วาจาจาวา – ตอนที่ ๒
เมท็อดของคลาส Object

สมชาย ประสิทธิ์จูตระกูล

เมท็อดของคลาส Object

- **`boolean equals(Object obj)`**
  - indicates whether some other object is "equal to" this one.
- **`int hashCode()`**
  - returns a hash code value for the object.
- **`String toString()`**
  - returns a string representation of the object.
- **`Object clone()`**
  - creates and returns a copy of this object.
- **`Class getClass()`**
  - returns the runtime class of an object.
- **`void finalize()`**
- **`void notify()`** **`void notifyAll()`**
- **`void wait()`** **`void wait(long timeout)`**
- **`void wait(long timeout, int nanos)`**

equals : เมื่อไรไม่ต้อง override

- คลาสซึ่งผลิตออปเจกต์ที่ถือว่าแตกต่างกันหมด
- superclass มี equals ที่เราใช้ได้อยู่แล้ว
- ไม่ต้องการให้บริการ equals หรือมั่นใจว่าไม่มีใครเรียก equals แน่ ๆ

```
public boolean equals(Object o) {
  throw new UnsupportedOperationException();
}
```

equals contract

- x.equals(null) ต้องเป็น false
- reflexive : x.equals(x) ต้องเป็น true
- symmetric : x.equals(y) เป็นจริง ก็ต่อเมื่อ y.equals(x) เป็นจริง
- transitive : ถ้า x.equals(y) เป็นจริง และ y.equals(z) เป็นจริง แล้ว x.equals(z) เป็นจริง
- consistent : ถ้าออปเจกต์ไม่เปลี่ยนแปลง เรียก x.equals(y) กี่ครั้ง ก็ต้องได้ค่าเหมือนเดิม

equals : Symmetric : ตัวอย่างผิด

```
class Point {
  int x, y;
  Point(int x1, int y1) { x = x1; y = y1; }
  public boolean equals(Object obj) {
    if (!(obj instanceof Point)) return false;
    Point that = (Point) obj;
    return x == that.x && y == that.y;
  }
  ...
```

```
class Point3D extends Point {
  int z;
  Point3D(int x1, int y1, int z1) { super(x1, y1); z = z1; }
  public boolean equals(Object obj) {
    if (!(obj instanceof Point3D)) return false;
    return super.equals(obj) && z == ((Point3D) obj).z;
  }
  ...
```

```
Point p = new Point(2,3);
Point3D p3 = new Point3D(2,3,4);
System.out.println(p.equals(p3)); // true
System.out.println(p3.equals(p)); // false
```

equals : Symmetric ถูก แต่ Transitive ผิด

```
class Point {
  int x, y;
  Point(int x1, int y1) { x = x1; y = y1; }
  public boolean equals(Object obj) {
    if (!(obj instanceof Point)) return false;
    Point that = (Point) obj;
    return x == that.x && y == that.y;
  }
  ...
```

mixed-type comparison เกิดปัญหา

```
class Point3D extends Point {
  int z;
  Point3D(int x1, int y1, int z1) { super(x1, y1); z = z1; }
  public boolean equals(Object obj) {
    if (!(obj instanceof Point3D)) return super.equals(obj);
    return super.equals(obj) && z == ((Point3D) obj).z;
  }
  ...
```

```
Point3D p3 = new Point3D(2,3,4);
Point   p  = new Point(2,3);
Point3D p4 = new Point3D(2,3,5);
System.out.println(p3.equals(p)); // true
System.out.println(p.equals(p4)); // true
System.out.println(p3.equals(p4));// false
```

equals : Symmetric ถูก Transitive ถูก

```
class Point {
  int x, y;
  Point(int x1, int y1) { x = x1; y = y1; }
  public boolean equals(Object obj) {
    if (obj==null || obj.getClass() != getClass()) return false;
    Point that = (Point) obj;
    return x == that.x && y == that.y;
  }
  ...
```

