Reusing Classes

Vittayasak Rujivorakul

Lecture 6

Reusing Classes

Vittayasak Rujivorakul

Lecture 6

Code Reuse

- 2 แนวทางในการ reuse class

  Composition (การประกอบ) :

  - การสร้าง class ใหม่จาก Object ของ class ที่มีอยู่แล้วใน class ใหม่

  – Inheritance (การสืบทอด) :

  - การสร้าง class ใหม่ให้มีชนิดเดียวกันกับ class ที่มีอยู่เดิมโดยเพิ่ม code ที่เป็นคุณลักษณะเพิ่มเติมโดยไม่ได้ทำการเปลี่ยนแปลงส่วนของ code ของ class เก่า (Compiler ทำหน้าที่ในการสืบทอดคุณลักษณะเก่าให้)

Composition Syntax

```
class WaterSource{
   private String s;
    WaterSource( ) {
     System.out.println("WaterSource( ) ");
     s = new String ("Constructed");
   }
   public String toString( ) {
            return s;
  }
}
public class SprinklerSystem{
  private String valve1,valve2,valve3,valve4;
  WaterSource s;
  int i;
  float f;
  void print( ) {
    System.out.println("valve1 = " + valve1);
   System.out.println("valve2 = " + valve2);
   System.out.println("valve3 = " + valve3);
   System.out.println("valve4 = " + valve4);
   System.out.println("i = " + i);
   System.out.println("f = " + f);
   System.out.println("s = " + s);
  }
  public static void main (String a[ ]) {
   SprinklerSystem x = new SprinklerSystem( );
   x.print( );
  }
}
```


toString ( ) Method

- ใน class จาวากำหนด toString ( ) เพื่อใช้ในการพิมพ์ object
- สามารถที่จะเรียกใช้ System. out. print ในการพิมพ์ object ได้ทุกตัว
- override toString ( ) เพื่อกำหนดสิ่งที่ต้องการพิมพ์ของ object นั้น
- หากแต่ toString ( ) มีจุดมุ่งหมายเพื่อการ debug มากกว่าที่จะเป็น fancy format ของ object ดังนั้นควรกำหนดลักษณะการพิมพ์ที่
  - ไม่ควรมีการขึ้นบรรทัดใหม่
  - ไม่มีรูปแบบมากเกินไป เพื่อที่จะพิมพ์ได้รวดเร็ว
  - ควรพิมพ์ชื่อของ class, fields, ค่าต่าง ๆ ที่สำคัญที่จะใช้บอกลักษณะหรือสถานะของ object นั้น

Initialization ของ object ใน composition

- ณ จุดที่กำหนด object => object ถูก initialized ก่อนที่จะมีการเรียก constructor

```
class Soap {
    String s = new String ("init at the declaration")
}
```

- ใน constructor เอง

```
Bath ( ) {
    s = new String ("init inside the constructor") ;
}
```

- ณ จุดก่อนที่จะมีการเรียกใช้ object นั้น

```
if (s == null)
    s = new String ("init at the point of use") ;
```


Inheritance Syntax

- Class สืบทอดสถานะและพฤติกรรมจาก Superclass
- Inheritance เตรียมกลไกสำหรับโครงสร้างและองค์ประกอบของโปรแกรม
- ประโยชน์ของ **Inheritance**
  - Subclasses จะทำพฤติกรรมพิเศษที่เพิ่มแตกต่างไปจากส่วนพื้นฐานที่มีอยู่ใน superclass นั่นคือทำให้สามารถจะ reuse code ใน superclass ได้
  - สามารถที่จะสร้าง superclass ที่เรียกว่า abstract class ซึ่งจะกำหนด "generic" behaviors โดยที่ abstract superclass กำหนดและทำการ implement เพียงบางส่วนของ behaviors นั้น นักพัฒนาอื่นสามารถมาใช้โดยทำการเพิ่มรายละเอียดผ่านทาง subclasses

การกำหนด class

```
Class Student
{
        // data members
        public static int NUM_OF_TESTS = 3 ;

        protected String name ;
        protected int total ;

        // methods
        public void computeGrade ( ) { }
        :
}
```

class data member

Protected fields สามารถเห็นได้จาก descendant object

การสร้าง Inheritance : extends

```
class GradStudent extends Student {
  public void computeGrade( ) {
    :
    if(total/NUM_OF_TESTS > 80)
      courseGrade = "Passed";
    else
      courseGrade = "Failed";
  }
}
```

```
class UndergradStudent extends Student {
  public void computeGrade( ) {
    :
    if(total/NUM_OF_TESTS > 70)
      courseGrade = "Passed";
    else
      courseGrade = "Failed";
  }
}
```

