



# IECS273/274 [1111 3670/3671] 物件導向設計與實習

## 05#1 OOP Fundamentals Encapsulation 2/2





03

ICES273\_04#1

► Variable & Reference

*To Be Continued*



04

ICES273\_04#1

► Immutable Objects  
and Classes

*To Be Continued*



01

## ► Static Variables and Static Methods

# ► Static Variables/Members

- A static variable is shared by all objects of the class. *The data members of some classes are not attached to the object, but are attached to the class.* This data member attached to the class should be defined as **static variable/member**.

Syntax: **static** <type> <member>;

- No matter how many objects a class has, **only one memory space** for static variables is attached to the class; *it does not change due to the number of objects.*
- Example: If **class** Student is a set students of Feng Chia University and, we want to record the school as a member of the class.
  - // General members will generate memory for each object, but all have the same value.  
String school = **new** String "Feng Chia University";
  - // Only one memory space is attached to the class.  
**static** String SCHOOL= "Feng Chia University" ;
- Example: To record the number of students, define **static int COUNT = 0;**
  - ✓ Each constructor method adds the statement COUNT++. Every time a student is created, COUNT is incremented by 1.
- For convenience, static member names use uppercase and underscores.



# ▶ Static Variables/Members

- If the static variable is declared **public**, for example the number of students is declared as **public static int COUNT=0**;
  - Program code outside the class can directly access COUNT, because the static variable is not attached to any object. Access does not need to pass through any Student object, as long as the **static** variable is preceded by the name of the class and a period. E.g.:  

```
System.out.println("Student numbers: " + Student.COUNT + " students.");
```
- A static variable is a variable that belongs to the class as a whole, and not just to one object.
- In general classes, **static** variables can be declared like members and methods.
  - However, **static** variables cannot be declared in methods, even in static methods.
  - If a class has *inner classes*, only static inner classes can *declare static variables*.

# ► Static Methods

- Class methods can also be defined as **static methods**, which is to add the keyword **static** *before the return type*. Syntax:

⟨access modifier⟩ **static** ⟨return type⟩ ⟨method name⟩ (⟨parameters⟩)  
{⟨method body⟩}

Example: **public static void** main(String[] args) {...}

- **Similarly, static methods are also *owned* by the class, but the objects can also be called.** Static methods are generally used to *operate on static variables*.
- In static methods, *non-static variables cannot be directly referenced*. Because non-static variables belong to *an instance of an object*, if the object has not been created, there is no instance variable. How can we refer to it? If you want to refer to a non-static variable, you can only use ***object reference***.

# ► Static Methods

- When a static method is defined, the keyword **static** is placed in the method header

```
public static returnType myMethod(parameters) {  
    ...  
}
```

- Static methods are invoked using the class name in place of a calling object.

```
returnedValue = MyClass.myMethod(arguments);
```

- A static method is one that can be used without a calling object.
- A static method still belongs to a class, and its definition is given inside the class definition.



# ► Static Methods

- The most common static method is method `main()`. Because the main method must be called before all other objects are created, `main()` can only be called through the class. In `main()`, *non-static variables cannot be directly referenced*.

```
public class StaticMethod {  
    private int age = 0 ;  
    private static String VENDOR = "ABC";  
  
    public static void main(String[] args) {  
        System.out.println(age);  
        System.out.println(VENDOR);  
    }  
}
```

*non-static variable, compiler error*  
*static variable, OK*

```
public class StaticMethod {  
    private int age = 0 ;  
    private static String VENDOR = "ABC";  
  
    public void show() {  
        System.out.println("Non-Static");  
    }  
  
    public static void staticShow() {  
        System.out.println("Static");  
    }  
  
    public static void main(String[] args) {  
        System.out.println(VENDOR);  
        staticShow();  
        StaticMethod staticMethod = new StaticMethod();  
        System.out.println(staticMethod.age);  
        staticMethod.show();  
    }  
}
```

*non-static method*  
*static method*  
*static method*  
*static variable, direct reference*  
*static method, direct call*  
*non-static variable/method, referenced and called via an object.*

```

public class demo {
public static void main(String[] args) {
    Rectangle rect1 = new Rectangle();
    System.out.println("The number of rectangle after rect1 is " +
        rect1.numberOfObjects);

    Rectangle rect2 = new Rectangle(5,6);
    System.out.println("The number of rectangle after rect2 is " +
        rect2.getNumberOfObjects());
}
}

class Rectangle {
    static int numberOfObjects; // Static Variable
    double height = 1;          // The height and width of rectangle
    double width = 1;
    Rectangle() {                // Construct a rectangle object
        numberOfObjects++;
    }
    Rectangle(double newHeight, double newWidth) {
        numberOfObjects++;
        height = newHeight;
        width = newWidth;
    }
    static int getNumberOfObjects() { // Static Method
        return numberOfObjects;
    }
    //return height;
    //java.lang.Error: Unresolved compilation problem:
    //Cannot make a static reference to the non-static field height
}
}

```

## Static.class

The number of rectangle after rect1 is 1

The number of rectangle after rect2 is 2

The number of rectangle after rect1 is 1

Exception in thread "main" java.lang.Error: Unresolved compilation problem:  
Cannot make a static reference to the non-static field height

at Rectangle.getNumberOfObjects([demo.java:33](#))  
at demo.main([demo.java:9](#))

### Rectangle

height : double

width : double

NumberOfObjects: int

getNumberOfObjects(): int

getArea(): double

### rect1: Rectangle

height = 1   Width = 1

NumberOfObjects: 2

### rect2: Rectangle

height = 5   Width = 6

NumberOfObjects: 2

After two Circle Objects were *instantiated*,  
numberOfObjects is 2.

