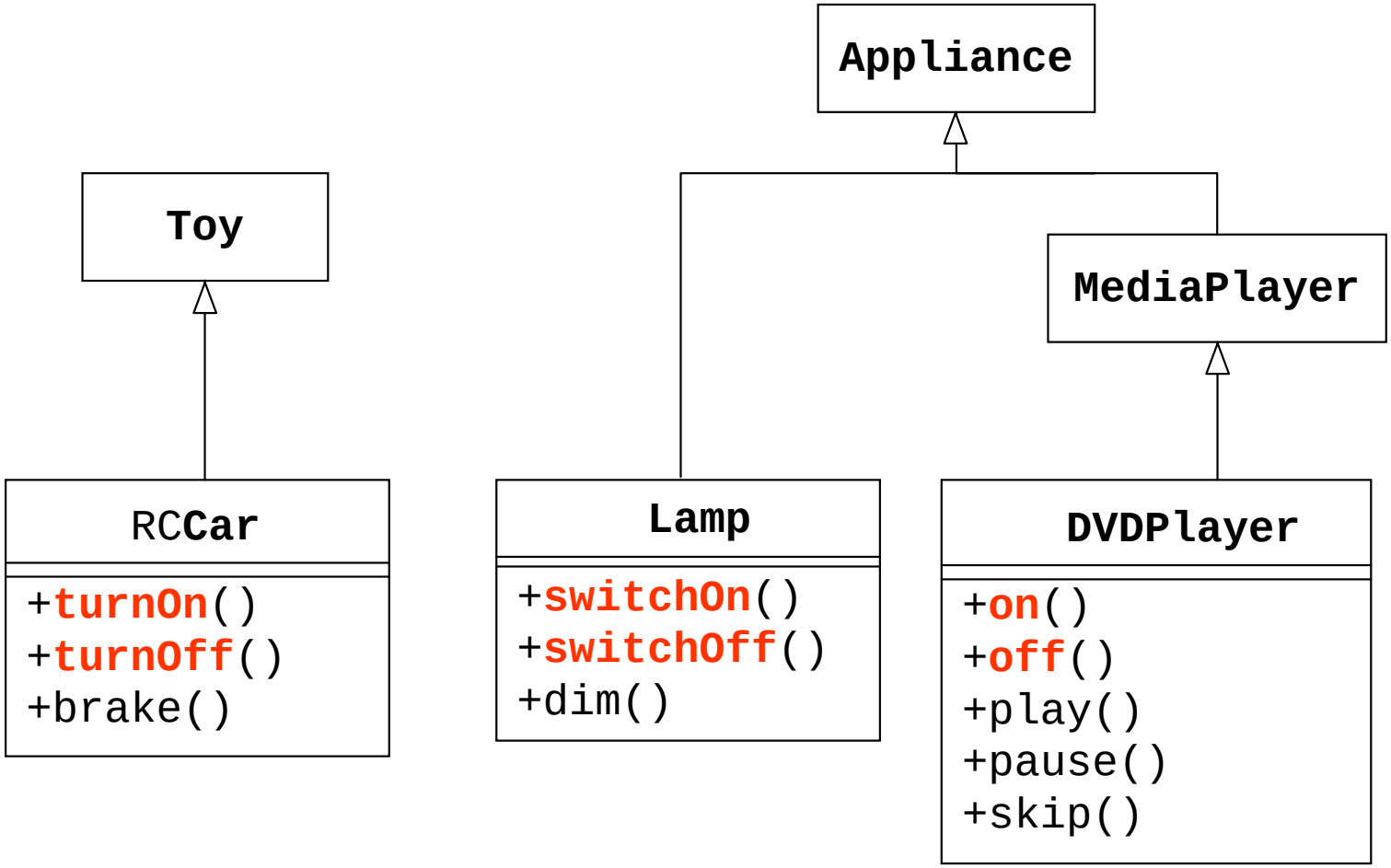
จาวา : Interfaces & Abstract Classes

สมชาย ประสิทธิ์จูตระกูล

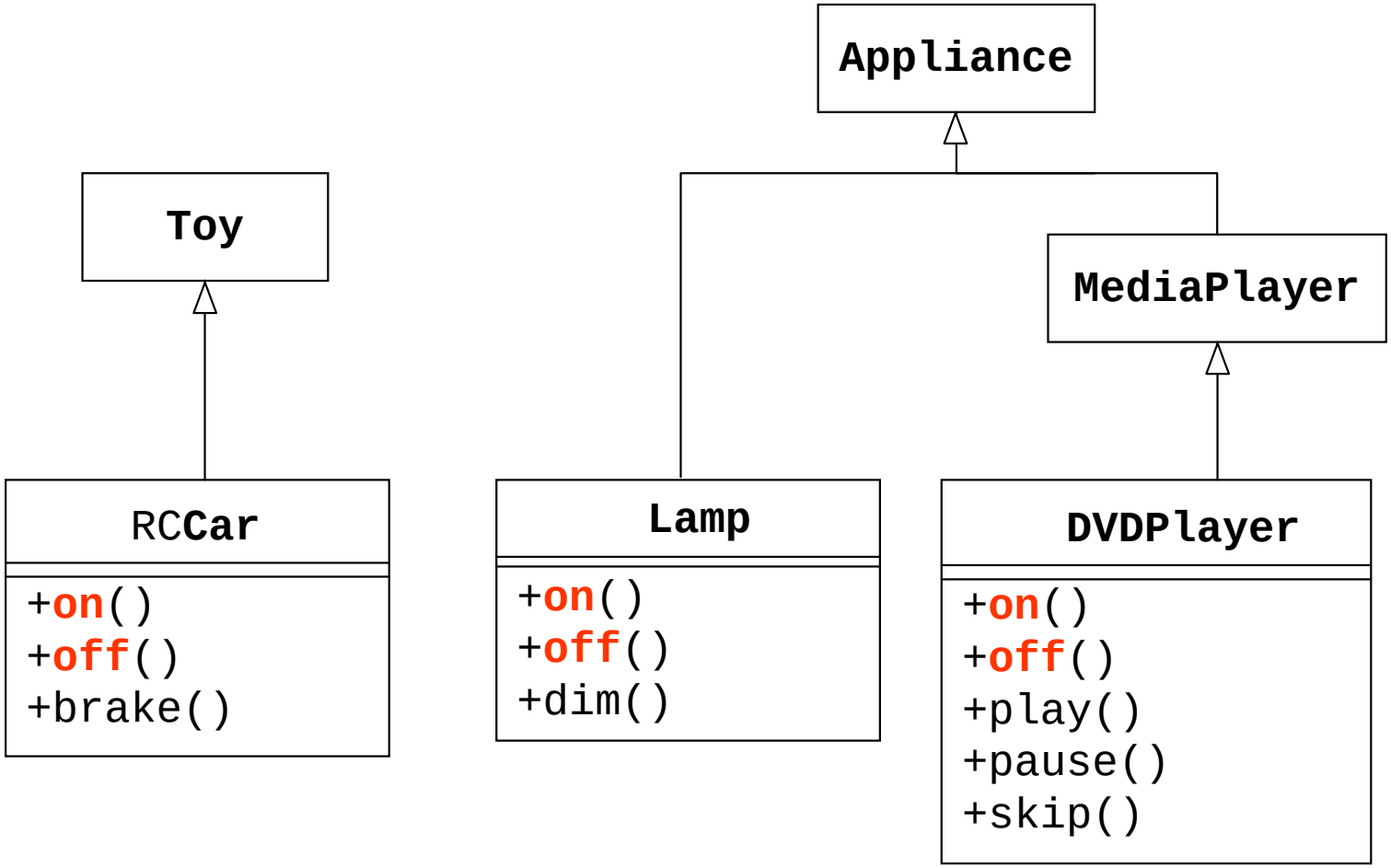
A Class Hierarchy



Generic Objects

```
class MonitoringSystem extends Appliance {
  public void reset( Object[] eq ) {
    for (int i = 0; i < eq.length; i++) {
      if (eq[i] instanceof Lamp) {
        Lamp a = (Lamp) eq[i];
        a.switchOff();
        a.switchOn();
      }
      if (eq[i] instanceof DVDPlayer) {
        DVDPlayer s = (DVDPlayer) eq[i];
        s.turnOff();
        s.turnOn();
      }
    }
  }
  ...
```

