Java

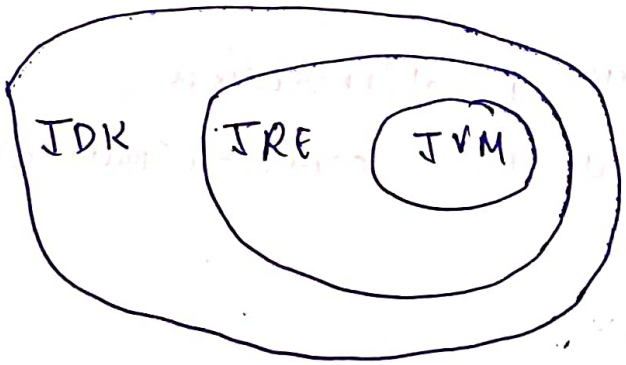
* Java is an object oriented programming language developed by Sun Microsystem of USA in 1991

  JAVA - purely object oriented.

* Java is compiled into the bytecode & then it is interpreted to machine code.

* 

*



JDK - (Java development kit) developers tools

JRE (Java Runtime Env.) libraries & toolkit

JVM - (Java Virtual Machine) runs java programs

JDK — Collection of tools used for developing & running the Java program.

JRE - Helps in executing programs developed in Java.

* Java is a case sensitive language
* Java is a platform independent (write once run everywhere)

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

do not refuse anything

Main functions

→ Helps in running this main functions without creating the objects of this class

can be access from anywhere

- For classes we use PascalConvention
- For functions we use camelCaseConvention
- // - comment
- /* - multiline comment */

Datatypes

primitive (intrinsic)

- (4byte) int
- (8-byte) long
- (8-byte) double
- (2-byte) short
- byte (1-byte)
- float (4 bytes)
- char (2-byte) — because it supports UNICODE
- bool (1-bit)

Non-primitive (derived)

* Unlimited
* store an address
* could hold more than 1 value
* more memory
* slower

* Variables - It is a container that stores a value

```
int number = 8;
```

int → data type; number → variable name; 8 → value it stores

```
float a = 1.2f;
double b = 4.6D;   or   4.6;
long c = 111111111111l;   or   111111111111L;
char symbol = '@';
String name = "Ash";
```

* Reading data from the keyboard

In order to read data from the keyword, Java has a scanner class.
Scanner class has a lot of methods to read the data from the keyboard

```
Scanner s = new Scanner(System.in);
```

↳ read from the keyboard

```
int a = s.nextInt();
```

↳ method to read from the keyboard (integer in this case)

* x = 3.14
z = 3.9

y = Math.round(x); → 3.0

y = Math.ceil(x); → 4.0

y = Math.floor(z); → 3.0

* after int or float ↓

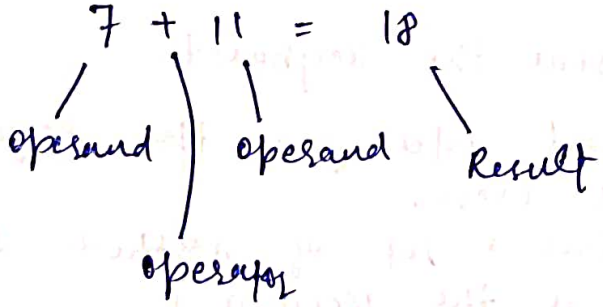
```
float b = input.nextFloat();
input.nextLine();
String c = input.nextLine();

System.out.println("..."+b);
                   ,,    ,,+c);
```

→ important if using String just because it clears the integer cursor

Operators

* Operators are used to perform operations on variables & values.



* Arithmetic operators → +, -, *, /, %, ++, --
Assignment ,, → =, +=
Comparison ,, → ==, >=, <=
Logical ,, → &&, ||, !
Bitwise ,, → &, | (operates bitwise)

- Asithmetic operators cannot works with booleans
- % operator can work on floats & doubles

* precedence of operators
  - (+, -) < (*, /)
  - In case we like to change this order, we use parenthesis.

