JAVA

Lecture 002 - Fundamentals

RECAP

- Java Development Kit (JDK)
- Java Runtime Environment (JRE)
- Java Virtual Machine (JVM)
- The Compilation and Execution of Java Program
  - Create a program
  - Compile a program
  - Load bytecode into memory
  - Verification
  - Execution
- Main Method

JAVA IDENTIFIERS

- An identifier is a name given to a method, variable, or other user-defined item.
- Identifiers: $,_,[A-Za-z],[0-9]
  - In fact, you can use Unicode characters, 中文
- Rules:
  - Cannot be empty, i.e., one or more characters
  - The first character cannot be a digit, e.g., 9sees
  - Case-sensitive: e.g., myvar and myVar are different

KEYWORDS

```
abstract continue for new switch assert default goto package synchronized
boolean do if private this break double implements protected throw byte
else import public throws case enum instanceof return transient catch
extends int short try char final interface static void class finally long
strictfp volatile const float native super while
```

- Top Tip! - Learning what each of these means takes you a long way in learning the language …also useful for exams
- https://www.geeksforgeeks.org/list-of-all-java-keywords/

VARIABLE NAMING CONVENTION

- Avoid keywords
- No white space
- Make variables descriptive (therefore self documenting)
- Chinese characters are allowed (Unicode), but no!
- CamelCase Syntax: If the name is combined with multiple words, later words will start with uppercase letters.
  - `e.g., topLevel`

VARIABLE NAMING CONVENTION

Type	Rules	Example
Class	• Start with uppercase letter • Noun	`class Employee{` `}`
Interface	• Start with uppercase letter • Adjective	`interface Runnable{` `}`
Method	• Start with lowercase letter • Verb • CamelCase, if multiple words	`void draw{` `}`
Variable	• Start with lowercase letter • CamelCase, if multiple words	`int id;`
Constant	• In uppercase • Multiple words should be separated by _ • May contain digits	`MIN_AGE = 18;`

EXERCISE1: SPOT THE IDENTIFIERS!

```
public class Identifiers{
    public static void main(String[] args){
        int a = 20;
    }
}
```

EXERCISE1: SPOT THE IDENTIFIERS!

```
public class Identifiers{
    public static void main(String[] args){
        int a = 20;
    }
}
```

Class name: Identifiers, String[]

Method name: main

Variable name: args, a

DATA TYPES IN JAVA

- Data Types in Java:
  - OO types.
  - 8 Non-OO (Primitive) types: byte, short, int, long, float, double, char, boolean.
- Integer Types: byte, short, int, long
  - int is the most commonly used integer type.
  - We use L to indicate long numbers, e.g., 100L

Data Type	Width in Bits	Range
byte	8	-2^7 to $2^7 - 1$
short	16	-2^{15} to $2^{15} - 1$
int	32	-2^{31} to $2^{31} - 1$
long	64	-2^{63} to $2^{63} - 1$

PRIMITIVE DATA TYPES

- Floating-Point Types: float and double.
  - Size differences.
  - Range differences.
  - Precision differences.
  - double is the most commonly used floating-point type, as all math function in Java use double
  - We use f to indicate float type, e.g., 2.91f
  - d for double type, e.g., 2.31d

Data Type	Width in Bits	Maximum
float	32	3.4×10^{38}
double	64	1.8×10^{308}

PRIMITIVE DATA TYPES

- Characters:
  - 16-bit characters to support Unicode characters (0 – 65,535)
  - Represented using '', e.g., 'X'
- Boolean Type:
  - true or false

Datatype	Default value(for fields)
byte	0
short	0
Int	0
long	0L
float	0.0f
double	0.0d
char	'\u0000'
boolean	false

A SECOND EXAMPLE

```
class Example2 {
        public static void main(String[] args) {
                int var1; // this declares a variable
                int var2; // this declares another variable
                var1 = 1024; // this assigns 1024 to var1
                System.out.println("var1 contains " + var1);
                var2 = var1 / 2;
                System.out.println("var2 contains " + var2);
        }
}
```

Declare variables

Value assignment

BASIC SYNTAX

- Variable declaration:
  - `type var-name; // e.g., int var1;`
  - `type var-name1, var-name2, …; // e.g., int var1, var2;`
- Assign a value to a variable:
  - `var-name = value; // e.g., var1 = 1024;`
  - `type var-name = value; // e.g., int var1 = 1024;`

