Programming in Java.

→ Hello World program,

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

* Parameters,

① class → Used to declare a class

② Public → Access modifier (represents visibility)

③ Static → Adv. of static method is, there is no need ~~for~~ to create an object to invoke the static method.

④ void → It means that the code does not return a value.

⑤ main → Starting pt. of the program.

⑥ String[] args
or
String args[]
→ used for command line argument.

① System.out.println() is used to print

3 WHAT IS JDK, JRE AND JVM?

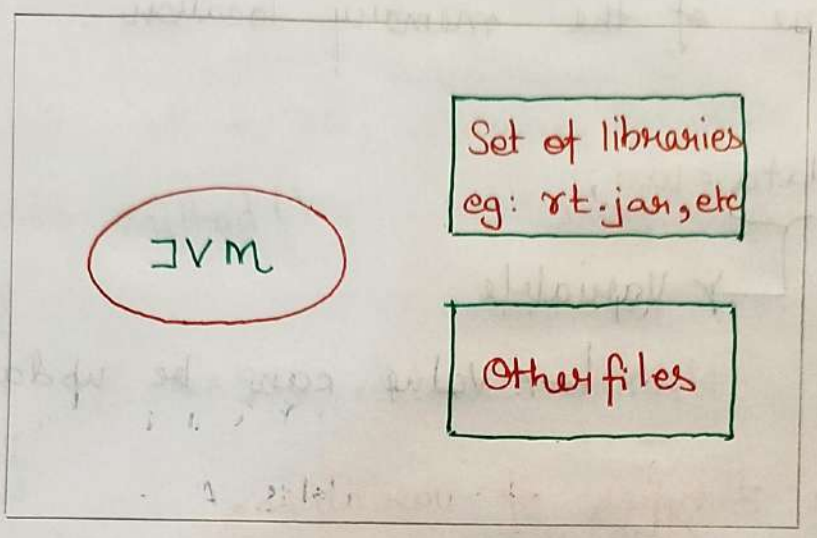
* JVM, (Java Virtual Machine)
  - Abstract Machine
  - It is called a virtual machine because it does not exists PHYSICALLY.
  - Provides a runtime environment in which Java bytecode can be executed.

It performs the following main tasks,

- Loads code
- Verifies code
- Executes code
- Provides Runtime Environment.

* JRE, (Java Runtime Environment)
  - Set of software tools
    - used for developing Java applications.
  - It exists physically

↳ contains set of libraries.



JRE fig.

* JDK, (Java Developers Kit)
  - ↳ Used for Java Applications
  - ↳ Physically Exists.
  - ↳ It contains JRE + development tools.

→ It has 3 editions,
- ↳ Standard Edition
- ↳ Enterprise Edition
- ↳ Micro Edition

* Variables in Java,

↳ Name of the memory location.

Eg:

```
int data = 100;
```

↳ Variable.

↳ Value can be updated.

↳ There are 3-types of variables,

- LOCAL → · Declared inside the body of a method. · can only used by that particular method.
- INSTANCE → · Declared inside a class but outside the body of the method · It's value is instance specific and is not shared among other instances.
- Static.

· Basically, Global.

↳ Shared among all methods.

↳ Memory allocation happens only once.

Eg:

```
public class A
{
    static int m=100; //Static Variable
    void method()
    {
       int n=90; // local variable
    }
    public static void main(String args[])
    {
        int data=50; //instance variable.
    }
}
```

---

* Data Types in Java,

-> There are two types of datatypes in Java,

- Primitive -> boolean, char, byte, short, int, long, float, double.
- Non Primitive
  - Classes, interfaces and Arrays.

