2110211 : โครงสร้างข้อมูล

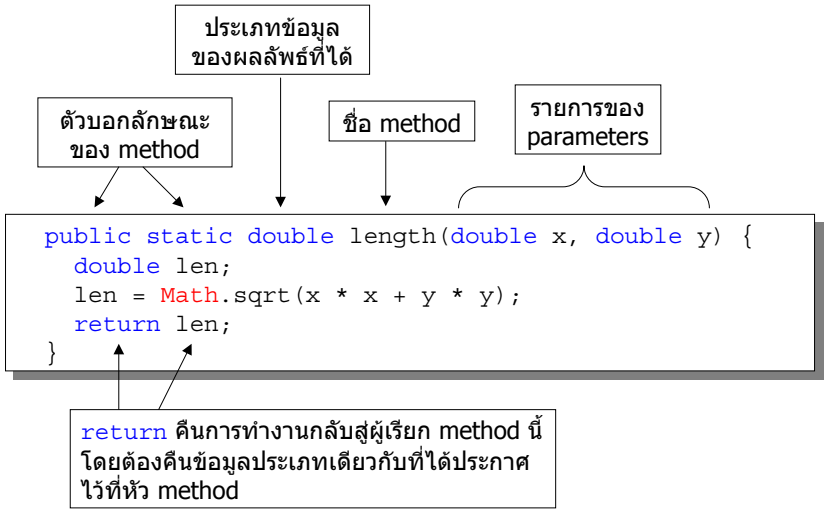
แนะนำจาวาตอน ๒

สมชาย ประสิทธิ์จูตระกูล
ภาควิชาวิศวกรรมคอมพิวเตอร์
จุฬาลงกรณ์มหาวิทยาลัย

หัวข้อ

- Methods
- Arrays
- Classes and Objects

องค์ประกอบของเมท็อด



หัวเมท็อด : ประเภทข้อมูลของผลลัพธ์

- ประเภทข้อมูลพื้นฐาน (int, double, boolean, ...)
- ประเภทข้อมูลแบบคลาส (String, Point, ...)
- ถ้าเป็น method ที่ไม่คืนผลลัพธ์ ให้ใส่คำว่า void

```
public static int min(int a, int b)
int m = min(a, 0);

public static String toString(int i, int radix)
String s = toString(a, 16);

public static void gc()
gc();
```

หัวเมท็อด : ชื่อ

- การตั้งชื่อใช้กฎเหมือนกับตัวแปร
  - ประกอบด้วยตัวอักษร ตัวเลข ตัว $ หรือ _ ก็ได้
  - ห้ามขึ้นต้นด้วยตัวเลข
  - ไม่มีข้อจำกัดเรื่องความยาวของชื่อ
  - ตัวอักษรตัวใหญ่ไม่เหมือนตัวเล็ก
  - ต้องไม่ซ้ำกับคำสงวนของภาษาจาวา
- โดยทั่วไปชื่อเมท็อดขึ้นต้นด้วยตัวเล็ก
- มักตั้งชื่อเมท็อดให้เป็นกริยา

```
fillCircle  setEnabled  turnLeft  println  start  run
```


หัวเมท็อด : รายการของพารามิเตอร์

- รายการของ parameters
  - แต่ละ parameter เขียนเหมือนการประกาศตัวแปร (แต่ไม่ต้องปิดท้ายด้วย semi-colon)
  - แต่ละ parameter คั่นด้วยจุลภาค (comma)
  - รายการของ parameters อยู่ภายในวงเล็บ (ถ้าไม่รับ parameter ใดๆ ก็ไม่ต้องใส่อะไรในวงเล็บ)

```
public static void turnLeft(double angle)
```

```
public static double pow(double a, double b)
```

```
public static String readString()
```


สิ่งที่มักผิดบ่อย

```
static printError(String msg) {
  System.err.println( msg );
}
```

```
public static int abs(int a) {
  if ( a < 0 ) a = -a;
}
```

```
static float max(float x, y) {
  if ( x > y ) return x; else return y;
}
```

```
static double length(double dx, double dy) {
  double dx = Math.abs(dx);
  return Math.sqrt( dx*dx + dy*dy );
}
```

```
public static float getBlackColor() {
  return 0.0;
}
```