- All variables must be declared (with their type) before they are used
- All statements end with ";"

OPERATORS

- Arithmetic Operators
  - +: addition, adds together two values. x + y
  - -: subtraction, subtracts one value from another. x - y
  - *: multiplication, multiplies two values. x * y
  - /: division, divide one value by another. x / y
  - %: modulus, return the division remainders. x % y
  - ++: incremental, increase the value of a variable by 1. x++ or ++x
  - --: decremental, decrease the value of a variable by 1. x-- or --x

EXAMPLE

```
public class ArithOps{
    public static void main(String[] args){
       int a = 10;
       int b = 20;
       int d = 25;
       System.out.println("a + b = " + (a + b));
       System.out.println("a - b = " + (a - b));
       System.out.println("a * b = " + (a * b));
       System.out.println("b / a = " + (b / a));
       System.out.println("b % a = " + (b % a));
       System.out.println("a++ = " + (a++));
       System.out.println("a-- = " + (a--));
       System.out.println("d++ = " + (d++));
       System.out.println("++d = " + (++d));
    }
}
```

OPERATORS

- Comparison Operators
  - ==: equals to — x == y
  - !=: not equals to — x != y
  - >: greater than — x > y
  - <: less than — x < y
  - >=: greater than or equals — x >= y
  - <=: less than or equals — x <= y

EXAMPLE

```
public class CompOps{
    public static void main(String[] args){
     int a = 10;
     int b = 20;
     System.out.println("a == b is " + (a == b));
     System.out.println("a != b is " + (a != b));
     System.out.println("a > b is " + (a > b));
     System.out.println("a < b is " + (a < b));
     System.out.println("a >= b is " + (a >= b));
     System.out.println("a <= b is " + (a <= b));
   }
}
```

OPERATORS

- Logical Operators
  - &&: logical and — x && y
  - ||: logical or — x || y
  - !: logical not — !x
- Assignment Operators
  - =: value assignment — x = 5
  - +=: assign the value of addition — x += 5 x = x + 5
  - -=: assign the value of subtraction — x -= 5 x = x – 5
  - ….
- Conditional Operators
- Expression ? Value1 : Value2 (a == 10) ? 5 : 2

EXERCISE 2

- Given x = 3, which of the following does NOT return 4?
  - 1) 2+2
  - 2) (x % 2 == 1) ? 4 : 2
  - 3) 6-(--x)
  - 4) x++

EXERCISE 2

- Given x = 3, which of the following does NOT return 4?
  - 1) 2+2
  - 2) (x % 2 == 1) ? 4 : 2
  - 3) 6-(--x)
  - **4) x++**

CONTROL STATEMENT

- Execution processes from top to bottom.
- The if Statement
  - Simplest form:
  - if (condition) statement;
- Example:
  - if(3 < 4) System.out.println("yes");
- The if-else Statement
  - If (condition) statement; else statement;

EXAMPLE OF IF STATEMENTS

```
class IfDemo {
	public static void main(String[] args) {
		int a, b;
		a = 2;
		b = 3;
		if(a < b)
			System.out.println("a is less than b");

		if(a == b)
			System.out.println("you won't see this");
	}
}
```

What is the problem?

EXAMPLE OF NESTED IF

```
class IfDemo {
	public static void main(String[] args) {
		int a, b;
		a = 2;
		b = 3;
		if(a < b)
			System.out.println("a is less than b");
		else{
			if(a == b)
				System.out.println(”a equals to b");
			else
				System.out.println(”a is larger than  b");
		}
	}
}
```

Looks messy…

THE IF-ELSE-IF LADDER

```
if(condition)
        Statement;
else if (condition)
        Statement;
else if (condition)
        Statement;
….
else
        Statement;
```

REVISED EXAMPLE

```
class IfDemo {
	public static void main(String[] args) {
		int a, b;
		a = 2;
		b = 3;
		if(a < b)
			System.out.println("a is less than b");
		else if(a == b)
			System.out.println(”a equals to b");
		else
			System.out.println(”a is larger than b");
	}
}
```

MORE IF-ELSE

```
int season = 0;
String seasonString;
if(season == 1){
    seasonString = "Spring";
}else if(season == 2){
    seasonString = "Summer";
}else if(season == 3){
    seasonString = "Autumn";
}else if(season == 4){
    seasonString = "Winter";
}else{
    seasonString = "Not a season";
}
System.out.println(seasonString);
```

