2110101 : การทำโปรแกรมคอมพิวเตอร์

Composition & Inheritance

ภาควิชาวิศวกรรมคอมพิวเตอร์
จุฬาลงกรณ์มหาวิทยาลัย

หัวข้อ

- Composition
- Inheritance
  - subclass และ superclass
  - Overriding vs. Overloading
  - class Object
  - Inheritance Hierarchy
  - upcasting และ downcasting
- Composition vs. Inheritance

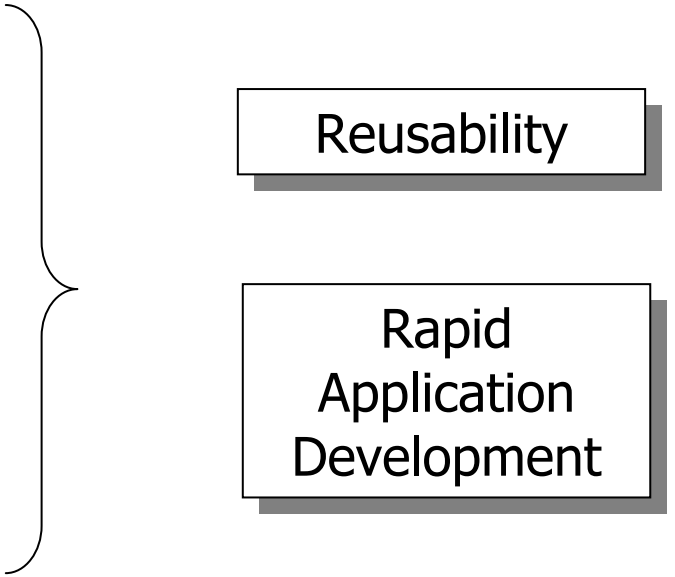
เมื่อต้องการสร้างคลาสใหม่

- คิดถึง
  - คลาสเก่าที่เคยเขียน
  - คลาสมาตรฐานของระบบจาวา
    - ค้นจาก j2sdk help file (มี ~ 3000 คลาส)
  - คลาสที่คนอื่นเขียนและอนุญาตให้ใช้
    - ค้นในอินเตอร์เน็ต (JNL, JSci, JLAPACK, JEdit, Lucene, ...)
  - คลาสที่ขายในท้องตลาด
- นำมาใช้ด้วยวิธี
  - copy & change source code (โบราณ)
  - composition
  - inheritance

avoid reinventing the wheel

เลือกของดี

- well defined
- carefully tested
- well documented
- portable
- highly rated
- open source