mixed-type comparison → false

```
class Point3D extends Point {
  int z;
  Point3D(int x1, int y1, int z1) { super(x1, y1); z = z1; }
  public boolean equals(Object obj) {
    if (!(obj instanceof Point3D)) return super.equals(obj);
    return super.equals(obj) && z == ((Point3D) obj).z;
  }
  ...
```

```
Point3D p3 = new Point3D(2,3,4);
Point   p  = new Point(2,3);
Point3D p4 = new Point3D(2,3,5);
System.out.println(p3.equals(p)); // false
System.out.println(p.equals(p4)); // false
System.out.println(p3.equals(p4));// false
```

equals : เมื่อไม่รับ mixed-type comparison

```
class A {
  public boolean equals(Object obj) {
    if (obj == null || obj.getClass() != getClass()) return false;
    A that = (A) obj;
    ...
    return ...
  }
  ...
}
```

```
class B extends A {
  public boolean equals(Object obj) {
    if (!super.equals(obj)) return false;
    B that = (B) obj;
    ...
    return ...
  }
  ...
}
```

```
class C extends B {
  public boolean equals(Object obj) {
    if (!super.equals(obj)) return false;
    C that = (C) obj;
    ...
    return ...
  }
  ...
}
```

equals : ข้อแนะนำ

- ใช้ instanceof ก่อน cast ต้องระวังเรื่องการเปรียบเทียบออปเจกต์ของ subclass (mixed-type)
- เพิ่มความเร็วด้วย if (this == obj) return true;

```
public boolean equals(Object obj) {
  if (obj == this) return true;
  ...
}
```

- อย่าเขียนหัวเมท็อดผิด

```
class A {
  public boolean equals(A obj) { // overload equals
   if (obj == this) return true;
  }
}
```

- override equals แล้วก็ override hashCode ด้วย

hashCode

- คืน int ที่ใช้คำนวณตำแหน่งใน hash table (HashSet, HashMap, Hashtable, ...)
- hashCode ของคลาส Object คืน memory address ที่เก็บออปเจกต์นั้น
- ข้อบังคับ :
  - ถ้า x.equals(y) เป็นจริง
    x.hashCode() ต้องเท่ากับ y.hashCode()

```
// if Point doesn't override hashCode
Set s = new HashSet();
s.add(new Point(2, 3));
System.out.println(s.contains(new Point(2, 3))); // false !!
```

hashCode : แบบง่ายสุด (แบบแย่สุด ๆ)

- คืนค่าคงตัวค่าหนึ่งเสมอ
- ตรงตามข้อบังคับ :
  - x.equals(y) เป็นจริง x.hashCode() ก็เท่ากับ y.hashCode()
- ข้อเสีย
  - "ชน" แหลก !!!
  - ประสิทธิภาพการทำงานกับ hash table ช้าสุด ๆ

```
class Point {
  ...
  public int hashCode() {
    // a correct (and worst) hashCode
    return 0;
  }
}
```

hashCode : ตัวอย่าง

```
class Point {
  int x, y;
  ...
  public int hashCode() {
    int code = 17;
    code = 37 * code + x;
    code = 37 * code + y;
    return code;
  }
}
```

```
class Polyline {
 Point[] points;
  ...
  public int hashCode() {
    int code = 1;
    for (int i = 0; i < points.length; i++) {
      Object obj = points[i];
      code = 31 * code + (obj == null ? 0 : obj.hashCode());
    }
    return code;
  }
}
```

hashCode : การทำงาน

- การเปลี่ยนแต่ละ field f เป็น c (c เป็น int)
  1. boolean : (f ? 0 : 1)
  2. byte, char, short, int : (int) f
  3. long : (int) (f ^ (f >>>32))
  4. float : Float.floatToIntBits(f)
  5. double : Double.doubleToIntBits(f) ได้ long แล้วเปลี่ยน long เป็น int ด้วยวิธีที่ 2
  6. object reference : (f==null ? 0 : f.hashCode())
  7. array : นำแต่ละช่องใน array มาคิดตาม 1 – 6
- นำ c ของแต่ละ field มารวมกัน
  - result = 37 * result + c;

