Lecture Notes

Variables and Data Types

In this session, you learnt the following:

- Variables
- Primitive data types
- Arithmetic operations on various data types
- How to convert one data-type to another
- The string and character data types
- Arrays for storing more than one value in a single variable
- The Boolean data type and logical operators
- Types of errors and debugging

Variables

We started with **Variables**. You learnt what variables are, how you initialise and declare them, how you update and print their values, and last but not the least, you learnt some variable naming conventions that Java follows.

Data Types

We then moved to **data types**. There, you learnt about eight primitive data types. Those were byte (e.g. byte b = 50;), short (e.g. short s = 100;), int (e.g. int distance = 9000;), long (e.g. long l = 999999;), float (e.g. float f = 45.5f;), double (e.g. double d = 6.25;), char(e.g. char c = '5';), and boolean (e.g. boolean b = true;). You learnt the range and kind of the values each of these data types stores.

Arithmetic Operations

Then we covered **arithmetic operations.** You learnt the syntax used in Java for addition (+), subtraction (-), multiplication(*), and division (/). We discussed that dividing two integers always gives back an integer, and dividing 2 doubles will give back a double as well.

We told you that if you are looking to store an integer such as 2 as a double in a double variable, it will store it as 2.0. We also discussed the precedence order in which these arithmetic operators are used and justified the use of parentheses to prioritise a calculation. Finally, in Arithmetic Operations, you learnt how to take the user's input for some values. The Scanner class was used for this.

Casting

You then learnt about **casting** or type casting. Here, you learnt to convert one data type to another. To convert a double to an int, you wrote (int)<double variable name>, and you worked the other way around to convert an int to a double. This can be done with any other data type as well, and not just with int or double.

Strings

Next, you learnt about **strings**. You learnt that a string is a combination of characters, with each character having a unique index starting from 0. 'String' is the keyword used to declare a string variable, and anything you want to store as a string should be enclosed in double quotes.
Example: String s = "learn" initialises and declares the string variable 's' with the value "learn".

You then learnt about two methods that change the case of a string:

1. **toUpperCase** was used to convert the string to upper case.
   Example: System.out.println("ankit".toUpperCase()); will give the output ANKIT.
2. **toLowerCase** was used to convert the string to lower case.
   Example: System.out.println(UPGRAD.toLowerCase()); will give the output upgrad.

We introduced you to the substring function as well, which allowed you to access a part of the string using the indices of the characters. <variable name>. substring(lower range, upper range) was used for the same. Here, 'lower range' is the index of the character from where you want to start, and you access the string till the index **upper range -1.**
Example: For the string variable 's' with the value "learn", System.out.println(s.substring(0,4)) will give "lear" as the output.

Arrays

What followed next was **arrays.** We used them to store more than one value in a single variable. <datatype><arrayname>[] is the syntax used for declaring an array. The **square brackets** ensure that this is an array. Then, this was assigned to the values it was supposed to store by writing them inside the curly brackets ({}), separated by commas. For example, if you want to initialise and declare an integer array containing five elements, you can do it like this:
int numbers[] = {2,3,4,5,6};
Here, 2 will be stored at index 0, 3 at index 1 , 4 at index 2, 5 at index 3, and so on.

Another way of doing this, that we discussed, is something like this:
int numbers = new int[5];
numbers[0] = 2;
numbers[1] = 3;
numbers[2] = 4;, and so on.
Here, 0, 1, and 2 written inside the square brackets are the indices of the corresponding elements.

We also discussed that each element in an array has a unique index using which, it can be accessed. For example, to access the fourth element of an array 'named members', you write members[3] where 3 is the index of the fourth element.

Boolean Data Types

Then, we moved on to the **Boolean data type.** Here, you learnt how Boolean data types help you make decisions by storing a true or false value inside. You could also assign a condition to a Boolean variable, which would eventually be evaluated to TRUE or FALSE, depending on the condition.

You also learnt about three logical operators AND(&&), OR(||), and NOT(!). The following are the truth tables we discussed for AND and OR:

AND Operator Truth Table

Condition1	Condition2	Result
TRUE	TRUE	TRUE
TRUE	FALSE	FALSE
FALSE	FALSE	FALSE

As you can see in the truth table, if both the conditions connected by the AND operator are true, only then will the result be TRUE. If any of the conditions is false (Rows 2 and 3 of the table), the result is FALSE.

So, boolean(2 < 5 && 3 > 6); will return FALSE as one of the two conditions is false, and the AND statements will return FALSE if one of the two conditions is false.

OR Operator Truth Table

Condition1	Condition2	Result
TRUE	TRUE	TRUE
TRUE	FALSE	TRUE
FALSE	FALSE	FALSE

So, boolean(2 < 5 || 3 > 6); will return TRUE since 2 < 5 is true, and OR will return TRUE if one of the two conditions in the OR statement is true. However, boolean !(2<5||3>6) will return FALSE because the NOT operator will reverse the value of the Boolean variable next to the ! operator. For example, !(2 < 5) will return FALSE and !(3 > 6) will return TRUE.

After this, we discussed how these operators helped in condensing the complicated conditions to fewer lines while coding. Also, the precedence order in which these operators are used was discussed to be —

1. NOT
2. AND
3. OR

Again, the use of parentheses was discussed to prioritise any one of these operators.

Code Comments

We then moved on to Code Comments. You used code comments to make notes in codes. You can refer to these comments later to understand the different sections of the code that you're working on. You learnt that anything written after '//' till the end of the line is not compiled by Java, and Java ignores it. For more than one line, you used '/*' to start the comment, and once the comment was written, you used '*/' so that Java ignores anything written between /* and */ on multiple lines.

For example, in the following code, all the lines written after // are code comments which are ignored by the Java compiler.

```
package com.company;
public class Main {
    public static void main(String[] args) {
        //Declaring a variable
        int distance;

        //Initialising a variable
        distance = 0;

        //Updating a variable
        distance = distance + 200;

        //Displaying the current value of the variable
        System.out.println(distance);

        //Updating the distance variable by adding 200 to the original value
        distance = distance + 200;

        System.out.println(distance)
    }
}
```

Program Errors

Last but not the least, we discussed different types of errors. They were —

Syntax error: This error comes up when the grammar rules of Java are not followed and the code cannot be compiled completely by the compiler.

Example: int d = 500 will throw an error. This will happen because you missed a semicolon after 500.

Run-time error: This is an error that the Java compiler does not detect during the compile time. It is capable of causing the program to crash while running.

Example: If you ask the program to divide a number by zero, or you use an illegal index (e.g. negative index) on an array

Logic error: This error won't stop your program from compiling or cause it to crash while running. A logic error is an incorrect fundamental assumption about the program.

Example: You expect a decimal value when you divide two ints, such as 3/5, and the program returns 0 instead.

You then learnt how to run a program in debug code, which enables you to run the code step by step to find out where you made the logical error.