Future Technology for Everyday Life

Unlocking the World of Java Programming…..

Prof. (Dr.) Priti Srinivas Sajja

Visit pritisajja.info for detail

An introduction and basic java programming concepts as specified in Unit 2, PGDCA PS02CDCA33:Object Technology, S P University.

Unit 2: Course Content

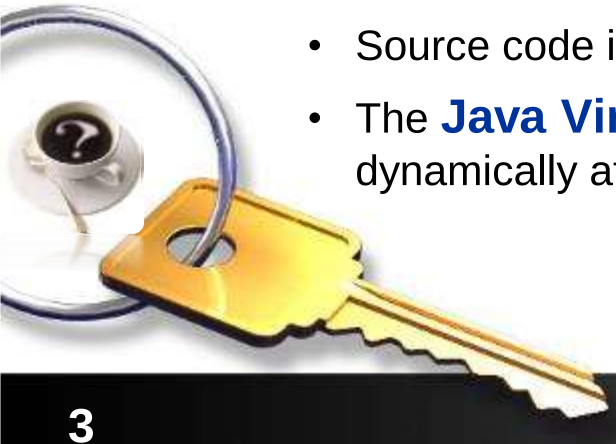
Basic Java Programming Concepts

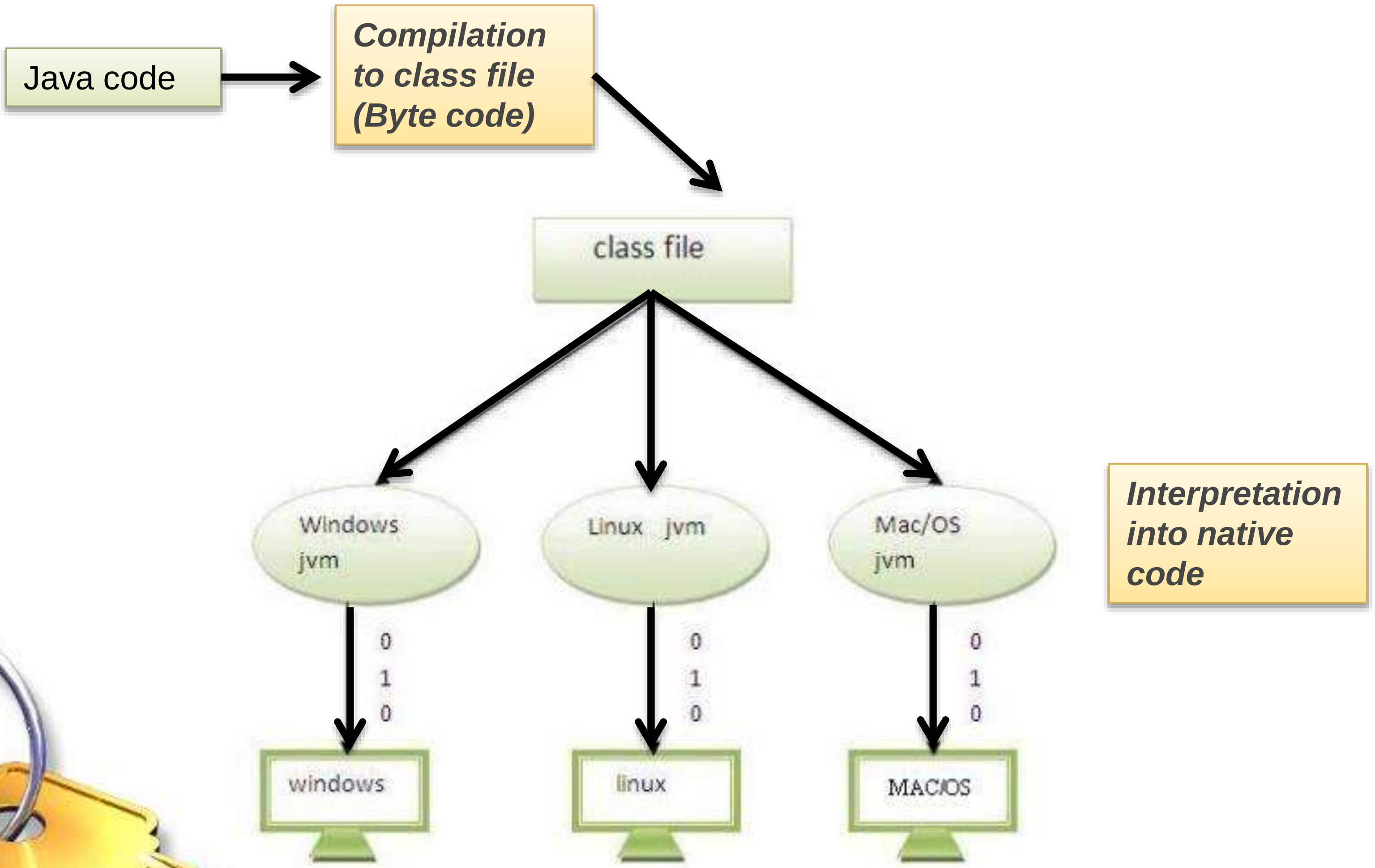
- Structure of Java Program
- Concept of Bytecodes and platform independence
- Primitive Data Types, Variable Names, Scope, Operators, Expressions,
- Control Flow Statements
- Arrays

