Date.....

- Java development Kit. {Essential}
- IntelliJ – Code editor → Project SDK (software Dev. Kit)

Main.java → Main file
.java : extension

Boilerplate –

```
package com.apnaCollege;

public class Main {
    public static void main (String [] args ) {
     // comment
    }
    }
```

basic : System.out.println ("Hello World");
(To print the text)

– Variables –

```
String name = "H";
int age = 30;
String neighbour = "Ak";
```

can be assigned to each other

Date.....

Java Types

Primitive — Non-Primitive (complex)
String (methods can be performed)

→ byte (-128 to 127) 1 byte
short - 2 byte
int - 4 b
long - 8 b
float - 4 b
char - 2 b , boolean - (1 b)

float pi = 3.14F;

— Keywords can be used such as new —
(Not compulsory)

Strings & Methods —

— Concatenation —
(Addition)
String1 + String2

```
String name1 = "Anu";
String name2 = "Aish";
String name3 = name1 + name2;
String name4 = name1 + "and" + name2;
System.out.println (name3);
System.out.println (name4);
```

O/P → anuAish
Anu and Aish (no space)

Spiral

Date.....

- charAt –

```
String Name = "Aman";
System.Out.println(Name.charAt(1));
```

O/P → A

- Replace –

```
String name = "Ana"
String name2 = name.replace('a', 'b');
System.Out.println(name2);
```

O/P → Anb

Strings are immutable in java

Substring –

```
String name = "Aman and Akku";
System.Out.println(name.substring(0,4));
```

↳ Last index isn't included

O/P → Aman

— Arrays —

collection of same data type.

→ can be made of diff data types.

```
datatype [] name = new int [no.];
int [] marks = new int [3];
marks [0] = 97;
marks [1] = 98;
marks [2] = 95;
System.Out.println (marks[0]); (marks[1]);
```

Spiral

Date.....

Arrays.sort(marks);
↳ Package so install
import java.util.Arrays;

∴ can also be initialized as:
↳ Assigning a value

```
import java.util.Arrays;

public class Main {

    public static void main(String[] args){
        int[] marks = {97,98,95};
    }
}
```

- Two-Dimensional Arrays:

```
int[][] finalMarks = {{97,98,95},{95,95,98}};
System.out.println(finalMarks[0][0]);
```
O/P - 97
if (finalMarks[1][1]); O/P → 95

Casting -
converting from one type to another

- Implicit casting → small to big
  e.g → int to double.
- Explicit Casting → big to small
  int fp = p + (int) 18.0;
  ↳ parenthesis + data type

Spiral

Date .....

Constants :—

Variables that can't be changed.
defined using final
(usually written in capital letters)

e.g- final float PI = 3.14F;

Operators —

- Comparison
- Arithmetic (+, -, *, /, %) Remainder
- Assignment (=) (-=, +=, *=, /=)
- logical

Modulo ÷ % [To find the remainder]

unary operators :—

++ (num++) → first og value then increment
(++num) → increment & increment

-- (num--) decrement

Maths Class —

- math → function

math.max (num1, num2) (To find the maximum value)
math.min (1, 2) (minimum)
math.random () (prints random value)

Input in Java (import java.util.scanner;)

```
Scanner sc = new Scanner (System.in);
System.out.println ("Input your Age :");
int age = sc.nextInt ();
System.Out.println ("Age");
```

Spiral

Date.....

String name = sc.next(); → only takes tokens.
System.out.println(name); (sc.nextline) inputs line.