DECISIONS -SWITCH

```
switch (expression){
        case constant1:
                statement sequence
                break;
        case constant2:
                statement sequence
                break;
        …
        default:
                statement sequence
                break;
}       // expression can be byte, short, int, char, enum and  String.
```

DECISIONS -SWITCH

```
public class Season {
    public static void main(String[] args) {
        int season = 0;
        String seasonString;
        switch (season) {
            case 1:  seasonString = "Spring";
                     break;
            case 2:  seasonString = "Summer";
                     break;
            case 3:  seasonString = "Autumn (Winter is Coming)";
                     break;
            case 4:  seasonString = "Winter";
                     break;
            default: seasonString = "not a season";
                     break; // you don't need this
        }
        System.out.println(seasonString);
    }
}
```

EXERCISE 3

- Given the following piece of code, what will be the result of y?

```
int x = 2;
int y =0;
switch(x){
        case 1: y+=1;break;
        case 2: y+=2;
        case 3: y+=3;
}
```

LOOPS

- For
- While
- Do While
- More advanced (for) loops
  - Labelled for
  - Enhanced for
  - For each

THE **FOR** LOOP

- **General form:**

```
for( initialization ; condition ; iteration )
        statement;

for( initialization ; condition ; iteration ){
        statement sequence;
}
```

- Initialization is normally for setting a loop control variable.
- Condition is for stopping the loop.
- Iteration is normally for updating the loop control variable.

EXAMPLE OF A FOR LOOP

```
class ForDemo {
        public static void main(String[] args) {
                int count;
                for(count = 0; count < 5; count = count+1)
                        System.out.println("This is count: " + count);
                System.out.println("Done!");
        }
}
```

VARIATIONS OF FOR LOOP

```
for( initialization ; condition ; ){
                        iteration
}
```

```
Initialization;
for(; condition ; ){
                        iteration
}
```

VARIATIONS OF FOR LOOP

- What is (;;) ?
- Loops with no body

```
for(int i = 0; i < 5; sum += i++);
```

- What does it mean?

WHILE LOOP

```
while (condition)
        statement;

while (condition){
        statements;
}
```

WHILE LOOP

```
class WhileDemo {
    public static void main(String[] args){
        int count = 0;
        while (count < 5) {
            System.out.println("Count is: " + count);
            count++;
        }
    }
}
```

DO WHILE LOOP

```
class DoWhileDemo {
    public static void main(String[] args){
        int count = 0;
        do {
            System.out.println("Count is " + count);
            count++;
        } while (count < 5);
    }
}
```

ARRAYS

- Array: a collection of variables of the same type.
  - Fixed size.
  - Variables are of the same type.
  - One or more dimensions.
  - Implemented as objects. (unused arrays will be garbage collected)
- One-dimensional arrays

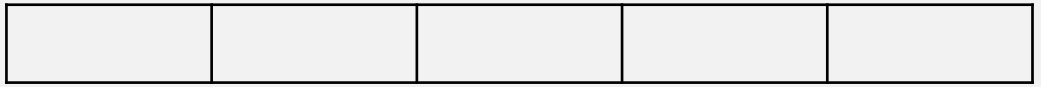
```
type[] array-name = new type[size];
```

- Type can be primitive types or any objects.

```
int[] intArray = new int[3];
String[] stringArray = new String[5];
```

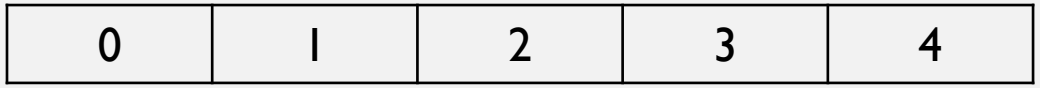
ARRAYS

- Each individual element within an array is accessed by its index.
- Index starts from zero, int[] xs = new int[5];



- Value assignement:
  - xs[3] = 5;
- Following code assign values to each element in an array

```
int[] xs = new int[5];
for(int i = 0; i < 5; i++)
        xs[i] = i;
```



ARRAYS

- Difficult to create an array and assign values like:

```
int[] xs = new int[4];
xs[0] = 1;
xs[1] = 3;
xs[2] = -13;
xs[3] = 20;
```

- Alternative, we could…

```
int[] xs = new int[]{1, 3, -13, 20};
int[] xs = {1, 3, -13, 20};
```

- Length of an array

```
xs.length;
```