Composition

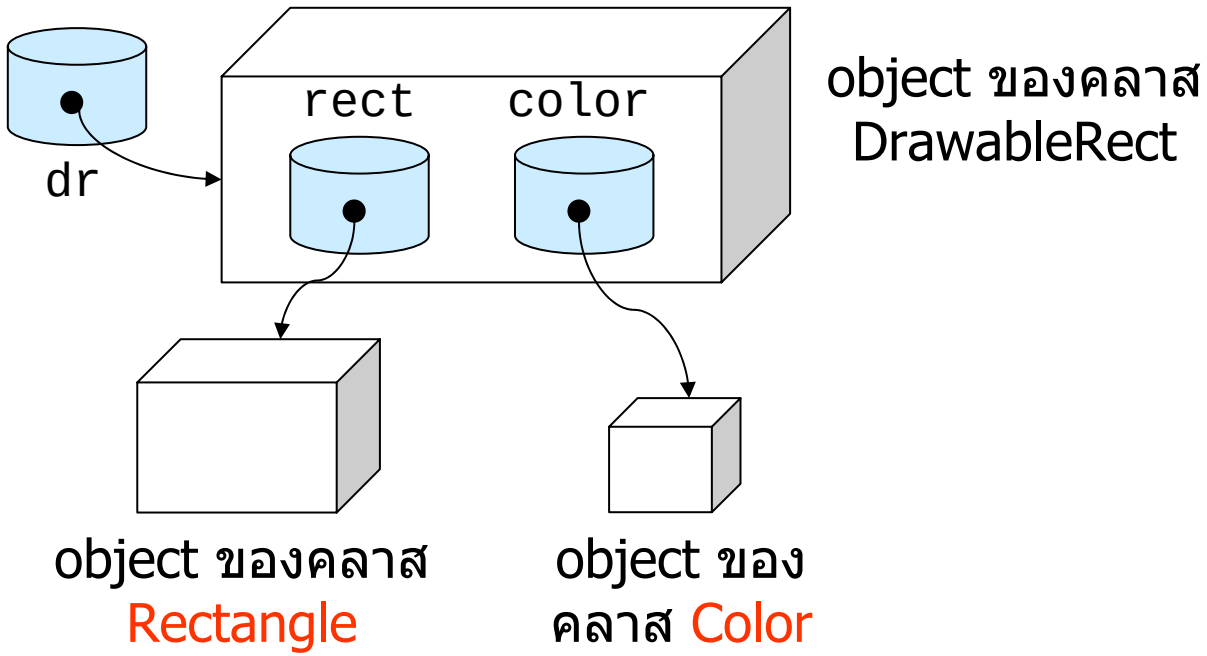
- สร้างคลาสใหม่ที่ประกอบด้วย fields ที่เป็น objects ของคลาสอื่น
- เขียน methods ของคลาสใหม่ โดยใช้ fields และ methods ขององค์ประกอบภายในให้เป็นประโยชน์
- รายละเอียดของ methods ใหม่เป็นเสมือน "กาว" ที่นำของที่มีอยู่มาปะติดกันให้ได้สิ่งประดิษฐ์ใหม่ๆ

อยากได้ "สี่เหลี่ยมที่วาดได้"

- อยากได้คลาส DrawableRect (มีสีด้วย) ซึ่งมีเมท็อด
  - equals(DrawableRect r) : ทดสอบว่า r เหมือน this ?
  - draw(Graphics g) : วาด this ?
- ~~เขียนใหม่หมด~~ (หาเรื่อง)
- ใช้คลาสมาตรฐานของจาวา Rectangle, Color
- Rectangle มีสมาชิก:
  - fields :
    - x, y, width, height
  - methods :
    - add, contains, equals, getWidth, getHeight, grow, intersection, intersects, move, toString, union, ...

Composition

- สร้างคลาสใหม่ประกอบด้วย fields
  - ออบเจกต์ของคลาส Rectangle
  - ออบเจกต์ของคลาส Color



Composition

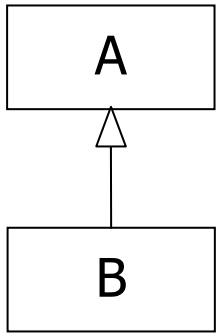
```
import java.awt.*;
public class DrawableRect {
  public Rectangle rect;
  public Color color;

  public DrawableRect(int x, int y, int w, int h, Color c) {
    this.rect = new Rectangle(x, y, w, h);
    this.color = c;
  }
  public boolean equals(DrawableRect r) {
    if (r == null) return false;
    return this.rect.equals(r.rect) &&
           this.color.equals(r.color);
  }
  public void draw(Graphics g) {
    g.setColor(this.color);
    g.fillRect(this.rect.x, this.rect.y,
               this.rect.width, this.rect.height);
  }
}
```

ประกอบด้วยออบเจกต์ของคลาส Rectangle และ Color

Inheritance

- เขียนคลาสใหม่ (B) ซึ่ง "ขยาย" คุณสมบัติจากคลาสเก่า (A) ที่มีอยู่ (class B extends A)
- เรียกคลาส B ว่า "รับมรดก" จากคลาส A
  - B มีทุกๆ fields ของ A (โดยอัตโนมัติ)
  - B มีทุกๆ methods ของ A (โดยอัตโนมัติ) (ยกเว้น constructors)
  - เรียก B ว่าเป็น subclass ของ A
  - เรียก A ว่าเป็น superclass ของ B
  - ตีความได้ว่า B เป็นคลาสที่เป็นกรณีพิเศษของ A
    - ออบเจกต์ของ B เป็น A
    - แต่ออบเจกต์ของ A ไม่จำเป็นต้องเป็น B
    - ตัวอย่างเช่น class Cat extends Mammal