— Comparison Operators —

==
!=
<
>
<=
>=

{less than & greater than}

Conditional statements

```
if()
    // work
else
    //
```

Logical operators —

→ && (And operator)
if both statements are true then, true

→ || (OR operator)
(one condition should be true)

```
if (a<50 || b<50)
System.out.println("atleast one less than 50");
```

! (if false then true & vice versa)

Spiral

Date.....

```
boolean isAdult = False;
if (!isAdult)
    System.Out.println ("is adult");
else
    System.Out.println ("not adult");
    }
```

Switch Statement –

```
Switch (day) {
    case 1:
        System.Out.println ("monday");
        break;
    case 2:
        System.Out.println ("Tuesday");
        break;
    default:
        System.out.println ("Wed-Sun");
```

– [Loops] –

for — Initialization, condition, increment/decrement

```
for (int i = 0; i<=100; i++) {
    System.Out.println (i)
    }
```

(initialization is only in the loop)

While

```
int i=100;
while (i>=1) {
    System.Out.println(i);
        i = i-1;
    }
```

Date.....

Do while

```
int k = 100;
do {
    system.out.println(k);
    k = k-1;
} while (k>=1);
```

Exercises to Do —

→ Take input of no. from user until a negative.

— Break & continue — (KeyWords)

↳ (Repeating)

↳ Jearing Out of the block

```
int i=0; → Loop runs infinite
while(true) {
    system.out.println(i);
    i = i+1;
    if (i>5) {
        break;
    }
```

continue — It starts again (the loop) even if unfinished

```
if (i==3) {
    i=i+1
    continue;
}
```

Date.....

- Exception handling (like errors but not)
  → lines doesn't execute after exception
  Try - Catch : used when we know exception can come in the block.

— e.g —

```
int [] marks = {97, 98, 95};
    try {
        System.out.println(marks[5]);
    } catch (Exception exception) {
        //something to do
    }
    System.out.println("The name is Anu");
    }
```

— Methods / functions —

Class contains many methods ← Takes input.

— e.g —

```
public static void printJava() {
        System.out.println("Hello Java");
    }
public static void main (String [] args) {
        printJava(); → call fnx.
        printJava();
        printJava(); }
```

— To print sum —

```
public static void printSum (int a, int b) {
        int sum = a+b;
        System.out.println(sum);
    }
```

```
printSum (a:1, b:6);
}
```

Spiral

Date.....

— OOPS —

- Classes & objects: introduced to tackle real world problems. (Blueprint = class)

```
class Pen {
    string color;
    String type;

    public void write() {
        System.Out.println("Writing Something");
    }
}

public class OOPS {
    public static void main(String args[]) {
        Pen pen1 = new Pen();
        pen1.color = "blue";
        pen1.type = "gel";

        pen1.write();
    }
}
```

blueprint

object

this.color → object that's calling

Constructors: Used to make objects
- → constr name = class name
- → Returns nothing
- → can be called only one time for an object

- Non parameterised
- parameterised
- copy

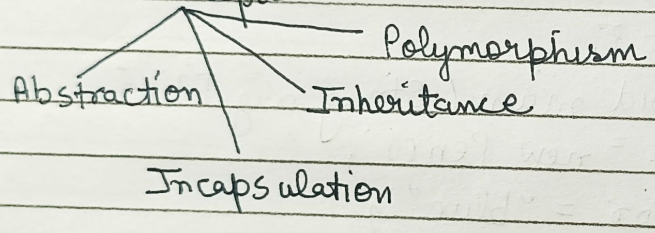
Date.....

1) Student () {
// work
}
→ non-parameter constructor (gets runned first)

2) Student (String name, int age) {
// work this.name = name;
}
→ with parameter

Student (student S2) {
this.name = S2.name;
}
→ copy constructor

— Concepts —



→ There are automatic destructors in java (has garbage collectors)

PolyMorphisms (Many + Forms) < Function Overloading (compile Time) (Static)
Function overriding (Runtime) Dynamic

→ It is the ability to present the same interface for different forms

1) Compile Time — Implemented at compile Time
Run Time — Implemented at Run Time

Spiral

(allows multiple methods in same class with same name but diff. parameters)

Date.....