# ▶ Static Block Statement

■ A **block statement** is a sequence of statements enclosed with a pair of { }. A block statement in Java can also be preceded by the keyword **static**, which is called a **static block statement**.

- The static block statement is *only executed when the class is loaded*, because the class is loaded only once and *the static block statement is executed only once*.
- Static block statement is often used to initialize static variables of class members.
- Non-static block statement is executed every time an object is created.
- Static block statements also have their limitations.
  - ✓ For example, the code size of static blocks *cannot exceed a certain value specified by the JVM*, the keywords **this** and **super** *cannot be used*, and *values cannot be returned from static blocks*.

```
public class StudentCount {  
    private static String COLLEGE;    // Private data member  
    private static int COUNT;  
    // Static block statement  
    static {                          // Initialize private data member  
        COLLEGE = new String("Feng Jia University");  
        COUNT = 0;  
        System.out.println("Static block code, only executed once  
when the class is loaded. " + COLLEGE + "=" + COUNT);  
    }                                // Executed only once  
    // Non-static block statement  
    {  
        COUNT++;  
        System.out.println("Non-static block code, which is  
executed every time an object is created." +  
COLLEGE + "=" + COUNT);            // Executed 3 times  
    }  
}
```

StudentCountApp.java

```
public class StudentCountAPP {  
  
    public static void main(String[] args) {  
        new StudentCount();    // Create three objects  
        new StudentCount();  
        new StudentCount();  
    }  
  
}
```



StudentCountApp.class

Satic block code, only executed once when the class is loaded.

Feng Jia University=0

Non-static block code, which is executed every time an object is created.Feng Jia University=1

Non-static block code, which is executed every time an object is created.Feng Jia University=2

Non-static block code, which is executed every time an object is created.Feng Jia University=3

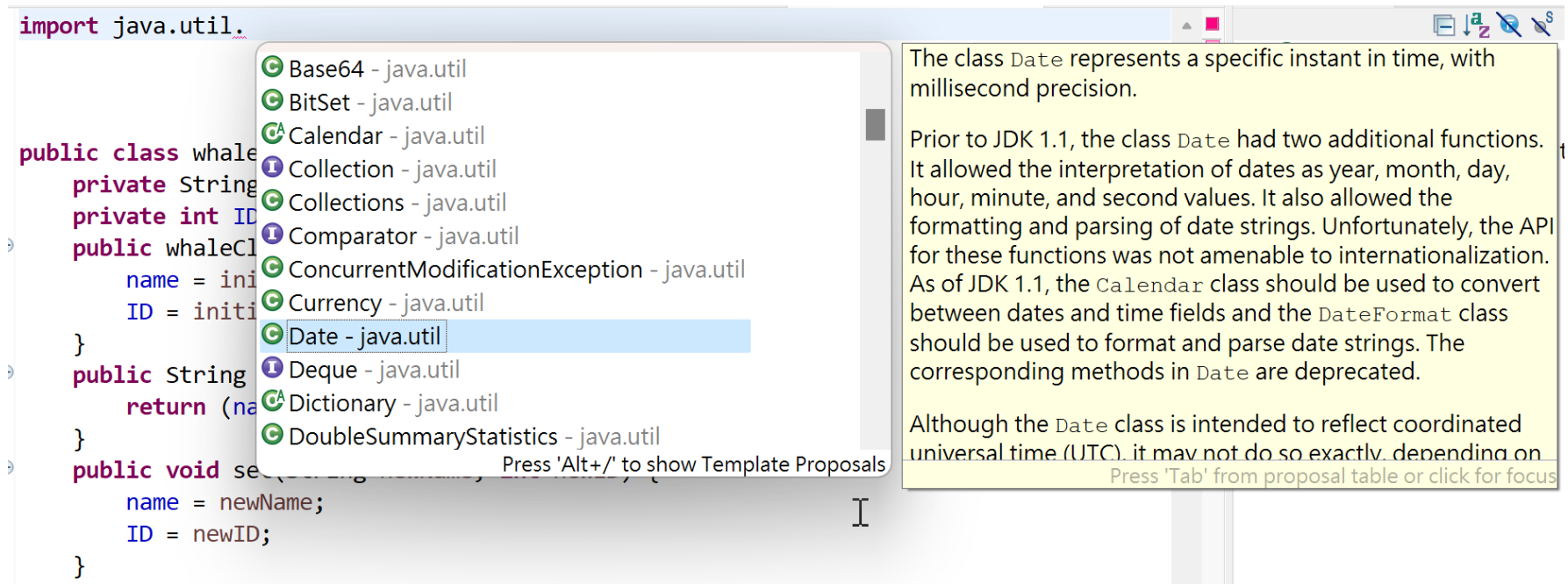


# 02

► Using Classes from the Java Library

# Java APIs

- The Java API contains a rich set of classes for developing Java programs.



```
import java.util.  
  
public class whale  
    private String  
    private int ID  
    public whaleC  
        name = ini  
        ID = initi  
    }  
    public String  
        return (na  
    }  
    public void se  
        name = newName;  
        ID = newID;  
    }
```

- Base64 - java.util
- BitSet - java.util
- Calendar - java.util
- Collection - java.util
- Collections - java.util
- Comparator - java.util
- ConcurrentModificationException - java.util
- Currency - java.util
- Date - java.util**
- Deque - java.util
- Dictionary - java.util
- DoubleSummaryStatistics - java.util

Press 'Alt+/' to show Template Proposals

The class `Date` represents a specific instant in time, with millisecond precision.