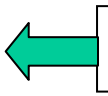
Inheritance : แบบง่ายสุดๆ

- เขียนคลาสใหม่ เพียงแค่บอกว่าขยายจากอีกคลาส

```
import java.awt.*;
public class R1 extends Rectangle {
}
```

- เหมือน Rectangle ทุกประการ (ยกเว้น constructors)
- R1 มี fields และ methods ต่างๆ ตามที่ Rectangle มี
- ไม่มี constructor อะไรเลย ระบบจะเติม constructor แบบไม่รับพารามิเตอร์ให้

```
import java.awt.*;
public class R1 extends Rectangle {
  public R1() {
    super();
  }
}
```



เรียก constructor ของ superclass

Inheritance : แบบที่ควรทำ

- กำหนดให้คลาสใหม่ขยายจากคลาสเก่าที่มี
- เพิ่ม fields ที่จำเป็น
- เขียน constructors ที่อยากให้บริการ
- เพิ่ม methods ใหม่ๆ ที่ superclass ไม่มี
- เปลี่ยน methods เดิมของ superclass ให้มีพฤติกรรมใหม่ (เรียกว่า overriding method)
- จาวาไม่อนุญาตให้ลบ methods ของ superclass ที่ subclass ไม่อยากได้

Inheritance

```
import java.awt.*;

public class DrawableRect extends Rectangle {
  public Color color;

  public DrawableRect(int x, int y, int w, int h, Color c) {
    super(x, y, w, h);
    this.color = c;
  }

  public boolean equals(Object r) {
    if ( !super.equals(r) ) return false;
    if ( this.getClass() != r.getClass() ) return false;
    return this.color.equals(((DrawableRect) r).color);
  }

  public void draw(Graphics g) {
    g.setColor(this.color);
    g.fillRect(super.x, super.y, super.width, super.height);
  }
}
```

เพิ่ม field ใหม่

เขียน constructor

เปลี่ยน method เดิม

เพิ่ม method ใหม่

การใช้ fields ของ superclass

- อ้างอิง fields ของ superclass ได้โดยใช้ super. นำหน้าชื่อ field
- หรือจะใช้ this. นำหน้าชื่อ field ของ superclass ก็ได้ (เพราะ "รับมรดก" มาแล้ว)
- หรือจะใช้แค่ชื่อ field ก็ไม่ผิด (เพราะตัด this. ได้)

```
public void draw(Graphics g) {
  g.setColor(this.color);
  g.fillRect(super.x, super.y, super.width, super.height);
}
```

เขียนแบบนี้
รู้ที่มา

```
public void draw(Graphics g) {
  g.setColor(this.color);
  g.fillRect(this.x, this.y, this.width, this.height);
}
```

```
public void draw(Graphics g) {
  g.setColor(this.color);
  g.fillRect(x, y, width, height);
}
```

การใช้ methods ของ superclass

- อ้างอิง methods เดิม ของ superclass ได้โดยใช้ super. นำหน้าชื่อ method
- หรือจะใช้ this. นำหน้าชื่อ method (ที่ไม่ได้ถูก overriden) ของ superclass ก็ได้ (เพราะ "รับมรดก" มาแล้ว)

```
public boolean equals(Object r) {
  if ( !super.equals(r) ) return false;
  if ( this.getClass() != r.getClass() ) return false;
  return this.color.equals(((DrawableRect) r).color);
}
```

เรียก equals ของ superclass เพื่อทดสอบว่าเป็น Rectangle ที่เหมือนกันหรือไม่ (ตรงนี้จะใช้ this.equals ไม่ได้ เพราะ.....)