ตั้งชื่อให้เหมือน ๆ กันทำให้ใช้งานง่าย

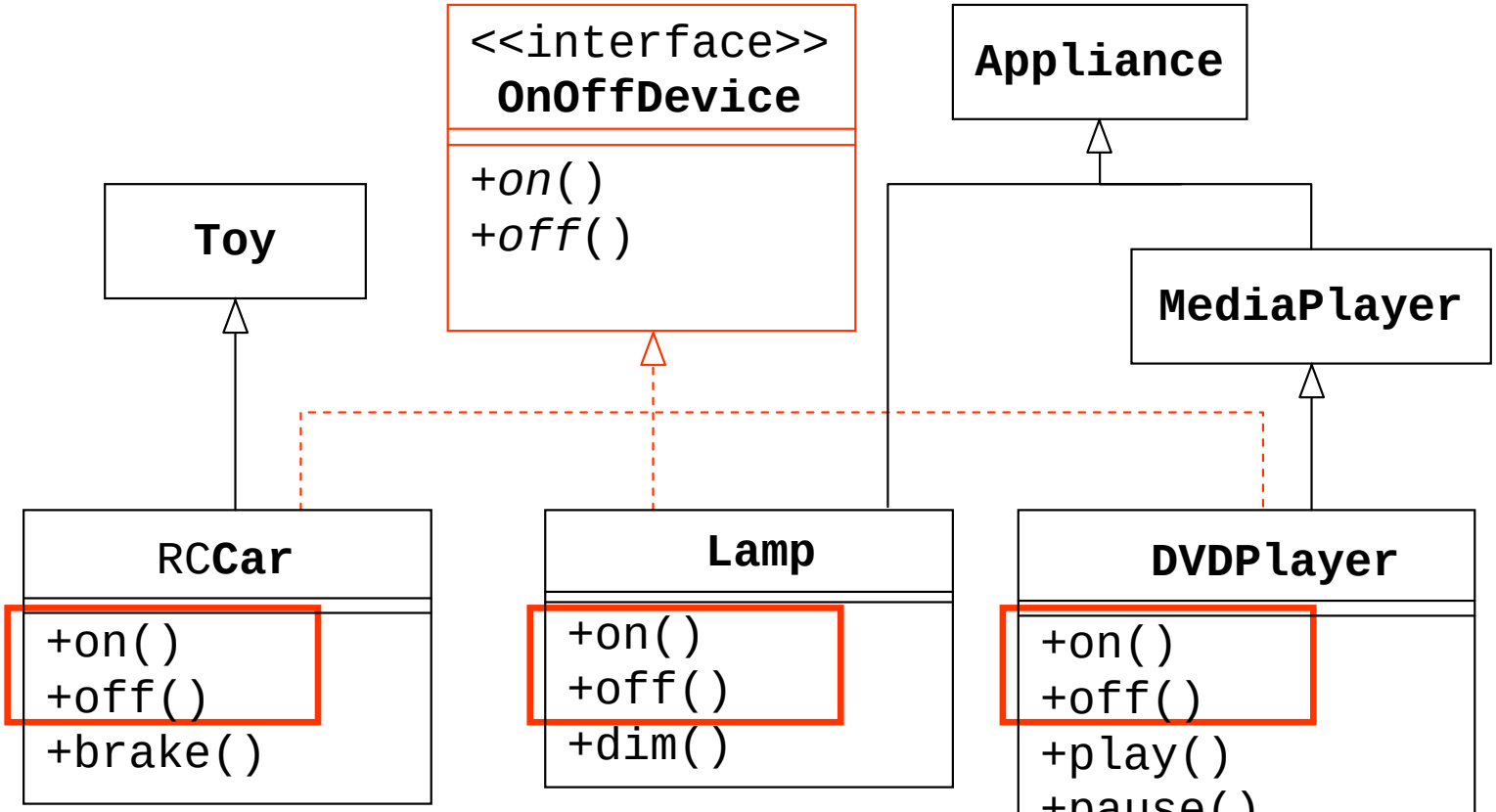


Generic Objects

```
class MonitoringSystem extends Appliance {
  public void reset( Object[] eq ) {
    for (int i = 0; i < eq.length; i++) {
      if (eq[i] instanceof Lamp) {
        Lamp a = (Lamp) eq[i];
        a.off();
        a.on();
      }
      if (eq[i] instanceof DVDPlayer) {
        DVDPlayer s = (DVDPlayer) eq[i];
        s.off();
        s.on();
      }
    }
  }
  ...
```

```
class MonitoringSystem extends Appliance{
  public void reset(Object[] eq) {
    for (int i = 0; i < eq.length; i++) {
      eq[i].off(); // Object don't have
      eq[i].on();  // on and off.
    }
  }
  ...
```

Interface




```
DVDPlayer dvd = new DVDPlayer();
MediaPlayer mp = dvd;
Appliance ap = dvd;
OnOffDevice ood1 = dvd;
OnOffDevice ood2 = new RCCar();
```

Interfaces

abstract methods (object methods)

```
interface OnOffDevice {
  public abstract void on();
  public abstract void off();
}
```

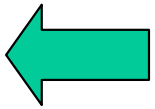
สัญญาว่าจะมี methods ต่างๆ ที่ปรากฏใน OnOffDevice ถ้าไม่ครบ compile ไม่ผ่าน

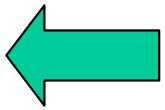
```
class RCCar extends Toy implements OnOffDevice {
  public void on() {...}
  public void off() {...}
  public void brake() {...}
}
```

```
class Lamp extends Appliance implements OnOffDevice {
  public void on() {...}
  public void off() {...}
  public void dim() {...}
}
```

OnOffDevice : เรียก on( ) และ off( ) ได้

```
class MonitoringSystem extends Appliance {
  ...
  public void reset( OnOffDevice[] eq) {
    for (int i = 0; i < eq.length; i++) {
      eq[i].off();
      eq[i].on();
    }
  }
  public void shutdown( OnOffDevice[] eq) {
    for (int i = 0; i < eq.length; i++) {
      eq[i].off();
    }
  }
}
```