การเรียกใช้เมท็อด : การส่งข้อมูล

- จำนวนข้อมูลที่ส่งต้องตรงกับที่ประกาศไว้ที่หัวเมท็อด
- สิ่งที่ส่งไปต้องเป็นประเภทที่ตัวแปรที่หัวเมท็อดรับได้ (ถ้าจำเป็น ระบบจะทำ argument promotion ให้)

```
public static float max(float x, float y) {
  . . .
}
```

ถูก

```
x = max( (float) (a/3.0), 1f );
y = max( Float.parseFloat(str), 1 );
```

ผิด

```
x = max( (float) (a/2) );
y = max( "1.0", 1 );
```

การเรียกใช้เมท็อด : ผลที่คืนกลับ

- ในกรณีที่เมท็อดมีการคืนค่า ก็ต้องมีตัวแปรซึ่งมีประเภทที่รับค่าของข้อมูลที่คืนมาได้

```
public static float max(float x, float y) {
  . . .
}
```

ถูก

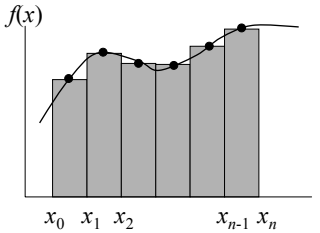
```
double x = max( 1.0f, (int) (x/3) );
int y = (int) max( 1, 2 );
float z = max( max(x, y), -10.5f ) );
```

ผิด

```
long x = max( 4.0f, 2.3f );
float y = max( 4.0, 1.0 );
max( 2.0f, 1.0f );
```


ตัวอย่าง : Integration

- แบ่งพื้นที่ใต้กราฟเป็นแถบๆ
- หาพื้นที่ของแต่ละแถบ แล้วหาผลรวม
- ใช้ midpoint rule ประมาณพื้นที่ใต้กราฟดังนี้



$$h = \frac{b-a}{n}$$

$$\int_a^b f(x)dx \approx \sum_{i=1}^{n} hf\left(\frac{x_{i-1}+x_i}{2}\right)$$

Method Overloading

- ชื่อเมท็อดซ้ำได้ ถ้ารายการพารามิเตอร์ไม่เหมือนกัน
- ระบบจะเรียกเมท็อดที่เหมาะสมให้

```
static int max(int x, int y) {
  if ( x > y ) return x; else return y;
}
static double max(double x, double y) {
  if ( x > y ) return x; else return y;
}
static double max(double x, double y, double z) {
  if ( x > y && x > z ) return x;
  else if ( y > x && y > z ) return y;
  else return z;
}
```


Method Overloading

- ไม่สามารถ overload เมท็อดที่รายการพารามิเตอร์เหมือน แต่ประเภทข้อมูลที่คืนไม่เหมือน
- ความเหมือนไม่เหมือนของพารามิเตอร์ อยู่ที่ประเภทข้อมูล ไม่เกี่ยวกับชื่อที่ใช้

```
static int max(int x, int y) {
  if ( x > y ) return x; else return y;
}
static double max(int x, int y) {
  if ( x > y ) return (double) x;
  else return (double) y;
}
static int max(int a, int b) {
  if ( x > y ) return a; else return b;
}
```

การประกาศและการสร้างอาเรย์

- ก่อนใช้อาเรย์ต้อง ประกาศตัวแปรอาเรย์ และ สร้างตัวอาเรย์

```
int [] data;
```

ประกาศตัวแปรชื่อว่า data มีไว้อ้างอิงอาเรย์ของ int

```
data = new int[10];
```

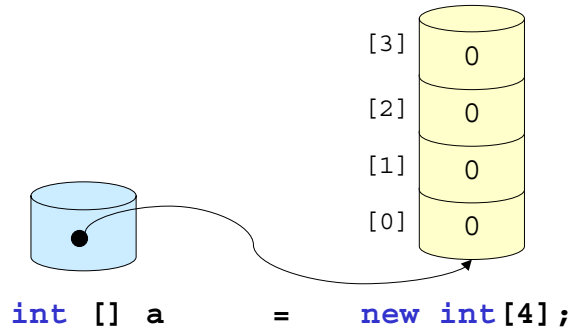
สร้างอาเรย์ของ int จำนวน 10 ช่อง แล้วให้ data เป็นตัวแปรสำหรับอ้างอิงอาเรย์ที่ได้สร้างขึ้น

```
int [] data = new int[10];
```