Keyword “super”

- Keyword “super” ใช้ในการอ้างถึง class แม่ (superclass) ของ class ปัจจุบัน
  - super.methodName( ); // อ้างถึง methodName ที่อยู่ใน class แม่
  - super( ); // เรียกใช้ constructor ของ class แม่
- เพื่อให้ class ลูกสามารถสืบทอดจาก class แม่และค่าต่าง ๆ ถูก initialized จาวาจะ automatic เพิ่มการเรียก base class constructor โดยอัตโนมัติใน derived class constructor

Inheritance และ Constructor

```
class Person {
    public void sayHello {
        System. out. println (“Hello”) ;
    }
}
```

มีความหมายเท่ากับ

```
class Person {
    public Person ( ) {
        Super ( ) ;
    }
    public void sayHello ( ) {
        System. out. println (“Hello”) ;
    }
}
```

Compiler ทำการเพิ่มโดยอัตโนมัติใน Java ถ้าไม่ extends class ใดจะ extends จาก Object by default

Constructor : No Default Constructor

- หากประกาศ Constructor ภายใน class เอง ไม่มีการสร้าง default constructor ให้อัตโนมัติ

```
class MyClass {
    public MyClass (int a) {
        . . .
    }
}
MyClass mC = MyClass ( ) ;          // INVALID
                                    // no matching constructor
```


User-Defined Constructor

- หากกำหนด Constructor โดยไม่เรียก super class constructor compiler ทำการเพิ่มให้โดยอัตโนมัติ

```
class MyClass {
  private myInt;
  public MyClass( ){
    myInt = 10;
  }
}
```

```
class MyClass {
  private myInt;
  public MyClass( ){
    super( );
    myInt = 10;
  }
}
```

ตัวอย่างให้ระวัง (Caution) !!

```
class Vehicle {
   private String vid;
   public  Vehicle(String vNo) {
      vid = vNo;
   }
   public void getVid( ) {
      return vid;
   }
}
```

```
class Car extends Vehicle {
   private int noOfSeats;
   public void setSeat(int n){
      noOfSeats = n;
   }
   public int getSeat( ) {
      return noOfSeats;
   }
}
```

Compilation Error เนื่องจากไม่มีการกำหนด constructor สำหรับ car( ) compiler ทำการเพิ่ม public Car( ) { super( ); } แต่ไม่มี constructor ที่ไม่มี argument ใน superclass Vehicle

ตัวอย่าง (ต่อ)

```
class Vehicle {
   private String vid;
   public Vehicle(String vNo) {
      vid = vNo;
   }
   public void getVid( ) {
      return vid;
   }
}
```

```
class Car extends Vehicle {
   private int noOfSeats;
   public Car( ) {
      super("unknow vid");
   }
   public void setSeat(int n){
      noOfSeats = n;
   }
   public int getSeat( ) {
      return noOfSeats;
   }
}
```


สังเกตุ

- Constructors ของ super class ไม่สืบทอดมายัง subclass
- แยกความแตกต่างจากการที่ compiler เพิ่ม default constructor ให้ในกรณีไม่กำหนด

```
student1  =  new  UndergradStudent ( ) ;
student2  =  new  GradStudent ( ) ;
```

- ถูกต้องเนื่องจากใช้ default constructor ที่เพิ่มโดย compiler ไม่ใช่เนื่องจาก inheritance

Abstract superclasses

- Abstract superclasses =
  - class ที่กำหนดโดยมี modifier abstract
  - ไม่สามารถสร้าง instance จากมันได้

```
abstract  class  student  {
   protected  static  int  Num_Tests  =  3 ;
      :
   abstract  public  void  computeGrade ( ) ;
   public  String  getGrade ( )  { return  courseGrade ;  }
      :
}
```

Abstract class

Abstract method

Abstract methods

- Abstract method =
  - method ที่มี keyword abstract เป็น modifier
  - ปิดท้ายด้วย ; = => ไม่มี method body
- Private และ Static methods ประกาศให้เป็น abstract methods ไม่ได้
- Class เป็น abstract class ถ้า
  - class มี method ที่เป็น abstract method หรือไม่ได้ทำ implementation ของ inherite abstract method

Overloading

- Overloading ใช้เมื่อต้องการใช้ method ชื่อเดียวกัน กับ data type ต่างชนิดกัน (I. e., + ใช้กับ String = การเชื่อมคำ ใช้กับ int = การบวกค่าตัวเลข)
- method สามารถถูกทำการ overloading ได้ เช่น System. out. println ( ) สามารถใช้พิมพ์ค่า int, long, String, etc)
- method ใน class ที่เรากำหนดเองก็สามารถ overload ได้ โดยการเขียน method ชื่อเดียวกันแต่มี argument ต่างกัน