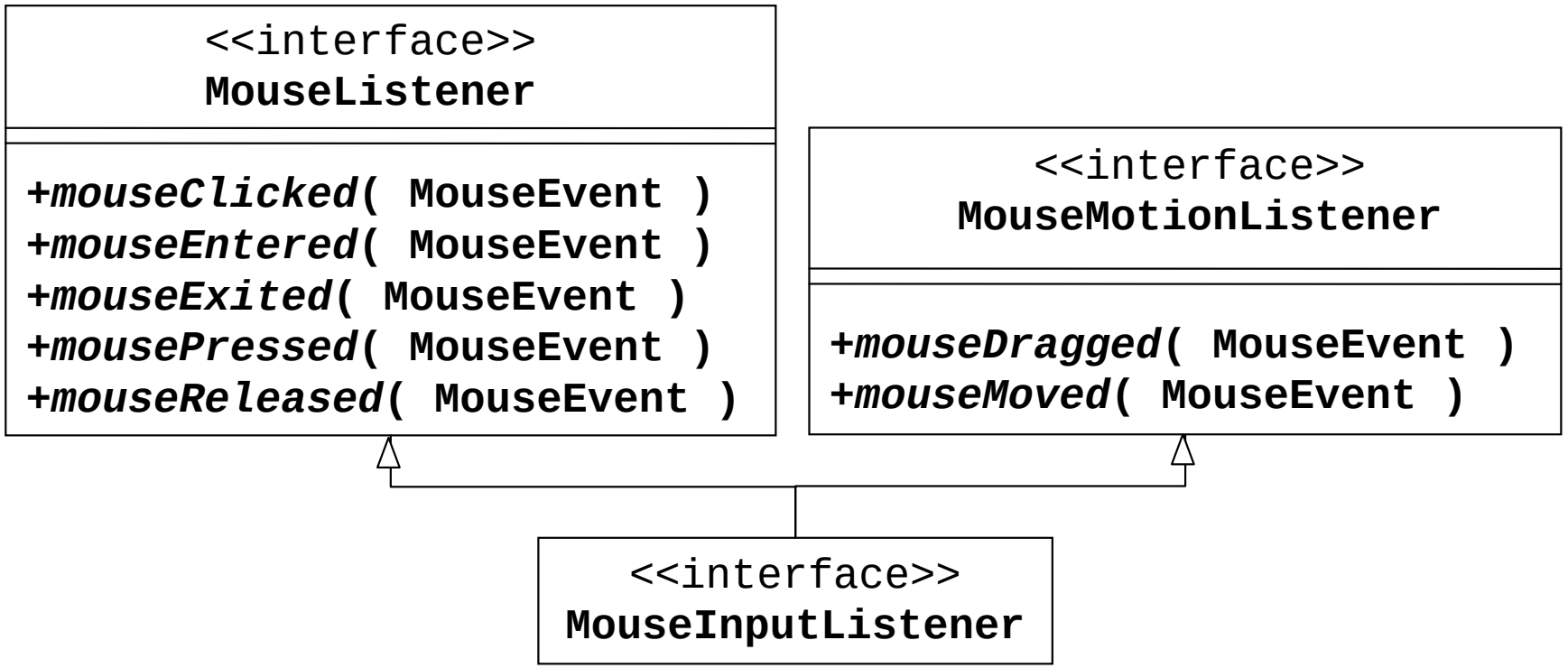
Interfaces

- ใช้ interface เป็นตัวนิยามประเภทข้อมูลใหม่
- interface เป็นเสมือน "สัญญา" ระหว่างผู้ใช้คลาสกับผู้พัฒนาคลาสว่าคลาสที่ถูกพัฒนาที่จะมี methods ทั้งหมดที่ปรากฏใน interface ที่กำหนด
- methods ใน interface
  - บอกแค่ว่าเรียกใช้อย่างไร (มีแค่หัวเมท็อด : abstract method)
  - ไม่บอกว่าทำอะไร (ใช้ javadoc comment บอกแทน)
  - ไม่บอกว่าทำงานอย่างไร (คลาสที่ implements บอกว่าทำอย่างไร)
- คลาสเชื้อสายต่างกัน implement interface ตัวเดียวกันได้
- หนึ่งคลาส implements หลาย interface ที่ไม่เกี่ยวกันก็ได้

```
public class ArrayList extends AbstractList implements
             Collection, List, Iterable, RandomAccess,
             Cloneable, Serializable { ... }
```