Primitive Data Types:

Data Type	Default Value	Default Size
• boolean	false	1 bit
• char	'\u0000'	2 byte
• byte	0	1 byte
• short	0	2 byte
• int	0	4 byte
• long	0L	8 byte
• float	0.0f	4 byte
• double	0.0d	8 byte

Syntax:

```
datatype  variable =    ;
```

Eg

```
Boolean  one = false;

   int   a = 50;
char  Letter = 'A';
```

- Operators in Java,
  - Unary
    - Postfix → expr++ , expr--
    - Prefix → ++expr , --expr , +expr ...
  - Arithmetic → multiplicative → * , / , %
    - Additive → + , -
  - Shift → Shift → <<, >>, >>>
  - Relational → comparison → < , > , <= , >=
    - Equality → == , !=
  - Bitwise → Bitwise AND → &
    - XOR → ^
    - OR → |
  - Logical
    - Logical AND → &&
    - Logical OR → ||
  - Ternary
  - Assignment

Assignment

= , += , -= , *= , /= ,

%= , etc.

Ternary

? :

Java Control flow Statements,

* THEY ARE OF THREE TYPES,

- Decision Making Statements
  - if Statements
  - Switch Statements
- loop Statements
  - do while
  - while
  - for
  - foreach
- Jump Statements
  - break
  - continue

* Decision Making Statements,

EXAMPLE-6

```
int number = -5;
int day = 3;
int age = 20;

if (number > 0) {
    System.out.println("The number is +ve");
}
else if (number < 0) {
    System.out.println("The number is -ve");
}
else {
    System.out.println("The number is 0");
}
```

```
switch (day) {
    case 1:
        System.out.println("Monday");
        break;
    case 2:
        System.out.println("Tuesday");
        break;
    case 3:
        System.out.println("Wednesday");
        break;
    default:
        System.out.println("Invalid day");
}
```

* Loop Statements,

↳ FOR
  ↳ Syntax,

```
for (start, stop, step) {
    //block of statements
}
```

↳ FOR EACH
  ↳ Syntax,

```
for (data-type var: array-name) {
    //block of Statements
}
```

↳ WHILE
  ↳ SYNTAX,

```
while (condition){
    // Looping Statements
}
```

- If while loop is true, the looping statements will execute infinite times until it becomes false.

Do WHILE

Syntax,

```
do {
    //statements
} while (condition);
```

* JAVA COMMENTS,

~~> Single Line → //comment

Multiline → /* . . . . . .
. . . . . .
. . . . . */

OBJECT ORIENTED PROGRAMMING IN JAVA.

* CLASS ,

- Group of objects which have common properties.
- Blueprint of an object.
- It is a logical entity.

→ Syntax ,

```
class <class_name>
{
    field;
    methods;
}
```

Imp Note: → new Keyword is used to allocate memory at runtime. (temporary)
→ All objects get stored in Heap memory area.

INSTANCE VARIABLE

- created inside the class but outside the method.
- Gets memory at runtime when an object is created

METHODS

- Method is basically a function in Java.
- It allows user to reuse the code and optimizes it.

Example:

```
class Student {
    int id;
    String name;
public static void main(String args[]){
    Student s1 = new Student();
    System.out.println(s1.id);
    System.out.println(s1.name);
    }
}
```

instance variable →
main method →
Object Creation →
Reference variable (s1) →

- 3 ways to Initialize object,
  - By reference variable
  - By method
  - By constructor.

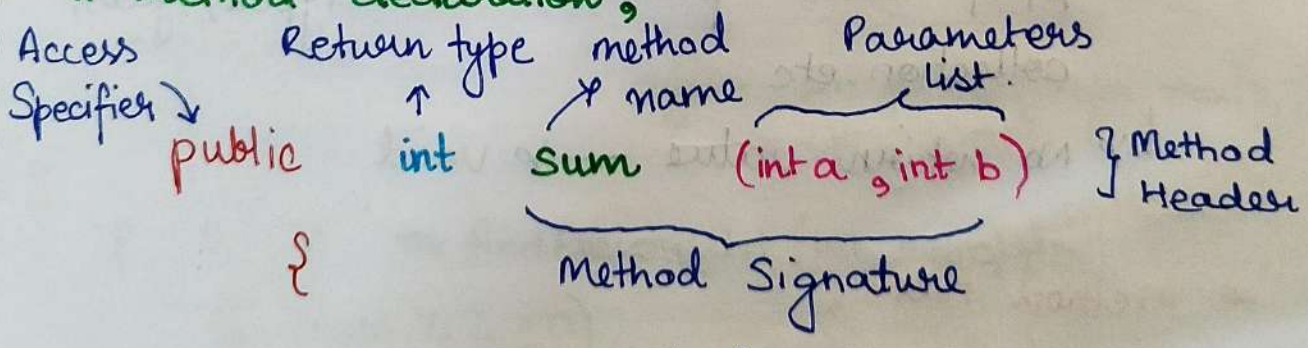
* Methods in Java,

Collection of instructions that performs a specific task or operation.