แบบฝึกหัด

- เขียนเมท็อด toString ให้กับ DrawableRect
- ข้อแนะนำ : ควรใช้ toString ของ superclass

```
public class DrawableRect extends Rectangle {
  ...
  public String toString() {









  }
}
```

Overriding ไม่เหมือน Overloading

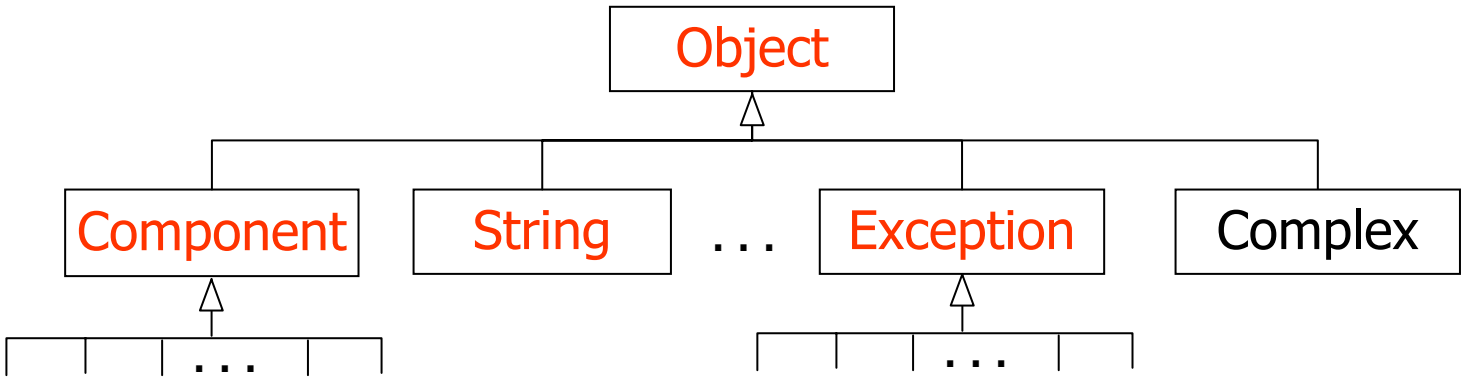
- Overriding เป็นการเขียนเมท็อดใน subclass ซึ่งมีชื่อและรายการพารามิเตอร์ปรากฏอยู่แล้วใน superclass (เป็นการ "ลบล้าง" ของเก่าที่มีอยู่)
- Overloading เป็นการเขียนเมท็อดใหม่ ซึ่งมีชื่อซ้ำ แต่รายการพารามิเตอร์ไม่เหมือนกับของเดิม (เป็นการ "เพิ่มภาระ" ให้กับชื่อเมท็อดว่าทำงานหลายแบบ ขึ้นกับรายการพารามิเตอร์ที่ได้รับ)

```
public class A {
  ...
  public int f() {
    ...
  }
}
```