Prior to JDK 1.1, the class `Date` had two additional functions. It allowed the interpretation of dates as year, month, day, hour, minute, and second values. It also allowed the formatting and parsing of date strings. Unfortunately, the API for these functions was not amenable to internationalization. As of JDK 1.1, the `Calendar` class should be used to convert between dates and time fields and the `DateFormat` class should be used to format and parse date strings. The corresponding methods in `Date` are deprecated.

Although the `Date` class is intended to reflect coordinated universal time (UTC), it may not do so exactly, depending on

Press 'Tab' from proposal table or click for focus

# ▶ Java.util.Date

```
1 import java.util.Date;
2
3 public class demo {
4     public static void main(String[] args) {
5         Date date = new Date();
6         System.out.println("The elapsed time since Jan 1, 1970 is " +
7             date.getTime() + " milliseconds");
8         System.out.println(date.t);
```

Converts this Date object to a String of the form:

dow mon dd hh:mm:ss zzz yyyy

where:

- dow is the day of the week (Sun, Mon, Tue, Wed, Thu, Fri, Sat).
- mon is the month (Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, Dec).
- dd is the day of the month (01 through 31), as two decimal digits.
- hh is the hour of the day (00 through 23), as two decimal digits.
- mm is the minute within the hour (00 through 59), as two decimal digits.

Press 'Tab' from proposal table or click for focus

- toGMTString() : String - Date
- toLocaleString() : String - Date
- toString() : String - Date
- toInstant() : Instant - Date
- after(Date when) : boolean - Date
- compareTo(Date anotherDate) : int - Date
- getDate() : int - Date
- getDay() : int - Date
- getHours() : int - Date
- getMinutes() : int - Date
- getMonth() : int - Date
- getSeconds() : int - Date

Press 'Alt+/' to show Template Proposals

The elapsed time since Jan 1, 1970 is 1664350581283 milliseconds  
Wed Sep 28 15:36:21 CST 2022

# ▶ Java.util.Random

- You can use the *java.util.Random* to generate a random int, long, double, float, and boolean value. Another way, you have used *Math.random()* to obtain a random double value between 0.0 and 1.0 (excluding 1.0).

```
1 import java.util.Random;
2
3 public class demo {
4     public static void main(String[] args) {
5         Random generator1 = new Random(3);
6         System.out.print("From generator1: ");
7         for (int i = 0; i < 10; i++)
8             System.out.print(generator1.nextInt(1000) + " ");
9         Random generator2 = new Random(3);
10        System.out.print("\nFrom generator2: ");
11        for (int i = 0; i < 10; i++)
12            System.out.print(generator2.nextInt(1000) + " ");
13    }
14 }
```

Returns an effectively unlimited stream of pseudorandom double values, each between zero (inclusive) and one (exclusive).

A pseudorandom double value is generated as if it's the result of calling the method [nextDouble\(\)](#).

Specified by: [doubles\(\)](#) in [RandomGenerator](#)

Returns:  
a stream of pseudorandom double values

Since:  
1.8

Impl Note:  
This method is implemented to be equivalent to `doubles(Long.MAX_VALUE)`.

- [doubles\(\)](#) : DoubleStream - Random
- [doubles\(long streamSize\)](#) : DoubleStream - Random
- [doubles\(double randomNumberOrigin, double randomNumberBound\)](#) : DoubleStream - Random
- [doubles\(long streamSize, double randomNumberOrigin, double randomNumberBound\)](#) : DoubleStream - Random
- [equals\(Object obj\)](#) : boolean - Object
- [getClass\(\)](#) : Class<?> - Object
- [hashCode\(\)](#) : int - Object
- [ints\(\)](#) : IntStream - Random
- [ints\(long streamSize\)](#) : IntStream - Random
- [ints\(int randomNumberOrigin, int randomNumberBound\)](#) : IntStream - Random
- [ints\(long streamSize, int randomNumberOrigin, int randomNumberBound\)](#) : IntStream - Random

Press 'Alt+/' to show Template Proposals

From generator1: 734 660 210 581 128 202 549 564 459 961  
From generator2: 487 92 474 424 506 605 854 691 722 321



03

► Array of Object



# ▶ Single-Dimensional Arrays

- An *array* is a data structure used to process a collection of data that is all of the same type, such as a list of numbers of type double or a list of strings.
- A single array variable can reference a large collection of data.
  - Once an array is created, its size is fixed.
  - Array type is an ordered set of elements of the same type.
  - An array reference variable is used to access the elements in an array using an index.
  - The number of indexed variables in an array is called the length or size of the array.
  - To use an array in a program, you must declare a variable to reference the array and specify the array's element type.
  - Syntax - declare:  

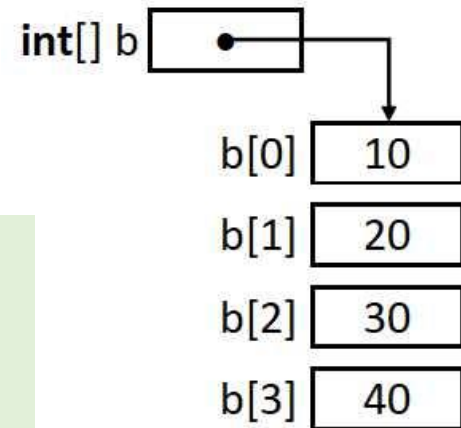
***elementType[] arrayRefVar;***  
or ***elementType arrayRefVar[];***
  - Syntax - create  