- used to achieve reusability of code.

- Most important method in java is the main() method.

Method declaration,



* Method Signature,

  - Method name + parameter list.

* Access Specifiers,

  - 4 types,
    - Public → Accessible by all classes.
    - Private → Only accessible in the defined class.
    - Protected → within the same package or subclasses in different package.
    - Default.
      - visible only from the same package.

* Return type,

  - Datatype that the method returns.
  - may have primitive datatypes, objects, collection, etc.
  - No return value → void

* Method Name,

  - used to define the method {unique}

* Parameter list,

  ↳ contains the data type and variable name.

  ↳ No parameter → leave the parentheses empty.

* Method Body,

  ↳ contains all the actions to be performed.

  ↳ {;}

Types of Methods,

* Predefined → length(), equals(), print(), sqrt(), etc.

* User defined

  ↓

  Modified acc to the requirement.

  ↓

  Eg:

```
p s v findEvenorOdd (int num) {
    if (num %2 = 0)
        S.o.p (num + "is even");
    else
        S.o.p (num + "is odd");
}
```

Eg:

```
system.out.print( Math.max(9,7));
```

→ Static Method,

- We can call it without creating an object.
- Example: main()

→ Instance Method,

- Non static method defined in the class.
- Before calling or invoking the instance method, it is necessary to create an object of its class.

* Accessor Method, → returns the value of the private field.
- getters.

Syntax:

```
public int getId();
  {
   return Id;
  }
```

* Mutator Method, —> It does not return anything.
  - setters or modifiers.
  - Accepts a parameter of the same data type that depends on the field.

* Abstract Method,

Syntax:

```
abstract void method_name();
```

* Constructors,

* Mutator Method, → It does not return anything.
  - setters or modifiers.
  - Accepts a parameter of the same data type that depends on the field.

* Abstract Method,

Syntax:

```
abstract void method_name();
```

* Constructors,
  - Similar to a method
  - It is called when an instance of the class is created.
  - Special type of method used to initialise the object.

→ Rules,

① → Constructor name must be the same as its class name.

② → A constructor must have no explicit return type.

③ → Constructor cannot be abstract, static, final or synchronized.

→ TYPES,

- → DEFAULT → Without a parameter.
- → PARAMETERIZED → With a specific no. of parameters.

★ DEFAULT SYNTAX,

~~class-name () { }~~

<class-name> () { }

★ PARAMETERIZED SYNTAX,

```
class class_name {
    public class_name (Type1 param1, Type2 param2...)
    {
        // constructor body
    }
}
```

★

→ Difference between Method & Constructor,

Constructor	Method
→ A constructor is used to initialise the state of an object.	→ A method is used to expose the behaviour of an object.
→ Must not have a return type.	→ Must have a return type.
→ Invoked implicity.	→ Invoked explicity.
→ The constructor name must be same as the class name.	→ The method name may or may not be same as the class name.

★ THIS Keyword,

→ It can be used to invoke current class method (implicitly).

→ It can be used to refer current class instance variable.

→ It is a reference variable that refers to the current object.

→ this() can be used to invoke current class constructor.

```
class Student{
int rollno;
String name;
float fee;
Student (int rollno,String name,float fee){
    this.rollno = rollno;
    this.name = name;
    this.fee = fee;
}
void display()
{
   System.out.println(rollno+" "+name+" "+fee);
}
}
class Testthis {
public static void main (String args[]){
Student s1 = new Student(111,"ankit",5000f);
Student s2 = new Student(112,"sumit",6000f);

s1.display();
s2.display();
}}
```

→ Output :

```
111  ankit   5000.0
112  sumit   6000.0
```

→ SYNTAX →

* Accessing Instance Variables,

```
this.instanceVariableName
```

* Invoking other constructors,

```
this(argument)
```

* Returning the current object,

```
return this;
```

* FINAL Keyword, ——→ Used to restrict the user.

