★ Lecture 4 - First Java Program - Input/Output, Debugging & Datatypes

#1.) Explaination of Java Program :-

While creating a Java Program we mainly do above mentioned 3 things :-

1.) Creating of Java file.
2.) Writing code in Java file
3.) Running code that is written in Java file.

Note :- We cover point 1 & 3 first and point 2 at last because every other topic is fall under point 2 like I/p, O/p, classes, constructor etc.

• 1.) Creating Java file :-

a.) Every file with .java extension is a class itself.
ex;- filename.java or Main.java

It is mandatory that this file contains a class called Main.

Note :- Name Convention :- It is imp. to name your class files as first letter capital.

ex;- class Main{} ✗
class Main { } ✓

b.) You can't write public to a class which name is not same as file name.

c.) Public keyword before class means that this particular class can be access from anywhere. We ~~can~~ talk about Access Modifiers (public, private, protected, final etc) later.

3.) Running code that is written in Java file :-

for running our source code of java we have to follow below 2 steps :-

1.) Compile Source code (Means convert .java file to .class file i.e Byte code) :-

Command for this :- javac -d bin src/filename.java

- javac ⇓ java compiler
- -d ⇓ destination to
- bin ⇓ class file storage folder
- src/filename.java ⇓ source code folder / name of .java file

b.) Run the compiled Byte code :-

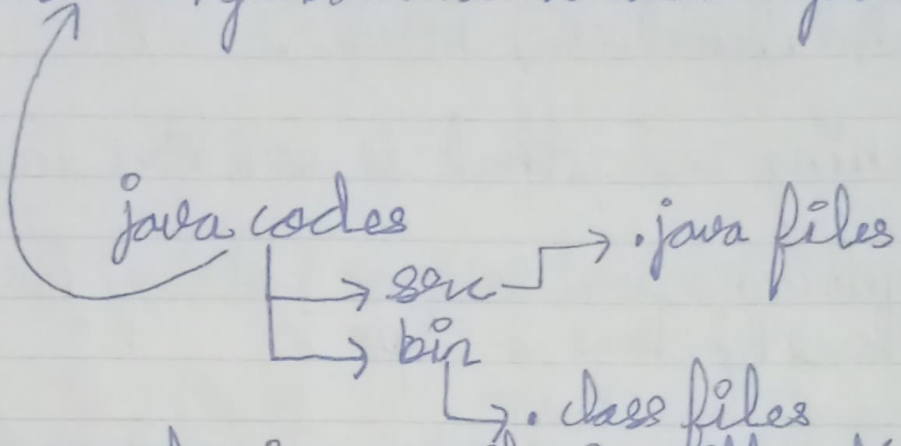
Command for this :- java -cp bin filename (or classname)

- -cp ⇓ class path
- bin ⇓ path where .class files are stored
- filename (or classname) ⇓ Name of class that you want to run (in case of filename and classname is same this only filename)

• ~~2.) Wri~~

Note :- Considering folders for above commands are,-

a.) Opened cmd / git bash at location java codes



b.) You can see where java compiler is with cmd :- where javac

• 2.) Writing code in Java file :-

⇒ Code 1 :- Code inside Main.java [or in P1HelloWorld.java]

```
package com.tejas ;
public class Main {
    public static void main(String [] args) {
        System.out.println("Hello world");
    }
}
```

→ Explainations of each word :-

① public :- Access modifier, necessary to write with class that have same name as filename to make it accessible from anywhere.

② class :- Necessary to write for making class in java
syntax class Classname { }
⇑
(capital cos of Naming convention

③ Main :- 1) It is a name of the class
2) written public before this means this is the master class (contains main method) and that's why named same as filename. Public cos main method's class should be accessed from anywhere

④ public :- Again access modifier and used here cos as main method is a starting point of program so isn't it obvious that it should accessed from anywhere cos it is necessary to start program for anywhere (file location) to run.

⑤ static :- static fu^ns & variables belongs to class
and non static " " " " " objects

so as main method is starting point it makes no sense that we make object of its class in order to access that cos it doesn't considered cos program is going to start from main itself and before it no code can execute so static indicates that this method belongs to class to avoid deadlock.