***arrayRefVar = new elementType[arraySize];***  
or ***elementType arrayRefVar[] = new elementType[arraySize];***

# ► Single-Dimensional Arrays


- `int [] a;`
  - ✓ Declare an integer array `a`.
  - ✓ When the program is executed, for the declared array variable `a`, the system only allocates a memory space pointing to integer elements, but no memory space are allocated for integer elements.
  - ✓ Variable `a` is a **one-dimensional array**.
- An array can have definition, initialization, and assignment written together.
  - ✓ `int [] b = {10, 20, 30, 40};`

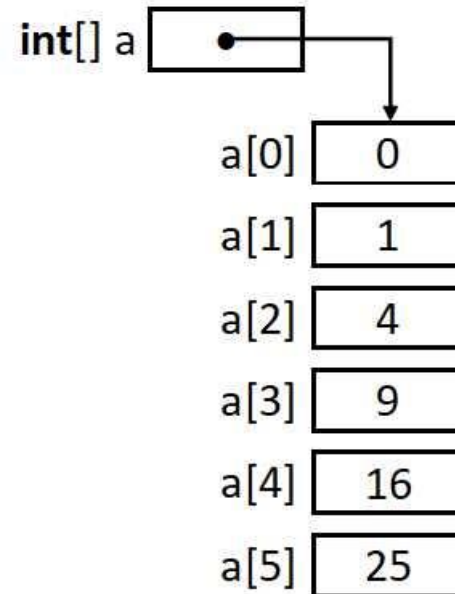
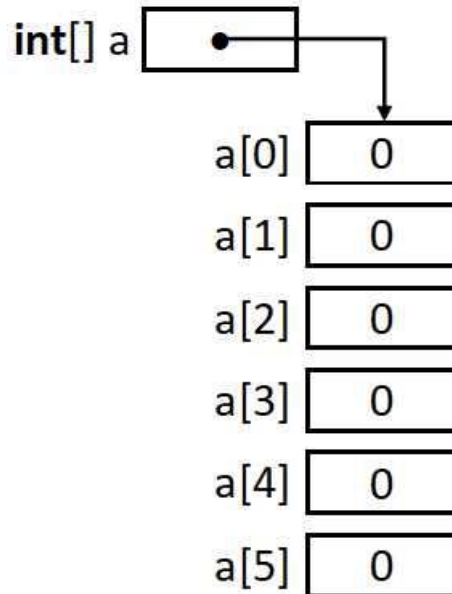
```
double[] score = new double[8];  
int[] lotte = new int[200];  
char[] name = new char[80];  
Person[] student = new Person[100];
```



# ► Single-Dimensional Arrays

- `a = new int [6];`
  - ✓ The system allocates six integer memory locations with an initial value of 0.
  - ✓ Integer values that can be assigned to array elements:  
for (i=0; i<6; i++) {a[i] = i \* i;}

int[] a 





## SingleArray.class

Enter 5 numbers:

20 30 50 40 10

The highest number is 50.0

The numbers are:

20.0 differs from max by 30.0

30.0 differs from max by 20.0

50.0 differs from max by 0.0

40.0 differs from max by 10.0

10.0 differs from max by 40.0

The average of numbers is 30.0

# ▶ Method

- **Method:** a function in object-oriented programming is called a method and function call is called message passing. Syntax:

***⟨method⟩ ::= ⟨modifier⟩ ⟨return type⟩ ⟨identifier⟩  
(⟨type⟩ ⟨parameter⟩, ..., ⟨type⟩ ⟨parameter⟩) {⟨statement⟩;}***

- **⟨modifier⟩** : access modifier (public/private/protected) 、 static modifier (static) 、 final modifier (final).
- **⟨return type⟩**: The data type of the value returned at the end of the message passing.
- **⟨parameter⟩**: a variable name, the data to receive when the message is passed.
- **⟨statement⟩** ; The program code of the method execution, including the return statement to return the data of the execution result.

factorial function: →

Iterative Function	Recursive Function
<pre>static int factorial(int n) {     int i, result = 1;     for (i=n; n&gt;0; n--) result *= i;     return result; }</pre> <p>for loop</p>	<pre>static int factorial(int n) {     if (n&lt;=1) { return 1; }     else { return n * factorial(n-1); } }</pre> <p>recursion, call <i>factorial</i> itself</p>



```
import java.util.Scanner;

public class SelctionSort {

    // The selection sort method for sorting the numbers
    public static void selectionSort(double[] list) {
        for (int i = 0; i < list.length - 1; i++) {
            // Find the minimum in the list[i..list.length-1]
            double currentMin = list[i];
            int currentMinIndex = i;
            for (int j = i + 1; j < list.length; j++) {
                if (currentMin > list[j]) {
                    currentMin = list[j];
                    currentMinIndex = j;
                }
            }

            // Swap list[currentMinIndex] if necessary
            if (currentMinIndex != i) {
                list[currentMinIndex] = list[i];
                list[i] = currentMin;
            }
        }
    }
}
```

```
public static void main(String[] args) {  
    double[] b = {8.2, 6.5, 10.1, 54.8, 3.3, 21.5, 4.2, 18.3,  
                  15.6, 32.6};  
    System.out.println("Before sorting:");  
    int i;  
    for (i = 0; i < b.length; i++)  
        System.out.print(b[i] + " ");  
    System.out.println();  
  
    selectionSort(b);  
  
    System.out.println("After sorted:");  
    for (i = 0; i < b.length; i++)  
        System.out.print(b[i] + " ");  
    System.out.println();  
}  
}
```

