

Java Fundamental

Pertemuan 2



Objectives :

- Java Intro
- Basic Java Syntax
- Control Flow
- Arrays



Java Intro



Java Intro

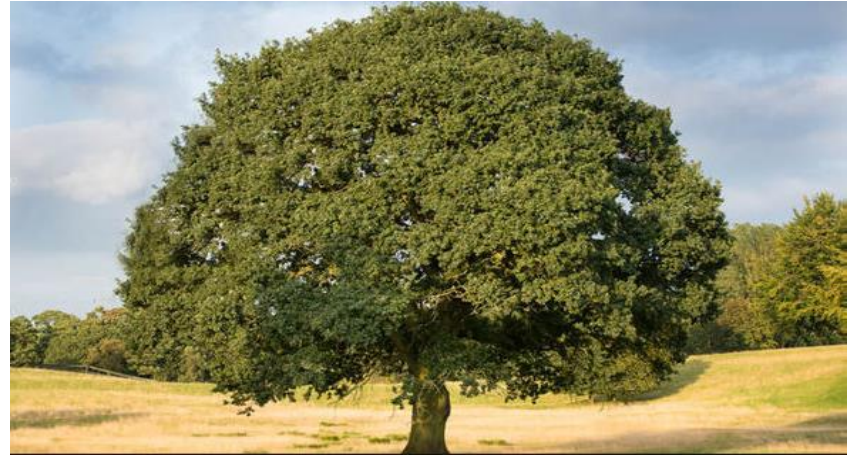
- **James Gosling, Mike Sheridan, and Patrick Naughton** initiated the Java language project in June 1991
- The language was initially called **Oak** after an **oak tree** that stood outside Gosling's office. It went by the name **Green** later, and was later renamed **Java**, from a list of random words
- Gosling aimed to implement a virtual machine and a language that had a familiar **C/C++ style of notation**
- Sun Microsystems released the first public implementation as **Java 1.0 in 1995**
- On May 8, 2007, Sun finished the process, making all of Java's core code available under **GNU Public License**



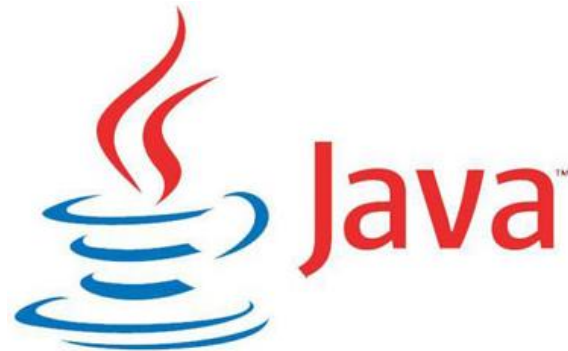
Contd..



- James Gosling



- Oak tree



Contd..

- Java has some interesting features:
 - automatic type checking,
 - automatic garbage collection,
 - simplifies pointers; no directly accessible pointer to memory,
 - simplified network access,
 - multi-threading!



Java Advantages

- Portable - Write Once, Run Anywhere
- Security has been well thought through
- Robust memory management
- Designed for network programming
- Multi-threaded (multiple simultaneous tasks)
- Dynamic & extensible (loads of libraries)
 - Classes stored in separate files
 - Loaded only when needed



Basic Syntax



Primitive Data Type & Variables

- Java is strongly typed language.
- boolean, char, byte, short, int, long, float, double etc.
- These basic (or primitive) types are the only types that are not objects (due to performance issues).
- This means that you don't use the new operator to create a primitive variable.
- Declaring primitive variables:

`float initVal;`

`int retVal, index = 2;`

`double gamma = 1.2, brightness`

`boolean valueOk = false;`



Initialization

- If no value is assigned prior to use, then the compiler will give an error
- Java sets primitive variables to zero or false in the case of a boolean variable
- All object references are initially set to null
- An array of anything is an object
 - Set to null on declaration
 - Elements to zero false or null on creation



Assignment

- All Java assignments are right associative

```
int a = 1, b = 2, c = 5
```

```
a = b = c
```

```
System.out.print("a= " + a + "b= " + b + "c= " + c)
```

- What is the value of a, b & c
- Done right to left: **a = (b = c);**



Declarations

```
int index = 1.2;           // compiler error
boolean retOk = 1;         // compiler error
double fiveFourths = 5 / 4; // no error!
float ratio = 5.8f;        // correct
double fiveFourths = 5.0 / 4.0; // correct
```

- 1.2f is a float value accurate to 7 decimal places.
- 1.2 is a double value accurate to 15 decimal places.



Constant Variable

- Untuk membuat variable yang bersifat konstan (tetap), keyword yang digunakan adalah **final**.
- Keyword **final** mengindikasikan variable hanya boleh dideklarasikan 1 kali dan tidak dapat diubah.
- Umumnya variable ditulis dengan UPPERCASE.

final double PI = 3.14 //nilainya tetap



Mathematical Operator

- $*$ $/$ $\%$ $+$ $-$ are the mathematical operators
- $*$ $/$ $\%$ have a higher precedence than $+$ or $-$

```
double myVal = a + b % d - c * d / b;
```

- Is the same as:

```
double myVal = (a + (b % d)) -  
                ((c * d) / b);
```



Statement & Blocks

- A simple statement is a command terminated by a semi-colon:

```
name = "Fred";
```

- A block is a compound statement enclosed in curly brackets:

```
{
```

```
    name1 = "Fred"; name2 = "Bill";
```

```
}
```

