

JAVA BASICS

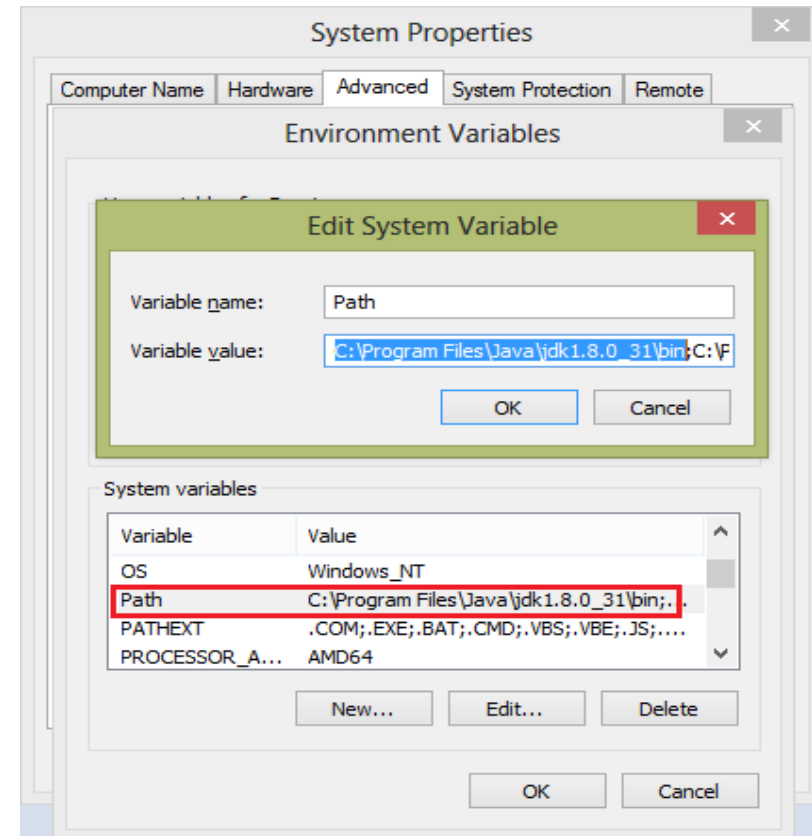
Tanjina Helaly

TOOLS/SET-UP



STEP1: INSTALL JAVA AND PATH SET-UP

- Need to install Java(JDK and JRE). Get the latest version from Java Standard Edition(SE) from <https://www.oracle.com/java/technologies/javase-downloads.html>
 - After installing Java you may need to set-up the “**Path**” environment variable which is available from **My Computer** under **Advanced Properties** tab.
 - **Note:** Do not delete anything in “Path” variable. Just add your path “C:\Program Files\Java\jdk1.8.0_31\bin;” (Depending on your version the path will change) at the beginning of the existing value.



STEP 2: INSTALL IDE

- Need an IDE: Eclipse or NetBeans or IntelliJ IDEA.

Or

- A Text Editor e.g. TextPad
- ***You can install***
 - eclipse from : <https://www.eclipse.org/downloads/>
 - NetBeans:
<https://netbeans.apache.org/download/index.html>
 - IntelliJ IDEA:
<https://www.jetbrains.com/idea/download/#section=windows>



COMPILE & RUN JAVA APPLICATION



WITHOUT IDE

- Using JDK you can compile and run java program from command line.
 - Compile
 - Enter the following command in command prompt.
`c:> javac HelloWorld.java`
 - This will compile the HelloWorld.java file and
 - It will produce HelloWorld.class file which will have the bytecode.
 - Run (need to run the compiled version)
 - Enter the following command in command prompt
`c:>java HelloWorld`
 - It will run java byte code on native machine



WITH JAVA IDE

- Creating, Compiling, Debugging and Executing using command prompt is not user friendly. IDE is provided for that. A list of IDEs are:
 - Eclipse
 - Netbeans.
 - IntelliJ IDEA



DATA TYPES

- Divided into two broad categories:
 - primitive types
 - class/reference types.
- Primitive data : eight types
 - Logical: boolean (true or false)
 - doesn't hold integer (unlike C)
 - Textual: char (16 bits)
 - use the Unicode(International: 0-255) not ASCII(1 byte: 0-127)
 - Integral: byte (8 bits), short (16 bits), int (32 bits), and long (64 bits)
 - Floating point: float (32 bits) and double (64 bits)
- Class or reference data: two types
 - Textual: String
 - All classes that declare by yourself



CASTING

- **Converting from one data type to another.**
- e.g. assigning an int value to a long variable

- **Example**

```
public class TestCast {  
    public static void main(String[] args) {  
        byte b= 5;  
        int a = b; // OK. Auto Casting  
        byte c = a; // Compiler error. Need Casting  
        c = (byte)a; // Casting  
  
        float f = 1.2f;  
        a= f; // Compiler error. Need Cast  
        a = (int)f; // Explicit Cast  
        f = a;  
    }  
}
```



