Q10021 - INFO: Overloading, Overriding, Polymorphism คืออะไร ?

Created on 4/27/2007 5:26 PM.

Last Modified on 5/3/2007 9:39 PM.

Last Modified by admin.

Skill Level: Beginner.

Article has been viewed 2134 times.

Rated 6 out of 10 based on 12 votes.

Print Article

Email Article

Overloading, Overriding, Polymorphism ?

Method Overloading

Method overloading คือการที่เราสามารถสร้าง method หลายตัว ที่มีชื่อเหมือนกัน แต่มี signature ต่างกัน ภายใน class เดียวกันได้ อย่างเช่น

```
// TestOVL.java
class TestOVL {
public void m() {} // (1)
public void m(int i) {} // (2)
public void m(double d) {} // (3)
public void m(int i, double d) {} // (4)
public void m(double d, int i) {} // (5)
//public int m(int i) {} // (6)
}
```

signature ของ method จะประกอบไปด้วย 1. ชื่อของ method 2. จำนวน parameter 3. ตำแหน่งและ type ของแต่ละ parameter ส่วน return type นั้นไม่รวมอยู่ใน method signature นั่นหมายความว่า เราไม่สามารถสร้าง method ที่มี return type ต่างกัน แต่มี signature เหมือนกันได้

จากตัวอย่าง

(1) มี signature คือ ชื่อ m และ ไม่มี parameter

(2) มี signature คือ ชื่อ m, รับ parameter 1 ตัว และ มี type เป็น int

(3) มี signature คือ ชื่อ m, รับ parameter 1 ตัว และ มี type เป็น double

(4) มี signature คือ ชื่อ m, รับ parameter 2 ตัว, โดยตัวแรกเป็น int และ ตัวที่สองเป็น double

(5) มี signature คือ ชื่อ m, รับ parameter 2 ตัว, โดยตัวแรกเป็น double และ ตัวที่สองเป็น int ในกรณีนี้ signature ของ (5) ไม่เหมือนกับ (4) เนื่องจากลำดับของ parameter ไม่เหมือนกัน

(6) มี signature คือ ชื่อ m, รับ parameter 1 ตัว และ มี type เป็น int ในกรณีนี้จะเห็นว่า signature ของ (6) นั้นเหมือนกับ (2) เนื่องจาก return type ไม่ได้รวมอยู่ใน signature ด้วย ดังนั้น ถ้าเรา compile โดย uncomment (6) จะได้เป็น compile error

```
TestOVL.java:8: m(int) is already defined in TestOVL
public int m(int i) {} // (6)
^
1 error
```

ประโยชน์ของ method overloading ก็คือ ในกรณีที่เราต้องการสร้าง method ที่ทำงานแบบเดียวกัน (มีความหมายหรือ semantics แบบเดียวกัน) แต่มี parameter ที่ใช้แตกต่างกัน เราไม่จำเป็นต้องเขียนเป็น method ใหม่ ที่มีชื่อต่างกัน แบบนั้นจะทำให้สับสนและใช้งานลำบาก

method overloading ถือว่าเป็น polymorphism แบบหนึ่ง ที่เรียกว่า adhoc polymophism ซึ่งเป็น polymorphism แบบหลอก ๆ เพื่อให้ได้ผลในลักษณะคล้ายกันกับ polymorphism ของจริง (pure polymorphism) คือการเรียก method หนึ่ง แต่ผลลัพท์ของการเรียกนั้น อาจมีได้หลายรูปแบบ (many forms) ขึ้นกับว่า argument ที่ส่งมานั้นเป็นแบบไหน (ต่างจาก pure polymophism ตรงที่ว่า pure polymorphism จะดูจาก actual type ของ object ที่ reference ที่ใช้เรียกนั้นชี้อยู่ ในการตัดสินใจว่าจะเรียก method() ของ class ใด)