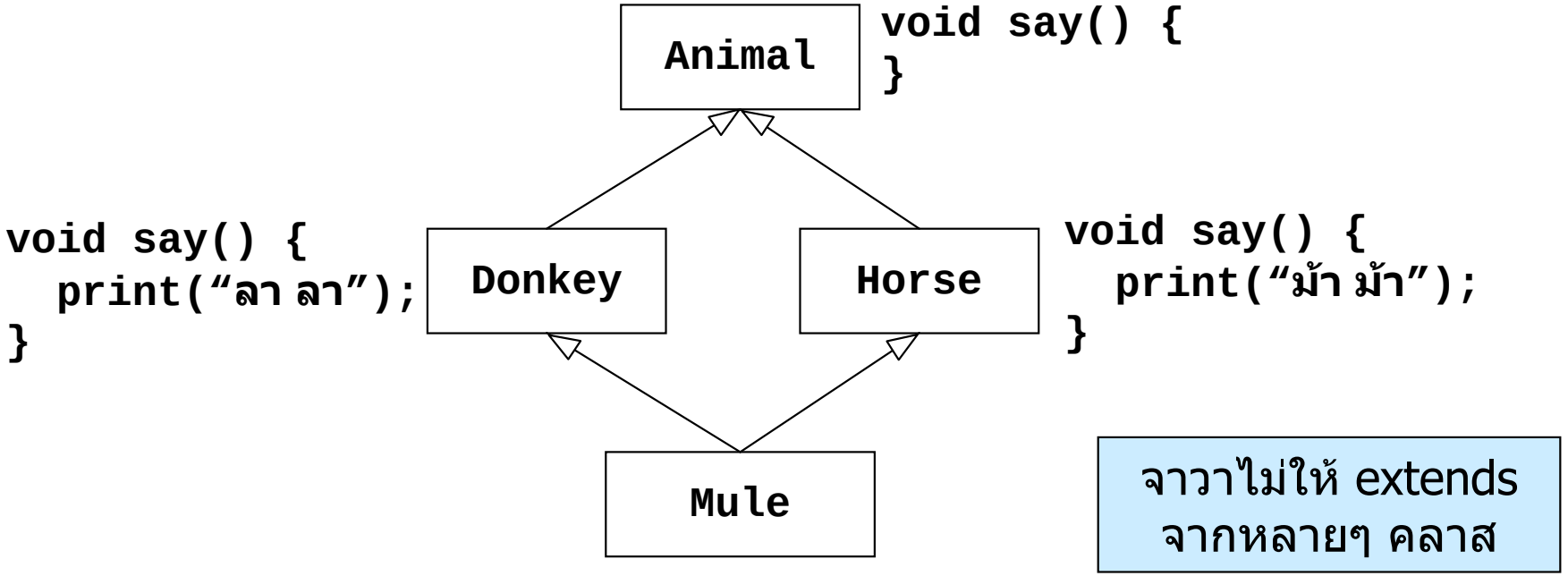
Multiple Inheritance Interface

- หนึ่ง class extends ได้เพียงหนึ่ง superclass
- แต่หนึ่ง interface extends หลาย ๆ superinterfaces ได้




```
interface MouseInputListener extends
          MouseListener, MouseMotionListener  {
}
```

Diamond Problem




```
Animal a = new Mule();
a.say(); // which "say" is invoked ?
```

interface : public static final fields

- เติม public static final fields ให้กับ interface ได้
- ไม่ใส่ public static final ก็เป็น public static final
- จะถูกเพิ่มเข้าไปในคลาสที่ implements

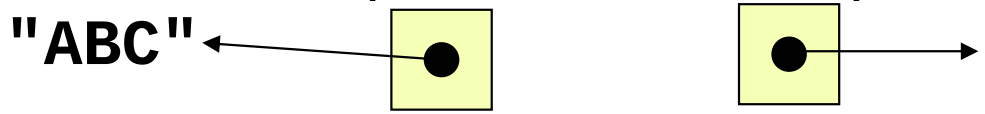
```
interface EnergySaving extends OnOffDevice {
  public static final int OFF = 0;
  public static final int ON = 1;
  public static final int STANDBY = 2;
  public abstract int getStatus();
  public abstract void setStandbyIdleTime(int seconds);
}
```

```
interface EnergySaving extends OnOffDevice {
  int OFF = 0;
  int ON = 1;
  int STANDBY = 2;
  int getStatus();
  void setStandbyIdleTime(int seconds);
}
```

การเปรียบเทียบออปเจกต์

- ถ้า s และ t เป็น object reference
  - s == t : อ้างอิงออปเจกต์เดียวกันหรือไม่
  - s.equals(t) : อ้างอิงออปเจกต์ที่มีค่าเท่ากันหรือไม่

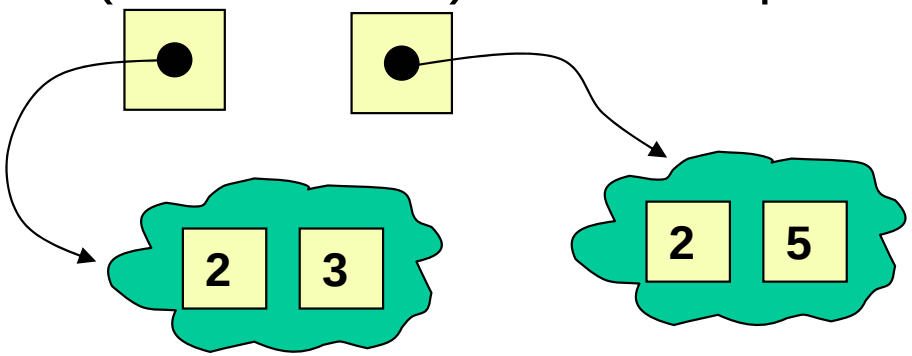
```
if ( s   ==    t )   // false
```




```
if ( s.equals( t ) ) // true
```

- ใช้ < <= > >= กับ object references ไม่ได้

```
if ( a  <  b )  // compilation error
```



Comparable Interface

```
public interface Comparable {
 /**
  * Compares this object with the specified object for
  * order.  Returns a negative integer, zero, or a positive
  * integer as this object is less than, equal to, or
  * greater than the specified object.
  * ...
  * @param   o the Object to be compared.
  * @return  a negative integer, zero, or a positive integer
  *    as this object is less than, equal to, or greater
  *    than the specified object.
  *
  * @throws ClassCastException if the specified object's
  *    type prevents it from being compared to this Object.
  */
  public int compareTo(Object o);
}
```

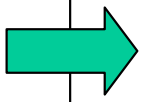
Comparable Interface

- Comparable object ต้องมี compareTo( )

```
static Comparable min(Comparable[] c) {
  Comparable m = null;

  if (c.length > 0) {
    m = c[0];
    for (int i = 1; i < c.length; i++) {
      if (c[i].compareTo(m) < 0) {
        m = c[i];
      }
    }
  }
  return m;
}
```

sort ใน Java API ก็รับ Comparable objects



```
package java.util;

public class Arrays {
 /**
  * Sorts the specified array of objects into ascending
  * order, according to the natural ordering of its
  * elements.  All elements in the array must implement
  * the Comparable interface.  Furthermore, all elements
  * in the array must be mutually comparable (that is,
  * e1.compareTo(e2) must not throw a ClassCastException
  * for any elements e1 and e2 in the array). ...
  */
  public static void sort(Object[] a) {
    ...
  }
  ...
}
```

```
String[] data = { "hello", "how", "are", "you" };
Arrays.sort(data);
```

