Initialization and Cleanup

Vittayasak Rujivorakul

Lecture 4

Initialization and Cleanup

Vittayasak Rujivorakul

Lecture 4

Class VS Object

- **Class:**
  - พิมพ์เขียว (Blueprint) หรือ Template ที่ใช้ในการสร้าง object
  - กำหนดลักษณะของ object ในส่วนของ data members และ methods ที่ใช้ได้กับ object นั้น หรือกล่าวคือการสร้าง data type ชนิดใหม่ขึ้นนั่นเอง

```
class Circle   {
      double  x , y ;       / /   coordinates of the center
      double radius ;       / /   radius
                  return  0.5 * Math. PI * radius * radius ;
      }
}
```


Class VS Object (cont.)

- **Object :** Instance ของ class
  - จาก class ที่สร้างขึ้น สามารถที่จะสร้าง object ใหม่ของ class นั้น

    Circle c ;
    - เป็นการประกาศตัวแปร c ซึ่งเป็นชื่อที่ใช้ในการอ้างถึง circle object
    - c ไม่ใช่ object โดยตัวมันเอง
  - การสร้าง object จะทำโดยคำสั่ง new

    Circle c ;

    c = new Circle ( ) ;
    - ซึ่งทำการสร้าง **instance** ของ **Circle class** หรือ **Circle object** โดยที่มีตัวแปร c เป็นตัวที่ใช้ระบุหรืออ้างถึง object นั้น

การสร้าง object กับ new

Circle r = new Circle ( ) ;

- การเรียก new Circle ( ) ทำให้เกิดการจองเนื้องที่ในหน่วยความจำพร้อมกับกำหนดค่าเริ่มต้น (Initialization) ให้กับ Object ใหม่ที่กำลังสร้างขึ้น โดยมีขั้นตอน
  - ทำการจองเนื้อที่ให้กับ object พร้อมกันนั้นจะทำการกำหนดค่าเริ่มต้นให้เป็นค่า default (Default Initialization)
  - ทำการกำหนดค่าเริ่มต้นของ data members (fields) ตามที่ระบุใน class (Explicit initialization)
  - ทำการ execute Constructor method

Constructor

- ชื่อของ method ต้องเหมือนกับชื่อของ class
- Constructor จะต้องประกาศโดยไม่มีการคืนค่า : เป็น method พิเศษที่ไม่ใช่ void และไม่ใช่ method ที่ return ค่าใด ๆ
- **Syntax :**
  <modifier> <class name> (<parameter>) { . . . }\
- **ตัวอย่าง**

```
class Circle {
      double x, y ; // coordinate of the center
      double radius ; // radius
      public Circle ( ) {
            x = 0.0 ; y = 0.0 ; radius = 0.0 ;
      }
}
```


ชนิดของ Constructors

- **Default Constructor** = No-Argument Constructor = constructor ที่ไม่มี parameter

  public ClassName ( ) { <statements> }

  - default constructor หากไม่มีการกำหนด constructor อื่นใด จะถูกกำหนด implicitly ใน class- - ถูกเรียกใช้ได้ (อัตโนมัติ) โดยไม่ต้องประกาศ
- **Constructor ที่มี parameter** อื่น
  - ใช้กำหนดconstructor ที่มีการรับ arguments เพื่อนำมาใช้ในกำหนดค่าเริ่มต้นให้กับ object ได้

  public ClassName (<parameter list>) { <statements> }

Method Overloding

- สามารถกำหนด constructor > 1 methods โดยที่มี parameter list แตกต่างกัน หรือที่เรียกว่าการทำ Method Overloading
- **Overloading** = การกำหนด method หรือ operator ชื่อเดียวกันแต่มีลักษณะของ arguments ต่างกัน
  - ทำให้ method ชื่อเดียวกันสามารถถูกใช้กับ data ต่าง ๆ ชนิดกัน
  - โดยแต่ละ method สามารถแยกออกจากกันได้จาก จำนวน, ชนิด และ ลำดับของ arguments ที่ส่งผ่านให้กับ method
  - ใช้ได้กับ method ทั่วไป ไม่เฉพาะแต่ constructor เท่านั้น

