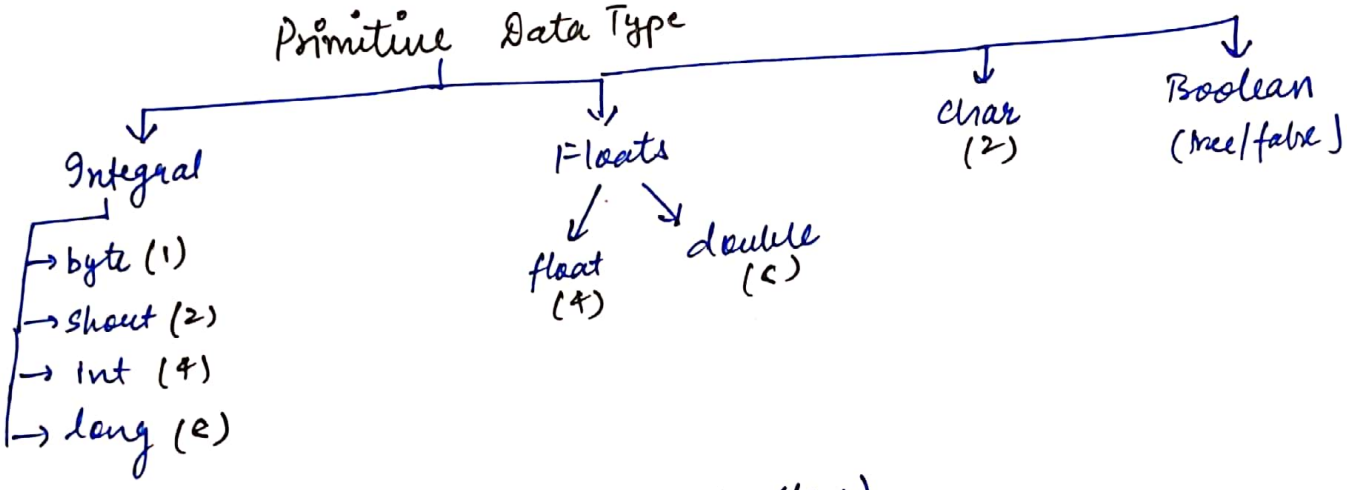
Java - Syntax - Cheatsheet

Source code → (javac) → byte code → (jvm) → Machine code

- JDK : Java development kit → tools for developing & running java programs
- JRE : Java Runtime Environment → executing programs developed in java



Strings (data type + class)

```
String name = "Rohit";
String s = new String("Rohit");
```

- name.length()
- name.toLowerCase()
- name.toUpperCase()
- name.trim()
- name.substring(int st)
- name.substring(int st, int ed)
- name.replace('r', 'p')
- name.startsWith("")
- name.endsWith("")
- name.charAt(2)
- name.indexOf(str)
- name.lastIndexOf(str)
- name.equals
- name.equalsIgnoreCase()

if-else-ladder

```
if( ){
}
else if( ){
}
else{
}
```

Logical : && , || , !

Switch - Case :

```
switch (var)
  {
     case c1 :
        ==
        break;
     case c2 :
        ==
        break;
     case c3 :
        ==
        break;
     default :
        ==
  }
```

var → integer, string or character

while loop

```
int i = 1;
while (i <= 5)
  {
     ==
     i++;
  }
```

For loop

```
for (int i = 0; i <= 5; i++)
  {
     ==
  }
```

do while

```
do
 {
    ==
 }
while ( );
```

Array

① int arr[] = new int [5];

② int arr[] = {2, 3, 4};

③ int arr[];
   arr = new int [5];

- index start from 0
- length → arr.length

for-each loop

```
for (datatype element : Arrayname)
   {
      sout(element);
   }
```

Methods

return datatype → int
name of method → sum

(1) public static int sum (int a, int b)

Syntax – 1

```
{
    return a+b;
}
```

Type I

//call ⇒ sum (2, 3); ⇒ (5) → output

(2) Class Method

Type II

Syntax – 2

```
{
    int sum(int a, int b)
    {
        return a+b;
    }
}
```

//cal → create object of this class

```
Method m = new Method();
m.sum(2,3); ⇒ (5) → output
```

Method Overloading → same name + diff. parameter

- Void :→ when don't want to return anything from method → return type also same

```
public static void telljoke()
{
    ≡
}

public static void telljoke(int a)
{
    ≡
}

public static void telljoke(String n, int b)
{
    ≡
}
```

Variable Argument (Varargs)