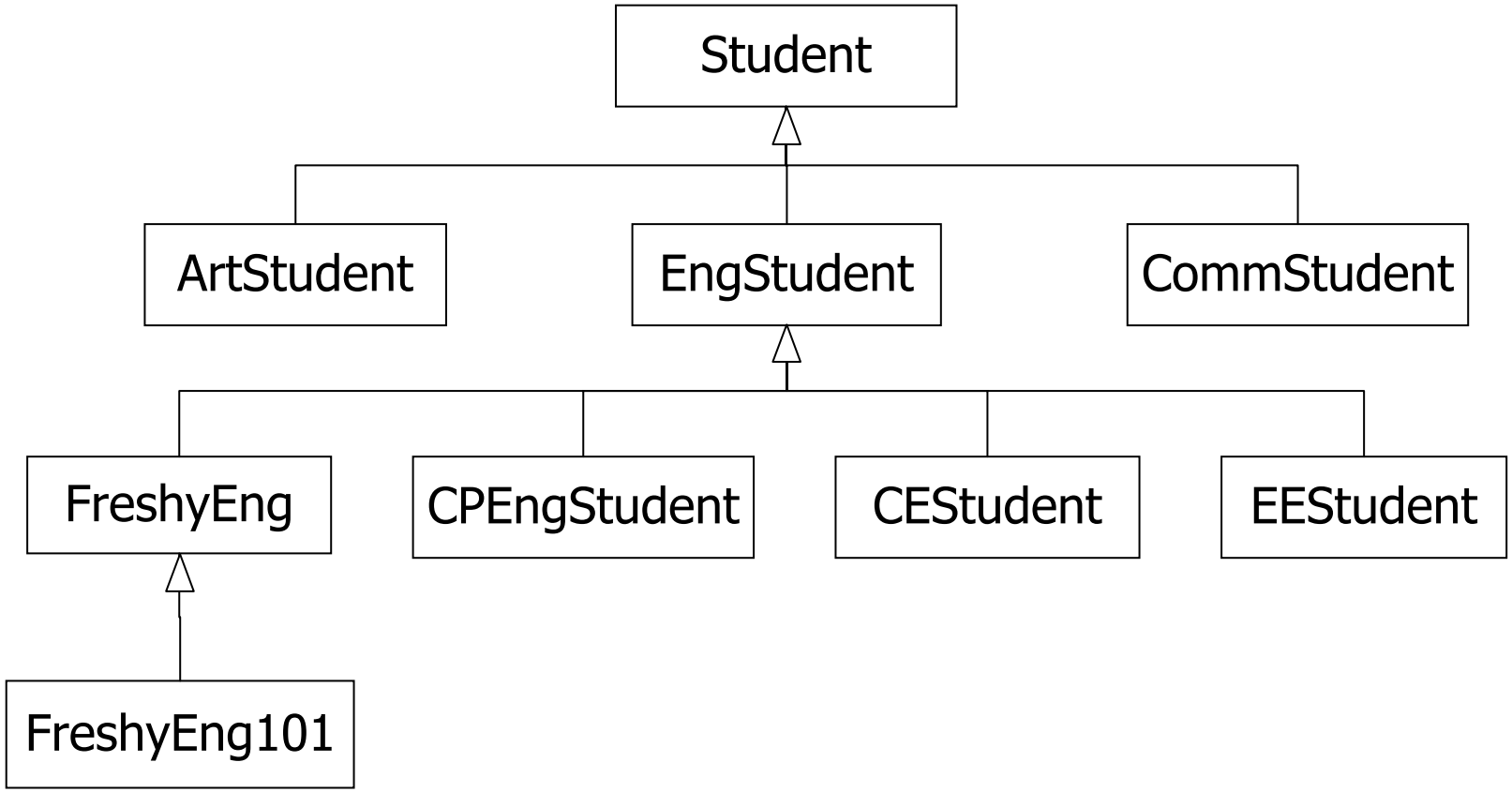
```
public class B extends A {
  public int f() {          // overriding
    ...
  }
  public int f(int x) {  // overloading
   ...
  }
```

คลาส Object

- จาวามีคลาสมาตรฐานชื่อว่า Object (ขอย้ำ ชื่อ Object นี้เป็นชื่อคลาส ขึ้นต้นด้วยโอใหญ่)
- คลาส Object เป็น "บรรพบุรุษ" ของทุกคลาสในจาวา
- การเขียนคลาสใหม่ที่ไม่ได้ extends จากคลาสใด ถือได้ว่า extends จากคลาส Object