hashCode : ข้อแนะนำ

- ใช้เฉพาะ fields ที่ใช้ใน equals มาคำนวณ
- ตัด field ที่ไม่จำเป็นออกได้
- อย่าตัด field ออกเพราะต้องการลดเวลาการทำงาน
- ถ้าเป็นคลาสแบบ immutable สามารถ cache ค่า hashCode ไว้ใช้ใหม่ในอนาคต (แล้วไม่ต้องคำนวณใหม่)
- ไม่จำเป็นต้องบอกวิธีคำนวณ hashCode ในคู่มือ

toString

- ... the toString method returns a string that "textually represents" this object. The result should be a concise but informative representation that is easy for a person to read ...

```
System.out.println(new Color(12, 13, 14).toString());
Object[] s = { "J2SE", new Date(), new Object() };
List list = Arrays.asList(s);
System.out.println(list.toString());
```

```
java.awt.Color[r=12,g=13,b=14]
[J2SE, Wed Oct 06 22:31:08 ICT 2004, java.lang.Object@757aef]
```

toString

- ระบบเรียก toString() ให้อัตโนมัติ
  - นำออปเจกต์ + สตริงอีกตัว
  - เมท็อดที่รับพารามิเตอร์เป็นสตริงมัก overload เมท็อดเดียวกันแต่รับ object reference แล้วเรียก toString() ให้
- toString() ของคลาส Object (ไม่ค่อยสื่อความหมาย)

```
public String toString() {
  return getClass().getName() + "@" +
         Integer.toHexString(hashCode());
}
```

- ควรเขียน toString() เพื่อความสะดวกของผู้ใช้

```
Date d1 = new Date();
Date d2 = new Date();
d2.setYear(d1.getYear() + 1);
String s = ">>" + d1;
System.out.println(d1);
assert d1.compareTo(d2) > 0 : d1;
```

toString : ตัวอย่าง

```
package com.somchai;

public class Point {
  int x, y;
  ...
  public String toString() {
    return getClass().getName() +
           "[x=" + x + ",y=" + y + "]";
  }
}
```

```
System.out.println(new Point(2, 3));
System.out.println(new Point(12, 52));
```

```
com.somchai.Point[x=2,y=3]
com.somchai.Point[x=12,y=52]
```

toString : รูปแบบของผลลัพธ์ที่ไม่แน่นอน

- บางคนใช้ toString( ) เพื่อเข้าถึงข้อมูลของออปเจกต์
- ถ้าเราไม่ต้องการให้ทำเช่นนั้น
  - เขียนไว้ให้ชัดเจนใน javadoc ว่า “อย่าทำ”
  - ให้บริการ accessor methods

```
public class Point {
  private int x, y;
  /*
   * Returns a brief description of this 2d-point.
   * The format of the returned string is subject to change.
   */
  public String toString() {
    return "[x=" + x + ",y=" + y + "]";
  }
  public int getX() { return x; }
  public int getY() { return y; }
  ...
```

toString : รูปแบบของผลลัพธ์ที่แน่นอน

```
public class Point {
  private int x, y;
  /**
   * Returns the string representation of this 2d-point.
   * The format is "(X,Y)" where X and Y are comma
   * separated, and can be any number of digits.
   */
  public String toString() {
    return "(" + x + "," + y + ")";
  }
  /**
   * Constructs a newly allocated Point object that
   * represents 2d-point indicated by the parameter whose
   * format is the same as returned by toString()
   * @param s the String to be converted to a Point object.
   * @exception  IllegalArgumentException
   *    if the String does not contain a parsable 2d-point.
   */
  public Point(String s) throws IllegalArgumentException {...}
  ...
}
```

