WHY Java ?

1. Platform or architecture independent

  - Supports Write once run anywhere philosophy.

  - Refer to diagram : "Java application development.png"

2. Simple & robust

3. Secure

4. Automatic memory management.

5. Inherent Multi-threaded support

6. Object Oriented support -- Encapsulation, Inheritance , polymorphism & abstraction

7. Excellent I/O(Input Ouptput) support

8. Inherent networking support for TCP/IP , UDP/IP programming & for URLs

9. Supports Functional programming

10. Supports web programming

& many more ...

Development & Execution of java application.

Objective 1 :   Create a java applicationn to display welcome message on the console.

Steps

1. Create a folder under Java workspace

eg : <day1>

2. Create 2 sub folders : src & bin

src : for Java source files

bin : for .class files (compiled output)

3. Write Java src file (eg. First.java) under <src>

```
class First {
 public static void main(String[] args)
 {
   System.out.println("Hello Java !");
 }
}
```

Brief Explanation –

Java is fully encapsulated language, meaning all of the data members & code has to be declared within a class.

public static void main(String[] args) – is the starting point of execution of Java application.

Argument of main method: String[] args

- represents command line arguments. (equivalent to argv in C)

The default package, inherently available to all java classes - java.lang

(You don't have to explicitly import it in your code)

System - name of the class, from java.lang package

out - static data member of the System class

Its data type : java.io.PrintStream - class

System.out => standard o/p stream (equivalent to printf in C | cout in C++)

System.in => standard i/p stream (equivalent to scanf in C | cin in C++)

print | println | printf - overloaded methods of PrintStream class

4. How to compile Java code?

Open command prompt in a folder : <day1>/src

Type -

```
javac -d ..\bin First.java
```

Java compiler Option

-d implies name of the folder , to place .class files.

5. How to run / launch java program(application) ?

Java classes are stored under <bin> folder.

```
 cd ..\bin
 java HelloWorld
```

(syntax - java Name of the class containing main method)

Objective 2 : Write Java application , to say hello to user. User name will be supplied as command line argument.

(eg : java SayHello Madhura

Desired O/P - Hello , Madhura !)

To refer to command line arguments

- Use args[0] , args[1] etc.

3. JDK , JRE & JVM

- (refer to diagram - JDK vs JRE vs JVM)

4. **Naming conventions in Java (Follow these strictly)**

1. For the class names- 1st letter of 1st word must start with upper case & then follow with camel case notation.

eg : HelloWorld

2. For data members and methods(functions) - 1st letter of 1st word must start with lower case & then follow camel case notation

eg : performanceIndex

calculateSalary()

3. For constants - use all uppercase.

eg : PI

5. **Java Tokens**

- Smallest elements of a program that is meaningful to the compiler. They are also known as the fundamental building blocks of the program. Tokens can be classified as follows:

1. Keywords
2. Identifiers
3. Constants/Literals
4. Operators
5. Separators

   Eg.

   ```
   class Student {
        int age = 20;
       String name = "Riya";
       double result=13.24*1.5;
   }
   ```

   Keywords: class, int, String

   Identifiers: Student, age, name

Literals: 20, "Riya"

Operators: =,*

Separators: { } ;

5.1 Rules on Identifiers

1. Identifiers must start with a letter, a currency character ($), or a connecting character such as the underscore (_),   cannot start with a number!

2. Can't use a Java keyword as an identifier.

3. They are Case sensitive

6. Legal Access specifiers for data members & methods

private : visible within the same class

default(package private) : no access modifier is required.

-visible within same package(i.e same folder)

protected : accessible within the same package & accessible to sub classess via inheritance , from any package.

public : accessible from anywhere.

7. Legal class level access specifiers -

1. default – class is accessible only from the same folder.

OR

2. public (class is accessible from anywhere)

Objective 3 : Accept 2 numbers as command line arguments , add them &

display the result.

8.Explore Java API documentation (java docs)

java.lang   is the default package name.

In java.lang package , Integer is a built in class

It's method -

public static int parseInt(String s) throws NumberFormatException

Input (i/p) - String

Output (o/p) - int

It will raise(throw) run time error(exception) , in case of un parseable integers.

Tip

Refer to java docs for any explanation of classes/methods..

8. **Introduce Scanner class** from java.util package

What is Scanner ?

