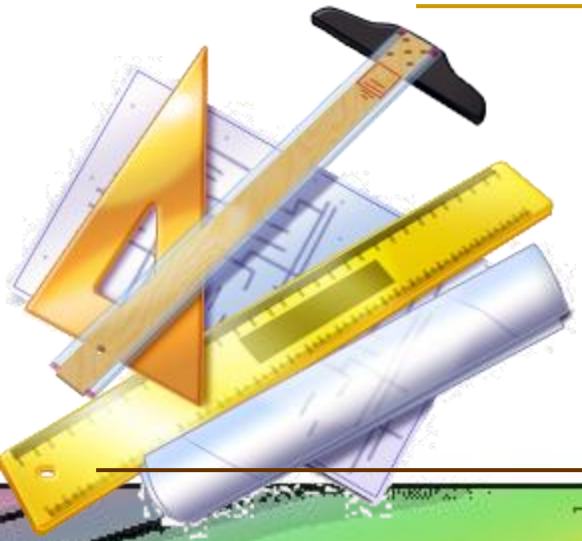
Interfaces และ Polymorphism



Lecture 8

Yaowadee Temtanapat

เยาวดี เต็มธนาภัทร์

วัตถุประสงค์ของการเรียนวันนี้

- เพื่อเรียนรู้เกี่ยวกับอินเทอร์เฟส (interfaces)
- สามารถที่จะแปลงระหว่าง supertype และ subtype references
- เข้าใจแนวคิดเกี่ยวกับ polymorphism
- เข้าใจการใช้ interfaces เพื่อแยก coupling ระหว่าง classes
- implement event listeners สำหรับเหตุการณ์เกี่ยวกับ timer

ดัดแปลง MovingGame1 สำหรับ Person

```
public class MovingGame1 {
   private Person object;

   // code for setting an object
   public void setObject(Person x) {
      object = x;
   }

   public void setPosition(int x, int y){
        object.setPositionX(x);
        object.setPositionY(y);
   }
   public Person getObject() {
      return object;
   }
}
```

ดัดแปลง MovingGame2สำหรับ Monster

```
public class MovingGame2 {
   private Monster object;

   // code for setting an object
   public void setObject(Monster x) {
      object = x;
   }

   public void setPosition(int x, int y){
        object.setPositionX(x);
        object.setPositionY(y);
   }
   public Monster getObject() {
      return object;
   }
}
```

อินเทอร์เฟส: Movable Interface

- เพื่อให้ MovingGame ทำงานได้กับทั้ง Person และ Monster
- สมมติให้คลาสเหล่านี้ตกลงที่จะมีเมท็อด setPositionX และ setPositionY แบบเดียวกัน:

```
object.setPositionX(x);
object.setPositionY(y);
```

  - คลาสจำเป็นต้องเป็น*ชนิด/ประเภทเดียวกัน*และมีเมท็อดแบบเดียวกัน
- **อินเทอร์เฟส** ช่วยกำหนดการเป็นชนิดเดียวกันและมีเมท็อดแบบเดียวกัน:

```
public interface Movable {
    void setPositionX(int x);
    void setPositionY(int y);
}
```

ข้อแตกต่างระหว่าง Interfaces กับ Classes

- ทุกเมท็อดในอินเทอร์เฟสเป็น abstract
  - ไม่มีส่วน body ของเมท็อด
- ทุกเมท็อดในอินเทอร์เฟสเป็น public โดยปริยาย
- อินเทอร์เฟสไม่มีตัวแปรวัตถุ (instance variables)

Generic MovingGame สำหรับ Movable Objects

```
public class MovingGame{
   private Movable object;

   // set movable object
   public void setObject(Movable x) {
      object = x;
   }

   public void setPosition(int x, int y){
        object.setPositionX(x);
        object.setPositionY(y);
   }
   public Movable getObject() {
      return object;
   }
}
```

การทำให้อินเทอร์เฟสเป็นจริง (Realizing an Interface)

- ข้อกำหนดในภาษาจาวา ในการกำหนดให้**คลาสเป็นชนิด**ของอินเทอร์เฟส
  - ประกาศคลาส พร้อมประโยค **implements** ที่ตามด้วยชื่ออินเทอร์เฟส
  - คลาสกำหนดการทำงานของ**ทุก**เมท็อดของอินเทอร์เฟส (มีส่วน body)

```
class ClassName implements Movable {
    public void setPositionX(int x) {
        implementation
    }
    public void setPositionY(int y) {
        implementation
    }
    additional methods and fields
}
```