Schildt H. : The Complete Reference Java 2, 5th Edition, McGraw-Hill / Osborne, 2002

What is Java?

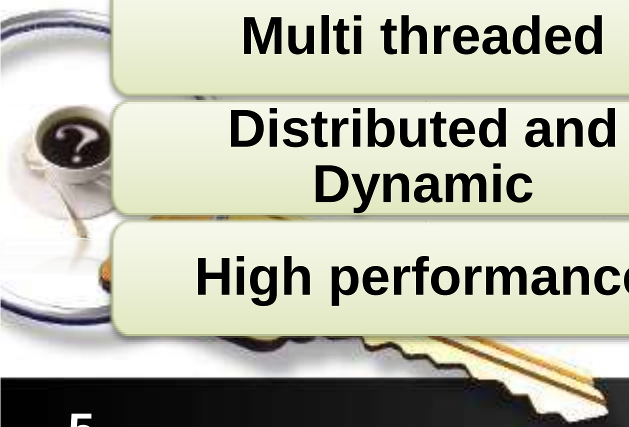
- Java is **object-oriented** with built in **Application Programming Interface** (API)
- It has borrowed its syntax from C/C++
- Java **does not have pointers** directly.
- **Applications** and **applets** are available.
- Java is *both* ***compiled*** *and* ***interpreted****.*
  - Source code is compiled to **bytecode**.
  - The **Java Virtual Machine (JVM)** loads and links parts of the code dynamically at run time (late or dynamic binding).




Java code
Compilation to class file (Byte code)
class file
Windows jvm
Linux jvm
Mac/OS jvm
Interpretation into native code
0 1 0
0 1 0
0 1 0
windows
linux
MAC/OS

Features of Java:

Simple	• To follow
Secure	• Remote applets are not trusted and not allowed to use local resources
Object-oriented	• Supports advantages of OOA
Platform independent and Architecture Neural	• Independent form hardware and software platforms
Interpreted	• It is complied also and interpreted also.
Robust	• Java is strong, replacing pointer by reference and provides automatic memory management
Multi threaded	• Supports concurrent procedures
Distributed and Dynamic	• Supports dynamic binding and links parts of code at the time of execution.
High performance	• Java provides native language support



Structure of java program: Hello World Application

Step 1: Write java code

```
package hello;
/**
The HelloWorld class implements an application that simply
    displays  "Hello World!" to the standard output (console)
*/
public class HelloWorld
    {
    public static void main (String args[])
            {
                System.out.println("Hello world!");
            } // end of main

    }// end of class
```

Output: Hello World!

Prototype of the main method

public static void main (String args[])

- **public** is the **access specifier**.
- **static** is the **storage class**.
- **void** is the **return type**.
- String **args[ ]** is an **array of arguments.**

Check public static void main( ) ? Will it cause any error? If yes, what?

About main method…

- **Several main methods** can be defined in a java class.
- The **interpreter will look for a main method with the prescribed signature** as the entry point.
- A method named main, which has some **other signature** is of **no particular significance**. It is like any other method
- in the class.
- Therefore, if the main method is not declared correctly, the application will not execute. There may not be any compilation problem.
- This class will compile correctly, but will not execute. The interpreter will say

In class NoMain: void main (String argv[]) is not defined

Is it true?