- variable → eg: final int speedlimit = 90;
- method → final void run()
- class → final class Bike { }

★ INHERITANCE IN JAVA,

- The process of creating a new class from an existing class.

- USE,
  - for method overriding.
  - for code reusability.

- Terms used in Inheritance,
  - Class
    - Sub Class / Child Class → derived class.
    - Super Class / Parent Class → subclass inherits its features from the superclass.
    - Reusability

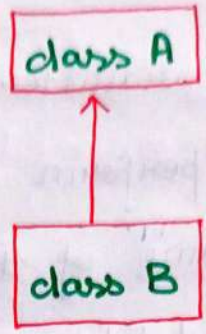
- Syntax,

```
class Subclass_name extends Superclass_name
{
    //methods and fields.
}
```

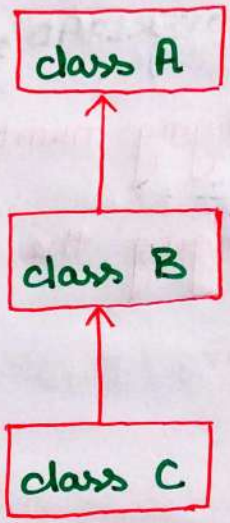
keyword. **

→ Types of Inheritance,

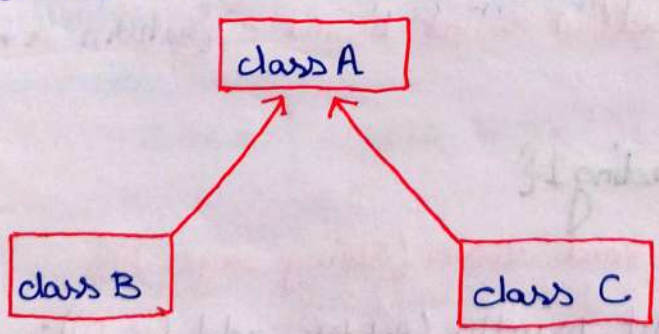
① Single →



② Multilevel →



③ Hierarchical →



• If we combine them, it becomes hybrid type of inheritance.

★ Method Overloading in Java,

- Same name.
- Different parameters.

→ If we have to perform only one operation, having same name of the ~~message~~ methods increases the readability of the program.

- TWO WAYS TO OVERLOAD,
  - By changing number of arguments.
  - By changing the data type.

- Example,

```
class Adder {
    static int add (int a, int b) {return a+b;}
    static int add(int a, int b, int c){return a+b+c;}
}

class TestOverloading1{
    public static void main (String args[]){
        System.out.println (Adder.add (11, 11));
        System.out.println (Adder.add (11, 11, 11));
}}
```

Same name, diff parameters

Method Overriding in Java,

- Same name
- Same parameters

* Method overriding is used to provide the specific implementation of a method which is already provided by its superclass.

→ Used for RUNTIME POLYMORPHISM.

- Example,

```
class Vehicle{
    void run(){System.out.println("Vehicle is running");}
}
class Bike extends Vehicle{
    void run(){System.out.println("Bike is running");}
public static void main(String args[]){
    Bike obj = new Bike();
    obj.run();
}
}
```

Same method
Same parameters

Output,

→ Bike is running.

★ SUPER Keyword :

↳ The SUPER keyword in Java is a reference variable which is used to refer immediate parent class object.

↳ Usage of Java super keyword :

① Super can be used to refer immediate parent class instance variable.

② Super can be used to invoke immediate parent class method.

③ super() can be used to invoke immediate parent class constructor.