  - คลาสต้องกำหนดเมท็อดของอินเทอร์เฟสเป็น **public**

ประกาศคลาส Person และ Monster ที่เป็น Measurable

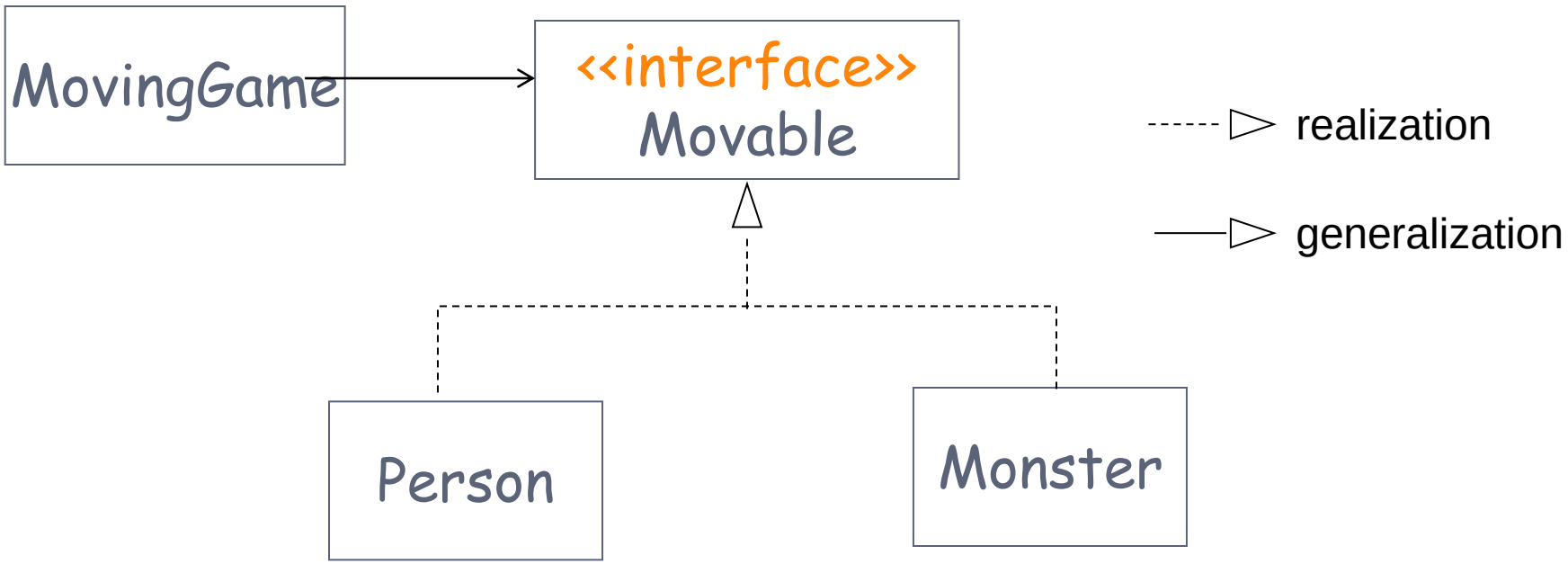
```
public class Person implements Movable {
    public void setPositionX(int x){
        positionX = x;
    }
     public void setPositionY(int y){
        positionY = y;
    }
        additional methods and fields
}
public class Monster implements Movable {
    public void setPositionX(int x){
        positionX = x;
    }
     public void setPositionY(int y){
        positionY = y;
    }
        additional methods and fields
}
```

File MovingGameTest.java (1)

```
1 /**   This program tests the DataSet class. */
2 public class MovingGameTest {
3   public static void main(String[] args) {
4     MovingGame game = new MovingGame();
5     if(Math.random() > 0.5)
6       game.setObject(new Monster());
7     else
8       game.setObject(new Person());
9
10    game.setPosition(50,100);
11
12
13    Movable obj = game.getObject();
17    System.out.println(obj);
15  }
16 }
```

Class Diagram ของ MovingGame และคลาสที่เกี่ยวข้อง

- สังเกตว่า MovingGame นั้นถูกแยก (*decoupled*) จาก Person , Monster



Syntax 9.1: การประกาศอินเทอร์เฟส (Interface)

- **Syntax**: การประกาศ Interface

```
public interface InterfaceName {
    method signatures
}
```

- ตัวอย่างเช่น:

```
public interface Movable {
     void setPositionX();
     void setPositionY();
}
```

- **จุดมุ่งหมาย:**
  เพื่อกำหนด interface และ method signatures โดยเมท็อดเหล่านี้เป็น *public โดยปริยาย*

Syntax 9.2: การ Implement โดยใช้ Interface

- **Syntax**: การประกาศ class ที่เป็นประเภท Interface

```
public class ClassName
             implements InterfaceName, InterfaceName, ...
{
    methods
    instance variables
}
```