## Selection Sort.class

Before sorting:

8.2 6.5 10.1 54.8 3.3 21.5 4.2 18.3 15.6 32.6

After sorted:

3.3 4.2 6.5 8.2 10.1 15.6 18.3 21.5 32.6 54.8

# ► Enumerated Types

- An **enumerated type** is a type for which you give all the values of the type in a typically short list. A value of an enumerated type is a kind of named constant and so, by convention, is spelled with all uppercase letters.

Syntax:

***enum Type\_Name {FIRST\_VALUE,SECOND\_VALUE, ..., LAST\_VALUE};***

```
enum Days {MON, TUE, WED, THU, FRI, SAT, SUN};
```

```
enum Flavor {CHOCOLATE, VANILLA, STRAWBERRY};
```

```
enum Berry {BLUEBERRY, RASPBERRY, STRAWBERRY};
```

```
import java.util.Scanner;

public class test{
    enum Month {JAN, FEB, MAR, APR, MAY, JUN, JUL, AUG,
                SEP, OCT, NOV, DEC};
    public static void main(String[] args){
        Month[] mon = Month.values();
        Scanner keyboard = new Scanner(System.in);
        int days = 0, sum = 0;
        for (int i = 0; i < mon.length; i++) {
            System.out.print("Enter days worked for " + mon[i] + " ");
            days = keyboard.nextInt();
            sum += days;
        }
        System.out.println("Total days work = " + sum);
    }
}
```

## Enumerated.class

```
Enter days worked for JAN 22
Enter days worked for FEB 20
Enter days worked for MAR 22
Enter days worked for APR 22
Enter days worked for MAY 23
Enter days worked for JUN 21
Enter days worked for JUL 24
Enter days worked for AUG 24
Enter days worked for SEP 21
Enter days worked for OCT 22
Enter days worked for NOV 21
Enter days worked for DEC 22
Total days work = 264
```



# ► Multi-Dimensional Arrays

- **Multi-dimensional arrays** are declared and created in basically the same way as one-dimensional arrays. You simply use as many square brackets as there are indices.

Syntax:

***Base\_Type [ ] . . . [ ] Variable\_Name = new Base\_Type [Length\_1]...[Length\_n];***

- Data in a table or a matrix can be represented using a two-dimensional array. An element in a two-dimensional array is accessed through a row and a column index.
- The instance variable length does not give the total number of indexed variables in a two-dimensional array.
  - ✓ `char[ ][ ] page = new char[80][120];`
    - `page.length` is equal to 80
    - `page[0].length` is equal to 120
  - ✓ `matrix = new int[2][3];`
    - `matrix[1][2] = 6;`
    - `matrix[0][1] = 4;`

```
double[][] table = new double[60][20];  
int[][][] figure = new int[12][24][36];  
char[][] page = new char[80][120];  
Student[][] = new Student[20][80];
```

	[0]	[1]	[2]
[0]	0	4	0
[1]	0	0	6

# Multi-Dimensional Arrays

- Arrays can be multi-dimensional, and the number of elements in the same dimension can vary.

➤ **int [][] m = new int[3][4];**

- ✓ The first dimension has 3 rows and 4 elements in each row.

✓ **int [][] s = new int[2][]**

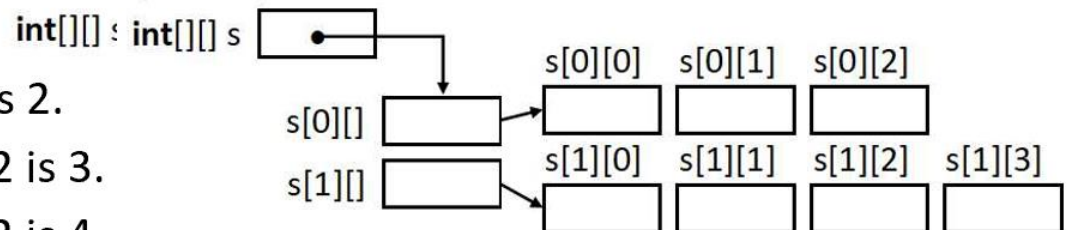
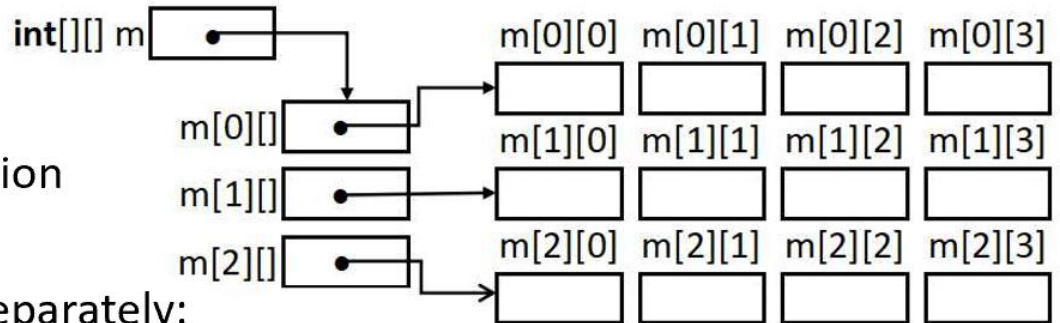
The first dimension has 2 rows,  
the length of the second dimension  
is undetermined。