⑥ void :- It means this method returns nothing.

⑦ main() :- primary method from where code is start to execute.

⑧ (String[] args) :- This is commandline arguments

a) String[] :- It is an array, a collection of similar datatypes (here it is string)

ex:-
```
public class P2CmdArgs {
    public static void main(String[] args) {
        System.out.println(args[0]);
    }
}
```

O/p :- javac -d bin src/P2CmdArgs.java

java -cp bin P2CmdArgs 30

⇑ provided value to args[0]

→ 30 [as output]

ex:-
```
class P2CmdArgs1 {
    public static void main(String[] args) {
        System.out.println(args[1]);
    }
}
```

o/p :- javac -d bin src/P2CmdArgs1 &

java -cp bin P2CmdArgs1 30 "Kunal"

→ Kunal //as output.

So this 30, "Kunal" etc which are commandline arguments whatever i/p you are giving in terminal as an array it is stored in String[] args

array of strings

Note: ① Array starts from 0th index
so args[0] = 30
args[1] = Kunal as in previous example

② All these things i.e javac, java, git, python3 etc are executable files that are located somewhere in your computer and we can see location with cmd :-

where javac
→ output address where it is located

we can see list of files under the outputed-addderess by cmd :-

ls output address
or
ls output address | grep javac (this is for filter only javac from list)

we can open this by cmd;-

open output address

So basically when we write, javac it only double clicking on the file that is located at the output address.
we can also write whole path like

output address / javac -d bin src/P1HelloWorld.java

But it is very tedious so to avoid that we give path in environment variables

whenever we write javac or git or python computer sees that if it is available in environment variables folder then run it otherwise not...

we can see environment variables folder location by cmd;-

$PATH

So Environment variable is only a folder addresses in which your computer will check whether the command that you have written is available or not.

(9) package ~~com~~ com.tejas; ;- Package is nothing but a folder to provide extra layer of security to sudden files

Ordinary way :- src → .class files

During Package :- src → com → tejas → P1HelloWorld.java, P2CallArgs.java
→ Kunal → P3trial.java, P4try.java

package com.tejas;

package com.Kunal;

(10) System.out.println("Hello World"); ;-

There are various predefined methods & classes which are already given by java to us for printing, i/p, debugging etc

System class contains all those methods & classes (press ctrl + click on System in your code in intellij to see code of System class)

2 things :-

1.) Outputs in java :-

System.out.println("Hello World");

System has a variable called out. out which is of type PrintStream and this out has a ~~xxx~~ method called println.

PrintStream.java

available to these files

System.Out.println("Helloword");

in commandline print (whatever in brackets)

So Obviously out is of type PrintStream means println method is also defined in PrintStream

We are accessing println through out from Print Stream. So out is like a reference variable for Println

Note;- println for new line
print to continue in same line

2) Inputs in Java;-

for output → System class and it is inside java.lang package
import java.lang.System;

for input → Scanner class and it is inside java.util package
import java.util.Scanner;

Scanner input = new Scanner(System.in)

means take i/p from keyboard
or,
we can also put filename here to provide i/p from filename

you can also take input / print o/p from file

ex:- for o/p ⇒ out = file.txt;
System.out.println(·);

for I/p ⇒ Scanner input = new Scanner(~~System.in~~ fileInputStream);

Code:- P3Scanner.java

```
package com.tejas;
import java.util.Scanner;
public class P3Scanner {
    public static void main(String[] args) {
        ~~System.out.println(~~
        Scanner ~~new~~ input = new Scanner(System.in);
        System.out.println(input.nextInt());
    }
}
```

o/p:- 456 //(Put int)
→ 456 //as o/p

⇒ Q: Also make program to take input from file

Code:- P4System.java

Simple hello world + Make program to take i/p from file and print its values

#2.) Primitive Datatypes in Java :-

Primitive datatypes are those which cannot further breakable into any basic datatypes unlike that primitive datatypes.