OPERATOR

- Assignment =
- Arithmetic + - * / %
- Equality == !=
- Relational < <= > >=
- Logical &&, ||
- increment/decrement ++ --
- Shift << >>



ARRAYS

- An **array** is a **collection** of data items, all of the **same type**, accessed using a **common name**.
- The data type can be either a **primitive** data type or a **reference** type.
- Major differences with C/C++ arrays:
 - Java arrays are references
 - Java arrays know their size (length property)
 - Java multidimensional arrays do not need to be rectangular.
 - Java array elements are initialized



ARRAY DECLARATION & INITIALIZATION

○ Declaration

```
int[] sampleArray;  
sampleArray = new int[10];  
Or  
int[] sampleArray = new int[10];
```

○ Initialization

• During declaration

```
int[] sampleArray = {1,2,3,4,5};  
int[] sampleArray = new int[] {1,2,3,4,5};
```

• After declaration

```
int[] sampleArray;  
sampleArray = new int[] {1,2,3,4,5};  
sampleArray = {1,2,3,4,5}; // compiler error
```



ARRAY SIZE & ACCESSING A SPECIFIC INDEX

- Getting size of array

```
int[] sampleArray = new int[10];  
int size = sampleArray.length; //this will return the size of the  
array, here 10
```

- Accessing a specific item

- Assigning a value

```
sampleArray[0] = 5;  
sampleArray[1] = 2;  
sampleArray[2] = 3;
```

- Getting/Reading a value

```
int value = sampleArray[2];
```



ARRAYS – EXAMPLE CODE

```
public class ArrayExample
{
    public static void main( String args[] )
    {
        // space to store Reference is allocated, no array space allocated
        double[] sampleArray;

        //allocate array locations on heap
        sampleArray = new double[ 10 ];

        // Indexing starts at 0 like C/C++
        sampleArray[ 0 ] = 5.5;

        // Reference refers to new array.
        // Old array available for garbage collection
        sampleArray = new double[ 2 ];
    }
}
```



MULTI-DIMENSIONAL ARRAY

- *multidimensional arrays* are actually arrays of arrays.

```
int twoD[][] = new int[4][5];
```

- Do not need to be rectangular
- During creation it's required to specify the size for the first/leftmost dimension. You can allocate the remaining dimensions separately.

```
int twoD[][] = new int[4][];
```



MULTI-DIMENSIONAL ARRAY

	Rectangular	Irregular Array
Declaration & Array Creation	<pre>int twoD[][] = new int[4][5]; or 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];</pre>	<pre>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];</pre>
Example of Array	<pre>0 1 2 3 4 5 6 7 8 9 1 1 1 3 4 5 6 7 8 9</pre>	<pre>0 1 2 3 4 5 6 7 8 9</pre>



CONTROL STATEMENT

- if –else
- switch
- Loop
 - for
 - while
 - do-while



CONTROL STATEMENT

○ “Enhance for” or “for-each”

- automatically cycles through an array in sequence from the lowest index to the highest.
- Syntax : *for(type itr-var : collection) statement-block*

- *Example:*

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

```
int sum = 0;
```

```
for(int x: nums)
```

```
    sum += x;
```

- Advantage: Avoid boundary error



JUMP STATEMENT

○ **break**

- Exits out of a loop or switch statement
- Unlabeled break exits out of the innermost loop or switch
- Use labeled break to exit out of nested loops or switch or block.



JUMP STATEMENT

```
public class BreakExample {  
    public static void main( String args[] ) {  
        for ( int row = 0; row < 5; row++ ) {  
            System.out.println("Outer loop: " +  
row);  
            for ( int column = 0; column < 4 ;  
column++ ) {  
                System.out.print(column + " " );  
                if ( ((row + column) % 2 ) == 0 ) {  
                    System.out.println("Break " );  
                    break;  
                }  
            }  
        }  
    }  
}
```

Output:

```
Outer loop: 0  
0 Break  
Outer loop: 1  
0 1 Break  
Outer loop: 2  
0 Break  
Outer loop: 3  
0 1 Break  
Outer loop: 4  
0 Break
```



JUMP STATEMENT – LABELED JUMP

```
public class BreakExample {  
    public static void main( String args[] ) {  
        Outer:  
        for ( int row = 0; row < 5; row++ ) {  
            System.out.println("Outer loop: " + row);  
            for ( int column = 0; column < 4;  
                column++ ) {  
                System.out.println(column + "\t");  
                if ( ((row + column) % 2 ) == 0 )  
                {  
                    System.out.println("Break " );  
                    break Outer;  
                }  
            }  
        }  
    }  
}
```

Output:

Outer loop: 0
0 Break



JUMP STATEMENT

○ continue

- A continue statement skips to the end of the current loop's body.
- The loop's boolean expression is then evaluated.

Code	Output
<pre>public class TestContinue { public static void main(String args[]) { for(int i=0; i<10; i++) { System.out.print(i + " "); if (i%2 == 0) continue; System.out.println(""); } } }</pre>	<pre>0 1 2 3 4 5 6 7 8 9</pre>

REFERENCE

- Java:Complete Reference Chapter 1-5