ประกาศตัวแปรและสร้างอาเรย์พร้อมกันเลยก็ได้

ตัวแปรอาเรย์กับตัวอาเรย์

- การประกาศตัวแปรอาเรย์ เป็นการสร้างที่เก็บ "ที่อยู่" (reference) ของอาเรย์ที่จะสร้าง

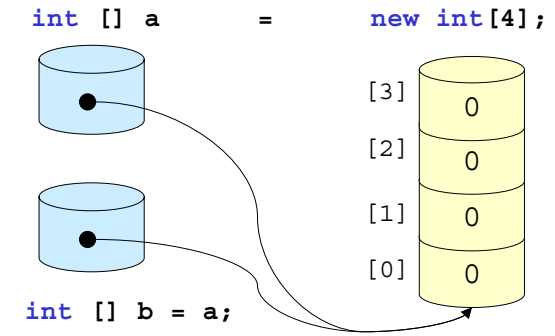


ข้อสังเกต : การประกาศ+การสร้าง

- การประกาศตัวแปรอาเรย์ ไม่ได้เป็นการสร้างตัวอาเรย์
- ขนาดของอาเรย์ถูกกำหนดตอนสร้างตัวอาเรย์ ไม่ใช่ตอนประกาศตัวแปร
- ขนาดของอาเรย์เป็นค่าของนิพจน์ก็ได้ (`data = new int[ 3*n + 1];`)
- อาเรย์ที่สร้างขึ้น ไม่สามารถเพิ่มหรือลดขนาดได้
- ระบบจะตั้งค่าเริ่มต้นของข้อมูลในอาเรย์โดยอัตโนมัติ (ให้ค่าศูนย์กับอาเรย์ของจำนวน และ ให้ค่า false กับอาเรย์ของ boolean)

ข้อสังเกต : การอ้างอิงอาเรย์

- อาเรย์แถวหนึ่ง อาจมีตัวแปรหลายตัวอ้างอิงได้



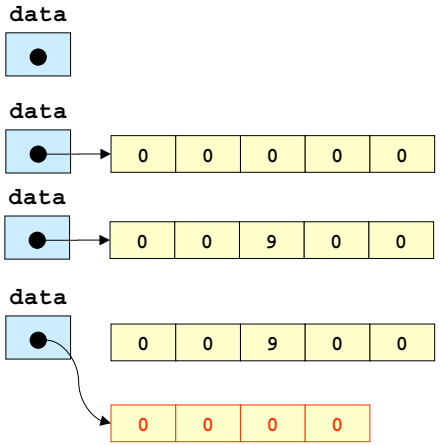
สร้างใหม่ได้ แต่ของเดิมหาย

```
int[] data;

data = new int[5];

data[2] = 9;

data = new int[4];
```



ประกาศ+สร้าง+ตั้งค่าเริ่มต้น

- ใช้ initializer list ระบุค่าเริ่มต้น
- ไม่ต้องกำหนดจำนวนช่องที่จะสร้าง
- จำนวนช่องที่สร้างเท่ากับจำนวนข้อมูลใน list

```
int[] data = {23, 3, 21, 47};
```

```
int[] data;
data = {23, 3, 21, 47};
```

เขียนแยกเป็นสองบรรทัดแบบนี้ไม่ได้

การอ้างอิงข้อมูลแบบผิดๆ

- ถ้าอาเรย์มีขนาด n ตัว
  index ต้องเป็นจำนวนเต็มมีค่าได้ตั้งแต่ 0 ถึง n-1
- index ไม่ใช่จำนวนเต็ม - compile error
- index เป็นจำนวนเต็มที่อยู่นอกช่วง
  - เกิด ArrayIndexOutOfBoundsException ขณะทำงาน