Datatypes

- Primitive
  ex :-

Bytes →	1	2	4	8
	Byte Boolean	Short Char	Int float	long Double

- Non-Primitive
  ex :- String
  ⇓
  as it can break into char.

Note:- 1.) Char represented by ' ' ex:- char b = 'a';
String represented by " " ex:- String s = "Tejas";

2.) for alternative of int to store large values we use long
" " " float " " " " " " double

3.) With float & Long we use small case f and Large case L respectively after values ex:-

```
float marks = 98.67f;
long largerInteger = 367895498549 7L;
```

4.) By default all decimal values are of type double that's why while using float we have to put f after value to specify float type.

Similary int is default and long with L at end specify long datatype

5.) We can also create classes of primitive datatypes that are called wrapper classes (see in detail in OOP).
ex:-

```
int rno1 = 64; // primitive datatype
Integer rno2 = 64; // wrapper class
```

Why wrapper class?
With, rno1 we get only basic functionality ex:- rno1.arg
but " rno2 " " More " ex:- rno2.doubleValue() cast etc
equals()
toString() etc

Code for primitive datatypes → P5Datatypes.java
~~P6Datatypes.java~~
~~P6Datatypes.java~~

```
package com.tjas;
public class P5Datatypes {
    public static void main(String[] args) {
        int rno = 64;
        char letter = 'r';
        float marks = 98.67f;
        double largeDecimalNumbers = 4567654.4567;
        long largeInteger = 3456798453218 9L;
        boolean check = false;
        Byte singlInput = 7;
        short simpleno = 150;

        Integer rno2 = 64; // like this we can create
                              wrapper class of all
                              primitive datatype (with
                              1st letter capital)
    }
}
```

#3.) Comments in java:-

Comments are sentences that used to explain codebase and they are ignored by compiler.

2 types:-

1.) Inline Comments :- // Ex: of Inline comment
2.) Multiline Comments :- /* Ex
of Multiline
comment */

#4.) Literals & Identifiers :-

→ Literals are synthetical representation of boolean, numeric, character or string data
or
Literal means a constant value assigned to a variable
ex:- int a = 10 ;
⇑
Literal

→ Identifiers are used for Identification purposes. An identifier can be a class name, method name, variable name, or label.

Identifier Rules :-

1.) Allowed characters are Alphanumeric characters i.e
[A-Z], [a-z], [0-9], '$', _
ex:- int tejas@ = 5; // error cos contain @ i.e Special character

2.) Should not start with digits i.e [0-9]
ex:- int 384tejas = 5; // error ✗
int t384jas = 5; // ✓

3.) Identifiers are case-sensitive.

4.) No limit on length but advisable to use an optimum length of 4-15 letters only

5.) Reserved words can't be use as identifiers eg:- while, for, abstract, int etc.
eg:- int while = 5; // ✗

<u>Note</u>; (1.) int a = 234,000,000; // gives errors code, is not allowed

instead use, int a = 234_000_000; // ✓

(2.)
```
next();          // for first word till space
nextLine();      // for whole inputted line
nextInt();       // for next integer value
nextFloat();     // "  "  "  float  "  (decimal)
nextDouble();    // "  "  "  double "  "
nextByte();      // "  "  "  byte   "
.
.
.
hasNext();       // for checking true or false (whether i/p value from keyboard is true or false)
hasNextInt();    // whether i/p value from keyboard is int or not.
.
.
.
```

(3) ~~int → float~~

#5.) Type Conversion & Type casting & Type promotion in Expressions :-

→ Type Conversion :- (Deal with I/p at runtime)

Conversion of data of one datatype into another datatype (both should compitible with each other) is called Type conversion.

- 2 cond^n in which type conversion happen :-

1.) Both datatype in type conversion should compatible.
   ex :- int & float are compatible (both for no.'s)
   float & char . " not " .