- ✓ Set the length of dimension 2, separately:

**s[0] = new int[3];**

**s[1] = new int[4];**

- ✓ s.length: length of dimension 1 is 2.
- ✓ s[0].length: row 1 of dimension 2 is 3.
- ✓ s[1].length: row 2 of dimension 2 is 4.



```
import java.util.Scanner;

public class test{
    public static void main(String[] args){
        int[][] matrix = new int[10][10];
        int row = 0, column = 0, total = 0;
        // Initializing arrays with random values
        for (row = 0; row < matrix.length; row++) {
            for (column = 0; column < matrix[row].length; column++) {
                matrix[row][column] = (int)(Math.random() * 100);
            }
        }
        // Printing arrays
        for (row = 0; row < matrix.length; row++) {
            for (column = 0; column < matrix[row].length; column++) {
                System.out.print(matrix[row][column] + " ");
            }
            System.out.println();
        }
        // Summing all element
        for (row = 0; row < matrix.length; row++) {
            for (column = 0; column < matrix[row].length; column++) {
                total += matrix[row][column];
            }
        }
        System.out.println("The sum of all elements is " + total);
    }
}
```

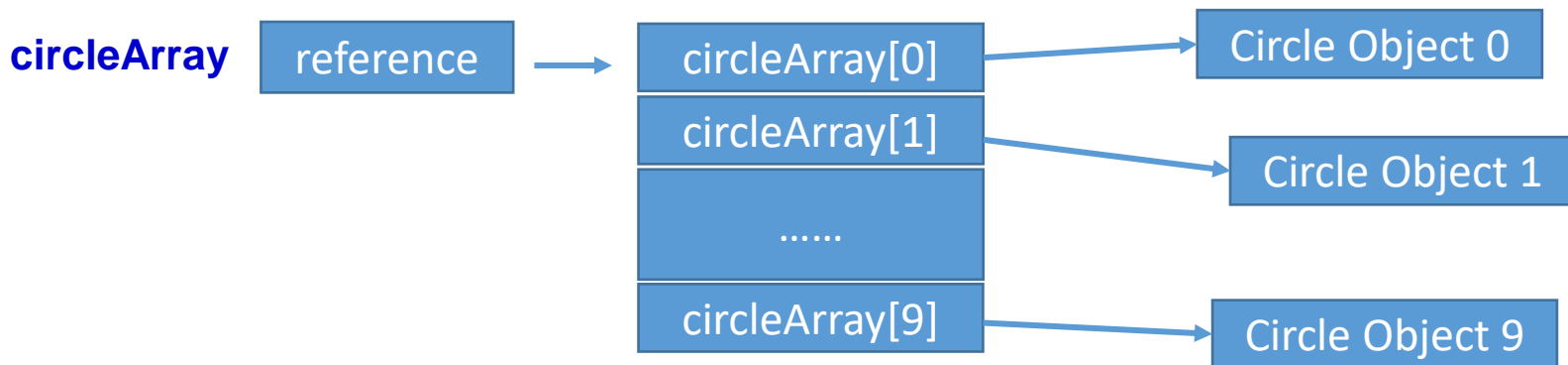
MultiArray.class

```
74 27 57 86 60 0 43 65 12 11
24 50 70 59 14 50 86 63 5 98
99 61 63 54 97 52 98 83 95 38
17 69 34 10 35 9 34 1 61 49
65 19 36 84 20 51 56 3 31 24
31 95 45 18 93 18 3 4 64 71
98 81 85 86 57 20 11 26 88 83
46 41 72 36 59 7 0 66 15 55
81 44 26 86 18 67 26 22 6 81
35 69 72 62 32 32 60 1 48 12
The sum of all elements is 4756
```

# ► Array of Objects

- An array can hold objects as well as primitive-type values.
- An array of objects is actually an array of reference variables.
- When an array of objects is created using the new operator, each element in the array is a reference variable with a default value of null.

```
Circle[] circleArray = new Circle[10];  
  
for (int i = 0; i < circleArray.length; i++) {  
    circleArray[i] = new Circle();  
}
```



*In an array of objects, an element of the array contains a reference to an object.*

```
public class demo {
    public static void main(String[] args) {
        Circle[] circleArray;
        circleArray = createCircleArray();
        printCircleArray(circleArray);
    }

    /** Create an array of Circle objects */
    public static Circle[] createCircleArray() {
        Circle[] circleArray = new Circle[5];
        for (int i = 0; i < circleArray.length; i++) {
            circleArray[i] = new Circle(Math.random() * 100);
        }
        return circleArray;
    }

    /** Print an array of circles and their total area */
    public static void printCircleArray(Circle[] circleArray) {
        System.out.printf("%-30s%-10s\n", "Radius", "Area");
        System.out.println("-----");
        for (int i = 0; i < circleArray.length; i++) {
            System.out.printf("%-30f%-10f\n",
                circleArray[i].getRadius(), circleArray[i].getArea());
        }
    }
}
```

```
System.out.println("-----");  
// Compute and display the result  
System.out.printf("%s% f\n", "The total area of circles is",  
    sum(circleArray));
```

```
}
```

```
/** Add circle areas */
```

```
public static double sum(Circle[] circleArray) {
```

```
    // Initialize sum
```

```
    double sum = 0;
```

```
    // Add areas to sum
```

```
    for (int i = 0; i < circleArray.length; i++)
```

```
        sum += circleArray[i].getArea();
```

```
    return sum;
```

```
}
```

```
}
```