A class (java.util.Scanner)   that represents text based parser(It has inherent small ~ 1K buffer for storing text)

It can parse text data from any source

 -Console input(System.in) |Text file | socket| string

Steps for attaching scanner , to accept inputs from User.(UI)

1. import java.util.*; or import java.util.Scanner;

2. Create instance of Scanner class(object)   using Scanner class constructor

public Scanner (InputStream in)

System.in represents - standard input (stdin , similar to cin)

usage -- Scanner sc=new Scanner(System.in);

3. Some methods of the Scanner class , to check data type , of the next token

available with Scanner. But these method won’t remove the token from the Scanner.

public boolean hasNextInt(),

public boolean hasNextByte(),

public boolean hasNextLong()

etc.

4. To actually read & parse data

public int nextInt() throws InputMismatchException

public double nextDouble() throws InputMismatchException

public boolean nextBoolean() throws InputMismatchException

To read the next token without parsing

public String next() throws NoSuchElementException

To read next line -

public String nextLine() throws NoSuchElementException

5. Before terminating application close scanner.

public void close();

Objective 4 : Accept 2 double values as user input ,divide them & display the result.

Objective 5 : Accept 2 int values as user input n compare them

If num1 < num2 , display mesg (1st no is < 2nd no)

If num1 > num2 , display mesg (1st no is > 2nd no)

If num1 == num2 , display mesg (1st no is same as 2nd no)

Objective 6 : Accept begin n end value from user(using scanner)

Print all odd numbers in the range.

Objective 7 : (while , switch-case)

Write java application , to accept month no from user , using Scanner

(Range 1-12 => Jan -Dec)

Display the name of a season accordingly

11,12,1,2 : Winter

3,4,5,6 : Summer

7,8,9,10 : Monsoon

The program should only exit , with error message , if user enters the invalid month no)

Objective 9: Write java application to accept int(emp id) , double(salary) , emp's first name(string) , is permanent : boolean from Scanner & display the same using printf

9. **Basic rules**

1. Java compiler doesn't allow accessing of un initialized data members.

Test it !

2. A java source file can have more than one non public class(default)

There is no restriction on name of the src file & name of the class.

3. There can be only one public class per source code file.

Public class name must match with the src file name.

Eg. If a class is declared as public class Example {....}

It must be in a source code file named Example.java.

10.

Java is

- **statically & strongly typed language**.
- It means you must declare variable types explicitly or even while using var keyword , javac ensures type safety , prevents run time errors
- Eg. int data=12345;
  data=”hello”; // javac error !

On the other hand , Javascript or Python
- **is dynamically & weakly typed**
- types are checked at run time & allows implicit conversions

Java is statically typed and a strongly typed language because

1. In Java, each type of data (such as integer, character, boolean etc) is predefined as part of the JLS (java language specification)

2. All variables or constants must be defined with one of the data types.

11. **Java Data Types**

- refer to diagram - " java-data-types.png"

Java has two categories in which data types are segregated .

1. Primitive Data Type: boolean, char, int, short, byte, long, float, and double

- value holding types

2. Non-Primitive Data Types : Reference Data Types

- address holding types

Types of Primitive Data Types

1. boolean

boolean data type represents only either true or false   BUT the size of the boolean data type is JVM specific

Default Value: false

2. byte

The byte data type is an 8-bit signed two's complement integer. The byte data type is useful in binary file handling or networking application.

Range of Values: -128 to 127 (-2^7 to 2^7-1)

Default Value: 0

3. Short

The short data type is a 16-bit signed two's complement integer.

Range of Values: -32, 768 to 32, 767 (-2^{15} to $2^{15}-1$)

Default Value: 0

4. int

It is a 32-bit signed two's complement integer.

It is a default data type in integer data types.

Range of Values: -2,147,483,648 to 2,147,483,647 (-2^{31} to $2^{31}-1$)

The default value is 0

Note : In Java SE 8 and later, we can use the int data type to represent an unsigned 32-bit integer, which has a value in the range [0--2^32-1] ,   using Integer class methods

5. long

The long data type is a 64-bit two's complement integer and is useful for those occasions where an int type is not large enough to hold the desired value.

Size: 8 bytes (64 bits)

Range of Values: -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 (-2^{63} to $2^{63}-1$)

Note: The default value is 0l.

Note : In Java SE 8 and later, you can use the long data type to represent an unsigned 64-bit long, which has a minimum value of 0 and a maximum value of 2^64-1.

6. float

The float data type is a single-precision 32-bit IEEE 754 floating-point. Use a float (instead of double) if you need to save memory in large arrays of floating-point numbers.