2.) ~~Source~~ Destination type should be greater that source datatype.
   ex :-
   Byte → Short → Int → Long → Float → Double
   → Datatype size is increasing

Widening or Automatic conversion

Code :- P6 TypeConversion.java

```
package com.tejas;
import java.util.Scanner;
public class P6 TypeConversion {
    public static void main (String [ ] args)
    {
```

→ ~~Type Casting~~ :-

```
    Scanner sc = new Scanner(System.in);
    float a = sc.nextFloat();
    System.out.println(a);
  }
}
```

O/p :- I/p :- 54 // int so print as it is compatible with float
O/p :- 54

I/p :- 'd' // char error not compatible
O/p :- Error

I/p :- 54382479.86589 // double means compatible but destination (~~float~~ double) is short than source (float) so get's round off (i.e floating point error)
O/p :- 54382486E7 (gets round off)
or
I/p :- 764378.987548
O/p :- 764379.0 (gets round off)

Note :- ① Type Conversion deal with i/p values at runtime,
If cond^ns satisfy then ok otherwise do narrowing automatically like float to double

② Type Casting deal with i/p values at compile-time,
means, float a = 764378.987548;
System.out.println(a); // it gives error that double can't store in float

→ ~~If compatible then manual casting conversion required~~

→ ~~If not compatible then automatically converted~~ even in type casting

→ Type Casting :- (Deal with compile time)

forcefully (Explicitly) converting one datatype to another (compatible ~~or not~~ automatically doesn't matter) is called Type casting.

~~Code :- P7TypeCasting.java~~

• 2 cond^n in which Type casting happen :-

1.) In case of Compatibility, manual ~~conversion~~ casting required.

2.) In case of ~~or~~ Non-Compatibility, automatically casting done by java.

2.) In this case, Destination type is shorter than source type.

Double → float → long → Int → Short → Byte

→ Datatype size is decreasing

Narrowing or Explicit Type Casting

Code :- P7TypeCasting.java

```
package com.tejas;
import java.util.Scanner;

public class P7TypeCasting {
    public static void main(String[] args)
    {
```

```
Scanner sc = new Scanner(System.in);
int a = 'a'; //char not compatible → Automatically converted
int f = (int) (7872073.7347); compatible → Manually "
float s = (float) (7878623.7673); compatible → " "
System.out.println(a);
                   "      (f);
                   "      (s);

}
}
```

O/p :- 97
7872073 (gets round off
7878624.0 (gets roundoff)

→ Type Promotion in Expressions :-

- Rules of Type Promotion :-

1.) All Byte, short, char values are promoted to integer.

ex :- ①
```
byte a = 40;
byte b = 50;
byte c = 100;
int d = a * b / c;
```
is 2000 so how it stored in byte as a and b are byte?
It's not cos byte convert to int automatically

②
```
byte b = 50;
byte b = b * 2;
```
Here b*2 converted to int and int can't store in byte i.e b that's why we get error.

③ Similarly, int no = 'A';
System.out.println("no");

O/p :- 65 // cos char also ~~converted~~ promoted to int ~~bydefaut~~ automatically by java.

2.) If any one of the operands is long or float or double (biggest) promotion the whole operation converted to that datatype.

ex :- Sout(3 * 5.6);
⇑ int ⇑ float

O/p :- 16.7999...7// float

3.) If during type promotion, destination is so short that its max^m value limit exceeds by the value it assigned then O/p is modulus b/w Assigned value % max^m value = Remainder

ex :- int a = 257;
byte b = (byte)(a); // 257 % 256 = 1
⇑ Assigned value ⇑ Max^m value of Byte ⇑ Remainder of Modulus.

For an example of above rules check - P9 TypeProEx.java

#6.)

Note :- ① Java follows UNICODE (Universal international standard character coding)

ex;- Sout("नमस्ते");

O/p ;- नमस्ते

② Type checking

a.) Primitive datatypes :-

```
int k = 5;
Sout((Object) k).getClass().getSimpleName());
```

b.) Non Primitive datatypes ;- [Arrays, Strings etc]

for String { String s = "Tejas Singh";
Sout(s.getClass().getSimpleName());

for Array { int[] arr = new int[5];
Sout(arr.getClass().getSimpleName());