Method Overriding

Method overriding เป็นกลไกที่ subclass สามารถใช้เพื่อเลือกที่จะไม่ใช้ implementation ที่มากับ instance

method ที่ inherit มาจาก superclass ได้ โดยระบุ implementation ของ method นั้นเองภายใน class

Note: การทำ method overriding จะใช้กับ instance method เท่านั้น ถ้าเป็น static หรือ class method จะเป็นการทำ hiding แทน

สมมุติให้ class B เป็น subclass ของ class A ถ้ามองในเรื่องของ inheritance class B จะ inherit (สืบทอด) state (variables) และ behavior (methods) ของ A มาทั้งหมด ในส่วนของ method นั้น ในการ inherit จะแบ่งได้เป็น 2 ส่วนคือ inherit ส่วนที่เป็น contract หรือพันธสัญญา ของ superclass และ inherit ส่วนที่เป็น implementation ของ contract นั้น ๆ ตัวอย่างเช่น ถ้า class A มี method m() {} class B จะ inherit ทั้ง contract หรือการที่ว่า class A สามารถทำ operation m() ได้ ซึ่ง m() จะรับและ return ค่าตามที่ได้ระบุไว้ นอกจากนี้ B ยัง inherit ส่วนที่เป็น implementation หรือเนื้อของ method m() ที่กำหนดโดย A ว่าจะทำงานอย่างไรมาด้วย

```
// TestOVR.java
class A {
public void m() {
System.out.println("A's implementation");
}
}
class B extends A {
}
class TestOVR {
public static void main(String[] args) {
B b = new B();
b.m();
}
}
```

Output จะได้เป็น

```
A's implementation
```

จาก code นี้ B จะ inherit method m() พร้อมกับ implemention ที่สร้างโดย A มาด้วย ถ้าเรามี object ของ class B แล้วเรียกใช้งาน method m() ผลที่ได้ก็คือข้อความ "A's implementation" ซึ่งเป็นสิ่งที่ class A ได้กำหนดไว้

จะทำอย่างไร ถ้า B ไม่ต้องการใช้ implementation ที่กำหนดไว้โดย A ? B สามารถทำได้โดยการทำ method overriding โดยเขียน method m() ไว้ใน class B เอง อีกครั้งหนึ่ง ดังนี้

```
// TestOVR2.java
class A {
public void m() {
System.out.println("A's implementation");
}
}
class B extends A {
public void m() {
System.out.println("B's implementation");
}
}
class TestOVR2 {
public static void main(String[] args) {
B b = new B();
```

```
b.m();
}
}
```

Output จะได้เป็น

```
B's implementation
```

จาก code จะเห็นว่า B ทำการ override method m() ที่ได้มาจาก A เพื่อที่ว่าจะได้ไม่ต้องใช้ implementation ของ method m() ที่กำหนดโดย A ในกรณีนี้ B เปลี่ยนให้ method m() พิมพ์ข้อความว่า "B's implementation" แทน ถ้าเรามี object ของ class B แล้วเรียก method m() ผลลัพท์ที่ได้ก็คือข้อความ "B's implementation" แทนที่จะเป็น "A's implementation"

เมื่อมีการทำ method overriding ใน subclass ถ้ามีการเรียกใช้ overriding method (method ใน subclass) ไม่ว่าจะเป็นการเรียกใช้ภายใน subclass นั้น หรือ class อื่นเรียกผ่าน reference ของ subclass implementation ที่กำหนดใน subclass นี้ จะถูกเรียกใช้เสมอ

ถ้าเราต้องการใช้ implementation ที่เป็นของ superclass เราสามารถทำได้โดยใช้ keyword super อย่างเช่น super.m() เพื่อเรียกใช้ method m() โดยใช้ implementation ที่มากับ superclass

สรุปง่าย ๆ ว่า method overriding เป็นกลไกที่ใช้โดย subclass เมื่อไม่ต้องการใช้ implementation ที่มากับ method ที่ inherit จาก superclass และต้องการระบุ implementation ที่ต้องการแทน