Size: 4 bytes (32 bits)

Range of values ~$\pm 3.4 \times 10^{38}$

Note: The default value is 0.0f

7. double

The double data type is a double-precision 64-bit IEEE 754 floating-point. For decimal values, this data type is generally the default choice.

Size: 8 bytes or 64 bits

Range of values ~ 4.9×10^{-324} to 1.8×10^{308}

The default value is taken as 0.0

Both float and double data types were designed especially for scientific calculations, where approximation errors are acceptable. If accuracy is the most prior concern then, it is recommended not to use these data types and use BigDecimal class instead.

8. char

The char data type is a single 16-bit insigned Unicode character.

Size: 2 bytes (16 bits)

Range of Values: ‘\u0000’ (0) to ‘\uffff’ (65535)

Note: The default value is ‘\u0000’

Why is the size of char 2 bytes in Java?

Other languages like C/C++ use only ASCII characters, and to represent all ASCII characters In that case - 8 bits are sufficient.

But java uses the Unicode system(universal encoding system) not the ASCII code system

To represent the Unicode system 8 bits is not enough to represent all characters so java uses 2 bytes for characters.

Unicode defines a fully international character set that can represent most of the world's written languages. It is a unification of dozens of character sets, such as Latin, Greek, Asian Languages and many more.

Why long → float is automatic (implicit / widening conversion) done by javac ?

- A float can represent **a wider *range*** of values than long.

-long range: ~$\pm 9.22 \times 10^{18}$

-float range: ~$\pm 3.4 \times 10^{38}$
So, every possible long value **fits within the range** of float(Meaning no *overflow risk*)

But in such conversion **, precision may be lost**.

- long is **exact** (integer up to 19 digits).

- float has only ~ 6–7 decimal digits of precision.
So, converting large long values may lose lower digits.

2nd Category of Java data types

Non-Primitive Data Type or Reference Data Types

The Reference Data Types will contain a memory address of objects, which will be created on heap

3 sub Types

1. Class type of reference

2. Array Type of reference

3. Interface type of reference

How are numbers stored in computer's memory ?

A positive number -

eg : 25

Hex : 0x19

Binary representation: 0001 1001

eg : -25

Using 2's complement = 1's complement + 1

```
     1110 0110   1's complement
    +         1
     -------------
      1110 0111
```

Pointers vs. Java references

Pointer arithmetic is not allowed in java

- Meaning you can't use any arithmetic operators with reference type of variables.

  Reference holds internal representation of address (equivalent to object pointer in c++)

11. **Conversions regarding primitive types**

Automatic conversions(Widening conversions or Automatic promotions)

1. byte→short→int→ long→float→double

eg - byte b1=100;

```
       short s1=b1;
```

2. char → int → long→float→double

```
     eg -char ch=65;

         int i=ch++;
```

3. float →double

What are the rules of automatic conversions?

Source & destination operand types must be compatible. Typically, destination data type must be able to store larger magnitude of values than that of source data type.

1. Any arithmetic operation involving byte or short will be automatically promoted to int

2. With int & long operands , resulting data type is long

3. With long & float , resulting data type is float

4. With byte,short,int , long , float & double, resulting data type is double.

Narrowing conversion is also known as forced conversion or type-casting. It has to be applied by programmer (javac cannot perform it automatically)

eg -

double → int

float → long

double → float

12. **Java Operators**

Arithmetic Operators

Unary Operators

Assignment Operator

Relational Operators

Logical Operators

Ternary Operator

Bitwise Operators

Shift Operators

Arithmetic Operators: They are used to perform simple arithmetic operations on primitive data types.

* : Multiplication

/ : Division

% : Modulo

+ : Addition

– : Subtraction

Unary Operators: Unary operators need only one operand. They are used to increment, decrement or negate a value.

– :**Unary minus**, used for negating the values.

eg : int a=20;

int b=-a;

++ :**Increment operator**, used for incrementing the value by 1. There are two varieties of increment operator.

Post-Increment : Value is first used for computing the result and then incremented.

Pre-Increment : Value is incremented first and then result is computed.

eg : int n1=10;

int n2=n1++;

System.out.println(n2+" "+n1);//10 11

-- : **Decrement operator**, used for decrementing the value by 1. There are two varieties of decrement operator.

Post-decrement : Value is first used for computing the result and then decremented.

Pre-Decrement : Value is decremented first and then result is computed.