* Associativity
  - left to right
  - it tells us the direction of execution of operators
  - */ → L to R
    +- → L to R
    ++, = → R to L

* Random no.

```
import java.util.Random;
public class RandomNum {
    public static void main(String[] args) {
        Random random = new Random();
        int n = random.nextInt(6) + 1;
        System.out.println(n);
    }
}
```

Random no b/w 1 to 6

* Resulting Datatypes after arithmetic operation

R = b+s → int
R = s+i → int
R = l+f → float
R = i+f → float
R = c+i → int
R = c+s → int
R = l+d → double
R = f+d → double

b → byte
s → short
i → integer
l → long
f → float
d → double
c → character

* increment / decrement

i++; first assign & then incremented
++i; first increment & then assign

Strings

A string is a sequence of characters.

- String name;
  name = new string("Ashish");

OR

- String name = new string("Ashish");

OR

- String name = "ashish";

* String is a class but can be used like a data type:

(Strings are immutable & cannot be changed)

String name = "Ashish";

name → reference
"Ashish" → object

* different ways to print string in java.

- System.out.print() → no new line at the end
- System.out.println() → Prints a new line at the end.
- System.out.printf()
- System.out.format()

System.out.printf("%c", ch)

↳ %d for int
%f for float
%c for char
%s for string

* Strings methods.

String name = "Hello";
0 1 2 3 4

1.) name.length() → Returns length of string name. (5 in this case)
2.) name.toLowerCase() → returns a new string which has all the lowercase characters.
3.) name.toUpperCase() → returns a new string which has all the uppercase characters
4) name.trim() → returns a new string after removing all the leading & trailing spaces from the original string

5.) name.substring(int start) → returns a substring from start to the end. String(3) returns → "lo".
[index starts from 0]

6.) name.substring(int start, int end) → returns a substring from start index to end index. Start index is included & end is excluded.

7.) name.replace('l', 'o') → returns a new string after replacing l with o, Heooo in this case.

8.) name.startsWith("He") → returns true if name starts with string "He."

9.) name.endsWith("lo") → returns true if name ends with string "lo".

10.) name.charAt(2) → returns character at a given index position.

11.) name.indexOf(el) → returns the index of the given string. ex- name.indexOf("el") returns 1 which is the first occurence of 'el' in string "Hello", -1 otherwise

12.) name.indexOf("s", 3) → returns the index of the given string starting from the index 3(int).

14.) name.lastIndexOf("l", 2) → returns the last index of the given string before index 2

13.) name.lastIndexOf("l") → returns the last index of the given string

15.) name.equals("Hello"). → returns true if the given string is equal to "Hello".

16.) name.equalsIgnoreCase("Hello") → returns true if two strings are equal ignoring the case of characters.

* Escape sequence

(" \" ");

↓

works as double quote

(" \\ ")

Conditionals

* if-else statement

```
if (condition) {
    statement;
}
else {
    statement2;
}
```

* Relationals

==, >=, >, <, <=, !=

equals, greater than equal to, not equal

* logical operators

&& → AND
|| → OR
! → NOT

Y && Y = Y	Y || Y = Y
Y && N = N	Y || N = Y
N && Y = N	N || Y = Y
N && N = N	N || N = N

- else if clause

```
if (condition 1) {
    }
else if (condition 2)
    {          }

else
    {
        }
```

* Switch case control instruction

```
switch (Var) {
    case c1:
        // code
        break;
    case c2:
        // code
        break;
```

```
case 'C3':
    // code
    break;
default:
    // code
}
```

- A switch can occur within another but in practice this is rarely done.

* Another method (Enhanced switch)

```
switch (var) {
    case 'C1' -> {
        // code
    }
    case 'C2' -> {
        // code
    }
    default ->
    {
        // code
    }
```

- No break is needed

Loops

* While loop
  - While(conditions) {
    // code
    i++;
    }
  - while (1) is not valid java.

* do-while
  - do {
    // code
    }
    while (condition); ← semicolon important.
  - The do while loop similar to while loops except the fact that it is guranteed to execute the code ~~~~ at least once.

* for loops.
  - for (initialize; check-bool-expression; update)
    {
    // code
    }
  - A for loop is usually used to execute a piece of code for specifying number of times

* for-each loop
  for(int element : Arr){
  sout(element); → prints all the elements
  }

* Nested loops.

```
for (----)
  {
    for (....)
      {   }
  }
```

loop inside the loops.

* break statement

```
for (---)
  {
    if (condition)
      { //code
        break;
      }
  }
```

- The break statement is used to exit the loop irrespective of whether the condition is true or false.