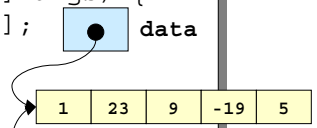
```
public class Array3 {
  public static void main(String [] args) {
    int [] data = new int[100];
    System.out.println( data[100] );
  }
}
```

```
JLab>java Array3
java.lang.ArrayIndexOutOfBoundsException
        at Array3.main(Array3.java:4)
Exception in thread "main"
JLab>
```


การส่งอาเรย์ไปยังเมท็อดอื่น

- ใช้ชื่ออาเรย์แทนทั้งอาเรย์ (ไม่มี [ ])
- การส่งอาเรย์จะไม่ได้ส่งข้อมูลของทั้งอาเรย์ไป แต่จะส่งตำแหน่ง (reference) ของอาเรย์ไปแทน

```
public static void main(String[] args) {
  double [] data = new double[5];
  ...
  double a = mean( data );
}
```




```
public static double mean(double[] x) {
  ...
}
```

ถ้าอยากรู้ขนาดของอาเรย์

- เติม `.length` ตามหลังชื่ออาเรย์

```
public class Array2 {
  public static void main(String [] args) {
    int [] data = new int[100];
    System.out.println( sum(data) );
  }

  public static int sum(int [] d) {
    int sum = 0;
    for(int i = 0; i < d.length; i++)
      sum += d[i];
    return sum;
  }
}
```


อาเรย์หลายมิติ (Multidimensional Array)

```
int [][] b = new int[3][5];
```

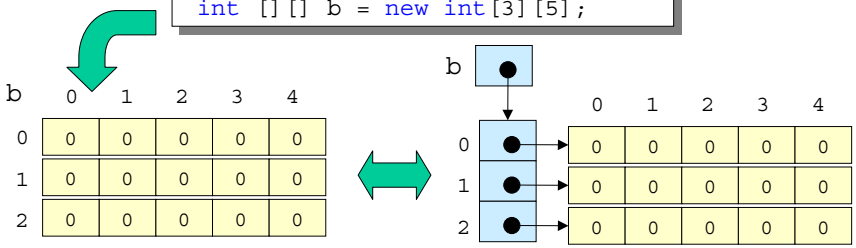
b	0	1	2	3	4
0	0	0	0	0	0
1	0	0	0	0	0
2	0	0	0	0	0

```
int [][] b = { {1, 2, 3, 4, 5},
               {5, 4, 3, 2, 1},
               {10, 20, 30, 40, 50} };
```

b	0	1	2	3	4
0	1	2	3	4	5
1	5	4	3	2	1
2	10	20	30	40	50

อาเรย์ของอาเรย์

```
int [][] b = new int[3][5];
```




```
int [][] b;
b = new int[3][];
```

```
b[0] = new int[5];
b[1] = new int[5];
b[2] = new int[5];
```


การ return ข้อมูลหลายๆ ค่า

- ใช้อาเรย์ในการคืนข้อมูลหลายๆ ตัวที่เป็นประเภทเดียวกัน (เช่นต้องการคืนรากของสมการทั้งสองตัว)

```
double [] quadraticRoot(double a,
                        double b, double c) {
  double [] root;
  double t = Math.sqrt(b * b - 4 * a * c);
  if (t > 0) {
    root = new double[2];
    root[0] = (-b + t) / (2 * a);
    root[1] = (-b - t) / (2 * a);
  } else {
    root = null;  // Oops, imaginary root
  }
  return root;
}
```

$f(x) = ax^2 + bx + c$

www.cartoonnetwork.com/ppg

- Powerpuff Girls
  - actions : kick, punch, onKicked, onPunched, duck, fly, whine, cry, ...
  - attributes : levels of power, stamina, color, ...
- Villains
  - actions : kick, punch, onKicked, onPunched, duck, fly, disintegrate, ...
  - attributes : levels of power, stamina, disgusting, ...
- Episode
  - actions : start, stop, ...
  - attributes : title, time, ...