- **The argument to the mandatory main function**

  public static void main (String args[])

  which is String args []

- **can also be written as**

  String [] args

Comments

There are three types of comments defined by Java.

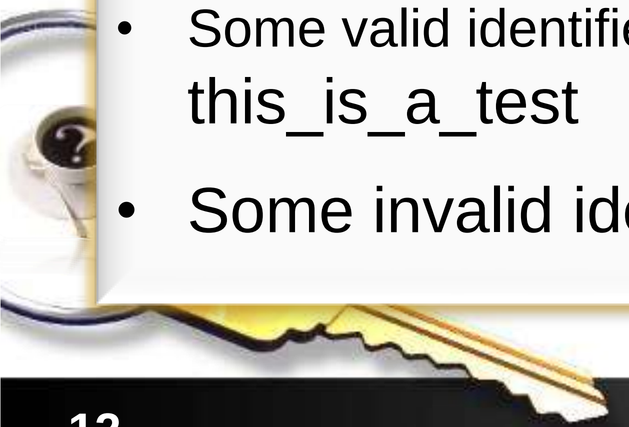
1. **Single-line comment :**Java single line comment starts from **//** and ends till the end of that line.
2. **Multiline comment:** Java multiline comment is between **/* and */.**
3. **Documentation comment :** Documentation comment is used to produce an HTML file that documents your program. The documentation comment begins with a **/** and ends with a */.**

Naming Conventions

- Java distinguishes between **UPPER and lower case** variables.
- The convention is to **capitalize the first letter of a class** name.
- The name of the **constructor is the same as the name of the class**.
- All **keywords** (words that are part of the language and cannot be redefined) **are written in lower case**.

Identifiers

- Identifiers are used for **class names, method names, and variable names.**
- An identifier may be **any sequence of uppercase and lowercase letters, numbers, or the underscore and dollar-sign** characters.
- Identifiers must **not begin** with a **number**.
- Java Identifiers are **case-sensitive.**
- Some valid identifiers are ATEST, count, i1, $Atest, and this_is_a_test
- Some invalid identifiers are 2count, h-l, and a/b



Operators

Java operators can be grouped into the following four groups:

- **Arithmetic,**
- **Bitwise,**
- **Relational, and**
- **Logical.**

Arithmetic Operators

The operands of the arithmetic operators must be of numeric type. You cannot use arithmetic operators on boolean types, but you can use them on char types.

Operator	Description
+	Addition
-	Subtraction (unary minus)
*	Multiplication
/	Division
%	Modulus
++	Increment
--	Decrement

Relational Operators

The relational operators determine the relationship between two operands.

Operator	Description
==	Equal to
!=	Not equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

Boolean Logical Operators

Operator	Result
&	Logical AND
|	Logical OR
!	Logical unary NOT
==	Equal to
!=	Not equal to
? :	Ternary if-then-else

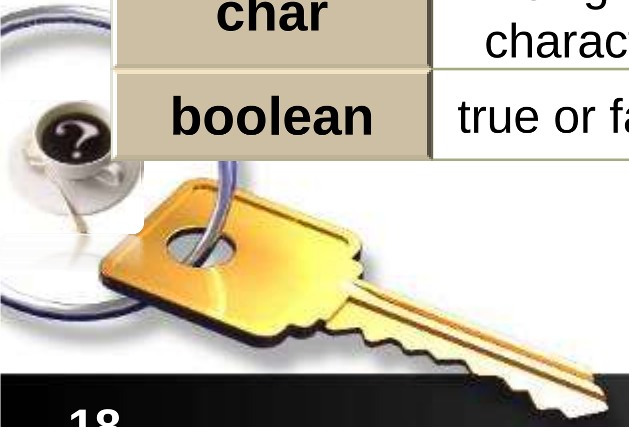
```
int num1=10;
int num2=20;
int max;
max= (num1>num2) ? num1 : num2;
System.out.println(max);
```

Data Types

- **Three kinds of data types are supported by Java.**
  - **primitive** data types
  - **reference** data types
  - the special **null** data type
    {*that is we may write if (obj!= null)*}

Primitive Data Types in Java

Type	Kind	Memory	Range
byte	integer	1 byte	-128 to 127
short	integer	2 bytes	-32768 to 32767
int	integer	4 bytes	-2147483648 to 2147483647
long	integer	8 bytes	-9223372036854775808 to -9223372036854775807
float	floating point	4 bytes	±3.40282347 x 10^{38} to ±3.40282347 x 10^{-45}
double	floating point	8 bytes	±1.76769313486231570 x 10^{308} to ±4.94065645841246544 x 10^{-324}
char	single character	2 bytes	all Unicode characters
boolean	true or false	1 bit	



There is no unsigned integer in java.