* continue statement

- The continue statement is used to immediately move to next iteration of the loop. The control is taken to next iteration thus skipping everything below "continue" inside the loop for that iteration.

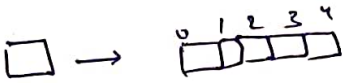
```
for (---)
  { if (condition)
    { //code
      continue;
    }
  }
```

Arrays

* Array is a collection of similar types of data

```
int [] marks = new int [5];   ⇒ [data type[] ArrName;]
```

marks → reference, new int[5] → Object

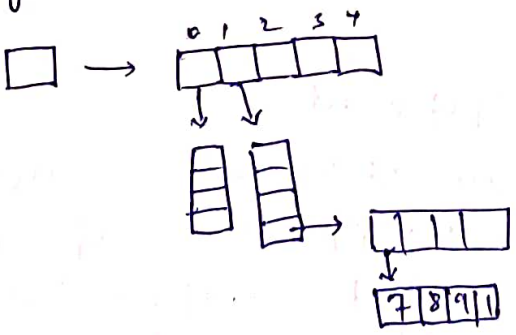


4 x 5 = 20 bytes

* • int [] marks; ⟶ declaration!
  marks = new int[5]; ⟶ Memory allocation
  • int[] marks = new int[5]; → Declaration + Memory allocation
  • int[] marks = {100, 70, 80, 89}; → Declare + initialize
* Array have a length property which gives the length of the array

```
marks.length
```

* Multidimensional arrays.
  arrays of array



marks[1][3][0][0] = 7

* Array List
  • ArrayList<String> food = new ArrayList<String>();

```
food.add("Pizza");
food.add("Burger");
```

```
for (int i=0; i<food.size(); i++) {
    sysout (food.get(i));
    }
```

- `food.set(0, "sushi");` replace 0th element
- `food.remove(2);` → remove 2nd element
- `food.clear();`

* 2-D ArrayList<String>.

- `ArrayList<ArrayList<String>> d = new ArrayList();`

`ArrayList<String> a = new ArrayList<String>();`

```
d.add(a);
d.add(b);
d.add(c);
sysout(d);
```

- `sysout(d.get(2).get(1));`

Methods

* Sometimes our program grows in size & we want to separate the logic of main method to others methods.

* Syntax

A method is a function written inside a class. Since Java is an object oriented language, we need to write the method inside the same class

- dataType name() {
  // method body
  }
- following method return sum of 2 num.

```
int mysum(int a, int b) {
    int c = a+b;
    return c;
}
```

* Calling a Method

A method can be called by creating an object of the classes in which the method exists followed by the method call.

```
Calc obj = new(calc();  → object creation
obj.mysum(a, b);  → Method call upon an object
```

* Void return type

When we dont want our method to return anything. we use void as the return type.

* Static ~~method~~ keyword is used to associate a method of a given class with the class rather than the object. Static method in a class is shared by all the objects.

* Method overloading

Two or more methods can have same name but different parameters. Such methods are called overload methods.
Method overloading cannot be performed by changing the return type of methods.

* Variable arguments (Varargs)

- A function with varargs can be created in Java using the following syntax

```
public static void foo(int...arr)
{
    // arr is available here as int[] arr
}
```

- foo can be called with zero or more arguments.

- we can also create a function bar

```
public static void bar(int a, int...arr)
{
    // code
}
```

↳ Atleast one integer is required

* A recursion in Java can call itself

Ex

$$factorial(n) = n * factorial(n-1) \quad \forall\, n \geq 1$$

* Method overriding

If the child class implements the same method present in the parent class again, it is known as Method overriding.

OOP

- object ~~oriented~~ oriented programming tries to map code instruction with real world making the code short & easier to understand.

* Class

a class is a blueprint for creating objects.

* Object

An object is an instantiation of a class. When a class is defined, a template (info) is defined. Memory is allocated only after object instantiation

* OOP Terminology

- Abstraction- Hiding internal details (show only essential info)
- Encapsulation- The act of putting various components together (in a capsule)
  - In java, encapsulation simply means that the sensitive data can be hidden from the users.
- Inheritance - The act of deriving new things from existing things.
- Polymorphism - one entity many forms.