Choosing Composition VS Inheritance

- Composition ใช้ในกรณีที่ต้องการเพิ่ม object ใน class ใหม่ แต่ไม่ต้องการแก้ไข interface ของ class นั้น
- Inheritance ใช้ในกรณีที่ต้องการลักษณะของ class นั้นแต่ต้องการเพิ่มลักษณะใหม่เพื่อทำให้เป็น class ใหม่ที่มีลักษณะพิเศษเพิ่มขึ้น
- **Protected**
  - อนุญาตสำหรับ class ลูกและ class อื่น ๆ ที่อยู่ใน package เดียวกัน

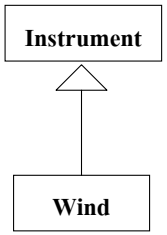
Upcasting

- ความสัมพันธ์ระหว่าง derived class กับ base class
  - derived class เป็นชนิดเดียวกันกับ base class

```
class Instrument {
    public void play ( ) { }
    static void tune (Instrument x) {
        x. play ( ) ;
    }
}
class Wind extends Instrument {
    public static void main (Sring [ ] args) {
        Wind flute = new Wind ( ) ;
        Instrument. Tune (flute) ;
    }
}
```

Upcasting

ทำไมต้อง “upcasting’?



Inheritance Diagram

- การ upcast เป็นการ casting ขึ้นตามลุกษณะของ inheritance diagram
- อาจกล่าวอีกอย่างในลักษณะของ super และ subset
  - base class นี้นเป็น superset ของ derived class หรือการ upcast จาก type ที่มีลักษณะเฉพาะไปยัง class ที่มีความทั่วไปมากกว่า

The “final” keyword

- final keyword ในจาวาหมายถึง สิ่งนั้นจะไม่มีการเปลี่ยนแปลง
  - เป็นส่วนหนึ่งของการออกแบบ หรือเพื่อต้องการเพิ่มประสิทธิภาย
  - สามารถใช้ได้กับ data, method และ class
- final data : ข้อมูลนั้นเป็นค่าคงที่
  - ค่าคงที่ในช่วง compile-time และค่าไม่มีการเปลี่ยนแปลง
  - ค่านั้นถูก initialized ในช่วง retime และค่าไม่มีการเปลี่ยนแปลงจากนั้น
  - static final = ค่าคงที่นั้นมีเพียงที่เดียวและไม่มีการเปลี่ยนแปลง

Final object

- final object : ตัวชี้ไปยัง object นั้นจะไม่มีการเปลี่ยนไปชี้ object อื่น
  - แต่ค่าภายใน object เองอาจถูกเปลี่ยนค่าได้

```
class Value { int x = 1; }
class FinalData {
  final int a[] = {1, 2, 3, 4};
  final Value v = new Value();
  public static void main (String args[]) {
        FinalData f = new FinalData();
        for (int i = 0; i < f, length; i++
              f. a [i]++; // object ไม่ได้เป็นค่าคงที่
        f. v = new Value (); // invalid
  }
}
```


Blank finals

- blank final : การประกาศ final โดยที่ยังไม่ได้มีการกำหนดค่าเริ่มต้น
  - ต้องทำการ initialize ก่อนที่จะมีการใช้
  - ทำให้แต่ละ object ที่อยู่ใน class เดียวกันสามารถมีค่าคงที่ภายในที่แตกต่างกันได้
  - ต้องทำการ initialize ค่าใน constructor

```
class BlankFinal {
    final int j; // blank final
    BlankFinal () {
          j = 10; // initialize blank final data
    }
}
```

Final argument

- Final argument : argument ไม่สามารถถูกเปลี่ยนไปชี้ที่อื่นได้

```
class Value  { }
class FinalArg  {
      void with (final Value v)  {
            v = new Value ( ) ;  // illegal  --v  is final
      }
      void without (Value v)  {
            v = new Value ( ) ;  //  ok --v is  not final
      }
]
```


Final methods

- final methods : เป็นการ lock ไม่ให้ method ถูกเปลี่ยนแปลงใน class ที่สืบทอดไป
  - ป้องกันการทำ overridden
  - เพิ่มประสิทธิภาพ ทำให้ compiler สามารถ call method นั้นได้ในลักษณะ inline (ลด overhead ต่าง ๆ ในการ call)
  - ไม่ควรที่จะกำหนด method ที่มีขนาดใหญ่มาก ๆ เป็น final เนื่องจากไม่ช่วยในแง่ประสิทธิภาพ
  - private method โดยนัยนั้นเป็น final แล้ว (ไม่สามารถ override) ไม่จำเป็นต้องกำหนดให้เป็น final อีก

