Java Foundation with Data Structures

Lecture 2 : Getting Started

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

How is Data Stored ?

a) How are integers stored ?

The most commonly used integer type is int which is a signed 32-bit type.
When you store an integer, its corresponding binary value is stored. The way integers are stored differs for negative and positive numbers. For positive numbers the integral value is simple converted into binary value and for negative numbers their 2’s compliment form is stored.

Let’s discuss How are Negative Numbers Stored?

Computers use 2's complement in representing signed integers because:

1. There is only one representation for the number zero in 2's complement, instead of two representations in sign-magnitude and 1's complement.

2. Positive and negative integers can be treated together in addition and subtraction. Subtraction can be carried out using the "addition logic".

Example:

int i = -4;
Steps to calculate Two’s Complement of -4 are as follows:

Step 1: Take Binary Equivalent of the positive value (4 in this case)
0000 0000 0000 0000 0000 0000 0000 0100

Step 2: Write 1's complement of the binary representation by inverting the bits

1111 1111 1111 1111 1111 1111 1111 1011

Step 3: Find 2's complement by adding 1 to the corresponding 1's complement

```
 1111 1111 1111 1111 1111 1111 1111 1011
+0000 0000 0000 0000 0000 0000 0000 0001
 -----------------------------------------------------------
 1111 1111 1111 1111 1111 1111 1111 1100
```

Thus, integer -4 is represented by the binary sequence (1111 1111 1111 1111 1111 1111 1111 1100) in Java.

b) Float and Double values

In Java, any value declared with decimal point is by default of type double (which is of 8 bytes). If we want to assign a float value (which is of 4 bytes), then we must use 'f' or 'F' literal to specify that current value is "float".

Example:

```
float float_val = 10.4f;            //float value
double val = 10.4;          //double value
```

c) How are characters stored

Java uses Unicode to represent characters. As we know system only understands binary language and thus everything has to be stored in the form binaries. So for every character there is corresponding code – Unicode/ASCII code and binary equivalent of this code is actually stored in memory when we try to store a char.

Unicode defines a fully international character set that can represent all the characters found in all human languages. In Java, char is a 16-bit type. The range of a char is 0 to 65,536.

Example code:

```
public static void main(String[] args) {
        char ch1, ch2;
        ch1 = 88;     //ASCII value for 'X'
        ch2 = 'Y';
        System.out.println(ch1 +" " +ch2);
}
```

Output:
X Y

Adding int to char

When we add int to char, we are basically adding two numbers i.e. one corresponding to the integer and other is corresponding code for the char.

Example code:

```
public static void main(String[] args) {
        System.out.println('a' + 1);
}
```

Output:
98

Here, we added a character and an int, so it added the ASCII value of char 'a' i.e 97 and int 1. So, answer will be 98.

Similar logic applies to adding two chars as well, when two chars are added their codes are actually added i.e. 'a' + 'b' wil give 195.

Typecasting

1. Widening or Automatic type conversion:
   In Java, automatic type conversion takes place when the two types are compatible and size of destination type is larger than source type.

2. Narrowing or Explicit type conversion:
   When we are assigning a larger type value to a variable of smaller type, then we need to perform explicit type casting.

   Example code:

```
public static void main(String[] args) {
        int i = 100;
        long l1 = i;                //automatic type casting

        double d = 100.04;
        long l2 = (long)d;          //explicit type casting
        System.out.println(i);
```

```
        System.out.println(l1);
        System.out.println(d);
        System.out.println(l2);
}
```

Output:
100
100
100.04
100

Operators

a) Arithmetic operators

Arithmetic operators are used in mathematical expression in the same way that are used in algebra.

OPERATOR	DESCRIPTION
+	Adds two operands
-	Subtracts second operand from first
*	Multiplies two operands
/	Divides numerator by denominator
%	Calculates Remainder of division

b) Relational operators

Relational Operators are the operators that used to test some king of relation between two entities. The following table lists the relation operators supported by Java.

OPERATOR	DESCRIPTION
==	Check if two operands are equal
!=	Check if two operands are not equal.
>	Check if operand on the left is greater than operand on the right
<	Check if operand on the left is smaller than right operand
>=	Check if left operand is greater than or equal to right operand
<=	Check if operand on left is smaller than or equal to right

	operand

c) Logical operators

Java supports following 3 logical operators. The result of logical operators is a Boolean i.e. true or false.

OPERATOR	DESCRIPTION
&&	Logical AND
||	Logical OR
!	Logical NOT

Example:
Suppose a = true and b= false, then:
(a && b) is false
(a || b) is true
(!a) is false