- ตัวอย่างเช่น:

```
public class Monster implements Movable {
    // other Monster methods
    public void setPositionX() {
        // method implementation
    }
     public void setPositionY() {
        // method implementation
    }
}
```

- **จุดมุ่งหมาย:** ประกาศคลาสใหม่ที่มีการ implement methods ของ interface

การแปลงชนิด (Types)

- สามารถแปลงจาก class type ไปเป็นชนิดของ interface type ได้:

```
Person player = new Person();
Movable x = player; // OK
```

- ตัวแปรอ้างถึงชนิดอินเทอร์เฟส Measurable สามารถใช้อ้างถึง Monster ได้

```
x = new Monster(); // OK
```

- ไม่สามารถแปลงข้ามไปเป็นชนิดที่ไม่สัมพันธ์กันได้

```
x = new Random(); // ERROR
```

การทำ Casting

- การเพิ่มวัตถุ Person ใน MovingGame

```
MovingGame game = new MovingGame();
game.setObject(new Person());
```

- หาค่า สถานะของวัตถุในคลาส Person โดยใช้ getStatus method:

```
Movable obj = game.getObject();
```

- เพราะ obj ไม่ใช่ชนิดของ Person ดังนั้น

```
String name = obj.getStatus(); // ERROR
```

- เรารู้ว่า มันเป็นชนิดของ Person แต่คอมไพเลอร์ไม่รู้ จึงต้องทำ casting:

```
Person player = (Person) obj;
String name = player.getStatus();
```

- หาก cast ผิดชนิด (เช่น obj ไม่ใช่ Person) จะเกิดความผิดพลาดในช่วง Runtime

การตรวจสอบโดย instanceof Operator

- เพื่อให้การ cast ทำได้อย่างปลอดภัย ใช้ instanceof ตรวจสอบก่อน

```
if (obj instanceof Person){
   player = (Person)obj;
   . . .
}
```

Syntax 9.3: instanceof Operator

- **Syntax** ของ instanceof

  object **instanceof** *ClassName*

- ตัวอย่างเช่น:

```
if (obj instanceof Person) {
    player = (Person)obj;
}
```

- **จุดมุ่งหมาย:**
  - คืนค่า true ถ้าวัตถุที่ทดสอบเป็น instance ของ *ClassName* (หรือว่าหนึ่งใน subclasses ของ ClassName นั้น), ถ้าไม่เช่นนั้นเป็น false

ตัวแปรที่เป็นชนิดอินเทอร์เฟส

- ตัวแปร Interface สามารถเป็นที่เก็บตัวอ้างถึงวัตถุของคลาสใด ๆ ที่เป็นชนิด interface นั้น ๆ ได้

```
Movable obj;
obj = new Person();
obj = new Monster();
```

- แต่เราไม่สามารถที่จะสร้างวัตถุจาก interface

```
obj = new Movable(); // ERROR
```

- สามารถเรียก interface methods จากวัตถุใดๆ ที่ implement interface ได้:

```
obj.setPositionX(x);
```

- ปัญหา method ของวัตถุใดที่ถูกเรียก?

Polymorphism

- **Polymorphism** (Greek: many shapes): *ความสามารถชื่อเดียวกัน แต่ให้ผลการทำงานที่ต่างกัน โดยขึ้นกับชนิดที่แท้จริงของวัตถุ*

- ขึ้นอยู่กับว่า วัตถุที่แท้จริงที่ถูกอ้างถึงแท้จริงเป็นวัตถุใด
  - ถ้า obj อ้างถึง Person, เรียก Person.setPositionX
  - ถ้า obj อ้างถึง Monster, เรียก Monster.setPositionX

ตัวอย่างการประกาศอินเทอร์เฟส

- อินเทอร์เฟสสำหรับวัตถุที่พิมพ์รายงานได้
  - เมท็อด report

```
public interface Reportable {
   void report();
}
```

คลาสที่เป็นชนิด Reportable: Item

```
public class Item implements Reportable {
   private String name;
   private double weight;

    // ... omit other methods
   public void report() {
        System.out.println("Item name " + name +
                           " with weight " + weight);
   }
}
```

คลาสที่เป็นชนิด Reportable: Person

```
public class Person implements Movable, Reportable {
   private int treasure;
   private int positionX = 0, positionY = 0;

    // ... omit other methods
   public void report() {
     System.out.println("The number of treasure: " +
   treasure);
   }
   public void setPositionX (int x) {
       positionX = x;
    }
   public void setPositionY (int y) {
       positionY = y;
    }
}
```