! : **Logical not operator**, used for inverting a boolean value.

eg :

boolean jobDone=true;

boolean flag=!jobDone;

System.out.println(flag);

Assignment Operator : ‘=’ Assignment operator is used to assign a value to any variable. It has a right to left associativity.

eg : int a=200;

In many cases assignment operator can be combined with other operators to build a shorter version of statement called **Compound Statement**.

eg : int a=100;

a += 10;

System.out.println(a);

+=, for adding left operand with right operand and then assigning it to variable on the left.

-=, for subtracting left operand with right operand and then assigning it to variable on the left.

*=, for multiplying left operand with right operand and then assigning it to variable on the left.

/=, for dividing left operand with right operand and then assigning it to variable on the left.

%=, for assigning modulo of left operand with right operand and then assigning it to variable on the left.

Relational Operators : These operators are used to check for relations like equality, greater than, less than. They return boolean result after the comparison and are used in looping statements and conditional if else statements.

==, Equal to : returns true if left hand side is equal to right hand side.

!=, Not Equal to : returns true if left hand side is not equal to right hand side.

<, less than : returns true if left hand side is less than right hand side.

<=, less than or equal to : returns true if left hand side is less than or equal to right hand side.

>, Greater than : returns true if left hand side is greater than right hand side.

>=, Greater than or equal to: returns true if left hand side is greater than or equal to right hand side.

Logical Operators : These operators are used to perform “logical AND” and “logical OR” operation, i.e. the function similar to AND gate and OR gate in digital electronics. One thing to remember is the second condition is not evaluated if the first one is false, i.e. it has a short-circuiting effect. Used extensively to test for multiple conditions for making a decision.

Conditional operators are-

&& : Logical AND : returns true when both conditions are true.

|| : Logical OR : returns true if at least one condition is true.

eg :

```
int data=100;

int data2=50;

if(data > 60 && data2 < 100)

 System.out.println("test performed...");

else

 System.out.println("test not performed...");
```

Ternary operator : Ternary operator is a shorthand version of if-else statement. It has three operands and hence the name ternary.

General format is- condition ? if true : if false

The above statement means that if the condition evaluates to true, then execute the statements after the ‘?’ else execute the statements after the ‘:’.

eg :

```
int data=100;

System.out.println(data>100?"Yes":"No");
```

Bitwise Operators : These operators are used to perform manipulation of individual bits of a number. They can be used with any of the integer types. They are used when performing update and query operations of Binary indexed tree.

& : Bitwise AND operator: returns bit by bit AND of input values.

| : Bitwise OR operator: returns bit by bit OR of input values.

^ : Bitwise XOR operator: returns bit by bit XOR of input values.

~ : Bitwise Complement Operator: This is a unary operator which returns the one’s compliment representation of the input value, i.e. with all bits inversed.

eg :

```
    int a = 3; // 0011 in binary

    int b = 6; // 0110 in binary
```

int c = a | b; // 0111 in binary (7)

int d = a & b; // 0010 in binary (2)

int e = a ^ b;//0101 in binary (5)

Shift Operators : These operators are used to shift the bits of a number left or right thereby multiplying or dividing the number by two respectively. They can be used when we have to multiply or divide a number by two.

<< : **Left shift operator**: shifts the bits of the number to the left and fills 0 on voids left as a result. Similar effect as of multiplying the number with some power of two.

eg :

int a = 25;

System.out.println(a<<4); //25 * 16 = 400

a=-25;

System.out.println(a<<4);//-25 * 16 = -400

Signed right shift operator

The signed right shift operator '>>' uses the sign bit to fill the trailing positions.

Eg. if the number is positive then 0 will be used to fill the trailing positions and if the number is negative then 1 will be used to fill the trailing positions.

Assume if int a = 60 and int b = -60; now in binary format, they will be as follows −

a = 0000 0000 0000 0000 0000 0000 0011 1100

b = 1111 1111 1111 1111 1111 1111 1100 0100

As , in Java, negative numbers are stored as 2's complement.

Eg. a = 0000 0000 0000 0000 0000 0000 0011 1100

b = 1111 1111 1111 1111 1111 1111 1100 0100

Thus a >> 1 = 0000 0000 0000 0000 0000 0000 0001 1110   (30)

And b >> 1 = 1111 1111 1111 1111 1111 1111 1110 0010   (-30)

Unsigned right shift operator

The unsigned right shift operator '>>' does not use the sign bit to fill the trailing positions. It always fills the trailing positions by 0s.

Thus a >>> 1 = 0000 0000 0000 0000 0000 0000 0001 1110   (30)

And b >>>   1 = 0111 1111 1111 1111 1111 1111 1110 0010   (a large +ve value : 2147483618)