Try this….

```
/** This program demonstrates how Java
* adds two integers. */
public class BigInt
{
    public static void main(String args[])
    {
    int a = 2000000000; //(9 zeros)
    int b = 2000000000;
    System.out.println ( "This is how Java adds integers");
    System.out.println ( a + "+" + b + " = " + (a+b) );
    } // end of main

}// end of class
```

Output:

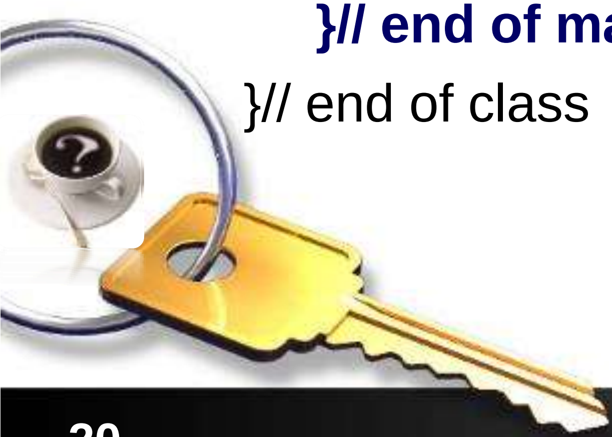
```
This is how Java adds integers
2000000000 + 2000000000 = -294967296
```

Try this….

```
public class Significant
{
    public static void main (String args[])
    {
    final float PI = 3.141519265359f;
    float radius = 1.0f;
    float area;
    area = PI * radius * radius;
    System.out.println (“The area of the circle = ” +
    area);
    }// end of main
}// end of class
```

Output:

```
area of the circle = 3.1415193
```



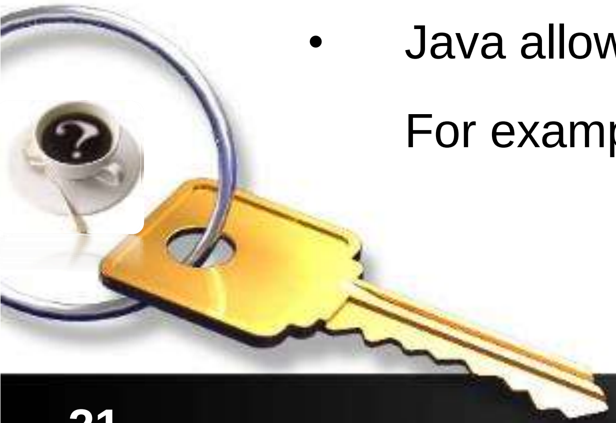
Declaration of variable

- A variable is defined by an **identifier, a type, and an optional initializer.**
- The variables also have a **scope(visibility / lifetime).**
- In Java, all variables must be declared before their use.
- The basic form of a variable declaration is :

type identifier [= value][, identifier [= value] ...] ;

- Java allows variables to be initialized dynamically.

  For example: **double** c = 2 * 2;



Scope and life of a variable:

- Variables declared inside a scope are not accessible Outside to the scope.
- Scopes can be nested. The outer scope encloses the inner scope.
- Variables declared in the outer scope are visible to the inner scope.
    - Variables declared in the inner scope are not visible to the outside scope.

```
public class Main
{ public static void main(String args[])
    { int x; // known within main
    x = 10;
    if (x == 10)
          { int y = 20;
          System.out.println("x and y: " + x + " " + y);
          x = y + 2; }
    System.out.println("x is " + x);
    }// end of main
}// end of class
```

Try this….

PSS

Output:

```
x and y: 10 20
x is 22
```

Try this….

```
public class Main2
{ public static void main(String args[])
    { if (true)

    {       int y = 20;
            System.out.println("y: " + y);
    } // end of if

    y = 100;

}// end of main
}// end of class
```

Output:

```
D:\>javac Main.java Main.java:9: cannot find symbol
symbol : variable y
location: class Main
          y = 100; // Error! y not known here
          ^
          1 error
```

PSS

```
public class Main3
{ public static void main(String args[])
    { int i = 1;
          {int i = 2;
          }
    }
}
```

Try this….