Episode : "Bubble in Trouble"

```
class BubbleInTrouble {
  final String title = "Bubble in Trouble"
  long time = 1000;

  void start() {
    PPG bubble = new PPG("Bubble", 80, 40);
    Villain[] villians = new Villain[100];
    for (int i = 0; i < villians.length; i++) {
      villians[i] = new Villain();
    }
    for (int i = 0; i < villians.length; i++) {
      villian[i].punch(bubble);
    }
  }

}
```


PPG

```
class PPG {
  double power, stamina;
  String name;
  PPG(String n, double p, double s) {
    this.name = n;
    this.power = p, this.stamina = s;
  }
  void punch(Villain v) {
    v.onPunched(this, this.power);
  }
  void onPunched(Villain v, double p) {
    this.power -= p / this.stamina;
    if (this.power <= 0)
      this.cry();
    else
      this.kick(v);
  }
  ...
```


Villain

```
class Villain {
  double power, stamina;
  Villain() {
    this.power = 1 + 20*Math.random();
    this.stamina = 1 + 10*Math.random();
  }
  void punch(PPG ppg) {
    ppg.onPunched(this, this.power);
  }
  void onPunched(PPG ppg, double p) {
    this.power -= p / this.stamina;
    if (this.power <= 0)
      this.disintegrate();
    else
      this.fly(Math.random());
  }
  ...
```

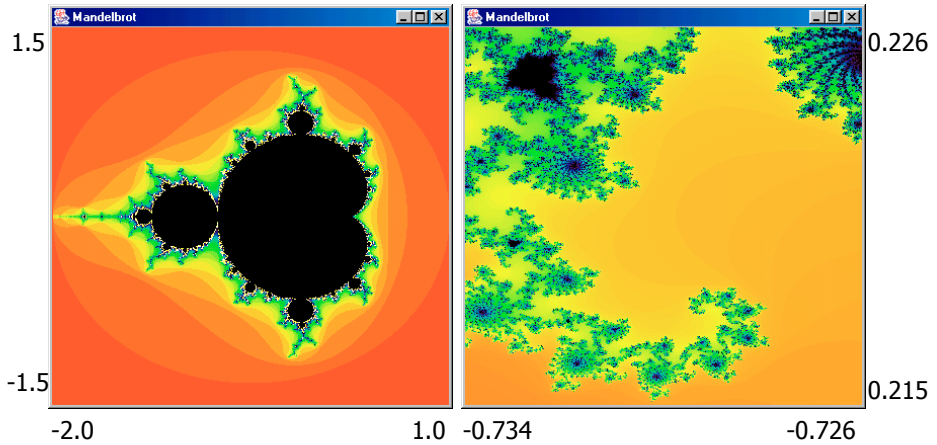
องค์ประกอบของคลาส

```
class Robot {
```

- data members
- constructor methods
- class & object methods

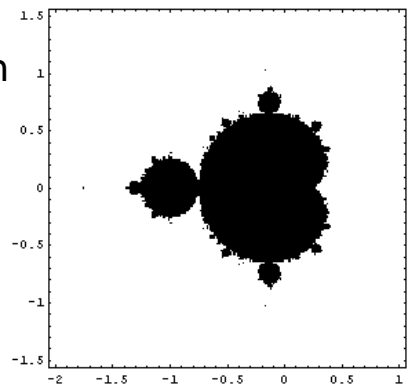
```
}
```


Mandelbrot Set



Mandelbrot Set

- ให้ลำดับของจำนวนเชิงซ้อน d_n คำนวณได้จาก $d_n = (d_{n-1})^2 + c$ โดยที่ $d_0 = 0+i0 = 0$ และ c เป็นจำนวนเชิงซ้อน
- c เป็นสมาชิกของ Mandelbrot set ถ้าลำดับของ $|d_n|$ ลู่เข้าหาค่าคงตัว