ตัวอย่าง

```
// 1) Default Constructor
public ClassA ( ) { . . . }
// 2) จำนวน parameter ต่างกับ 1
public ClassA (int x) { . . . }
// 3) ชนิดของ parameter ต่างจำนวนกับ 1 และต่างชนิดจาก 2
public ClassA (float x) { . . . }
// 4) Constructor ข้างล่างมีชนิดของ parameter และจำนวนเท่ากัน ถือเป็น
   invalid
public ClassB (int x, int y) { . . . }
public ClassB {int a, int b} { . . . } // invalid constructor
```

Overloading โดย primitive

- Primitive สามารถจะถูก implicitly promoted จาก data type ที่เล็กกว่าเป็น data type ขนาดที่ใหญ่กว่าโดยอัตโนมัติ
- ตัวอย่าง :

```
public Circle (double a, double b, double c) {
  . . . System. out. println ("method 1") ; . . .
}
public Circle (int a, int b, int c) {
  . . . System. out. Println ("method 2") ; . . .
}
สร้าง Circle โดย :
Circle c = new Circle (0, 0, 20.0) ;
```

กรณีนี้ method 1 จะถูกเรียกใช้งาน

Default Constructors

- Default Constructor จะถูกสร้างโดยอัตโนมัติหากนักพัฒนาไม่กำหนด Constructor ใด ๆ ขึ้นใน class
  - **กรณีที่มีการกำหนด Constructor ใด ๆ เอง Compiler จะไม่ทำการสร้าง default Constructor ให้**
  - ตัวอย่าง :

```
class Circle {
    double x, y, radius ;
    public Circle (int a, int b, int c) { . . . }
/
หากสร้าง Circle c โดย
Circle c = new Circle ( ) ;
```

Compilation error:
No constructor matching Circle()
found in class Circle

"this'

- **this** : ใช้ในการอ้างถึง ตัวมันเองภายในตัว object (explicit reference ถึงตัว object เอง)
  - เพื่อทำการอ้างถึง data members หรือ methods ใน object
  - เพื่อทำการอ้างถึง object เอง

```
class Circle {
    double x, y, radius ;
    public Circle (int x, int y, int r) {
        this.x = x ; / / reference to data members
        this.y = y ;
        this.radius = r ;
        }
        :
    }
```


ตัวอย่าง : this

```
/ / simple use of "this" keyword
public class Circle {
        double x, y, radius ;
        Circle expand (int i) {
                radius += I ;
                radius this ;
        }
        void print ( ) {
                System. out. println ("radius = " + this. radius);
        }
        public static void main (String args [ ]) {
                Circle c = new Circle ( ) ;
                c. expand (1) .expand (1) .expand (2) .print ( ) ;
        }
}
```

สามารถทำ operations หลายอย่าง (หรือหลายครั้ง) บน object เดียวกันได้

การเรียก Constructors จาก Constructors

- การเรียก constructor อื่นใน class เดียวกัน ทำได้โดยการใช้เรียก this โดยมี arguments ตรงกับ constructor อื่นที่ต้องการเรียกนั้น
  - ไม่สามารถทำการเรียก constructor โดยใช้ this 2 ครั้งซ้อนได้

```
public Circle ( )  {
        this (0.0) ;
        this (1, 1) ; / / invalid use of this :  call  > 1
}
```

  - ไม่สามารถทำการเรียก this เพื่ออ้างในแบบ constructor นอก constructor method

```
public void print  ( )  {
        this  (0) ; / /  invalid  "this" :  outside constructor . . .
}
```


ตัวอย่าง

```
Public class Circle  {
      double  x,  y,  radius ;
      public Circle  (double  x, double  y, double  r)  {
            this.x = x ;
            this.y = y ;
            radius  =  r ;
      }
      public Circle  ( )  {
            this  (1.0) ;  / / call below constructor
      }
      public Circle  (double radius)  {
            this (0.0 ,  0.0 ,  radius) ;  / / call top constr.
      }
}
```


