Classes & Objects:-

```
public class Demo
{ public static void main ( string a[] )
                  ↳ Return type        → Method
  {  int a = 2;
     ↳ Primitive variable

     Calculator obj = new Calculator ();
                ↳ Reference variable.
  }
}
~~public~~ class Calculator{

   public int add (int n1 , int n2)
   {
      return n1+n2;
   }

}
```

```
class Computer {  → Instance
   int a=5;
public String getMeAPen (int cost)
  { if (cost >=3)  → local variable
       int b=3
       return "Pen";

    else
       return "Nothing";
  }
}
```

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

red.nextInt();
```

Method Overloading :-

* Every Method has different parameter.

Eg:- public int add (int num1, int num2) { }

* But if we want more parameter or parameters of different data types for same method we use method Overloading

Eg:- public int add (int a, int b, int c) { }
public int add (double a, double b) { }

{ int a[ ] =
Scanner
~~System~~ sc = new Scanner (System.in);

~~int a[5] = sc.next~~

for(int i, i<= : }

Array :-

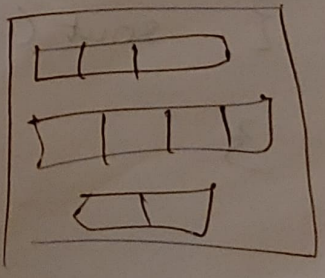
$ int a[ ] = { 1, 2, . . } ;
int a[ ] = new int[6];

Multi Dimensional array :-

int a[ ][ ] = new int [3][4];

Jagged array :-

int a[ ][ ] = new int[3][ ];
a[0]=[new int [3];
a[1] = new int [4];
a[2] = new int [2];



```
for (int i; i<nums.length; i++)
{
 for (int j=0; j<num[i].length; j++)
 {
   num[i][j] = (int)(Math.random()*
 }
}
```

```
class Student
{ String name;
  int RollNo;
}
```

```
Student stu[] = new Student[3]
```

or

```
Student stu[] = {s1, s2, s3}
```

for printing the array of objects & their values we can use Enhanced for loop

```
for (Student i : stu)
```

- Student → Variable type: reference variable, just int, string we can pass the class name
- i → each one-by-one
- stu → Array name

```
{ sout( i.name + i.RollNo);
}
```

* Static variables are same for every object. So if we change static variable value it will that variable value for every object.

* We can call static variable using class name instead obj name

Eg:-

```
class student
{   static string class = "B";
    string name;

public static void main (String a[])
{
    student st1 = new student;
sout (student.class);
}
```

* We can also create static method.
* We cannot use non-static variables in static method.
* A constructor is similar a method which has same name as the class and has no return type.

Eg:-

```
class Human{
    public Human()
    {sout ("I am a Human");
    }
}
```

* Constructor is ~~creat~~ called automatically once at the beginning of the after ~~creat~~ object is created.

* There are two types of constructor default constructor & parameterized constructor.

* Just like the method overloading we can have two constructor with different parameters

* We follow Camel casing in Java

→ for classes & interfaces – Class, Human, ...
Classes name start with capital letter

→ for methods & variables – marks, show()
methods & variables name start with small letter

→ constants – PIE, BRAND.

Stack contains :- A continuous block of memor

- Primitive types (int, bool, float, etc.,)
- Reference to objects (It stores reference to obj which are stored in heap)

- LIFO
- fast access

Heap memory :- Heap memory is used to store objs :-

- Objects (objects are instances of a class)
- Dynamic ~~storage~~ memory allocation (Memory can be allocated & de allocated as we need)
- Garbage collection
- Slower access (As Heap memory is not a contiguous memory block)
- static Variables (As they can be used at any part of code)

---

function :- A function is a block of code which is self contained piece of code that performs specific task and return value.

* A function can be called multiple time at any part of a program.

method :- Method is a block of code that is associated with a class or an object

Key difference :-

→ function must return a value whereas Methods may or may not return values.

→ Parameters:- Functions must have parameters whereas Methods may or may not take parameters.

Function:-