Example -

```
                                    ← parameter
public void printInfo (String name ) {
        System.out.println (name);
    }
public void printInfo (int age) {
        System.out.println (age);
    }
public void printInfo (String name, int age) {
    System.out.println (name + " " + age);
    }
}
public class OOPS {
        public static void main (String args[]) {
            Student s1 = new Student();
            s1.name = "Harshita";
              s1.age = 17;
            s1.printInfo (s1.name, s1.age);
```

— Inheritance —

→ when one class acquires the properties & behaviours of another. (child to parent)

Superclass (parent)

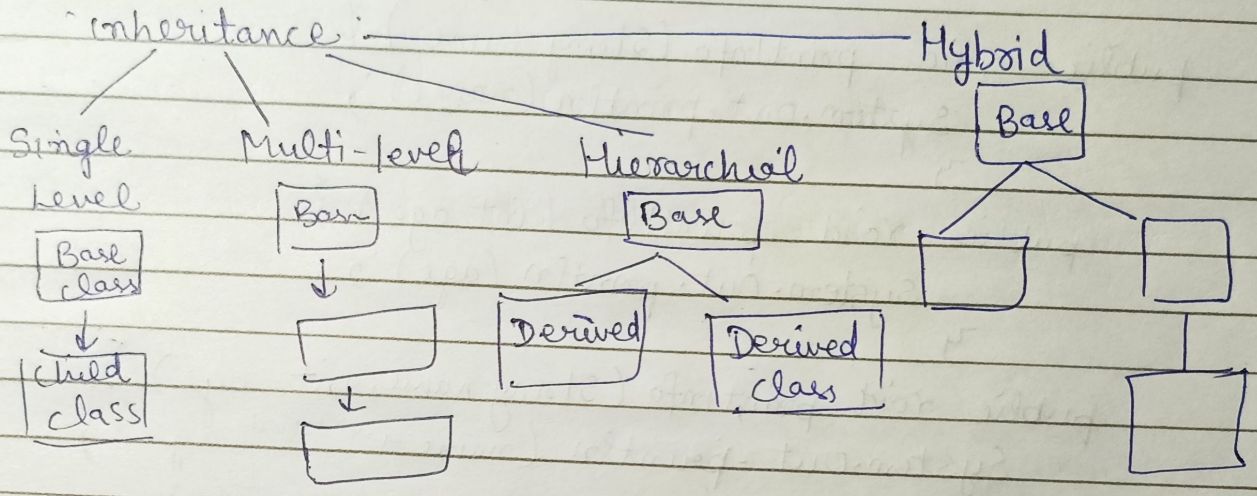
Subclass (child)

e.g-
```
class Shape {
        String color;
    }
class Triangle extends Shape {
    }
```
→ keyword to be used

Spiral

Date.....

- (increases reusability)



Packages → contains related classes, functions, methods etc

built-in user-made

package bank;

To import : import bank;

- Access Modifiers : To define what is accessible to whom

public → can be accessed by anyone

private string password; → can be accessed only by using getter & setters

protected string email;

Getters

↓

fnx

```
public String getPassword(){
    return this.password;
}
```

Setters

```
public void setPassword
(String pass){
    this.password = pass;
}
```

Spiral

Date.....

Incapsulation : combining data (variables) and code (methods) together.

→ Data Hiding can be done using access modifiers

— Abstraction - Hiding the implementation details and showing only important / useful parts to the user

→ can be done using abstract class or using interfaces

```
abstract class Names {
    // method e.g public void Indian() {
    }
```

→ methods can be both abstract or non-abstract
→ can have constructor (called first) & static method

— Interfaces (pure abstraction) —

defined using interfaces
→ can't have constructors

```
interface Animal {
    void walk();
}
class Horse implements Animal {
    public void walk()
        System.Out.println("Walks");
    }
}
```

(implements → keyword)

Spiral

Date.....

→ can have multiple inheritance.

— static keyword —

→ common for classes & accessible

can be used with blocks, functions, keywords etc.