* we cannot use word "~~class~~ public" more than one time in a class.

before V class name

* Access Modifiers

- Specifies where a property/method is accessible.
- 4 - access modifiers
  - Private
  - Default
  - Protected
  - Public
- Getters and setters

  Getter → Returns the value

  Setter → Sets/update the value

* Constructors

- A member function used to initialize an object while creating it.

```
Employee inp = new Employee();
inp.setName("Alone");
```

- In order to write our own constructor, we define a method with name same as class name.

```
public Employee() {
    name = "your name";
}
```

- Constructor overloading
  - Constructor can be overloaded just like other methods in java..
  - constructor can take parameters without being overloaded
  - There can be more than two overloaded constructors.

* Variable Scope.

- local - declared inside a method visible only to that method.
- global - declared outside a method, but within a class visible to all parts of a class.

* Inheritance

- It is used to borrow properties & methods from an existing class.
- Inheritance in Java is declared using "extends" Keyword
- Java doesn't support multiple inheritance
- public class Dog extends Animal {
  // code
  }
- constructor during constructor overloading
  — When there are multiple constructors in the parent class, the constructor without any parameters is called from the child class. If we want to call the constructor with parameters from the parent class, we can use super keyword.

  super(a,b); ⇒ calls the const. from parent class which takes 2 variables.
- constructor in Inheritance
  When a derived class is extended from the base class, the constructor of the base class is executed first followed by the constructor of derived class.

Abstract classes

* Abstract method- A method that is declared without implementation
  abstract void move to (double x, double y)
* objects of abstract class cannot be created
* Abstract class – If a class includes abstract methods, then the class itself

must be declared abstract as in,

- public abstract class PhoneModel {
  abstract void switchOff();
  // code
  }

"this" and "super" Keyword

* this is a way for us to reference an object of the class which is being created.
  - public void (int a) {
    this.a = a;
    }

* A reference variable used to refer immediate parent class object
  - can be used to refer immediate parent class instance variable, parent class methods & parent class constructors.

Interface

* In java interface is a group of related & method with empty bodies.
* A class can be created using several interface
* - A template that can be applied to a class is called interface
* Default methods
  - An interface can have static & default method Default methods enable us to add new functionality to existing interface.

- Classes implementing the interface need not implement the default methods.
- Interface can also include private methods for default methods to use.

* Inheritance in Interface

- Interface can extend another interface
- Interface cannot implements another interface, only classes can do that.

Polymorphism

The ability of an object to identify as more than one type.

Packages

Interpreter	Compiler
* Interpreter translate one statements at a time into machine code.	compiler scans the entire program & translate whole of it into machine code
* Partial execution if errors	No execution if an errors.
* Easy	Not as much as interpreter.

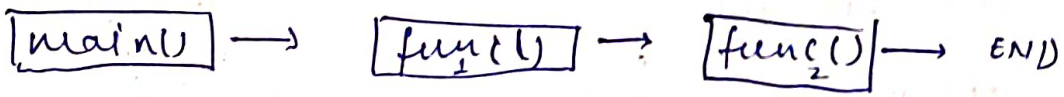
* A package is used to group related classes packages help in avoiding name conflicts
* 2- type packages
  - Built in packages → Java APIs
  - User defined packages → Custom packages

* using package
  - import java.lang.* → import everything from java.lang
  - import java.lang.String → import String from java.lang
  - s = new java.lang.String("Ash") → use without importing

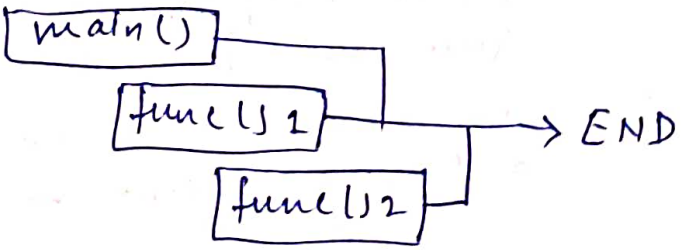
* creating a package
  - Javac Harry.java → creates Harry.class
  - javac -d . Harry.java → creates a package folder

Multithreading

* Threads allow a program to operate more efficiently by doing multiple things at same time
* It can be used to perform complicated tasks in the background without interrupting the main program.
* Multiprocessing and multitasking both are used to achieve multitasking.
* Thread is light-weight where as a process is heavyweight.
* flow of control
  - Without threading



