



Lecture 2 : Getting Started with Java

First Program

a) About Eclipse

Eclipse is an integrated development environment (IDE) for developing applications using the Java programming language and many other programming languages. The Java Development Tools (JDT) project provides a plug-in that allows Eclipse to be used as a Java IDE.

A new Java class can be created using the New Java Class wizard. The Java Class wizard can be invoked in different ways –

1. By clicking on the File menu and selecting New → Class, or
2. By right clicking in the package explorer and selecting New → Class, or
3. By clicking on the class drop down button and selecting class.

Note : We will understand what classes are when we will study Object Oriented Programming. For now you can assume them as a file. Also name of class and .java file inside which we have this class should be same.

b) About Main

Consider the following line of code:

```
public static void main(String[] args)
```

1. This is the line at which the program will begin executing. This statement is similar to start block in flowcharts. All Java programs begin execution by calling main()
2. We will understand what public, static, void mean in subsequent lectures. For now we should assume that we have to write main as it is.
3. The curly braces {} indicate start and end of main.

c) print / println

In order to print things to console we have to write --- System.out.println("Hello World"). Again for now we should leave System.out.print mean, and should write it as it is.

The built-in method print() is used to display the string which is passed to it. This output string is not followed by a newline, i.e., the next output will start on the same line. The built-in method println() is similar to print(), except that println() outputs a newline after each call.

Example Code:

```
public static void main(String[] args) {  
    System.out.println("Hello World");  
    System.out.println("Programming is fun");  
}
```

Output:

Hello World

Programming is fun

Variables

a) Add two numbers

Consider the following code for adding two numbers

```
public static void main(String[] args) {  
    int num1 = 10;  
    int num2 = 5;  
    int ans = num1 + num2;  
    System.out.println("Sum =" + ans);  
}
```

Output:

15

Here, we used variables to store values of two integers and their sum. Thus, a variable is a basic unit of storage in a Java program.

Syntax for Declaring a Variable:

type variable_name [= value];

Here, type is one of Java's primitive datatypes. The variable_name is the name of a variable. We can initialize the variable by specifying an equal sign and a value (Initialization is optional). However, the compiler never assigns a default value to an uninitialized local variable in Java.

While writing variable names you should be careful and follow the rules for naming them. Following are the rules for writing variable names ---

1. All variable names may contain uppercase and lowercase letters (a-z, A-Z), underscore (_), dollar sign (\$) and the digits 0 to 9. The dollar sign

character is not intended for general use. No spaces and no other special characters are allowed.

2. The variable names must not begin with a number.
3. Java is case-sensitive. Uppercase characters are distinct from lowercase characters.
4. A Java keyword (reserved word) cannot be used as a variable name.

b) Data types of variables

Based on the data type of a variable, the operating system allocates memory and decides what can be stored in the reserved memory. Therefore, by assigning different data types to variables, we can store integers, decimals, or characters in these variables.

There are eight primitive data types in Java:

DATA TYPE	DEFAULT VALUE	DEFAULT SIZE
char	'\0' (null character)	2 bytes
byte	0	1 byte
short	0	2 bytes
int	0	4 bytes
long	0L	8 bytes
Float	0.0f	4 bytes
Double	0.0d	8 bytes
Boolean	false	Not specified

c) Code for calculating Simple Interest

Example Code:

```
public class SimpleInterest {  
    public static void main(String[] args) {  
        double principal = 2500.0, rate = 6.0, time = 5.0;  
        double si = (principal * rate * time) / 100;  
        System.out.println("Simple Interest = " + si);  
    }  
}
```

Output:

Simple Interest = 750.0

Taking Input

a) Scanner

The Java Scanner class breaks the input into tokens using a delimiter that is whitespace by default. It provides many ways to read and parse various primitive values.

In order to use scanner you have to write this import statement at the top –
`import java.util.Scanner;`

Example Code:

```
//Code for adding two integers entered by the user
import java.util.Scanner;
class AddTwoNumbers
{
    public static void main(String args[])
    {
        int a, b, c;
        System.out.println("Enter two integers to calculate their sum:");
    };
    // Create a Scanner
    Scanner s = new Scanner(System.in);
    a = s.nextInt();
    b = s.nextInt();
    c = a + b;
    System.out.println("Sum of entered integers = "+c);
}
}
```

Sample Input:

10 5

Output:

15

Here, `s.nextInt()` scans and returns the next token as `int`. A token is part of entered line that is separated from other tokens by space, tab or newline. So when input line is: "10 5" then `s.nextInt()` returns the first token i.e. "10" as `int` and `s.nextInt()` again returns the next token i.e. "5" as `int`.

b) Code for calculating simple interest taking input from user

Example Code:

```

import java.util.Scanner;

public class SimpleInterest {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        double si, principal, rate, time;
        principal = input.nextDouble();
        rate = input.nextDouble();
        time = input.nextDouble();
        si = (principal * rate * time) / 100;
        System.out.println("Simple Interest= " + si);
    }
}

```

Sample Input:
2500.0 6.0 5.0

Output:
750.0

c) Taking character input

To read a character as input, we use `next().charAt(0)`. The `next()` function returns the next token in the input as a string and `charAt(0)` function returns the first character in that string.

Example code to read a character as input:

```

import java.util.Scanner;

public class ScannerDemo1 {
    public static void main(String[] args) {
        Scanner s = new Scanner(System.in);
        char ch = s.next().charAt(0); // character input
        System.out.println("input character = " + ch);
    }
}

```

Sample Input:

k

Output:

input character = k

Example code to take a string as input:

```

public static void main(String[] args) {
    Scanner s = new Scanner(System.in);
    String str;
    str = s.next();
    System.out.print(str);
}

```

Sample Input:

Coding Ninjas

Output:

Coding

Here, `s.next()` returns the next token as String. A token is part of entered line that is separated from other tokens by space, tab or newline. So when input line is --- "Coding Ninjas" then `s.next()` returns the first token i.e. "Coding".

d) Other scanner options

Some commonly used Scanner class methods are as follows:

METHOD	DESCRIPTION
<code>public String next()</code>	It returns the next token from the Scanner.
<code>public String nextLine()</code>	It moves the Scanner position to the next line and returns the value as a string.
<code>public byte nextByte()</code>	It scans the next token as a byte.
<code>public short nextShort()</code>	It scans the next token as a short value.
<code>public int nextInt()</code>	It scans the next token as an int value.
<code>public long nextLong()</code>	It scans the next token as a long value.
<code>public float nextFloat()</code>	It scans the next token as a float value.
<code>public double nextDouble()</code>	It scans the next token as a double value.

Example code:

```

public static void main(String[] args) {
    Scanner s = new Scanner(System.in);
    int a = s.nextInt();
    String str = s.nextLine();
    System.out.println(a);
    System.out.println(str);
}

```

```
}
```

Sample Input:

100 Hello World

Output:

100

Hello World

Here, `s.nextInt()` scans and returns the next token as `int`. A token is part of entered line that is separated from other tokens by space, tab or newline. So when input line is -- "100 Hello World" then `s.nextInt()` returns the first token as `int` i.e. "100" and `s.nextLine()` returns remaining part of line i.e " (space)Hello World"