ควรระบุรูปแบบนั้นไว้ใน javadoc

มี constructor ที่รับสตริงในรูปแบบดังกล่าว

toString : ข้อแนะนำ

- ควร override toString เมื่อเขียนคลาสใหม่
- toString มีไว้ใช้แสดงข้อมูลที่สำคัญของออปเจกต์
- ควรมีเมท็อดที่ให้บริการคืนข้อมูลทุก fields ที่ปรากฏในสตริงที่คืนจาก toString
- อย่าลืมใช้ getClass().getName()
- ต้องเขียนใน javadoc ให้ชัดเจนว่า ผลลัพธ์ที่ได้จาก toString มีรูปแบบที่แน่นอนหรือไม่

clone

- clone ให้บริการสร้าง object ใหม่อีกก้อนหนึ่งที่ “เหมือน” ออปเจกต์ที่ถูกเรียก
  - x.clone() != x เป็นจริง
  - x.clone().getClass() == x.getClass() เป็นจริง
  - x.clone().equals(x) เป็นจริง
- ถ้าคลาสหนึ่งให้บริการ clone
  - clone จะคืนออปเจกต์ประเภท Object
  - โดยทั่วไป ผู้ใช้บริการต้องเปลี่ยนประเภทข้อมูล (cast) ไปเป็นประเภทของออปเจกต์ที่ถูก cloned

```
Point pt1 = new Point(2, 3);
Point pt2 = (Point) pt1.clone();
```

```
int[] x = { 1, 2, 3, 4, 5 };
int[] y = (int[]) x.clone();
```

Cloneable Interface

- เป็น marker interface
  - ไม่มีเมท็อดใดใน interface นี้
  - ไม่ได้บังคับ public Object clone();
  - ทุกคลาสรับ protected Object clone() จากคลาส Object
  - มีไว้เพียงเพื่อให้ clone ที่คลาส Object ตรวจสอบว่าผู้ออกแบบคลาสใหม่อยากให้ clone หรือไม่

```
package java.lang;

public interface Cloneable {
}
```

clone : วิธีเขียนแบบมาตรฐาน

- เพิ่ม implements Cloneable
- เพิ่ม throws CloneNotSupportedException
- เรียก super.clone()
- ออปเจกต์ที่เรียก clone() ต้องอยู่ในคลาสที่ implements Cloneable ไม่เช่นนั้น CloneNotSupportedException

```
class Object {
 protected native Object clone()
                  throws CloneNotSupportedException;
  ...
}
```

การโคลนออปเจกต์เกิดขึ้นจริง ๆ ที่เมท็อดของคลาส Object

```
class B implements Cloneable {
  int b = 1234;
  public Object clone() throws CloneNotSupportedException {
    return super.clone();
  }
  ...
}
```

```
B b1 = new B();
B b2 = (B) b1.clone();
System.out.println(b1 == b2);     // false
System.out.println(b1.b == b2.b); // true
```

clone : super.clone()

- การโคลนออปเจกต์ที่ Object จะสร้างออปเจกต์ของคลาสเดียวกับที่เรียก clone ตัวแรกสุด

```
class Object {
  protected native Object clone()
                    throws CloneNotSupportedException;
}
```

```
class A {  // does not implement Cloneable
  ...
}
```

```
class B extends A implements Cloneable {
  public Object clone() throws CloneNotSupportedException {
    return super.clone();
  }
}
```

```
class C extends B {
  ...
}
```

```
A a1 = new A();
A a2 = (A) a1.clone(); // Clone not supported
B b1 = new B();
B b2 = (B) b1.clone(); // OK
C c1 = new C();
C c2 = (C) c1.clone(); // OK
```