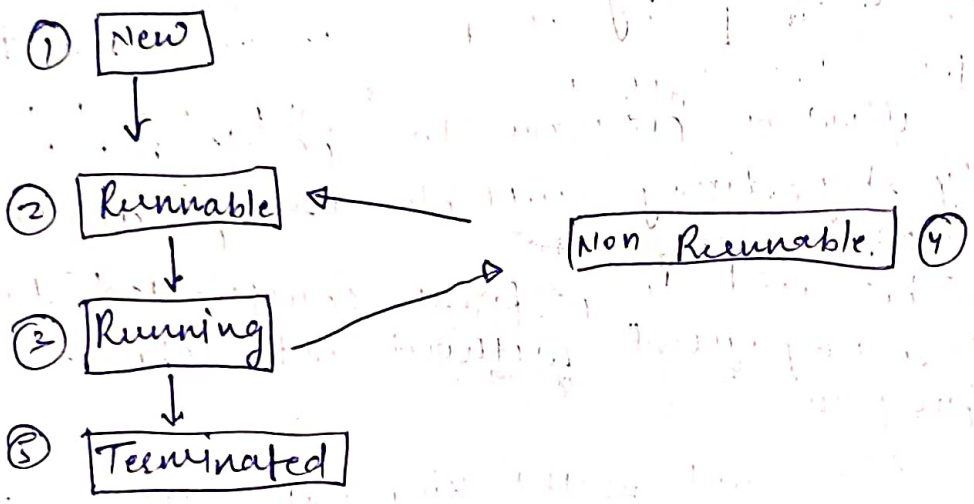
  - With treading



* Creating a threads
  - By extending Thread class
  - By implementing Runnable interface

- The major differences between extending and implementing Threads is that when a class extends the Thread class, you cannot extend any other class but by implementing the Runnable interface, it is possible to extend from another class as well.

* Life cycle of Thread:



Errors & Exception

* Error is an illegal operation performed by the user which result in abnormal working of the program

* Errors

→ Syntax errors – when compiler finds something with our program

ex- int a = 9 → No semicolon
d = 4 → Variable not declared

→ Logical errors – occurs when a program compiles & runs but does wrong thing

ex – message delivered wrongly
– incorrect redirects.

↳ Runtime errors - Also called exceptions

Java may encounter an error while the program is running

ex - user supplies 's' + 8 to a program which add two numbers.

* Syntax errors & logical errors are encountered by the programmer where as Runtime errors are encountered by the user.

* Exceptions

An exception is an event that occurs when a program is executed disrupting the normal flow of instructions

- Checked Exceptions - compile time exceptions (Handled by compiler)
- unchecked Exceptions - Runtime exceptions.

- try catch block in java
  - exceptions are managed using try-catch blocks
  - try {
    // code to try }
    catch (Exception e)
    // code if exception
    }

- Handling specific Exceptions

```
try {
    // code
}
catch (IOException e) {      → Handle I/O exceptions
    // code }
catch (ArithmeticException e) {      → Handle Arithmetic exceptions
    // code
}
catch (Exception e)      → all other exceptions
    // code
}
```

- Nested try-catch

```
try {
    try {
    }
    catch(Exc) {
    }
}
catch (Exc) {
}
```

- Exception class in java

- public class MyException extends Exception {
  // override methods
  }

- Imp methods
  → String toString() → executed when sout(e) is run
  → void printStackTrace() → prints stack trace
  → String getMessage() → prints the exception message.

- The Throw keyword

The throw keyword is used to throw an exception explicitly by the programmer.

```
if (b==0) {
    throw new ArithmeticException("Div by 0");
}
else {
    return a/b;
}
```

- Throws exception

- The java ~~throwsough~~ throws keyword is used to declare an exception.

- public void calc(int a) throws IOException
  {
  // code
  }

- Finally block

finally block containing the code which is always executed whether the entire exception is handled or not. It is used to execute code containing instruction to release the system resources, close connection etc.

Java File Handling

* Reading from & writing to files is an important aspect of any programming language

* We can
  - create file - myfile.createNewFile();
  - write file - ~~myfile~~ filewrite.write("....");
  - Read file -
  - delete file - myfile.delete()