```
public int add (int n1, int n2) {
    return n1+n2;
}
```

Method:-

```
public class Calculator{
    int n;
    public int add (int n1, int n2)
    {
        return n1+n2;
    }
}
```

Instance variable

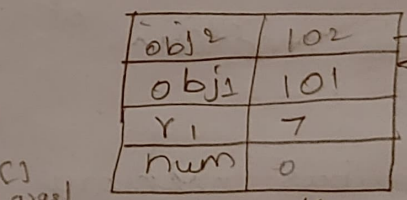
local variable

---

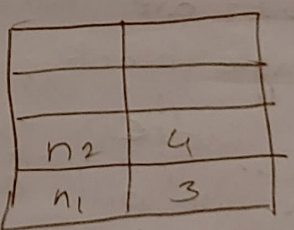
*Every method has it's own stack memory

from above example

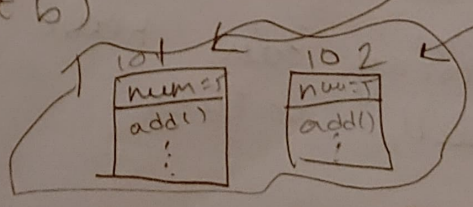
```
public
Class Main
{ Public static void main (String[] args)
{ int num=0;
  Calculator obj = new Calculator();
r1=obj.add(3,4);
  Sout(r1); }
}
Public
Class Demo {
  int num=5;
  public int add (int a, int b)
  {
    retur a+b;
  }
}
```



(Stack)
Main



add



Heap

* In the above example for method which is not part of "n" Instance variable ~~is stored in~~ any method is stored in Heap memory

List :- List is a collection of elements of same data type

The set of operation we can do on list are:-

add() - Add elements to the list

remove() - Remove elements from the list

get() - Access the element in the list

set() - Update elements in the list

* List is an interface which is a part of java.u package
* List interface is implemented by using subclasses like linked list, Arraylist, stack, vectors

Array	List
* Array has fixed size that is determined during the creation of array.	* List has dynamic size that can be chan at any time and add new elements.
* Contiguous memory locations, which means elements are stored in a single block.	* non-contiguous locati elements are stored in seperate location.
* Has indices through which we will access the elements.	* Has indices through which we access elem
* Array can have null values.	* Lists can also ha null values but not recommended as it throw null pointer exceptions.
* faster	*Slower

* Not type safe

* Arrays are not generic that means they can store elements of any type.

* Type safe

* Lists are generic as they can store the elements of only one type

→ Arrays are not type specific because they can store any type of elements eventhough they are declared to store a particular type of elements.

This is because JVM checks the type of the element being stored in runtime & throws an exception that this type is not compatible with the type of array

To avoid this issue Java introduced "Generics" which allows to specify the type of elements that an array or collection can have.

Adding an element to list:-

```
List<String> colours = new ArrayList<>();
colours.add("Red");
```

(Or we can specify the index & element)

```
colours.add(1, "Orange");
```

If we try to add an elements in to the list in the position where already an element exist it moves the element to one position & make space to add new element

Ex:- we have: Orange in "1" position
if we add yellow.

colours.add(1, "Yellow");

o/p:- [Red, Yellow, Orange]

remove elements:-

We can specify element

colours.remove("Red");

or

colours.remove(0);

Get elements:- Gets an element from the specified position

String c = colours.get(1);

Set elements:- Updates the elements in the specific space

colours.set(1, "Purple");

Contains element:- Check if list has the specified element

colours.contains("Red");

→ o/p Returns Bool true/false

isEmpty() → checks whether list is empty

size() → gets the size of list

clear() → clear the list

4/08/24

Collection

Class & Interfaces
- starts with capital letter

variable & methods
- starts with small letters.

- Constants - All cap

Abstract class/method:-

* An abstract method can only be defined in abstract class.
* But it is not necessary that an abstract class have only abstract methods it can have other methods too.

> We cannot create objects of an abstract class

* Abstraction is a process of hiding the implementation details and showing only functionality to the user.

* If we are extending the abstract class we must declare all the abstract methods of the abstract class

Eg:

```
abstract class Car
{
    public abstract drive();
    public abstract music();
    public void fly(){
        System.out.println("Car can't fly");
    }
}
```

abstract methods

Normal method

```
public class WagonR extends Car
{
    public void drive()
    {
        sout("Drive");
    }
    public void music()
    {
        sout("Play music");
    }
}
```

In the above example WagonR inherited from Car so it must define the abstract methods (drive, music()),

```
adstract class Car{
    public abstract void drive();
    public abstract void fly();
    public abstract void playMusic();
}
abstract class WagonR extends Car {
    Public void drive();
    {
    }
    public void fly();
    {
    }
}
// We didn't define playmusic() which is an abs" method then WagonR is also become "abstract" class
Public class Updated WR extends WagonR
{
    public void playMusic();
    { . . . }
}
// Here new class called Updated WR (concrete class) defined the playmusic() method.
```