ความหมายของ static

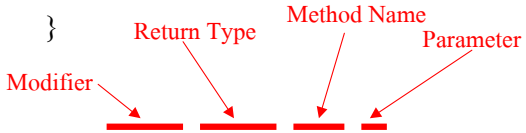
- การประกาศให้ fields หรือ methods เป็น static เป็นการทำให้ members นั้น หรือ methods เป็น class fields หรือ class methods
- ความหมายคล้ายกับการทำให้เป็น global functions (หรือ fields)
  - non-static method = instance method = ประกาศโดยไม่มี modifier static
  - static method = class method = ประกาศโดยมี modifier static
- ไม่สามารถเรียก non-static methods/fields จากภายใน static method (**Class method *ไม่*** access **instance methods/fields**)

การใช้งาน class methods/fields

- Class fields ใช้เพื่อกำหนด property ของ class ซึ่งทำให้ค่าใน field นี้เหมือนกันหมดในทุก object ของ class นั้น
  - ตัวอย่างเช่น เพื่อเก็บ version ของ class, counter ของ class
- Class method อาจใช้ในการกำหนด method ที่ไม่ต้องทำกับ instance fields หรือไม่ต้องเรียก instance methods
  - ตัวอย่างเช่น method ใน Math class (sqrt, random, sin. ctc.) ซึ่งทำการคำนวณกับ argument ที่ส่งผ่านให้ แล้วคืนค่าผลลัพธ์

รูปแบบการกำหนด method

```
<modifiers> <return type> <method name> (<parameter list>)
{
        <statement list>
}
```




```
        public double area ( ) {
                return 3.14 * radius * radius;
        }
```

statement

Default Initialization

- Data fields ใน class จะถูก initialize ด้วยค่า default ของ data type นั้น เมื่อเริ่มต้นสร้าง object
- ตัวอย่าง

```
class Test Initial  {
    public static void main
            (String args [ ] )
    {
            member Initial  m ;
            m = new Member Initial ( ) ;
            m.print ( ) ;
    }
}
```

```
Class MemberInitial {
            boolean t;          char    c;
            byte    b;          short   s;
            int     i;          long    l;
            float   f;          double  d;

            void print( ) {
            System.out.println(
"DTATA  TYPE    INITIAL-VALUE \n"
"boolean            " + t + "\n" +
"char               " + c + "\n" +
"byte               " + b + "\n" +
"short              " + s + "\n" +
"int                " + i + "\n" +
"long               " + l + "\n" +
"float              " + f + "\n" +
"double             " + d + "\n" );
            }
}
```


Explicit Initialization

- ค่าที่กำหนดให้กับ Data fields ใน class จะเป็นค่าเริ่มต้นสำหรับ data fields ของทุก object ที่สร้างขึ้น
- ตัวอย่าง

```
class Test Explicit  {
    public static void main
            (String args [ ] )
    {
            Circle  c = new Circle ( ) ;
            c. print ( ) ;
    }
}
```

```
Class Circle {
        double  x = 10.0;
        double  y = 10.0;
        double  radius = 1.0;
        void print( ) {
            System.out.println(
            "x  coordinate:  " + x +
            "\ny  coordinate: " + y +
            "\nradius:  " +radius);
        }
}
```


Constructor Initialization

```
Public class TestConstructor{
    public static void main(String args[ ]){
            Circle c;
            c = new Circle( );
    }
}
class Name {
    public Name( String s) {
            System.out.println(
             "Name: " + s);
    }
}
```

```
Class Circle {
   // explicit  init
   Name  n = new Name("circle");
   double x,y;
   double  radius;
   public Circle( ) {
      // default init
    System.out.println("x coordinate: " +
    x + "\ny coordinate: " + y +"\nradius: "
    + radius);
     // constructor init
     x = 10.0; y = 10.0;
     radius = 1.0;
     System.out.println("x coordinate: " + x +
"\ny coordinate: " + y + "\nradius: " + radius);
    }
}
```