PSS

Output:

Results in compilation error.

'i' is already defined……

Flow Control: if:

- **if**(condition) statement;
- Note: Write a java program that compares two variables and print appropriate message.
- The condition can be expression that result in a value.
- Expression may return boolean value.
- **if (b)** is equivalent to if (b== true).

Flow Control: if else:

if (condition) statement1;

else statement2;

- Each statement may be a single statement or a compound statement enclosed in curly braces (a block).
- The condition is any expression that returns a boolean value.
- Nested if statements are possible

Flow Control: if else ladder:

if(condition) statement;
else if(condition) statement;
else if(condition) statement;
…
…
else statement;

Example

```
public class Main4
{ public static void main(String args[])
{ int month = 4;
String value;
if      (month == 1) value = "A";
else if (month == 2) value = "B";
else if (month == 3) value = "C";
else if (month == 4) value = "D";
else value = "Error";
System.out.println("value = " + value);
} }
```

Switch statement:

```
switch (expression)
{ case value1:          statement sequence
                        break;
 case value2 :          statement sequence
                        break;
. . .
case  valueN:           statement sequence
                        break;
default:                default statement sequence }
```

.

Switch statement can be nested

Example of Switch Statement:

```
package demo2;
public class Demo2 {
    public static void main(String[] args) {
        int day = 4;
        switch (day) {
            case 1: System.out.println("Monday");   break;
            case 2: System.out.println("Tuesday");  break;
            case 3: System.out.println("Wednesday"); break;
            case 4: System.out.println("Thursday");  break;
            case 5: System.out.println("Friday");  break;
            case 6: System.out.println("Saturday"); break;
            case 7: System.out.println("Sunday"); break;
        }
    }
}
```

For Loop statement:

for (*statement 1*; *statement 2*; *statement 3*)

{ *// code block to be executed* }

- **Statement 1** is executed (one time) before the execution of the code block.
- **Statement 2** defines the condition for executing the code block.
- **Statement 3** is executed (every time) after the code block has been executed.

For Loop Example:

- The example below will print **all the numbers** 0 to 4:
  - for (int i = 0; i < 5; i++) { System.out.println(i); }
- This example will only print **even values** between 0 and 10:
  - for (int i = 0; i <= 10; i = i + 2) { System.out.println(i); }

For Each Loop Example:

- Open NetBeans Editor and create java application named as Demo1.

```
package demo1;
public class Demo1 {
    public static void main(String[] args) {
    String[] days  = {"Mon", "Tue", "Wed", "Thu"};
    for (String i : days) { System.out.println(i);    }
 }

}
```

Switch statement can be nested

While Loop Statement :

- The while loop loops through a block of code as long as a specified condition is true:
- while (*condition*) { *// code block to be executed* }
- Example:
  - int i = 0;
  - while (i < 5)
    - { System.out.println(i);
    -     i++; }

Do While Loop Statement :

- The do/while loop is a variant of the while loop. This loop will execute the code block once, before checking if the condition is true, then it will repeat the loop as long as the condition is true.
- do { *// code block to be executed* } while (*condition*);
- Example:
  - int i = 0;
  - do { System.out.println(i); i++; }
  - while (i < 5);

Recursion

```
class factorial{
    int fact(int n){
          if (n==1) return 1;
          else return (n*fact(n-1));}
    }
class factdemo{
    public static void main (String args[]){
          int a = 4; int fa=0;
          factorial f = new factorial ();
          fa=f.fact(a);
          System.out.println(fa);
    }
}
```

Arrays

- General form of one dim array declaration is
  type array-name[size];
- Examples are:
  - **int a[10];**
    - Defines 10 integers such as a[0], a[1], … a[9]
  - **char let[26];**
    - Defines 26 alphabets let[1]='B';
  - **float x[20];**
  - **Employee e[100];**
    - Employee is a class definition
  - **Tree t[15];**
    - Tree is a class

Array Definition with Initialization

- int maxmarks[6]= {71,56,67,65,43,66}
- char let[5]= {‘a’, ‘e’, ‘I’, ’o’, ’u’};