คลาส Report (คลาสทดสอบ)

```
import java.util.ArrayList;
public class Report {
    public static void main(String[] args) {
          ArrayList<Reportable> list = new ArrayList<Reportable>();
          list.add(new Person());
          list.add(new Item("Gold", 10));
          list.add(new Item("Silver", 15.0));
          list.add(new Person());

          System.out.println("--Report--");
          for (Reportable r : list) {
            r.report();

            if (r instanceof Person) {
                  Person p = (Person)r;
                  p.move((int)(Math.random()*100), (int)(Math.random()*100));
                  System.out.println("New state after move: " + p.getStatus());
            }
          }
    }
}
```

การทำงานกับ Timer (1/2)

- javax.swing.Timer สร้าง timer events และส่ง events ไปยัง action listener

```
public interface ActionListener {
     void actionPerformed(ActionEvent event);
}
```

- การทำให้เป็นชนิด interface

```
class MyListener implements ActionListener {
     public void actionPerformed(ActionEvent event) {
      // this action will be executed at each timer event
      // place listener action here
   }
}
```

การทำงานกับ Timer (2/2)

- เพิ่ม listener ให้กับ timer

```
MyListener listener = new MyListener();
Timer t = new Timer(interval, listener);
t.start();
```

- โปรแกรมจับเวลาถอยหลัง โดยพิมพ์ค่า

```
10
9
. . .
2
1
Happy New Year!
```

- ชะลอประมาณ 1 วินาทีในแต่ละบรรทัดการพิมพ์

File CountDown.java

```
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class CountDown implements ActionListener {
    private int count;

    public CountDown(int initialCount) {
          count = initialCount;
    }

    public void actionPerformed(ActionEvent event) {
          if (count > 0) System.out.println(count);
          if (count == 0) System.out.println("Liftoff!");
          count--;
    }
}
```

File TimerTest.java

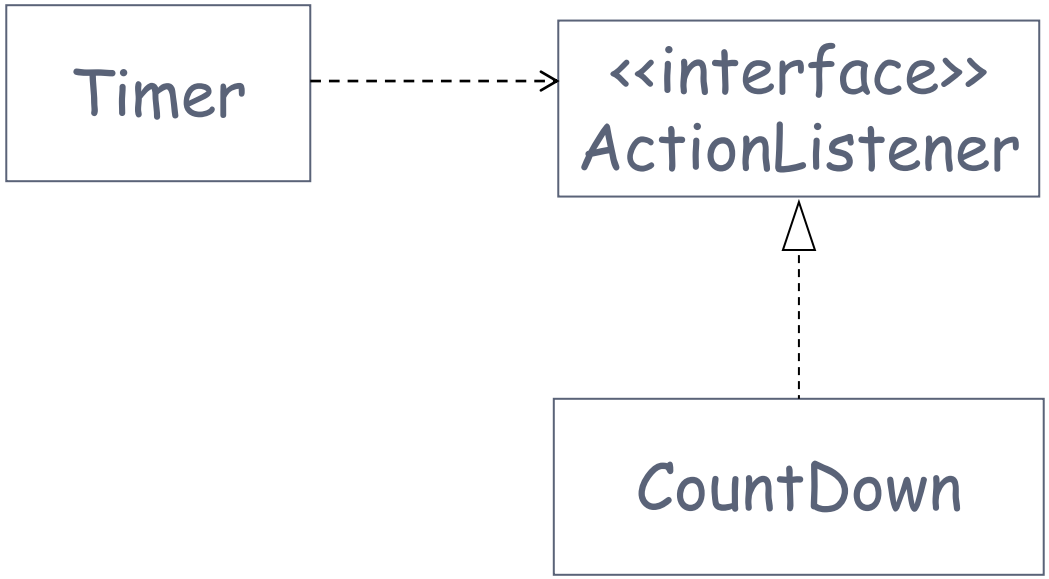
```
import javax.swing.JOptionPane;
import javax.swing.Timer;

/**
   This program tests the Timer class.
*/
public class TimerTest {
   public static void main(String[] args) {
         CountDown listener = new CountDown(10);

         final int DELAY = 1000; // milliseconds between timer ticks
         Timer t = new Timer(DELAY, listener);
         t.start();

         JOptionPane.showMessageDialog(null, "Quit?");
         System.exit(0);
   }
}
```

Class Diagram ของการใช้คลาส CountDown กับ Timer



สรุปการเรียนวันนี้

- เรียนรู้เกี่ยวกับอินเทอร์เฟส (interfaces)
- สามารถแปลงระหว่าง supertype และ subtype references
- เข้าใจแนวคิดเกี่ยวกับ polymorphism
- เข้าใจการใช้ interfaces เพื่อแยก coupling ระหว่าง classes
- implement event listeners สำหรับเหตุการณ์เกี่ยวกับ timer