การ initialize ค่าสำหรับ static data

- ในกรณีที่ class มี class data members/methods (static data/method)
  - ทำการ initialize static data นั้นเพียงครั้งแรกครั้งเดียวสำกรับ class นั้น จะไม่ทำการ reinitialize ใน object ตัวต่อไปที่สร้างตามมา

```
public class TestStatic {
    public static void main (String args [ ] )  {
                Circle c1 = new Circle ( ) ;
                Circle c2 = new Circle ( ) ;
    }
}
Class Name  {
    static int counter ;
    public Name (String s)  {
                System . Out . Println ("counter : "+ counter++) ;
                System . Out . Println ("Name: " + s) ;
      }
}
```


การทำ explicit initialization สำหรับ

- Java ยอมให้มีการทำ static initialization ภายใต้ static construction clause หรือ static block ใน class
- ตัวอย่าง

```
class Unitcircle {
        double.x, y;
        static {
                double radius - 1.0;
                double circumference = 2*Math.PI / radius;
                double area = 0.5*Math. PI*radius*radius;
        }
}
```


การ initialize non-static instance

- ทำ non-static initialization ได้คล้ายกับ explicit static initialization โดยกำหนด block ของการกำหนดค่าเริ่มต้นโดยไม่ต้องมี static (ใช้กับการทำ anonymous inner classes)
- ตัวอย่าง

```
class Circles {
        Circle c1 , c2;
        // instance initialization
        {
                c1 = new Circle (10,10,10) ;
                c2 = new Circle (0,0,10) ;
                System.out.println ("Created Circle c1 & c2") ;
        }
        :
}
```


สรุปการทำ Initialization

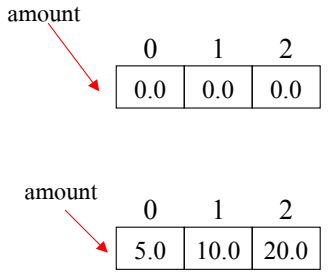
- สรุปขั้นตอนการทำ Initialization ใน Java
  - ครั้งแรกที่ class (I.e., Circle) นั้นถูกสร้าง หรือ ครั้งแรกที่ static method หรือ static fields ถูก accessed, Java interpreter ทำการหา Circle.class
  - ทำการกำหนดค่าเริ่มต้นให้กับ static (run static initializer)ซึ่งจะทำเพียงครั้งเดียว
  - เมื่อทำการสร้าง new Circle(), จะทำการจองเนื้อที่ในหน่วยความจำ
  - ทำ Default Initialization
  - ทำ Explicit Initialization
  - ทำ Constructor Initialization

Array

- Array ใน Java เป็น Object
  - การประกาศ : float [ ] amount; หรือ float amount [ ] ;
    - หากใช้ float [ ] x,y,z, หมายความว่า z,y,z, เป็น array
  - ใช้ new statement เพื่อสร้าง object: amount = new float[12];
  - ขนาดของ array ได้จาก data member length ของ array object
  - Index ของ array เริ่มต้นจาก 0 ถึง length - 1
  - Initializing Arrays เมื่อทำการสร้าง array ทุก element ใน array จะถูกกำหนดค่าเริ่มต้นให้ (e.g., array ของ char จะ set ค่า element เป็น null (\u0000) character)

ตัวอย่าง Array ของ Primitive Data Types

- ตัวอย่าง
  - double amount [ ] = new double [3];
  - amount[0] = 5.0;
  - amount[1] = 10.0;
  - amount[2] = 20.0;



- ขนาดของ array ได้จาก constant length เช่น
  - amount.length // ในกรณีค่าที่ได้ คือ 3

การสร้าง array พร้อมกับกำหนดค่า

- ทำการกำหนดค่าของ element ใน array เมื่อสร้างได้ เช่น

```
int montyInYear [ ] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 };
```

ไม่ต้องกำหนดขนาด

- ซึ่งจะมีค่าเท่ากับการทำ

```
int monthInYear [ ] ;
monthInYear  =  new int [12] ;
monthInYear  [0]   =   1 ;
:
monthInYear [11]   =   12;
```


Fixed-size Array VS Variable-size Array Declaration

- **Fixed-size Array :** รู้ขนาดของ array ในขณะ compile

```
int number [ ]  =  new int [100] ;
int [ ] amount  =  {1, 2, 3} ;
```