แกน x แทนเส้นจำนวนจริง แกน y แทนเส้นจำนวนจินตภาพ
จุดดำในรูปคือสมาชิกของ Mandelbrot set


```
public void paint(Graphics g) {
  int n;
  Complex c, d;
  double stepX = 3.0 / getSize().width;
  double stepY = 3.0 / getSize().height;
  for (double y = -1.5; y <= 1.5; y += stepY) {
    for (double x = -2.0; x <= 1.0; x += stepX) {
      c = new Complex(x, y);
      d = new Complex(0, 0);
      n = 0;
      while (d.abs() < 2 && n++ < 256) {
        d = d.multiply(d).add(c);
      }
      if (n >= 256) {
        g.drawRect((int)((x + 2) / stepX),
                   (int)((y + 1.5) / stepY), 1, 1);
      }
    }
  }
}
```

$|d_n| < 2$

$d_n = (d_{n-1})^2 + c$

อยากได้จำนวนเชิงซ้อน

- จาวามีแต่จำนวนจริง จำนวนเต็ม
- แต่ไม่มีจำนวนเชิงซ้อน ต้องออกแบบเอง
- เขียน class ใหม่ชื่อ Complex
- เขียน class = เขียนนิยามของประเภทข้อมูลแบบใหม่
- เก็บจำนวนเชิงซ้อนหนึ่งจำนวน = เก็บจำนวนจริงสองจำนวน $z = x + iy$

```
public class Complex {
  public double  x;
  public double  y;
}
```

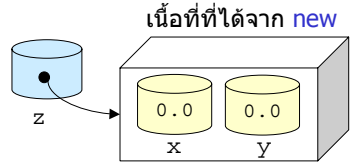
fields
(data members)

การประกาศและการสร้าง

```
public class Complex {
  public double  x;
  public double  y;
}
```

นิยามของคลาส

```
Complex z;
z = new Complex();
```



`Complex z;`
เป็นการประกาศและสร้างตัวแปร z ซึ่งมีไว้อ้างอิงเนื้อที่เก็บข้อมูลที่เป็นแบบ Complex

`z = new Complex();`
เป็นการสร้างเนื้อที่เก็บข้อมูลที่เป็นแบบ `Complex`

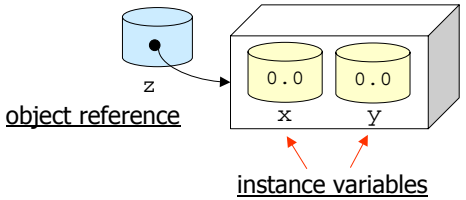
ศัพท์เทคนิค

class ชื่อ Complex

```
public class Complex {
  public double  x;
  public double  y;
}
```

fields (หรือ data members) ของ class

ตัวแปรอ้างอิง object ของ class Complex

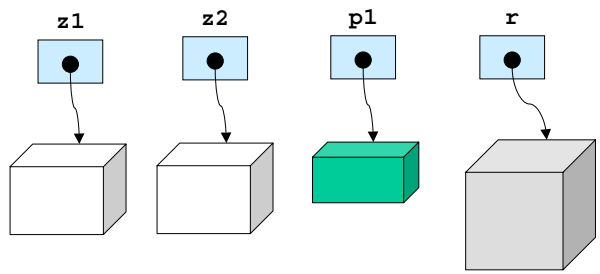


object หนึ่ง (หรือ instance) ของคลาส Complex

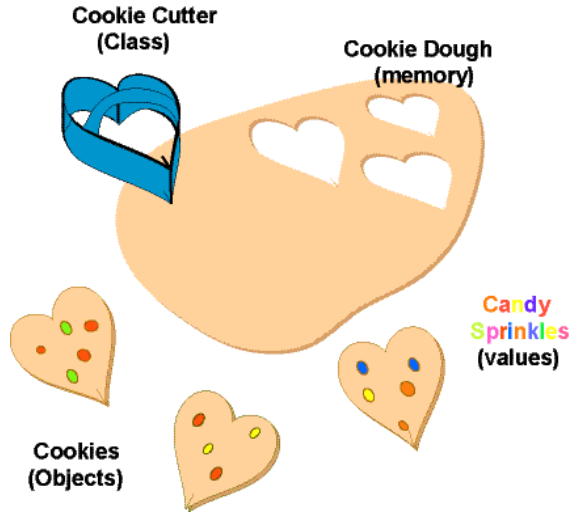
new เป็นเสมือนโรงงานสร้าง objects

- new จะสร้าง object ใหม่ของ class ที่กำหนดให้
- ต้องมีตัวแปรคอยรับตำแหน่งอ้างอิง object ใหม่

```
Complex   z1 = new Complex();
Complex   z2 = new Complex();
Point     p1 = new Point();
Rectangle r  = new Rectangle();
```