Point is Comparable

```
public Point implements Comparable {
  int x, y;
  public Point(int x, int y) {
    this.x = x; this.y = y;
  }
  public int compareTo(Object o) {
    Point that = (Point) o;
    double dThis = this.x * this.x + this.y * this.y;
    double dThat = that.x * that.x + that.y * that.y;
    if (dThis < dThat) return -1;
    return (dThis == dThat ? 0 : 1);
  }
  ...
}
```

ใช้ระยะทางวัดจากจุดถึง (0,0) เป็นตัวเปรียบเทียบ

ข้อควรระวัง

1

```
public Point {
   ...
   public int compareTo(Object o) {
      ...
   }
   ...
}
```

2

```
public Point implements Comparable {
   ...
   public int compareTo(Comparable o) {
      ...
   }
   ...
}
```

3

```
public Point implements Comparable {
   ...
   int compareTo(Object o) {
      ...
   }
   ...
}
```

Marker Interface

- an interface with no abstract methods

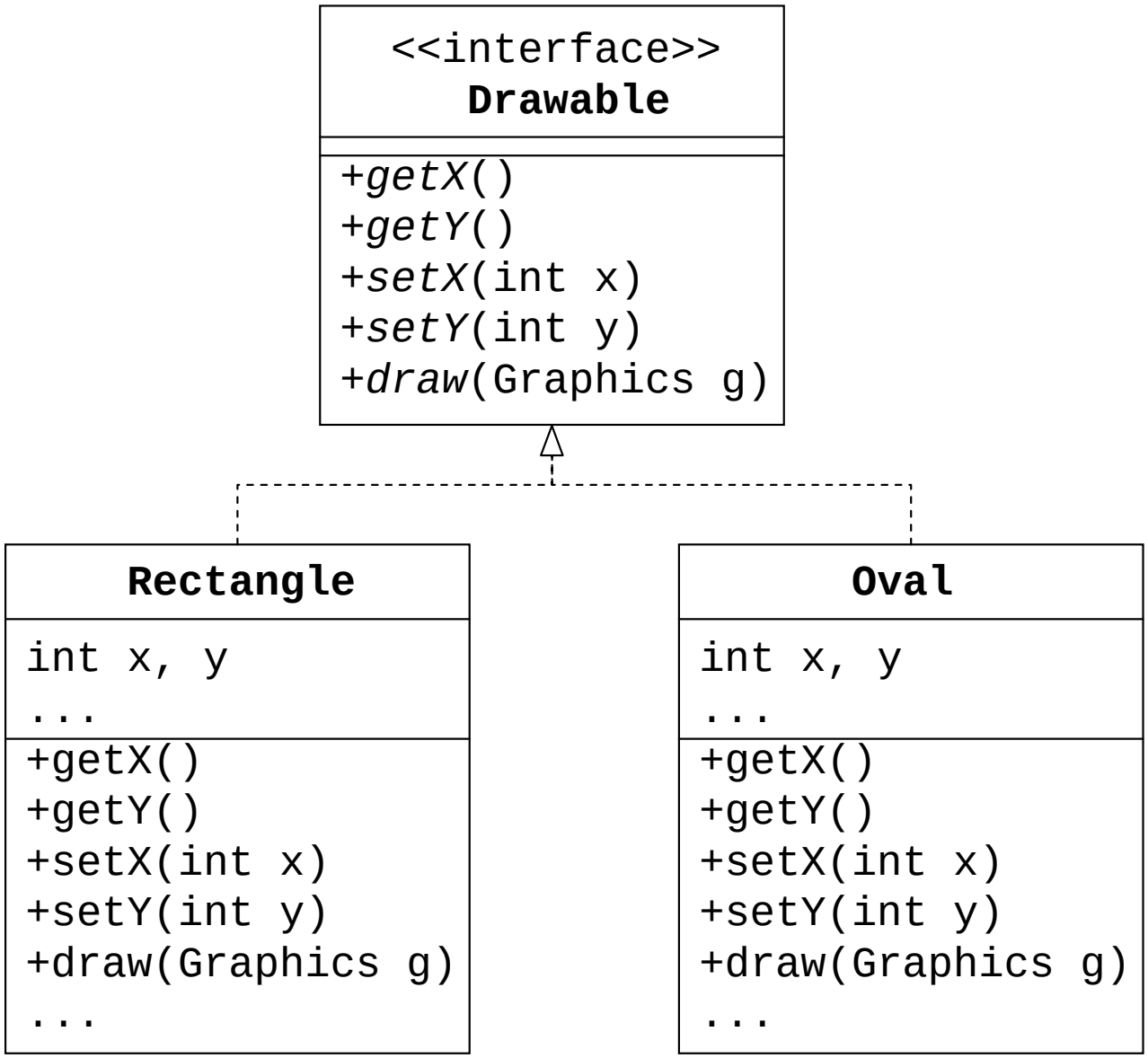
```
pacakage java.io;
public interface Serializable { }
```

```
public class Point implements java.io.Serializable {
  ...
```

```
FileOutputStream out = new FileOutputStream("filename");
ObjectOutputStream os = new ObjectOutputStream(out);
os.writeObject("10 Points");
for (int i = 0; i < 10; i++) os.writeObject(new Point(0, i));
os.close();

FileInputStream in = new FileInputStream("filename");
ObjectInputStream is = new ObjectInputStream(in);
System.out.println("read " + is.readObject());
Point[] p = new Point[10];
for (int i = 0; i < p.length; i++) p[i]=(Point) is.readObject();
```