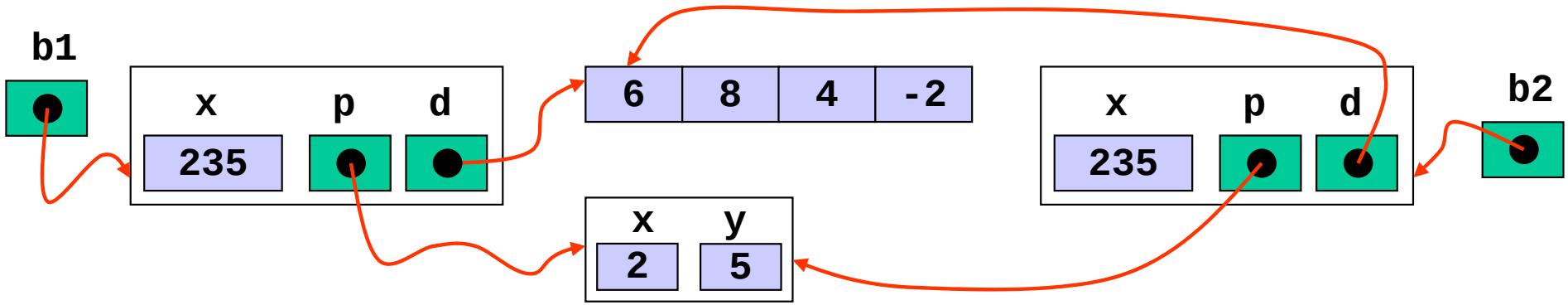
clone : Object.clone ทำ shallow cloning

- โคลนเฉพาะ fields ต่าง ๆ ของตัวเองและ super กรณีที่เป็น object reference หรือ array จะโคลนเฉพาะตัว references

```
class A {
  int [] d = new int[4];
  ...
}
```

```
class B extends A implements Cloneable {
  Point p;
  int x;
  public Object clone() throws CloneNotSupportedException {
    return super.clone();
  }
  ...
}
```

```
B b1 = new B();
...
B b2 = (B) b1.clone();
```



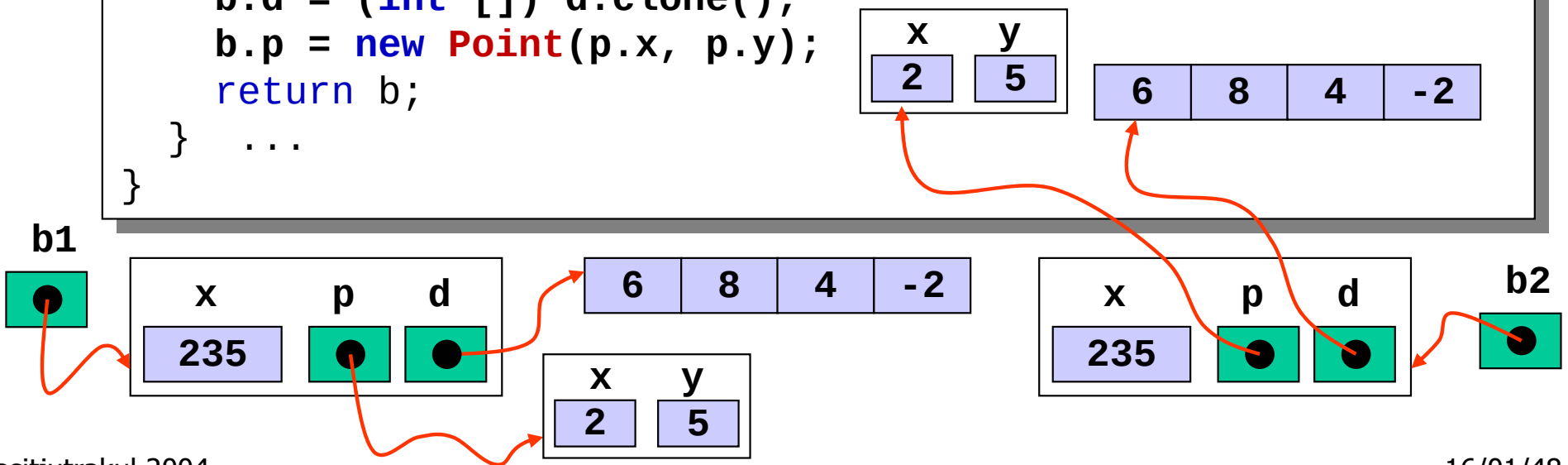
clone : deep cloning

- กรณีที่เป็น object reference หรือ array ต้องโคลนตัวออปเจกต์เอง

```
B b1 = new B();
...
B b2 = (B) b1.clone();
```

```
class A {
  int [] d = new int[4];
  ...
}
```

```
class B extends A implements Cloneable {
  Point p;
  int x;
  public Object clone() throws CloneNotSupportedException {
    B b = (B) super.clone();
    b.d = (int []) d.clone();
    b.p = new Point(p.x, p.y);
    return b;
  }  ...
}
```



