* Commenting in Java

We are using commenting not to execute specific lines of the code in a java program & to provide additional information in a java program.

Single line code //
Group of statements /* .... */

* Variables in Java

- local variables
- static variables
- Non-static variables.

To store the values which can be reused multiple times.

1. Local variables; The variables which we create inside the method/block & the scope of the local variable is only within the method.

2. Static variables: The variables which we create inside the class & these variables are loading while class is loading.
We will declare it with Static keyword.

```
public class StaticVar {
    static int s1 = 10; // Static variable
  psvm (String[] args){
    int i = 20; // local variable.
  }
}
```

3. Non-static variables: Creating inside the class & these variables are loading while object creating.

Syntax for creating Object

```
Classname objectname = new Classname();
```

eg.
```
public class NonStaticVar {
    public int a=40; //Creating NonStatic Variables
    public int b= 50;

    psvm(String[] args) {

    NonStaticVar n = new NonStaticVar();
        Sopln(n.a*n.b);
    }
}
```

* Access Specifiers
  - Public
  - Private
  - Protected
  - Default.

We are using access specifier to provide access permissions to variables, methods & constructors in Java.

1. Public : Allow variables, method, constructors anywhere (other package also).

2. Protected : Allows variables, methods, constructors only within the same package (All classes in package).

3. Private : Allows variable, methods, constructors only within the same class.

4. Default : Allows variable, methods, constructors only within the same package (All classes in package).

Access Specifier	Same class	Different class	Different package
Public	Yes	Yes	Yes
Private	Yes	No	No
Protected	Yes	Yes	No
Default	Yes	Yes	No.

* Methods

A block of code or collection of statements, or set of code grouped together to perform a certain task or operation. It is used to achieve the reusability of code.

→ Predefined methods : The method that is already defined in the Java class libraries. Also known as standard library method or built-in method.
eg. length(), equals(), charAt() etc,.

→ Userdefined methods : Written by the user or programmer. These are modified acc. to requirements.

* Types of Methods

1. Static method : Creating inside the class & loading while class is loading & can be access directly within the class, "classname.method" in diff class & package.

2. Non-static method : Creating inside the class & these are loading while object is creating, that's why can access only by creating the object.

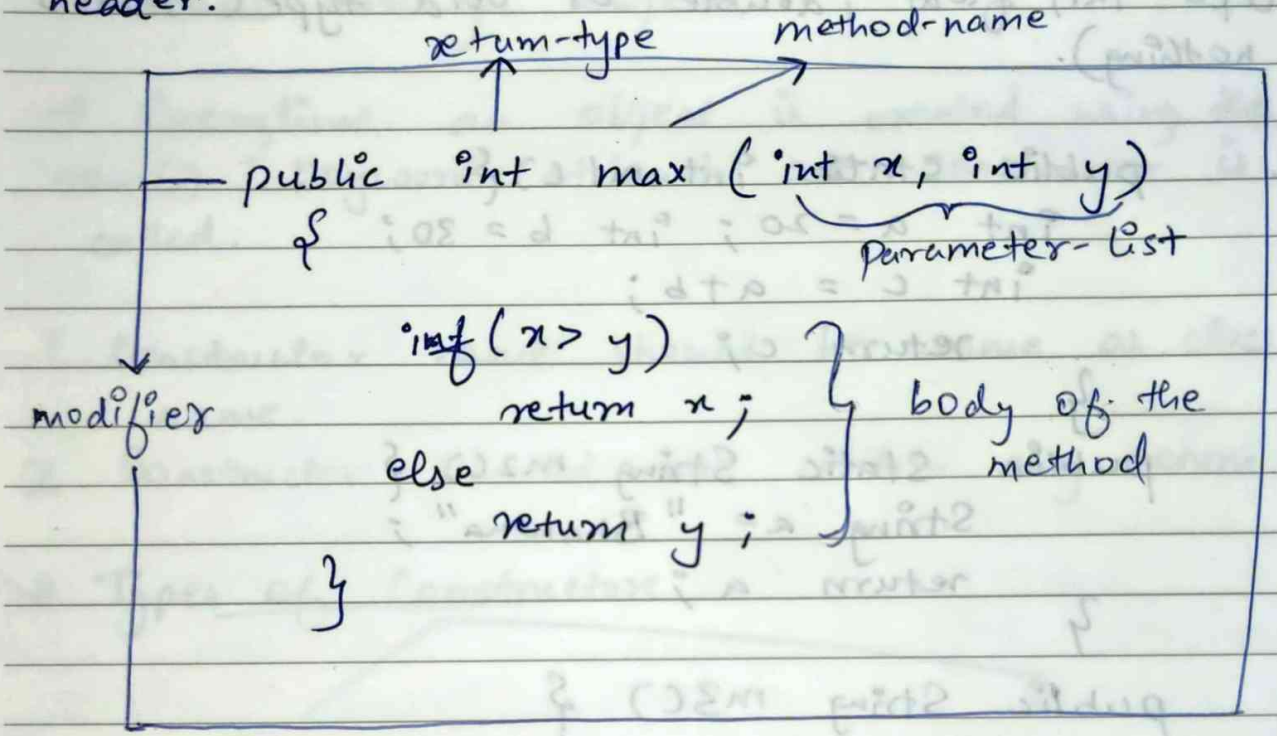
NOTE : Static variable can access into static & non-static methods but we can access non-Static variables into only non-static methods.