Abstract Classes



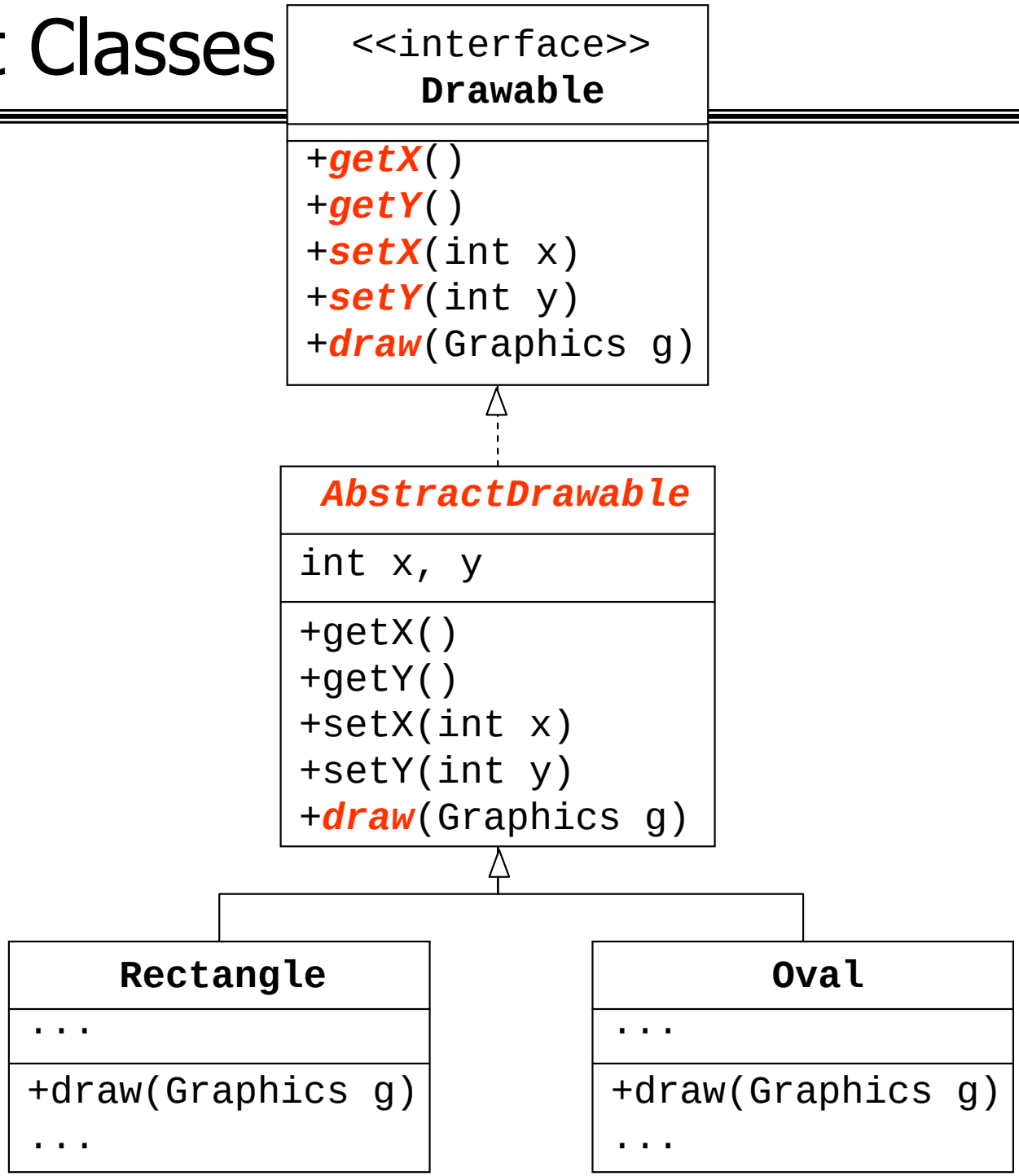
Abstract Classes

```
interface Drawable {
  void draw(Graphics g);
  int getX();
  int getY();
  void setX(int x)
  void setY(int y)
}
```

```
class Rectangle implements Drawable {
  private int x, y;
  public void draw(Graphics g) {...}
  public int getX() { return x; }
  public int getY() { return y; }
  public void setX(int x) { this.x = x; }
  public void setY(int y) { this.y = y; }
  public void draw(Graphics g) {...}
```

```
class Oval implements Drawable {
  private int x, y;
  public int getX() { return x; }
  public int getY() { return y; }
  public void setX(int x) { this.x = x; }
  public void setY(int y) { this.y = y; }
  public void draw(Graphics g) {...}
  ...
}
```