- **Variable-size Array:** รู้ขนาดของ array หลังจาก run

```
static int genRand (int mod)   {
        return  (int)  (Math.random ( ) *mod ) ) ;
}
int number [ ] ;
number  =  new  int [genRand (10) ] ;
```

Array ของ Object

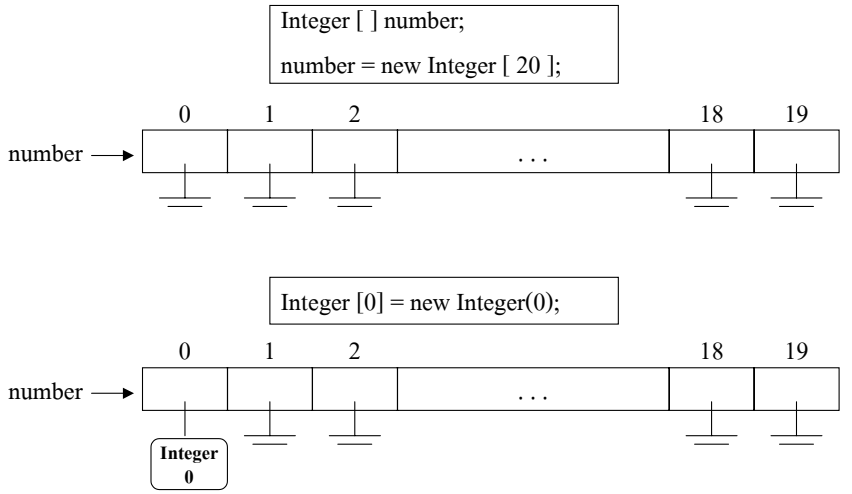
- การประกาศ และ initialize array ของ object ทำให้สร้าง array ของ object reference ซึ่งภายในมีค่าเริ่มต้นเป็น null

```
Integer [ ] number = new Integer [20] ;
```

- จึงยังไม่สามารถเรียกใช้ แต่ละ element ของ array ของ object ได้ ต้องทำการสร้าง element object แต่ละตัวก่อนโดย **new**

```
number [0] = new Integer (0) ;
   :
number [19] = new Integer (100) ;
```


Array ของ Integer (Wrapper)



การ initialize array ของ object โดย { }

```
public class ArrayInt {
    public static void main (String args [ ]) {
        // Java 1.0
        Integer [ ] a = {
            new Integer (1) ,
            new Integer (2) ,
            new Integer (3) ,
        } ;
        // Java 1.1 +
        Integer [ ] b = new Integer [ ] {
            new Integer (1) ,
            new Integer (2) ,
            new Integer (3) ,
        } ;
    }
}
```

อนุญาตให้มี , ของตัวสุดท้ายได้

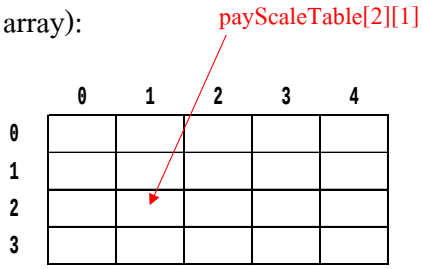
การสร้างและ initialize array ของ unknown type

```
Class A {int a ;}
public class VarArray {
    static void f (Object [ ] x) {
        for (int i = 0 ; i < x.length ; i++ ) {
            System. out. Println (x [i] ) ;
        }
    }
    public static void main (String args [ ] ) {
        f (new Object [ ] {new Integer (20) , new VarArray ( ) ,
                           new Double (1.2), new Long (10) } ) ;
        f (new Object [ ] {"aA", "bB", "cC"} ) ;
        f (new Object [ ] {new A ( ), new A ( ), new A ( ) } ) ;
    }
}
```

ทุก object ใน Java สืบทอดจาก Object จึงใช้ช่วยในการสร้าง object ใดๆ ได้

Multidimensional Array

- Java ไม่มี array ชนิดหลายมิติ แต่สามารถกำหนด array ของ array เพื่อสร้างให้เป็น array หลายมิติได้
- การประกาศ array 2 มิติ (2-dimensional array):
  - float [ ] [ ] payScaleTable ; หรือ
  - float payScaleTable [ ] [ ] ;
- สร้าง array ได้โดย
  - payScaleTable = new float [4] [5] ;