Final classes

- final class : เป็นการกำหนดว่า class นี้ไม่มีการสืบทอดต่อไป
  - method ใน class จะโดยนัยว่าเป็น final ด้วยเพราะสืบทอดต่อไม่ได้

```
final class Dinosaur  { }
public class Jurassic extends Dinosaur  { // illegal
   extends
   :
}
```


Initialization and Class Loading

- โดยทั่วไป class แต่ละ class ในจาวาจะอยู่ต่าง file กัน file เหล่านี้จะไม่ถูก load จนกว่าจะต้องการใช้
  - ถูก load เมื่อมีการสร้าง object นั้น หรือ
  - ถูก load เมื่อ class code นั้นถูกเรียกใช้เป็นครั้งแรก (กรณีของ static)

การพิจารณา access specifier

- กฎเบื้องต้นที่อาจใช้ในการกำหนดว่าอะไรควรเป็น public หรือ private
  - Classes ควรเป็น public
  - Fields ควรเป็น private
  - constructors ควรเป็น public
  - methods getter และ setter ควรเป็น public
  - methods อื่นให้พิจารณาเป็นกรณีไป
- ไม่ใช่กฎตายตัว เลือกเปลี่ยนแปลงได้ตามความเหมาะสมเป็นกรณี

การวิเคราะห์และการออกแบบ (Analysis & Design)

- กฎพื้นฐาน : พยายามทำการค้นหา
  - ค้นหา class : ว่าอะไรคือ objects? ต้องทำการแบ่ง project ของเราอย่างไรให้เป็นส่วนย่อย (component parts)
  - **กำหนด behavior ของแต่ละ class** : ต้องมี method อะไรบ้างใน class
  - **กำหนดความสัมพันธ์ระหว่าง class** : อะไรบ้างที่เป็นส่วน interfaces ของ object เหล่านี้, messages อะไรบ้างที่ต้องการเพื่อที่จะทำให้สามารถติดต่อกันได้ระหว่าง objects

CRC cards : Class, Responsibility, Collaborator

- เป็นวิธีการอย่างง่าย ที่ช่วยในการออกแบบ class วิธีการ :
  - ทำการค้นหา class โดยมองหาคำนามใน problem description คำนามที่มีลักษระที่อธิบายได้จะเป็น candidate ของ class จากนั้นใช้ index card หนึ่งใบสำหรับหนึ่ง class
  - หาคำกริยาที่ช่วยอธิบายหน้าที่ที่ต้องมีใน project ที่กำลังออกแบบ จากนั้นให้พิจารณาว่า class ใดควรจะต้องรับผิดชอบในการทำงานนั้น บันทึก คำกริยานั้นเพื่อกำหนดเป็น method หนึ่งของ class นั้น
  - สำหรับในแต่ละหน้าที่ที่ถูกบันทึกไว้ ให้พิจารณาว่ามี class ใดที่เกี่ยวข้องกับหน้าที่นั้น class นั้นจะเป็น collaborator

ตัวอย่าง CRC Card

- Problem Statement : ต้องการพิมพ์ invoice ลูกค้า

INVOICE

XXX
202 Nanglinchi Rd., Chongnonsri,
Yannawa, Bangkok, 12120

Item	Qty	Price	Total
paper	3	120.00	360.00
staples	12	100.00	120.00
Amount Due:			480.00

ขั้นตอน

- **หาคำนาม**
  - Invoice
  - Customer
  - Item
  - Quantity
  - Price
  - Total
  - Amount Due
- **Candidate :**
  - Invoice ,
  - Customer ,
  - Item

•หาคำกริยา: จุดมุ่งหมายของโปรแกรมเพื่อพิมพ์ invoice

Invoice	
print	

CRC cards (ต่อ)

Customer	
print	

Invoice ไม่สามารถพิมพ์ชื่อลูกค้าและรายการได้

Customer class ควรรับผิดชอบในการพิมพ์ชื่อลูกค้า

Item class ควรรับผิดชอบในการพิมพ์รายการ

Item	
print	

CRC cards (ต่อ)

Invoice	
print	Customer, Item

Collaborators

CRC card

Invoice	
print	Customer, Item
add item	Item
compute amount due	Item

Item	
print	Customer, Item
get total price	