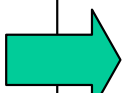
Abstract Classes



Abstract Classes : provide default behaviors

```
interface Drawable {
  void draw(Graphics g);
  int  getX();
  int  getY();
  void setX(int x);
  void setY(int y);
}
```

```
abstract class AbstractDrawable implements Drawable {
  private int x, y;
  public int  getX() { return x; }
  public int  getY() { return y; }
  public void setX(int x) { this.x = x; }
  public void setY(int y) { this.y = y; }
  public abstract void draw(Graphics g);
}
```



```
class Rectangle extends AbstractDrawable {
  public void draw(Graphics g) {...}
  ...
}
```

```
class Oval extends AbstractDrawable {
  public void draw(Graphics g) {...}
  ...
}
```

Interface vs. Abstract Class

- ใน interfaces มี
  - public abstract methods
  - public static final fields
- ใน abstract class มี
  - อะไรก็ได้ที่ class มี และ
  - abstract methods
  - มี constructor ไว้ให้ลูกเรียก super(...)
- class หนึ่ง
  - implements ได้หลาย interfaces
  - extends ได้แค่หนึ่ง class หรือหนึ่ง abstract class

Lab 5.1 : Rational is Comparable

- จงทำให้คลาส com.somchai.lab5.Rational implements Comparable
  - เพิ่ม implements Comparable ที่หัวคลาส Rational
  - เพิ่มเมท็อด public double doubleValue( ) ซึ่งคืนค่าจำนวนจริงเช่น `new Rational(2, 4).doubleValue()` ได้ 0.5
  - เพิ่มเมท็อด public int compareTo(Object obj)

```
public static void main(String[] args) {
  Rational[] r = {
    new Rational(2, 3),
    new Rational(4, 9),
    new Rational(4, 5)
  };
  System.out.println(r[0].compareTo(r[1]));
  System.out.println(r[0].compareTo(r[2]));
}
```

ลอง Run ดู

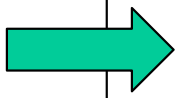
Lab 5.2 : Arrays.sort

- คลาส Arrays มีเมท็อด sort, ... ที่รองรับทั้ง primitive types และออปเจกต์ที่ Comparable
- ลองเขียน code ข้างล่างนี้ และ run (ใน com.somchai.lab5.Rational)

```
public static void main(String[] args) {
  Rational[] r = {
    new Rational(2, 3), new Rational(2, 5),
    new Rational(4, 9), new Rational(1, 3),
    new Rational(4, 5), new Rational(7, 9)
  };
  java.util.Arrays.sort(r);
  for (int i = 0; i < r.length; i++) {
    System.out.println(r[i]);
  }
}
```

Lab 5.3 : Copy Constructor

- ถ้าเราอยากได้ออปเจกต์ใหม่ตัวหนึ่งที่มีค่าเหมือนออปเจกต์อีกตัว ให้เขียน copy constructor
- Copy constructor : constructor ที่รับออปเจกต์ของคลาสตัวเองเป็นพารามิเตอร์
- จงเพิ่ม copy constructor ใน Rational และทดสอบ



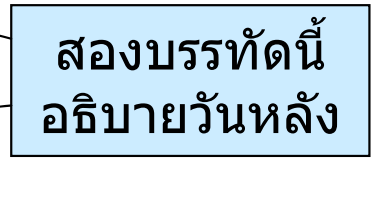
```
public class Rational {
  private int numerator, denominator;

  public Rational() { this(0, 1); }
  public Rational(int n, int d) {
    numerator = n; denominator = d;
  }
  public Rational(Rational r) {
    ...
  }
```

Lab 5.4 : clone

- ถ้าเราอยากได้ออปเจกต์ใหม่ตัวหนึ่งที่มีค่าเหมือนออปเจกต์อีกตัว ให้เขียนเมท็อด clone()
- clone เป็น protected method ของ Object
- อยากให้บริการ clone กับคลาสใด
  - ให้คลาสนั้น implements Cloneable
  - เพิ่มเมท็อด public Object clone() ในคลาสนั้น

```
class Rational implements Comparable, Cloneable {
  ...
  public Object clone() {
    Object c = null;
    try {
      c = super.clone();
    } catch (CloneNotSupportedException e) {}
    return c;
  }
```

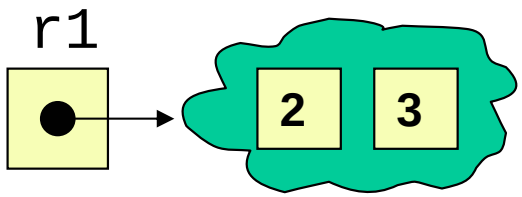


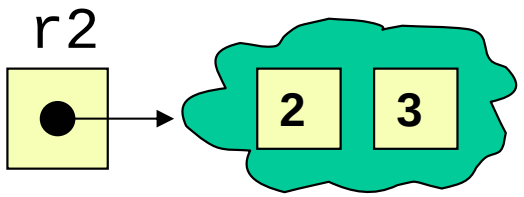
Lab 5.4 : clone

- จงเพิ่ม clone ใน Rational และทดสอบตาม code ข้างล่างนี้ใน main

```
Rational r1 = new Rational(2, 3);

Rational r2 = r1.clone();
System.out.println(r1 == r2);
System.out.println(r1.equals(r2));
System.out.println(r2.equals(r1));
```





ทดสอบ : Run JUnit ของ lab5