ArrayOfObject.class

Radius	Area
82.834559	21556.239736
31.889209	3194.753538
52.809309	8761.346107
37.082176	4319.965737

The total area of circles is 45736.979516





# 04 ▶ Package & javadoc

# ► Packages and Import Statements

- Java uses **packages** to form libraries of classes.
- A package is a group of classes that have been placed in a directory or folder, and that can be used in any program that includes an **import statement** that names the package.
  - The import statement must be located at the beginning of the program file:  
Only blank lines, comments, and package statements may precede it
  - The program can be in a different directory from the package.
- The package *java.lang* contains the classes that are fundamental to Java programming. It is imported automatically, so no import statement is needed.
- All the classes in the current directory belong to an unnamed package called the *default package*. As long as the current directory (.) is part of the *CLASSPATH* variable, all the classes in the default package are automatically available to a program.

# ► Import Statements

- You can use a class from a package in any program or class definition by placing an **import statement** that names the package and the class from the package at the start of the file containing the program (or class definition). The program (or class definition) need not *be in the same directory* as the classes in the package.

## SYNTAX

```
import Package_Name.Class;
```

## EXAMPLE

```
import java.util.Scanner;
```

- You can import all the classes in a package by using an asterisk in place of the class's name.

## SYNTAX

```
import Package_Name.*;
```

## EXAMPLE

```
import java.util.*;
```

# ▶ javadoc

- Java has a program called **javadoc** that automatically extracts the interface from a class definition and produces documentation.
  - This information is presented in HTML format, and can be viewed with a Web browser
  - If a class is correctly commented, a programmer need only refer to this API documentation in order to use the class
  - javadoc can obtain documentation for anything from a single class to an entire package
- The **javadoc** program *extracts class headings, the headings for some comments, and headings for all public methods, instance variables, and static variables.*
  - The comment must immediately precede a public class or method definition, or some other public item.
  - The comment must be a block comment `/** ... */`
  - The comment preceding a public method definition should include descriptions of parameters, any value returned, and any exceptions that might be thrown. This type of information is preceded by the @ symbol and is called an @ tag.

```
/**  
General Comments about the method . . .  
@param aParameter Description of aParameter  
. . .  
*/
```



# 05 ▶ Programming Practice

# ► Practice 1 : Magic Square

- A magic square of degree  $n$ , where  $n$  is an odd number greater than 3, is an  $n \times n$  integer square matrix; the element values of this magic square are  $1, 2, 3, \dots, n^2$ , and the sums of all rows, all columns, the diagonal, and the anti-diagonal are the same, i.e.  $(1+n^2)n/2$ .

For example, the following are magic squares for  $n=3$  and  $n=5$ .

$n=3$	6	1	8
	7	5	3
	2	9	4

$n=5$	15	8	1	24	17
	16	14	7	5	23
	22	20	13	6	4
	3	21	19	12	10
	9	2	25	18	11

Write a Java program that reads in an odd integer  $n$  from 3 to 19 and constructs a magic square of degree  $n$ . The algorithm for constructing a magic square of degree  $n$  is as below:

- Fill 1 into the top middle cell,
- After filling in  $k$ , fill in  $k+1$  according to the following steps until  $n^2$  cells are filled,
- Meanwhile, the  $n \times n$  square matrix is a cyclic structure connected as a torus in the horizontal and vertical directions,
  - ✓ If there is no integer in the upper left cell of  $k$ , then fill that cell with  $k+1$ ;
  - ✓ If the top left cell is occupied, then fill  $k+1$  into the cell below  $k$ .
- Also, add up the sum of each row, each column, the diagonal and the anti-diagonal to verify that the result is a magic square.

# ► Practice 1 : Magic Square

Enter the size of the magic square ( $3 \leq n < 20$ , odd integer): 3

The magic square of degree 3:

6	1	8
7	5	3
2	9	4

The magic square passes verification.

The sum of each row, each column, diagonal, and anti-diagonal is 15.

Enter the size of the magic square ( $3 \leq n < 20$ , odd integer): 7

The magic square of degree 7:

28	19	10	1	48	39	30
29	27	18	9	7	47	38
37	35	26	17	8	6	46
45	36	34	25	16	14	5
4	44	42	33	24	15	13
12	3	43	41	32	23	21
20	11	2	49	40	31	22

The magic square passes verification.

The sum of each row, each column, diagonal, and anti-diagonal is 175.

## ► Practice 2 : Matrix Class

- Write a Java program that defines an  $m \times n$  matrix, and addition, subtraction, and multiplication operations of two matrices. To add ( $A+B$ ) and subtract ( $A-B$ ) two matrices  $A$  and  $B$ ,  $A$  and  $B$  must have the same number of rows and columns. To multiply two matrices ( $A \times B$ ), the number of columns of  $A$  must equal to the number of rows of  $B$ .

*Assuming the values of the matrix elements are between -50 and 50, use a random number generator to generate the values of the matrix elements.*

Define  $A$ ,  $B$ ,  $C$ ,  $D$ , and  $E$  to be five matrices of  $5 \times 4$ ,  $5 \times 4$ ,  $5 \times 4$ ,  $4 \times 6$ , and  $5 \times 6$ , respectively.