Inheritance Hierarchy



การแสดง Inheritance ใน Help File

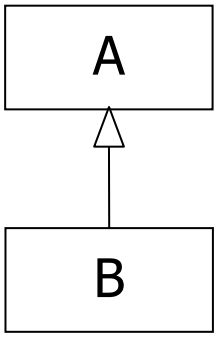
```
java.lang.Object
  |
  +-JSci.maths.Matrix
      |
      +-JSci.maths.ComplexMatrix
          |
          +-JSci.maths.ComplexSquareMatrix
              |
              +-JSci.maths.ComplexTridiagonalMatrix
                  |
                  +-JSci.maths.ComplexDiagonalMatrix
```

แบบฝึกหัด

- เขียน inheritance hierarchy ของคลาสต่อไปนี้
  - Quadrilateral, Trapesoid, Parallelogram, Rectangle, และ square
  - Vehicle, Car, Truck, Sedan, Coupe, PrickupTruck, SportUtilityVehicle, Minivan, Bicycle, และ Motocycle
  - Archosaurs, Thecodonts, Pterosaurs, Dinosaurs, Crocodilians, Saurischians, และ Ornithischians

Casting

- กำหนดให้ B extends A
  - B รับพฤติกรรมทุกอย่างจาก A
  - B "เป็น" A (คือเป็นกรณีพิเศษของ A)



- Upcasting
  - เราสามารถ cast ออบเจกต์ของ B ให้เป็น A ได้
  - upcast ได้เสมอ
- Downcasting
  - ออบเจกต์ของ B ที่เคยถูก upcast ให้เป็น A สามารถถูก cast กลับลงมาเป็น B ได้
  - ถ้า downcast ไม่ได้จะเกิด ClassCastException

มาดู equals ของ DrawableRect

- equals รับออบเจกต์ของคลาส Object ดังนั้น
  - equals นี้ override equals ของ Rectangle
- ใช้ super.equals(r)
  - เพื่อทดสอบว่าเป็น Rectangle ที่เหมือนกันหรือไม่
- ใช้ this.getClass() กับ r.getClass()
  - เพื่อเปรียบเทียบว่า r เป็นคลาสประเภทเดียวกับ this หรือไม่
  - ถ้าเป็นคลาสประเภทเดียวกันจะได้ downcast จาก Object มาได้ เพื่อจะได้หยิบ color : ((DrawableRect) r).color

```
public boolean equals(Object r) {
  if ( !super.equals(r) ) return false;
  if ( this.getClass() != r.getClass() ) return false;
  return this.color.equals(((DrawableRect) r).color);
}
```

แบบฝึกหัด

- เขียนคลาส Square โดยใช้คลาส Rectangle แบบ
  - Composition
  - Inheritance
- Fields :
  - x, y ระบุพิกัดตรงกลางสี่เหลี่ยมจัตุรัส
  - length ระบุความยาวด้าน
- Methods :
  - `boolean contains( int x, int y )`
  - `void translate( int dx, int dy )`
  - `int getLength( )`
  - `int setLength( )`
  - `boolean equals( Object s )`
  - `String toString( )`

อ่านความหมายของแต่ละเมท็อดได้จาก JDK help file ของ Rectangle

Composition vs. Inheritance

- ใช้ composition เมื่อเราใช้คุณสมบัติของคลาสอื่นมาเป็นส่วนประกอบเพื่อออกแบบของใหม่
  - เช่น รถ "มี" ล้อ เครื่องยนต์ หน้าต่าง ประตู
  - ดังนั้นคลาส Car ประกอบด้วย fields ของออบเจกต์จากคลาส Wheel, Engine, Window, Door เป็นต้น
- ใช้ inheritance เมื่อเราต้องการคลาสใหม่ซึ่งเป็นกรณีพิเศษของคลาสเดิม จึงต้องการคุณสมบัติเดิมส่วนใหญ่ของคลาสเก่า มีการเปลี่ยนหรือเพิ่มเติมคุณสมบัติใหม่บ้าง
  - เช่น นิสิตวิศวฯ จุฬาฯ "เป็น" นิสิตจุฬาฯ แต่มีอะไรบางอย่างเพิ่มเติมจากกรณีทั่วไป
  - ดังนั้นคลาส EngCUStudent extends CUStudent