* Methods with Arguments / Parameters

```
                                    args.
public static void add (int a, int b) {
        System.out.println(a+b);
    }   public void multiply (int a, int b){
                    Sopln(a*b); }
psvm (String[] args) {
    add (10, 15);
}
MethodWithArg m = new MethodWithArg();
    m.multiply(20*5);
}
```

* Method Declaration

Provides information about method attributes, such as visibility, return-type, name & arguments. It has six components that are known as method header.



* Method Overloading

Creating same method name with different arguments. We can differentiate method based on the arguments.

eg.
```
public static void add (int a, int b){
        Sopln (a+b); }
public static void add (int a, int b, int c){
        Sopln (a+b+c);
    }
psvm (String[] args) {
    add (10,15);
    add(2,5,6); }
```

* Return type in Java

Every method is declared with a return type and it is mandatory for all Java methods. A return type may be a primitive type like int, float, double or void type (returns nothing).

eg.

```
public static int m1(){
    int a = 20; int b = 30;
    int c = a+b;
    return c;
}
public static String m2() {
    String a = "Bhawna";
    return a;
}
public String m3() {
    String a = "Bhawna";
    return a;
}
psvm(String[] args) {

    Returntype obj = new Returntype();
    String value = obj.m3();
    Sopln(m1()); // 50
    Sopln(m2()); // Bhawna
    Sopln(value); // Bhawna
}
```

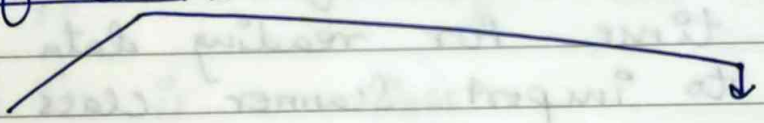
* Constructors in Java

Similar to methods which is loading while creating the object. With the help of constructor, we can pass the data into the methods through objects.

→ Everytime an object is created using the new() keyword, atleast one constructor is called.

1. Constructor name should be same as class name.
2. Constructor should not contain any return type.

* Types of Constructors



Constructor without args
Default Constructor

Syntax : <classname>() { }

eg.
```
class Stu {
    Stu() {
        sopln("Default Constructor")
    }
}
```

Constructor with args
Parameterized Constructor.

Syntax: classname(args){}

eg.
```
class Stu {
    int a; int b;
    public Stu(int m1, int m2)
    { a = m1;
      b = m2;
    }
}
```

* Constructor Overloading

Creating multiple constructor with diff args.

```
int a; int b; int c;
```

eg.
```
public St (int m1, int m2) {
    a = m1;
    b = m2;
}
public St (int m1, int m2, int m3) {
    a = m1;
    b = m2;
    c = m3;
}
```

* Scanner Class

To read the data from console during execution time. for reading data from console we need to import Scanner class by creating object.

```
Scanner sc = new Scanner (System.in).
```

NOTE : It is present in Java.util package.

eg.
```
import Java.util.Scanner;
public class SC {
    psvm (String[] args) {
        Scanner sc = new Scanner (System.in);
        Sopln("Enter value of a");
        int a = sc.nextInt();
    }
}
```

* Class

Collection of variables & methods.
A user-defined blueprint from which objects are created.

* Object

Instance of class. that means if you want to access the instance members of the class we need to create the reference object.

NOTE : Class is just only structure. Objects only have physical entity.

eg.

Fruit is a class
but banana is a object.
apple is a object. etc.

Similarly, Car is a class
but Swift desire is a object.
BMW is a object. etc,.

* Packages < User-defined package - created by user
Built-in package - created by vendor
Collection of classes & interface.

reusing multiple times