Test the following matrix expressions:

- $A + B - C$
- $B \times D + E$
- $E - A \times D$

In **package Matrix** program source code: **Matrix.Java** and application class **MatrixAPP.Java**.



# ► Practice 2 : Matrix Class

Matrix A:

-44	0	-50	-35
45	-10	12	-28
5	31	3	19
15	25	-8	-41
0	-43	26	44

---

Matrix B:

-35	40	-16	10
-4	15	-5	-44
1	5	42	-5
-11	18	-32	46
-38	-33	-34	31

---

Matrix C:

-35	-47	7	-34
27	21	39	-1
-42	36	21	46
-34	-40	24	13
-27	-30	-4	-34

---

Matrix D:

5	-49	50	-12	-7	39
3	-43	-13	-15	-11	-16
-32	-43	-22	-44	49	21
-24	-28	21	38	25	46

---

# ► Practice 2 : Matrix Class

Matrix E:

-42	-21	-30	23	-25	23
5	31	29	26	3	26
-50	-9	-17	-34	1	45
-15	-37	-41	-40	-17	-4
13	-41	23	9	-38	48

-----  
Resulting Matrix of A+B-C:

-44	87	-73	9
14	-16	-32	-71
48	0	24	-32
38	83	-64	-8
-11	-46	-4	109

-----  
Resulting Matrix of B\*D+E:

175	382	-1738	927	-754	-1858
1246	1029	-1180	-1603	-1479	-2499
-1254	-1939	-1061	-2159	1872	656
-96	-184	845	2978	-556	723
68	3834	-49	3634	-300	-194

-----  
Resulting Matrix of E-A\*D:

-2262	-5307	1805	-1375	2992	4399
-478	1538	-1499	2008	320	-853
384	2230	-197	-99	-245	-591
-1405	281	219	1721	1780	1865
2030	460	-888	-1164	-2885	-3210

-----



# 06 ▶ Assignment #2

# ► Assignment #2: Complex Number

- The **complex number**  $a+bi$ , contains two real numbers, the *real part*  $a$  and the *imaginary part*  $b$ . Write a Java program that defines **class Complex** of complex numbers. The class contains three constructors:
  - Default constructor, set value to  $0.0+0.0i$ .
  - Constructor of one real number parameter,  $r$ , set value to  $r+0.0i$ .
  - Constructor of two real number parameters,  $r$  and  $s$ , set value to  $r+si$ .

There are five complex number arithmetic operations:

- Complex addition:  $(a+bi)+(c+di) = (a+c)+(b+d)i$
- Complex subtraction:  $(a+bi)-(c+di) = (a-c)+(b-d)i$
- Complex multiplication:  $(a+bi) \times (c+di) = (a \times c - b \times d) + (a \times d + b \times c)i$
- Complex division:  $(a+bi) \div (c+di) = ((a \times c + b \times d) + (-a \times d + b \times c)i) \div (c^2 + d^2)$
- Complex absolute value:  $|a+bi| = (a^2 + b^2)^{1/2}$

Also, there are five supporting methods:

- ① Get real part of a complex number: `getRe()`
- ② Get imaginary part of a complex number: `getIm()`
- ③ Set real part of a complex number: `setRe(r)`
- ④ Set imaginary part of a complex number: `setIm(s)`
- ⑤ Print a complex number: `printComplex()`

# ► Assignment #2: Complex Number

- Create **package ComplexNumber** for your project and use file name HW2\_DXXXXXXXX.java for class and HW2APP\_DXXXXXXXX.java for the application program where DXXXXXXXX is your student ID..
  - ① **Write a program to demonstrate the above arithmetic operations of two input complex numbers.**
  - ② **And use class Complex to write an application program to solve a quadratic equation and verify the two roots.** In your output, print a real numeral four digits after the decimal point. When verify the two roots, consider the precision error up to six digits after the decimal point, i.e., the absolute value the result of substituting a root to the quadratic equation is less than 0.000001.
- Write **comments** in your program solution. Also, write a **report** to explain how you develop your assignment solution using HW2RPT\_DXXXXXXXX.pdf.

# ► Assignment #2: Complex Number

- ① Write a program to demonstrate the above arithmetic and supporting operations of two input complex numbers.

```
Enter the first complex number a + bi = 1 2
Enter the second complex number c + di = 3 4
```

The complex numbers are  $1 + 2i$  and  $3 + 4i$

After complex number arithmetic operations:

=====

Complex addition:  $4+6i$

Complex subtraction:  $-2-2i$

Complex multiplication: .....

# ► Assignment #2: Complex Number

- ② And use class Complex to write an application program to solve a quadratic equation and verify the two roots.

Enter coefficient a: 4

Enter coefficient b: -6

Enter coefficient c: 2

The two roots of quadratic equation  $4.0000X^2 - 6.0000X + 2.0000 = 0.0000$  are:  
1.0000 and 0.5000

Verification of the two quadratic equation roots PASSES.

-----

Enter coefficient a: 5

Enter coefficient b: 7

Enter coefficient c: 9

The two roots of quadratic equation  $5.0000X^2 + 7.0000X + 9.0000 = 0.0000$  are:  
 $-0.7000 + 1.1446i$  and  $-0.7000 - 1.1446i$

Verification of the two quadratic equation roots PASSES.