02. Java Fundamentals

Date
Page No.

(a) Basic Syntax

(i) Structure of a Java program

- Every file that ends with .java is a class.
  example → Main.java (class starts with capital letter)

⇒ A typical Java program is made up of classes and methods.

Basic structure :

```
                                    ← class name/filename → Helloworld.java
public class Helloworld {
public static void main (String[] args) { → main method
System.out.println("Hello, world!"); → prints the message
}
}
```

- Convert .java file to .class (Byte code)
  → javac filename
- To Run
  java Main or use intellij run.

→ Change location of .class file → javac -d ../classname/ ./classname.

- Explanation of class - (shortcut psvm)
  - public → It means, it can be accessed from anywhere.
  - class → class is name group of properties & functions.
  - Helloword → just the name of file.

psvm

Sout for

Date
Page No.

- Explanation of main function

public -> accessible from anywhere
static -> You don't need to create an object to call this method.
void -> The method does not return any value.
main -> This is the method that runs when the program starts.
String[] args -> A parameter that can take command-line arguments as a string array.

(ii) • Input / Output

- (outputs in java)
- Explanation of System.out.println("Hello, world");

System -> A built-in class in the java.lang package. It provides access to System-related functionality.

out -> A static object (specifically, a printstream) inside the System class. It represents the standard output (usually your screen).

println() / print() -> println() is a method of the printStream class. It prints the given message and moves to next line.

- print() also prints the give message but doesn't moves to next line.

Inputs in Java

```
import java.util.Scanner;
public class main() {
public static void main(String[] args) {
Scanner input = new Scanner( System.in);
System.out.println(input.nextInt());
}
}
```

(Before using scanner, need to import it)
(This tells Java that you want to use the scanner class from java.util package)
prints taken int input.
for int, for string → next()
just for word
for string line → nextLine()

(iii). Explanation of input taking

Scanner → is the name of a class that is use to take input from the user.

input → variable name

new Scanner (System.in) → creates a Scanner that reads input from the keyboard (standard input stream)

(iv) Comments

(1) // This is a single-line comment
2) /* */ This is a multi-line comment

(v) Identifiers

⇒ Name you give to classes, variables, methods, etc.

Rules:

a. Can use letters (A-Z, a-z), digits (0-9), underscore (_) & $.
b. Cannot begin with a digit.
c. Cannot be a key word.

(b) Data Types

~~int rollno = 64; → for integer numbers~~

⇒ In Java, datatypes define what kind of data (like numbers, text, etc) a variable can hold

- Two main categories of Data Types

1) Primitive Data Types (Basic Types)

These are built-in data types in Java. They are not objects - they store simple values directly in memory.

There are 8 primitive data types:

	Data Type	Stores	Example.
1.	int (4 byte)	Whole numbers	int age = 25;
2.	float (4 byte)	Decimal numbers (less precision)	float pi = 3.14f;
3.	double (8 byte)	Decimal numbers (more precision)	double rate = 99.99;
4.	Char	Single character (in single quotes)	char grade = 'A';
5.	boolean	True or false values	boolean ison = true;
6.	byte (1 byte)	Small number (-128 to 127)	byte a = 100;
~~7.~~	~~The Small Number~~		
7.	short (2 byte)	short number	short b = 1000;
8.	long (8 byte)	Big number (with L)	long big = 1234567890L;

- Characteristics of primitive Data types :-

(i) Fixed memory size

(ii) Fast and efficient

(iii) Not objects (can't call methods on them)

2. Non-primitive Data Types (Reference types)

These are not built-in values. They are objects created using classes.

	Data Type	Stores	Example
1.	String	Sequence of characters (text)	String name = "ravi";
2.	Array	Group of values	int[] marks = {90, 80, 25};
3.	Object	User-defined/custom types	Student s = new Student();

- Characteristics

(i) Can store multiple values or complex data.
(ii) Created using new keyword (except String)
(iii) Can call methods (like name.length())

(2) ⇒

(C) Java Operators

⇒ Operators are used to perform operations on variables and values.

→ Types of Operators

(1) Arithmetic Operators
(2) Assignment operators
(3) Comparison operators
(4) Logical operators
(5) Bitwise operators

- Explanation of Each

P.T.O

(1) Arithmetic Operators

⇒ They are used to perform common mathematical operations.

Operator	Name	Description	Example
+	Addition	Adds together two values	x + y
-	Subtraction	Subtracts one value from another	x - y
*	Multiplication	Multiples two values	x * y
/	Division	Divides one value by another	x / y
%	Modulus	Returns the division remainder	x % y
++	Increment	Increases the value of a variable by 1	++ x
--	Decrement	Decreases the value of a variable by 1	-- x

(2) Java Assignment Operators

⇒ They are used to assign values to variables.

Operator	Example / Same As
=	x = 5 / x = 5
+=	x += 3 / x = x + 3
-=	x -= 3 / x = x - 3
*=	x *= 3 / x = x * 3
/=	x /= 3 / x = x / 3
%=	x %= 3 / x = x % 3
&=	x &= 3 / x = x & 3
|=	x |= 3 / x = x | 3
^=	x ^= 3 / x = x ^ 3
>>=	x >>= 3 / x = x >> 3
<<=	x <<= 3 / x = x << 3

(3) Java Comparison Operators

=> They are used to compare two values (or variables). This is important in programming because it helps us to find answers and make decisions.

Operator	Name	Example
==	Equal to	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

(4) Java Logical Operators

=> They are used to determine the logic between variables or values.

Operator	Name	Description	Example
&&	Logical and	Returns true if both are true.	x<5 && x<10
||	Logical or	Returns true if one is true	x<5 || x<4
!	Logical not	Reverse the result, return false if the result is true	!(x<5 && x<10)

(5) Bitwise operators

(we will discuss this later).

P.T.O

(d) Java Type Casting and Conversion.

→ Type casting is when you assign a value of one primitive data type to another type.

- (i) Widening casting.

→ It automatically converts a smaller type to a larger type size.

byte → short → char → int → long → float → double

Ex →

```
int myInt = 9;
double myDouble = myInt; // Automatic Casting: int to double.
S.O.println(myInt); → without type casting → output 9
S.O.println(myDouble); → After type casting → output 9.0
```

- (ii) Narrowing Casting

→ It should be manually converting a larger type to a smaller size type.

double → float → long → int → char → short → byte.

Ex →

```
double myDouble = 9.78d;
int myInt = (int) myDouble;
```

manual type casting (double to int)

```
S.O.println(myDouble); → without type casting → output → 9.78
S.O.println(myInt); → After manual type casting → output → 9.
```

(e) Automatic Type promotion in java.

=> When you perform arithmetic operations on different data types (like int + float, or byte + short), Java automatically promotes smaller data types to larger compatible types to prevent data loss and ensure accuracy.

- Promotion Rules (Simplified)

1. All byte, short, and char are promoted to int when used in experience.
2. If one of the operands is long, the whole expression becomes long.
3. " " " " " " float; " " " " float.
4. " " " " " " double, " result becomes double.

- Examples (mixing Everything)

```
byte b = 5;
char c = 'A'; // ASCII value 65
int i = 10;
float f = 4.5f;
double d = 2.0;
double result = b + c + i + f + d;
// All are promoted to double
System.out.println(result); // output 86.5
```

Important: You'll Get a compile error if you Do this;

```
byte a = 10;
byte b = 20
byte c = a+b; * Error:
                result is int
```

Correct way

```
byte c = (byte)(a+b);
// Explicit Casting.
```