Polymorphism

poly = many (มีหลายอย่าง, หลายแบบ), morphos = form (รูปแบบ), polymorphic จึงหมายถึงการมีมากกว่า 1 รูปแบบ หรือ many forms

แนวความคิดของ polymorphism ก็คือการที่ object ที่เรียกใช้งาน method ของอีก object หนึ่ง ไม่จำเป็นต้องรู้ว่า object ที่รับ message นั้น เป็น object ของ class อะไร

```
a A ---m()---> ?
```

จากรูปจะเห็นว่า object a ของ class A สามารถเรียก method m() ของ object ใด ๆ ที่มี method m() ได้ โดยไม่ต้องรู้ว่า object ? เป็นของ class อะไร

อย่างไรก็ตาม ในความเป็นจริง เราจำเป็นต้องแน่ใจว่า object ที่ถูกเรียกใช้งาน method m() นั้น จะต้องมี method นั้นให้เรียกใช้จริง ๆ ดังนั้น ในภาษาจาวาจะใช้ class hierarchy ในการกำหนดว่า object ที่ถูกเรียก method นั้นจะเป็น object ของ class ใดได้บ้าง class ที่เรียกใช้ method m() จะรู้เพียงว่า ตัวที่ถูกเรียกนั้น เป็น class ในตระกูลใด แต่จะไม่รู้อย่างแน่ชัดว่าเป็น object ของ class ใด

```
a A ---m()---> x X
```

จากรูป object a ของ class A สามารถเรียกใช้ method m() ของ object x ได้ โดยที่ object x นั้นจะเป็น object ของ class X หรือของ subclass ของ X ก็ได้ (X เป็น class ที่มี method m() ให้เรียกใช้ได้)

(สมมุติว่า Y เป็น subclass ของ X) จะเห็นว่า ถ้า x เป็น object ของ class X method m() ของ class X ก็จะถูกเรียกใช้งาน แต่ถ้า x เป็น object ของ class Y method m() ของ class Y ก็จะถูกเรียกใช้งาน

จะเห็นว่าการเรียกใช้ method m() โดย A นั้นจะมีผลลัพท์ได้มากกว่า 1 แบบ (many forms) ส่วนผลลัพท์จะเป็นแบบไหนนั้น ขึ้นอยู่กับว่า object ของ class ใด เป็นตัวที่ถูกเรียก method จาก object ของ A

ในจาวานั้น variable ของ reference type จะมีลักษณะเป็น polymorphic คือมีหลายรูปแบบ ซึ่งหมายถึงว่า ถ้าเรามี reference ของ class A เราสามารถใช้ reference นี้ ชี้ไปที่ object ของ class A หรือ subclass ของ class A ก็ได้ เราเรียก variable แบบนี้ว่า polymorphic variable ตัวอย่างเช่น

```
A a = new A();
a = new B();
a = new C();
```

สามารถทำได้ เนื่องจาก object ของ B และ C เป็น object ของ subclass ของ A (สมมุติให้ B เป็น subclass ของ A และ C เป็น subclass ของ B)

ถ้าเรามี method f() ที่รับ parameter เป็น polymorphic variable อย่างเช่น

```
public void f(A a ) {
a.m();
```

```
}
```

เราจะสามารถส่ง object ของ class A, B หรือ C ก็ได้ ให้กับ method นี้ จะเห็นว่า เราสามารถเปลี่ยนการทำงานของ method f() ได้โดยที่ไม่ต้องแก้ code เลยแม้แต่น้อย แต่ทำโดยเปลี่ยน argument ที่ส่งมาให้กับ method นี้ ถ้าเราส่ง object ของ A มาให้ ก็จะได้ผลแบบหนึ่ง ถ้าเราส่ง object ของ B มาให้ ก็จะได้ผลอีกแบบหนึ่ง