↳ Example :

```
class Person {
int id;
String name;
Person (int id, String name) {
this.id = id;
this.name = name;
}
}
```

```
class Emp extends Person {
    float salary;
    Emp(int id, String name, float salary) {
        super(id, name);    // Reusing parent constructor.
        this.salary = salary;
    }
    void display(){
        System.out.println(id+" "+name+" "+salary);
    }
    class TestSuper {
        Public Static void main(String args[]){
        Emp e1 = new Emp(1,"Ankit", 45000f);
        e1.display();
    }
    }
```

Output :

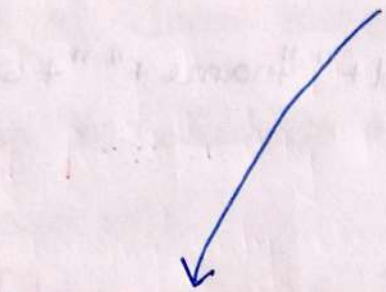
1 Ankit 45000

* instanceof Operator :

- Used to test whether the object is an instance of the specified type.
- It returns either TRUE or FALSE.

↳ Example,

```
class Simple{
    public static void main(String args[]){
        Simple s=new Simple();
        System.out.println(s instanceof Simple);
    }
}
```



This returns:

True.

* RECURSION,

↳ A process in which the method calls itself continously. Such method is called a recursive method.

↳ Syntax,

```
returntype methodname(){
    // code to be executed
    methodname();
}
```

Ex Example,

```
public class RecursionExample{
    static int count=0;
    static void p(){
        count++;
        if (count<=5){
            System.out.println("hello" +count);
            p();
    } }
} }
public static void main(String args[]){
        p();
}
}
```

Output:

hello 1
hello 2
hello 3
hello 4
hello 5

* ABSTRACT CLASS AND METHODS,

- Declared with the Abstract keyword.
- It can have abstract and non-abstract methods.

• WHAT IS ABSTRACTION?

→ Process of hiding the implementation details and showing only functionality to the user.

→ Basically, showing only essential things to the user and hides the internal details.

There are two ways of achieving abstraction in Java,

- Abstract Class. (0 to 100%)
- Interface. (100%)

→ Abstract class needs to be extended and its method implemented.

→ It cannot be instantiated.

- Eg:

```
abstract void A{
}
```

* Abstract Method

↳ A method which is declared as abstract and does not have implementation.

↳ Example,

```
abstract void printStatus(); //no method body and abstract.
```

↳ Example of abstract class that has an abstract method,

```
abstract class bike{
  abstract void run();
}
class Honda extends bike{
   void run(){System.out.println("running");}
   public static void main(String args[]){
     bike obj = new Honda();
     obj.run();
   }
}
```

★ INTERFACE

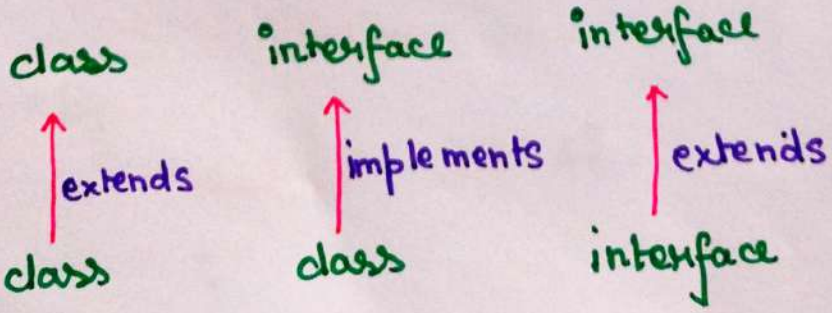
- Blueprint of a class.
- Has static constants and abstract methods.

- Mechanism to achieve abstraction.
- There can be only abstract methods in the Java Interface, not method body.
- It can be used to achieve loose coupling.

* Syntax,

```
interface <interface name>{
    // declare constant fields
    // declare methods that are abstract
    // by default.
}
```

- Relationship b/w class and interface,



↳ Example,

```
interface printable{
    void print();
}
class A6 implements printable{
    public void print(){System.out.println("Hello");}
    public static void main(String args[]){
        A6 obj = new A6();
        obj.print();
    }
}
```

↳ Can be used for MULTIPLE INHERITANCE,