clone : deep cloning

- แต่ละคลาสควรรับผิดชอบการทำ cloning ของตัวเอง

```
B b1 = new B();
...
B b2 = (B) b1.clone();
```

```
class A implements Cloneable {
  int [] d = new int[4];
  public Object clone() throws CloneNotSupportedException {
    A a = (A) super.clone();
    a.d = (int []) d.clone();
    return a;
  }
  ...
}
```

```
class B extends A implements Cloneable {
  Point p;
  int x;
  public Object clone() throws CloneNotSupportedException {
    B b = (B) super.clone();
    b.p = new Point(p.x, p.y);
    return b;
  }
  ...
}
```

clone : ถ้ารำคาญ CloneNotSupportedException

- เปลี่ยน CloneNotSupportedException เป็น Error

```
class A implements Cloneable {
  int [] d = new int[4];
  public Object clone() {
    try {
      A a = (A) super.clone();
      a.d = (int[]) d.clone();
      return a;
    } catch (CloneNotSupportedException e) {
      throw new Error("can't clone");
    }
  }
  ...
}
```

```
class B extends A {
  Point p;
  int x;
  public Object clone() {
    B b = (B) super.clone();
    b.p = new Point(p.x, p.y);
    return b;
  }
  ...
}
```

clone : เมื่อไม่อยากรับบริการ clone จากพ่อ

- ถ้า superclass ทำ clone ได้ แต่ subclass ไม่ต้องการ

```
class A implements Cloneable {
  public Object clone() throws CloneNotSupportedException {
    return super.clone();
  }
  ...
```

```
class B extends A {
  public Object clone() throws CloneNotSupportedException {
    throw new CloneNotSupportedException();
  }
  ...
```

```
class C implements Cloneable {
  public Object clone() {
    ...
  }
  ...
```

```
class D extends C {
  public Object clone() {
    throw new UnsupportedOperationException();
  }
  ...
```

clone : constructor หรือ static factory

- เขียน clone ทั้งยุ่ง ทั้งงง
- ใช้ copy constructor หรือ static factory ง่ายกว่า

```
public class A {
  int x;
  Point p;
  public A(A a) {
    this.x = a.x;
    this.p = new Point(a.p.getX(), a.p.getY());
  }
  ...
}
```

```
public class A {
  int x;
  Point p;
  public static A newInstance(A a) {
    A newa = new A();
    newa.x = a.x;
    newa.p = new Point(a.p.getX(), a.p.getY());
  }
  ...
}
```

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

- ไม่เกี่ยวอะไรเลยกับคลาส Object
- ใช้กับคลาสที่ให้บริการเปรียบเทียบออปเจกต์
- คลาสทั่วไปที่อาศัยการเปรียบเทียบข้อมูลเช่น TreeSet, TreeMap, Collections, Arrays (เมท็อด search, sort) อาศัย Comparable ทั้งสิ้น

```
public Point implements Comparable {
  ...
  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

Comparable : Exceptions

- ภายใน compareTo
  - cast เลย ถ้าทำไม่ได้ก็ให้เกิด ClassCastException
  - ใช้เลย ถ้าเป็น null ก็ได้เกิด NullPointerException

```
public Point implements Comparable {
   ...
   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);
   }
   ...
}
```

Comparable : ข้อแนะนำ

- อย่าออกแบบ subclass ที่มีพฤติกรรมของ compareTo ที่ต่างจากของ superclass
- ควรให้ compareTo ทำงานสอดคล้องกับ equals **`(x.compareTo(y) == 0) == (x.equals(y))`**
- อย่าเขียน **`public int compareTo(Comparable o)`**
- อย่าลูกเล่นมากใน compareTo

```
public class A implements Comparable {
  int x;
  A(int x) { this.x = x; }
  public int compareTo(Object o) {
    return x - ((A) o).x; // WRONG
  }
}
```

```
A a1 = new A(Integer.MAX_VALUE);
A a2 = new A(-1);
System.out.println(a1.compareTo(a2));
```