นี่คือประโยชน์ที่ได้รับจาก polymorphism ซึ่งก็คือ flexibility (ความยืดหยุ่น) ในแง่ที่ว่า เราสามารถเปลี่ยนการทำงานของ code เดิมที่มีอยู่แล้วได้ โดยไม่ต้องแก้ไข code เดิมเลย เราเพียงแต่ส่ง object ของ subclass ตัวใหม่ไปให้เท่านั้นเอง ก็สามารถเปลี่ยนการทำงานของ method ได้

การที่ code ของเรา flexible (มีความยืนหยุ่น) นั้นมีความสำคัญตรงที่ทำให้เราสามารถปรับเปลี่ยนการทำงานของ code นั้นได้ โดยแก้ไขไม่มาก ตรงนี้จะมีประโยชน์อย่างมาก เพราะว่าในการพัฒนา software นั้น มักจะมี change หรือการเปลี่ยนแปลงอยู่เสมอ (อาจจะเกิดจาก requirement ที่เปลี่ยนไป)

polymorphism จะช่วยให้เราสามารถใช้งาน code เดิมที่มีอยู่แล้วได้ โดยแก้ไขเพียงเล็กน้อย เมื่อมีความจำเป็นต้องการเปลี่ยนแปลงการทำงานบางอย่าง

ตัวอย่าง

```
// TestOVR3.java
class X {
public void m() {
System.out.println("X's implementation");
}
}
class Y extends X {
public void m() {
System.out.println("Y's implementation");
}
}
class Z extends Y {
public void m() {
System.out.println("Z's implementation");
}
}
class A {
public void f(X x) {
x.m();
}
}
class TestOVR3 {
public static void main(String[] args) {
A a = new A();
X x = new X();
Y y = new Y();
Z z = new Z();
x.m(); // (1)
y.m(); // (2)
z.m(); // (3)
x = y; // (4)
```

```
x.m(); // (5)
y = z; // (6)
y.m(); // (7)
}
}
```

Output จะได้เป็น

```
X's implementation
Y's implementation
Z's implementation
Y's implementation
Z's implementation
```

ในการเรียกใช้ instance method ในจาวานั้น จะมีการทำ dynamic binding คือทำการหาว่า method ที่จะเรียกนั้นเป็นของ class ใด ณ run-time การเรียก method จะดูจากว่า actual type หรือ type ของ object ที่ reference นั้นชี้อยู่เป็นของ class ใด อย่างเช่น ถ้า reference เป็นของ class X แต่ชี้อยู่ที่ object ของ class Y ที่เป็น subclass ในกรณีนี้ method ของ class Y ก็จะถูกเรียก

จากตัวอย่าง

ที่ (1) method m() ของ class X จะถูกเรียก เนื่องจาก reference x ชี้อยู่ที่ object ของ X

ที่ (2) method m() ของ class Y จะถูกเรียก เนื่องจาก reference y ชี้อยู่ที่ object ของ class Y

ที่ (3) method m() ของ class Z จะถูกเรียก เนื่องจาก reference z ชี้อยู่ที่ object ของ class Z

ที่ (4) มีการเปลี่ยนให้ reference x มาชี้ที่ object ของ class Y แทน มีผลทำให้การเรียก method ที่ (5) method m() ของ class Y จะถูกเรียก เนื่องจากมีการทำ dynamic binding โดยดูจาก actual type ของ object ที่ x ชี้อยู่ ซึ่งก็คือ object ของ Y

ที่ (6) มีการเปลี่ยนให้ reference y มาชี้ที่ object ของ class Z แทน มีผลทำให้การเรียก method ที่ (7) method m() ของ class Z จะถูกเรียก ด้วยเหตุผลเดียวกัน

ตัวอย่าง

```
// TestOVR4.java
class X {
public void m() {
System.out.println("X's implementation");
}
}
class Y extends X {
public void m() {
System.out.println("Y's implementation");
}
}
class Z extends Y {
public void m() {
System.out.println("Z's implementation");
}
}
class A {
```

```
public void f(X x) {
x.m();
}
}
class TestOVR4 {
public static void main(String[] args) {
A a = new A();
X x = new X();
Y y = new Y();
Z z = new Z();
a.f(x); // (1)
a.f(y); // (2)
a.f(z); // (3)
}
}
```