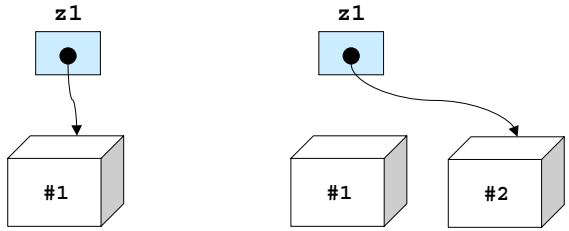
คุกกี้



ขยะ

- Object ที่ถูกสร้างขึ้น แต่ไม่มีตัวแปรใดอ้างอิง
- เนื้อที่สูญเปล่า
- ระบบจาวาจะ "เก็บขยะ" นำเนื้อที่เหล่านี้ไปใช้ใหม่

```
Complex   z1 = new Complex();  // #1
...
z1 = new Complex();            // #2
```

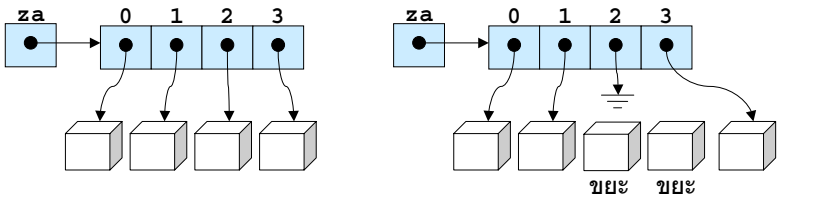


การสร้างและทำลาย object

- สร้างโดยใช้ new
- ทำลายโดยอย่าให้มีตัวแปรใดไปอ้างอิงถึง

```
Complex [] za  = new Complex[4];
for(int i=0; i< za.length; i++) {
  za[i] = new Complex();
}
za[2] = null;
za[3] = new Complex();
```

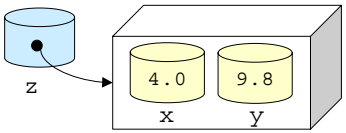
null เป็นค่าคงตัวพิเศษแทน object reference ที่ไม่ได้อ้างอิงอะไร



การใช้ Instance Variables

- instance variables คือตัวแปรต่างๆ ภายในของ object ที่ได้ถูกสร้างขึ้น
- อยากจะใช้ ให้เขียนในรูป *object* . *variableName*

```
public class Complex {
  public double  x;
  public double  y;
}
```




```
  Complex z = new Complex();

  z.x = 4;
  z.y = 9.8;
```

Class Methods vs. Object Methods

- ออกแบบเมท็อดให้ผู้อื่นเรียกใช้ได้สองวิธี
  - class methods *className* .*methodName*(...)

```
double a = Complex.abs(z1);
Complex z3 = Complex.add(z2, z1);
```

  - object methods *objectName* .*methodName*(...)

```
double a = z1.abs();
Complex z3 = z2.add(z1);
```


Class Methods

```
public class Complex {
  public double  x;
  public double  y;

  public static double abs( Complex z ) {
    return Math.sqrt( z.x * z.x + z.y * z.y );
  }
}
```

บรรยายคลาส Complex

```
public class Test {
  public static void main( String [] args ) {
    double a = JLabIO.readDouble("real part : ");
    double b = JLabIO.readDouble("imag part : ");
    Complex z = new Complex();
    z.x = a;
    z.y = b;
    System.out.println( Complex.abs(z) );
  }
}
```

สร้าง object และใช้ method ของ Complex

Object Methods

```
public class Complex {
  public double  x;
  public double  y;

  public static double abs( ) {
    return Math.sqrt(this.x*this.x + this.y*this.y);
  }
}
```