- Initialization of an array can be done using new statement as follows:
  - int a[j]; // defines a as an array contains j integrs
  - a=new int [10] // assigns 10 integers to the array a

- This can also be written as
  - int [] a = new int [10];

Example of array

```
class array{
   public static void main (String args[ ]){
   int score [ ] = { 66,76,45,88,55,60};
    for (int i=0; i<6; i++)
        System.out.println(score[i]);
    System.out.println(“================”);
   }
}
```

Example of array

```
public class Main4 {
public static void main(String[] args)
    { int[] intArray = new int[] { 1, 2, 3, 4, 5 };
    // calculate sum
    int sum = 0;
    for (int i = 0; i < intArray.length; i++)
    { sum = sum + intArray[i]; }
    // calculate average
    double average = sum / intArray.length;
    System.out.println("average: " + average);
    }
 }
```

Example of array

```
public class Main6
{ public static void main(String args[])
    { int a1[] = new int[10];
      int a2[] = {1, 2, 3, 4, 5};
      int a3[] = {4, 3, 2, 1};
    System.out.println("length of a1 is " + a1.length);
    System.out.println("length of a2 is " + a2.length);
    System.out.println("length of a3 is " + a3.length);
    }
}
```

```
import java.util.*;
     public class  array{
       public static void main(String[] args){
               int num[] = {50,20,45,82,25,63};
               int l = 6;  // you may use l= num.length;
               int i,j,t;
               System.out.print("Given number : ");

               for (i = 0;i < l;i++ )    { System.out.print("  " + num[i]);   }

               System.out.println("\n");
               System.out.print("Accending order number : ");

               Arrays.sort(num);

               for(i = 0;i < l;i++){
               System.out.print("  " + num[i]);
               }
       }
}
```

Two Dimensional Arrays

Declaration of a two dimensional array called twoD with size 4*5

- **int** twoD[][] = **new int**[4][5];

(0,0)			**(0,3)**	**(0,4)**
(1,0)	(1,1)	…	…	(1,4)
(2,0)	…	(2,2)	…	(2,4)
(3,0)	…	…	(3,3)	(3,4)

Matrix

```
public class Main {
public static void main(String args[]) {
    int twoD[][] = new int[4][5];
    for (int i = 0; i < 4; i++)
    { for (int j = 0; j < 5; j++)
            { twoD[i][j] = i*j; } }
    //------------------------------------------------------------------------------
    for (int i = 0; i < 4; i++)
    { for (int j = 0; j < 5; j++)
            { System.out.print(twoD[i][j] + " "); }
    System.out.println(); }
    }
}
```

Initialization of Two Dimensional Array

```
public class Main{
public static void main(String args[]) {
    double m[][] = {        { 0, 1, 2, 3 },
                            { 0, 1, 2, 3 },
                            { 0, 1, 2, 3 },
                            { 0, 1, 2, 3 } };

    for(int i=0; i<4; i++)
    { for(int j=0; j<4; j++)
          { System.out.print(m[i][j] + " "); }
          System.out.println(); }
    }
}
```

Jagged array

- When you allocate memory for a multidimensional array, **you can allocate the remaining dimensions separately.** For example, the following code allocates the second dimension manually.

```
public class Main {
public static void main(String[] argv)
    { int twoD[][] = new int[4][];
           twoD[0] = new int[5];
           twoD[1] = new int[5];
           twoD[2] = new int[5];
           twoD[3] = new int[5]; } }
```

```
public class Main {
public static void main(String args[]) {
   int twoD[][] = new int[4][];
   twoD[0] = new int[1];
   twoD[1] = new int[2];
   twoD[2] = new int[3];
   twoD[3] = new int[4];
```

```
for (int i = 0; i < 4; i++)
{ for (int j = 0; j < i + 1; j++)
    { twoD[i][j] = i + j; } }
    //--------------------------------------------------
    for (int i = 0; i < 4; i++)
     { for (int j = 0; j < i + 1; j++)
           System.out.print(twoD[i][j] + " ");
    System.out.println(); }
  }
}
```

References:

- Patrick Naughton and Herbert Schildt, The Complete Reference Java 2, Seventh, Tata McGraw Hill Pub., 2007
- https://www.w3schools.com/