Output จะได้เป็น

```
X's implementation
Y's implementation
Z's implementation
```

จะเห็นว่า เรามี method f() ที่ประกาศไว้ให้รับ parameter เป็น object ของ class X อย่างที่ได้บอกแล้วว่า ในจาวา variable ของ reference type มีลักษณะที่เป็น polymorphic ทำให้เราสามารถส่ง object ของ class X หรือ object ของ subclass ของ X ไปเป็น argument ของ method นี้ได้ สังเกตว่าเราสามารถเปลี่ยนการทำงานของ method f() ได้โดยไม่ต้องแก้ไข เพียงแต่เปลี่ยน object ที่เป็น argument เท่านั้น ถ้าเราส่ง object ของ X ไปให้ในการเรียกที่ (1) method m() ของ X ก็จะถูกเรียก แต่ถ้าเราส่ง object ของ subclass ไปให้ ที่ (2), (3) method m() ของ subclass ก็จะถูกเรียก โดยขึ้นกับว่า object ของ class ไหนที่ถูกส่งเข้ามา

method overloading กับ polymorphism มีลักษณะที่คล้ายกัน (ในแง่ที่ว่า ผลลัพท์ของการเรียก method นั้น สามารถมีได้มากกว่า 1 แบบ ขึ้นกับว่า argument list ที่ใช้เป็นอย่างไร (สำหรับ overloading) หรือ actual type ของ object ที่ reference ชี้อยู่เป็นของ class ใด (สำหรับpolymorphism)

สองเรื่องนี้จะต่างกันตรงที่ว่า method overloading จะเป็นการเรียก instance/class method ของ class หนึ่งเท่านั้น และสามารถทำได้ตั้งแต่ compile-time สังเกตว่าตัวชื่อ method นั้นจะมีลักษณะเป็น polymorphic คือชื่อเดียวแต่ทำงานได้หลายแบบ

ในขณะที่ polymorphism เป็นเรื่องของการเรียก instance method ของ object หนึ่ง ซึ่ง method ที่จะถูกเรียกจริง ๆ นั้น อาจจะอยู่ได้ในหลาย class (แต่ใน class hierarchy เดียวกัน) ขึ้นกับว่า object ที่ถูกเรียกใช้นั้นเป็น object ของ class ใด polymorphism จำเป็นต้องใช้ dynamic binding ในการทำงาน ซึ่งทำงานตอน run-time สังเกตว่าในกรณีของ polymorphism variable ของ reference type จะมีลักษณะเป็น polymorphic คือ variable ตัวเดียว แต่สามารถชี้ไปยัง object ของ class ตัวเอง หรือ subclass ได้

นอกจากนี้ จะเห็นว่าในกรณีของ overloading จะมี method ชื่อเดียว แต่มี body (เนื้อ) ของ method หลายตัว ในขณะที่ polymorphism มี method ชื่อเดียว และมี body ของ method ตัวเดียว (ในแต่ล่ะ class)

Related Articles

No Related Articles Available.

Article Attachments

No Attachments Available.

Related External Links

No Related Links Available.

Help us improve this article...

What did you think of this article?

	1	2	3	4	5	6	7	8	9	10	
poor	○	○	○	○	○	○	○	○	○	○	excellent

Tell us why you rated the content this way. (optional)

Submit

Approved Comments...

ขอบใจมากครับ แต่ขอวิธีการเขียนโปรแกรมเชิงวัตถุทีนะครับแบบล่ะเอียดนะครับขอบคุณล่วงหน้าครับ	Approved: 3/7/2008 1:45 PM
เยี่ยมเลย	Approved: 2/13/2008 11:23 AM