ตัดคำว่า static ทิ้ง ก็หมายถึง object method

this ใช้อ้างถึง object ที่ถูกเรียกให้มาทำงานที่เมท็อด

```
public class Test {
  public static void main( String [] args ) {
    double a = JLabIO.readDouble("real part : ");
    double b = JLabIO.readDouble("imag part : ");
    Complex z = new Complex();
    z.x = a;
    z.y = b;
    System.out.println( z.abs() );
  }
}
```


ย้อนอดีต

- String และ array เป็น object
- String มีเมท็อด length()
- array มี instance variable ชื่อว่า length

```
String s = "Hi Ho Hi Ho;
int a = new int[100];

System.out.println( s.length() );
System.out.println( a.length );
```

Constructor Methods

- เป็นเมท็อดพิเศษของคลาส ซึ่งระบบเรียกใช้โดยอัตโนมัติหลัง new คลาสนั้น
- มีไว้เพื่อตั้งค่าเริ่มต้นให้กับ instance variables ของ object ใหม่ซึ่งเพิ่งถูกสร้าง
- เป็นเมท็อดพิเศษ
  - ชื่อต้องเหมือนชื่อ class
  - ต้องไม่ระบุ return type ของเมท็อด

```
public class Complex {
  public double x, y;

  public Complex( ) {
    this.x = 0.0;
    this.y = 0.0;
  }
```


Constructor Methods

- Overload ได้ด้วยการกำหนดรายการของพารามิเตอร์ที่แตกต่างกัน

```
public class Complex {
  public double x, y;

  public Complex( ) {
    this.x = 0.0;
    this.y = 0.0;
  }
  public Complex( double x1, double y1 ) {
    this.x = x1;
    this.y = y1;
  }
  public Complex( Complex z ) {
    this.x = z.x;
    this.y = z.y;
  }
```

```
Complex z1 = new Complex();
Complex z2 = new Complex(2, 3.5);
Complex z3 = new Complex( z2 );
```


Constructor Methods

- Constructor เรียกกันเองได้ โดยใช้ this

```
public class Complex {
  public double x, y;

  public Complex( ) {
    Complex(0, 0);          // WRONG
  }
  public Complex( double x1, double y1 ) {
    this
    this
  }
```

```
public class Complex {
  public double x, y;

  public Complex( ) {
    this(0, 0);             // CORRECT
  }
  public Complex( double x1, double y1 ) {
    this.x = x1;
    this.y = y1;
  }
```


เรื่องน่ารู้

- Utility class คือคลาสที่มีแต่ class methods (เช่น `Math.*`, `Arrays.*`, `JLabIO.*`)
- คลาสไม่จำเป็นต้องมีเมท็อด main เมื่อเป็นคลาสที่ผู้ใช้ไม่ได้เรียกใช้งานโดยตรง
- ถ้าคลาสไม่มี constructor ระบบจะเติมให้แบบไม่มีการรับพารามิเตอร์
- ตั้งชื่อคลาสเป็น noun ตั้งชื่อเมท็อดเป็น verb
- ตัด this. ทิ้งก็ได้

```
public class Complex {
  public double x, y;
  ...
  public boolean equals( Complex z ) {
    return x == z.x && y == z.y;
  }
```

คำถาม

```
public Complex {
  ...
  public Complex sqr( ) {
    return                          ;
  }
  ...
}
```

จงเขียนเมท็อดยกกำลังสองในคลาส Complex โดยการใช้เมท็อด multiply

แบบฝึกหัด : อยากได้คลาส "Rational"

- Rational เป็นคลาสเก็บเลขตรรกยะ
  - ต.ย. 3/4, 15/7, 8/1
- มีสอง fields : numerator และ denominator
  - ต.ย. 15/7 มี numerator เป็น 15 และ denominator เป็น 7
- มีเมท็อดดังนี้
  - add, subtract, multiply, divide, และ equals
- ควรเก็บแบบลดรูป เช่น 4/8 ควรเก็บเป็น 1/2

```
public class Rational {
  public int numerator;
  public int denominator;
  ...
}
```