2 = new C(2);
     public C c3;

     public B(){
          c3 = new C(3);
     }
}
```

```
constructor A
create C1
create C2
create C3
```

Summary

- เมื่อเข้า constructor ของ B จะกระโดดเข้าไปยัง constructor ของคลาส A ก่อน
- จากนั้นจึงมากำหนดค่า instance variable ต่างๆที่อยู่นอกเมธอด
- จากนั้นจึงทำตามคำสั่งต่างๆภายในตัว constructor

** ส่วนมากชอบคิดว่า ประกาศไว้ข้างนอกแล้ว มันจะมีค่าก่อนที่จะเข้า constructor มันจะมีปัญหาแบบนี้

```
public class A {
     public A(int a){

     }
}
public class B extends A{
     public int a = 5;

     public B(){
          super(a); // Error เพราะ a ยังไม่ถูกกำหนดค่า
     }
}
```

Part V : .equal & ==

Scenario 1

```
String s1 = "Apple";
String s2 = "Apple";
String s3 = new String("Apple");
String s4 = new String("Apple");

System.out.println(s1 == s2);    // true
System.out.println(s1 == s3);    // false
System.out.println(s3 == s4);    // false
System.out.println(s1.equals(s3));    // ture
```

Summary

- == จะเช็คว่า obj. ทั้งสองฝั่งมี address เดียวกันรึเปล่า

- s1 s2 แบบคล้ายๆ primitive type ระบบจะเก็บข้อมูลไว้ที่ address เดียวกัน

**มีเฉพาะ String นี้เหละที่งงแบบนี้

- s3 s4 สร้างแบบ obj. ดังนั้นจะมี address คนล่ะที่กัน

- .equal เป็นเมธอดที่เฉพาะของแต่ล่ะคลาส สำหรับคลาส String จะทำการเช็คทุกตัวว่าเหมือนกันหรือไม่

Scenario 2

```
public class A {
	public static C c0 = new C(0);
	public final C c1 = new C(1);
	public C c2 = new C(2);
	public C c3;
	public C c4;

	public A(C c3){
		this.c3 = c3;
		c4 = new C(4);
	}
}
```

```
C c3 = new C(3);
A a1 = new A(c3);
A a2 = new A(c3);

System.out.println(a1.c0 == a2.c0);		// true
System.out.println(a1.c1 == a2.c1);		// false
System.out.println(a1.c2 == a2.c2);		// false
System.out.println(a1.c3 == a2.c3);		// true
System.out.println(a1.c4 == a2.c4);		// false
```

Summary

- c0 ถูกประกาศเป็น static นั้นหมายความว่าจะเป็นตัวแปรของคลาส ไม่ว่าจะเรียกจาก obj. ตัวไหน ก็คิดตัวเดียวกัน (c0 เป็นของคลาส A ไม่ว่า a1 a2 มันก็คือตัวเดียวกัน)
- c1, c2, c4 instrace variable ไม่ว่าจะประกาศตรงไหน ก็จะเป็นของ obj ใคร obj มัน (c1 ของ a1 กับ c1 ของ a2 เป็นคนละตัวกัน)
- c3 สร้างจากภายนอก แล้วส่งเข้าไป นั่นคือส่ง obj ตัวเดียวกันให้ทั้ง a1 a2 ทั้งสองตัวเลยใช้ obj.ตัวเดียวกันอยู่

Part VI : String Method

```
String s1 = "Apple";
String s2 = "Apple";
String s3 = "Cee";
String s4 = "apple";
String s5 = "Ab";
String s6 = "    adw    \n";
String s7 = "1 3 5 6";
```

```
System.out.println(s1.charAt(4));     // e

System.out.println(s1.compareTo(s2)); // 0
System.out.println(s1.compareTo(s3)); // -2
System.out.println(s1.compareTo(s5)); // 14

System.out.println(s1.contains("ple"));    // true
System.out.println(s1.contains("Bee"));    // false

System.out.println(s1.equals(s4));         // false
System.out.println(s1.equalsIgnoreCase(s4));   // true

System.out.println(s1.substring(3)); // le
System.out.println(s1.substring(1, 3));    // pp

System.out.println(s6.trim());   // adw

System.out.println(s1.length()); // 5

System.out.println(s1.indexOf('p')); // 1
System.out.println(s1.indexOf("c")); // -1
System.out.println(s1.indexOf('p', 2));    // 2

System.out.println(s1.lastIndexOf('p'));  // 2
System.out.println(s1.replace("pp", "abc"));   // Aabcle
System.out.println(s1.replace('p', 'c'));      // Accle
```

```
String[] ls = s7.split(" ");
for(int i=0; i<ls.length; i++){
     System.out.print(ls[i] + ",");
}
// 1,3,5,6,

System.out.println(s1.toUpperCase()); // APPLE
System.out.println(s1.toLowerCase()); // apple
```