```
public static void main()
{
    Car obj = new UpdatedWR();
    obj.drive();
    obj.fly();
    obj.playMusic();
}
```

we can create obj of concrete class

We can create obj of concrete class.

Inner Class:- Class inside a class.

* We cannot create obj of an inner class directly; for example let us consider classes A and B. B is inside class A

~~A obj = new A();~~
~~A.B obj1 = obj.new B();~~

then

B obj = new B(); ✗

~~Todd~~ A obj = new A(); ✓
A.B obj1 = obj.new B();

To create an obj of inner class "B" we must create obj of class "A" first.

to call all non-static methods.

* Method overriding is a feature which allows the child class to override (provide a diff implementation) the parent class
* Method Overloading same method has diff parameters

5/08/24

Anonymous class:- Class that is defined without name. They are used as Event listener, Callback interfaces, Subclasses & lambda expressions.

* Anonymous classes are diff from Inner class while inner class has access to outer class Anonymous class does not have access to the outer class

* Syntax of Anonymous inner class:-

```
public class Extra{
    public void show()
    { sout("show");
    }
}
Public class Main{
    Extra obj = new Extra()
    {
        Public void show()
        { sout("Hello");
        }
    };
}
```

→ Anonymous Inner class

We can override a method without creating child class using anonymous inner class we declare an object and open curly brackets & override the methods

We cannot create an object of an Abstract class but we can create obj of an abstract class and define the abstract methods in Anonymous inner classes.

```
Abstract class A{
   public abstract void print();
}
public class Main {
   A obj = new A()
   {  public void print(){
        sout ("Hello");
      }
   };
   obj.show();
```

→ Anonymous Inner class we are providing implementation of abstract class in this.

Interface:- Interface is not a Class.
By default every method in the interface are "public abstract". We can't make obj of interface we need a concrete class that "implements" the interface rather than extends

```
~~public class A~~
Interface A{
   methods();
}
public class B implements A
{  methods(){
   "
   }
}
```

Variables in an interface are either "final" & "static" by default.

* Interface does not have heap memory.

```
public class Main
{
    public static void main (String [] args)
    {
        A obj;   // A → Interface (It is a Type)
        obj = new B;
        obj.methods();
    }
}
```

final Keyword

~~The methods / Variable~~

Final:-

→ The variables declared as final cannot be reassigned once initialized

→ The methods declared as final cannot be overriden by any subclass

→ A class declared as final cannot be subclass. That means any other class cannot extends/modify the final class

* To access the static variables/methods we don't need to create obj of the class we can just call it by using Class name . variable/method();

Enum:- Enum is a short form for Enumeration

* It is a special type of class that contains constants or error codes
* We can't extends Enum class
* Enum is a data type as classes.

6/08/24

```
enum Status {
    Started, Running, Paused → Every constant represent as an instance of enum class
}
Public class Main{
public static void main(String[] args)
{ status obj = status.Running;
sout(obj)
sout(obj.ordinal()); → Print index no. of constants in enum class
Status[] ss = Status.values();
```

here enum acts as data type

```
for( Status i : ss)
{ sout(i + " "); } } }
```

Meta data:- data provides info about other data (Discription) Ex:- file name, file type, webpage title, date, etc...

Annotation:- It is a comment which is used to provide metadata to a compiler & JVM about the program

@Override → is an annotation

↓

this annotation is used to override methods of a class.

There are diff types of annotation from method level to class level.

9/08/24

Thread:- Thread light weight smallest unit of work

Framework:- It is a collection of classes & interfaces which provide ready to use architecture

Collection:- collection is a group of objects that are stored and retrive data. They provide a way to perform operations on the data.

Set:- Set is an interface in Collection that supports only unique values.

* Set does not maintain any sequence so we cant sort the values.
* Set does not support index values
* we use add() to add an element.

Set<Integer> S = new HashSet<Integer>();

*If we want a sorted set we will use Treeset # set<Integer> S = new Treeset<Integer>();

Map:- Map is a data structure that is a collection of Key-value pairs.

* They are also known as "Dictionaries".

Types of Maps
- Hash Map
- LinkedList Map
- Tree Map
- Hash Table

* Map is an Interface

MAPS:-

To add element into map

map.put(Key, value); → the corresponding

map.get(Key); → for getting the value of Key

map.remove(Key);

map.containsValue(value) → To check whether the value is there in the map.

map.containsKey(Key) → // keys.

map.size(); → To get the size of the map.

map.isEmpty(); → To get whether map is empty

map.clear(); → To remove all Key-value pairs

map.KeySet(); → Returns all keys

map.values(); → Returns all values

for Printing all the elements of the map

```
for (Map.Entry<Integer, Integer> entry:
        Map.entrySet())
{ sout (entry.getKey() + " " +
        entry.getValue()); }
```