- Blocks may contain other blocks



Type Casting

- Type casting pada java dilakukan untuk mengkonversi suatu variable dengan tipe data yang berbeda dari tipe data saat deklarasi.
- Contoh :

```
double x = 9.997;
```

```
int nx = (int) x;
```



Control Flow



Flow of Control

- Java executes one statement after the other in the order they are written
- Many Java statements are flow control statements:

Alternation: if, if else, switch

Looping: for, while, do while

Escapes: break, continue, return



If – Conditional Statement

- The if statement evaluates an expression and if that evaluation is true then the specified action is taken

`if (x < 10) x = 10;`

- If the value of x is less than 10, make x equal to 10
- It could have been written:

`if (x < 10)`

`x = 10;`

- Or, alternatively:

`if (x < 10) { x = 10; }`



Relational Operators

==	Equal (careful)
!=	Not equal
>=	Greater than or equal
<=	Less than or equal
>	Greater than
<	Less than



If... else

- The if ... else statement evaluates an expression and performs one action if that evaluation is true or a different action if it is false.

```
if (x != oldx) {  
    System.out.print("x was changed");  
}  
else {  
    System.out.print("x is unchanged");  
}
```



Nested if ... else

```
if ( myVal > 100 ) {  
    if ( remainderOn == true) {  
        myVal = mVal % 100;  
    }  
    else {  
        myVal = myVal / 100.0;  
    }  
}  
else  
{  
    System.out.print("myVal is in range");  
}
```



Else if

- Useful for choosing between alternatives:

```
if ( n == 1 ) {  
    // execute code block #1  
}  
else if ( j == 2 ) {  
    // execute code block #2  
}  
else {  
    // if all previous tests have failed, execute code block #3  
}
```



A Warning..

WRONG!

```
if( i == j )
    if ( j == k )
        System.out.print(
            "i equals k");
else
    System.out.print(
        "i is not equal    to j");
```

CORRECT!

```
if( i == j ) {
    if ( j == k )
        System.out.print(
            "i equals k");
}
else
    System.out.print("i is
not equal to j"); //
Correct!
```



The switch statement..

```
switch ( n ) {  
    case 1:  
        // execute code block #1  
        break;  
    case 2:  
        // execute code block #2  
        break;  
    default:  
        // if all previous tests fail then  
        // execute code block #4  
        break;  
}
```



The for loop

- Loop n times

```
for ( i = 0; i < n; n++ ) {  
    // this code body will execute n times  
    // ifrom 0 to n-1  
}
```

- Nested for:

```
for ( j = 0; j < 10; j++ ) {  
    for ( i = 0; i < 20; i++ ){  
        // this code body will execute 200 times  
    }  
}
```



While loops

```
while(response == 1) {  
    System.out.print( "ID =" + userID[n]);  
    n++;  
    response = readInt( "Enter ");  
}
```

- What is the minimum number of times the loop is executed?
- What is the maximum number of times?



do {... } while loops

```
do {  
    System.out.print( "ID =" + userID[n] );  
    n++;  
    response = readInt( "Enter " );  
} while (response == 1);
```

- What is the minimum number of times the loop is executed?
- What is the maximum number of times?



Break

- A break statement causes an exit from the **innermost** containing **while**, **do**, **for** or **switch** statement.

```
for ( int i = 0; i < maxID, i++ ) {  
    if ( userID[i] == targetID ) {  
        index = i;  
        break;  
    }  
} // program jumps here after break
```



Continue

- Can only be used with while, do or for.
- The continue statement causes the innermost loop to start the next iteration immediately

```
for ( int i = 0; i < maxID; i++ ) {  
    if ( userID[i] != -1 ) continue;  
    System.out.print( "UserID " + i + " : " +  
        userID);  
}
```



Array



Array

- An array is a list of similar things
- An array has a fixed:
 - name
 - type
 - length
- These must be declared when the array is created.
- Arrays sizes cannot be changed during the execution of the code



Contd..

myArray =

3	6	3	1	6	3	4	1
0	1	2	3	4	5	6	7

myArray has room for 8 elements

- the elements are accessed by their index
- in Java, array indices start at 0



Declaring Arrays

`int myArray[];`

declares *myArray* to be an array of integers

`myArray = new int[8];`

sets up 8 integer-sized spaces in memory, labelled *myArray[0]* to *myArray[7]*

`int myArray[] = new int[8];`

combines the two statements in one line



Assigning Values

- refer to the array elements by index to store values in them.

```
myArray[0] = 3;
```

```
myArray[1] = 6;
```

```
myArray[2] = 3;
```

- can create and initialise in one step:

```
int myArray[] = {3, 6, 3, 1, 6, 3, 4, 1};
```



Iterating Through Array

- **for** loops are useful when dealing with arrays:

```
for (int i = 0; i < myArray.length; i++) {  
    myArray[i] = getsomevalue();  
}
```



Arrays of Objects

- So far we have looked at an array of primitive types.
 - integers
 - could also use doubles, floats, characters...
- Often want to have an array of objects
 - Students, Books, Loans
- Need to follow 3 steps.



Declaring Array Object

1. Declare the array

```
private Student studentList[];
```

- this declares studentList

2. Create the array

```
studentList = new Student[10];
```

- this sets up 10 spaces in memory that can hold references to Student objects

3. Create Student objects and add them to the array:

```
studentList[0] = new Student("Cathy",  
"Computing");
```



Question