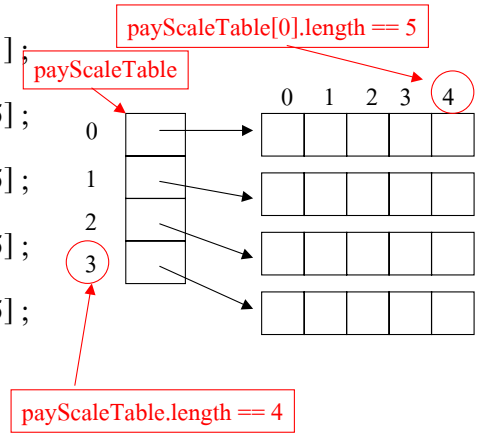
payScaleTable = new float [4] [5]

- เป็นการรวมการทำงานของขั้นตอนต่อไปนี้
  - payScaleTable = new float [4] [ ] ;
  - payScaleTable [0] = new float [5] ;
  - payScaleTable [1] = new float [5] ;
  - payScaleTable [2] = new float [5] ;
  - payScaleTable [3] = new float [5] ;
- หรือ

```
payScaleTable = new float [4] [ ] ;
for (int j = 0 ; j < 4 ; j++)
    payScaleTable [ j ] = new float [5] ;
```


การทำงานที่เกิดขึ้น

```
payScaleTable = new float [4] [ ] ;
payScaleTable [0] = new float [5] ;
payScaleTable [1] = new float [5] ;
payScaleTable [2] = new float [5] ;
payScaleTable [3] = new float [5] ;
```



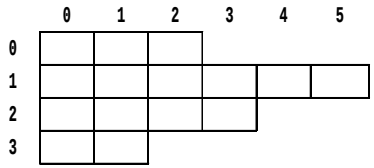
การ initialize ค่าสำหรับ multidimensional array

```
// fixed size 2dimensional 3x3array declared { }
int [ ] [ ] a = {{1, 2,3} , {4, 5, 6}} ;
// fixed size 3 dimensional 2x2x4 array
int [ ] [ ] [ ] b = new int [2] [2] [4] ;
// variable size 2 dimensional array
int [ ] [ ] c = new int [random (20)] [ ] ;
for (int i = 0 ; i < c.length ; i++) {
        c {I} = new int [random (10) ] ;
}
// array of non-primitive
Integer [ ] [ ] d = {{new Integer (1) , new Integer (2) } ,
                    {new Integer (3) , new Integer (4) } , }
```

subarray

- Subarray = array ซึ่งเป็นส่วนหนึ่งของอีก array
- ตัวอย่างก่อนหน้านี้ (payScaleTable มี 4 subarrays) ที่มีขนาดเท่ากัน
- อาจสร้าง subarray ขนาดที่ต่างกันได้ เช่น

```
mixArray = new float [4] [ ] ;
mixArray [0] = new float [3] ;
mixArray [1] = new float [5] ;
mixArray [2] = new float [4] ;
mixArray [3] = new float [2] ;
```



Cleanup : Garbage Colection

- Java ทำ automatic garbage collection, finalize () ใน Java มีไว้เพื่อช่วยในการทำการ cleanup ที่ต้องการก่อนที่จะถึงเวลา garbage collect โดย Java

  หาก object มี finalize method, มันจะถูกเรียกก่อนที่จะมีการทำ garbage collection

  - การทำงานไม่เหมือนกับ destructor ใน C + +
  - Garbage collection ไม่ใช่การทำ destruction : หากต้องมีการทำงานบางอย่างก่อนที่จะไม่ใช้ Object นั้น ผู้ใช้ต้องทำเอง ไม่เช่นนั้นอาจไม่ garbage collected ของ object นั้น
  - object อาจไม่ถูก garbage collected เลยก็ได้ : Java อาจไม่ทำ garbage collection จนสิ้นสุดการทำงานของโปรแกรมก็ได้หากเนื้อที่ยังมีมากพอ
  - Garbage collection จึงเกี่ยวข้องเฉพาะกับการคืนเนื้อที่ให้กับระบบ