→ no argument compulsory

```
public static int sum(int ...arr)
{
    int s = 0;
    for (int a : arr)
    {
        s+=a;
    }
    return s;
}
```

```
sout(sum()); ✓
sout(sum(4,3,5)); ✓
```

1 arg. compulsory

```
public static int sum(int b, int ...arr)
{   int s = b;
    for (int a : arr)
    {
        s+=a;
    }
    return s;
}
```

```
sout(sum()); ✗
sout(sum(4,3,5)); ✓
```

Creating Class

```
class Employee
{
    int id;
    String nm;

    public void printDetails()
    {
        sout(id);
        sout(nm);
    }
}
```

```
main Method()
{ ↓
Employee Rahul = new Employee();
    Rahul.id = 25;
    Rahul.name = "Rahul Aryan";
    Rahul.printDetails();
}
```

Getter - Setter Methods

```
class Employee
 {
    private int id;
    private String name;
  //getter - name
  public String getName()
   {
      return name;
   }
  //setter - name
  public void setName(String n)
     {
        name = n;
          or this.name = n;
     }
  //getter - id
  public int getId()
   {
      return id;
   }
  //setter - id
  public void setId(int i)
     {
        this.id = i; or id = i;
}//class }
```

```
Main Method()
{
    Employee rK = new Employee();
    rK.setId(25);
    rK.setName("Rahul");
    Sout(rK.getName + rK.getId);
} // main
```

Constructor

```
class Construct
{
    string name; int id;
    public Construct() → without argument
    {
        sout();
    }
    public Construct(String n) → with argument
    {
        name = n;
    }
    public Construct(int i, String n)
                    ↓ multiple argument
    {
        name = n;
        id = i;
    }
}
main()
{
    Construct c1 = new Construct();
    Construct c2 = new Construct("Rahul");
    Construct c3 = new Construct(25, "Rahul");
}
```

all together Overloading of Constructor

Inheritance

```
class Base
{
    int x ;
    // - getter - setter for x
}
class Derived extends Base
    {
        int y ;
        // getter - setter for y
    }
Main()
{
    Base b = new Base() ;
        b.setX((4)) ;
    Derived d = new Derived() ;
        d.setX((6)) ;
        d.setY((8)) ;
}
```

Method - over - ridding

```
class A
{
    public void met2()
        {
            sout(" - - ");
    }   }
```

```
class B extends A
{
    @Override
    public void met2()
    {
        sout("...");
    }
}

Main()
{
    A a = new A();
    B b = new B();
    a.meta();  → A class meth2
    b.met2();  → B class meth2
}
```

Dynamic Dispatch or Polymorphism

```
class Phone
{
    ===
}

Class SmartPhone extends Phone
{
    ===
}

main()
{
    Phone p = new SmartPhone();
    p.all methods of phone will be executed.
}
```

But if any method name appears on both then smartphone gets executed.
→ only print method present in phone.

}//main

Constructor - Inheritance

```
class EK
 {
    int a ;
    public EK() → Constructor
      { sout(" ... ") ;
      }
    public EK(int a) → Constructor with Arg.
      {
        this.a = a ;
      }
 }

Class DO extends EK
  {
     public DO (int c) → construct with Arg.
       {
          super(c) ; → to call const. with Arg from Base class
          sout(" ... ");
       }
  }
```

otherwise throws error if super not used else use her const. without Argument.

Abstract class & methods

```
abstract class Parent
{
    public void say()
    {
        sout("Good Morning");
    }

    abstract public void greet();
    abstract public void greet2();

}

class child extends Parent
{   @Override
    public void greet()
    {   sout("Hii...");
    }
    @Override
    public void greet2()
    {   sout("Hiieee...");
    }

}

main()
{
    Parent p = new Child();
    Child  c = new Child();

    Parent p = new Parent(); → abstract class ka object nahi hota
```

Interface

```
interface Bicycle
{
    void applyBrake();
    void speed();
}

interface HornBicycle
{
    void Horn();
}

Class Avon implements Bicycle, HornBicycle
{
    public void Horn()
    {
        sout("Peee...");
    }
    public void speed()
    {
        sout("Hm....");
    }
    public void applyBrake()
    {
        sout("Br...");
    }
}

main()
{
    Avon a = new Avon();
}
```

don't create object with interfaces

Inheritance in Interface

```
Interface Sample1
{
    void met1();
    void met2();
}

interface Sample2 extends Sample1
{
    void met3();
    void met4();
}

class Sampleclass implements Sample2
{
    -
    =
    -
}

main()
{
    Sampleclass sc = new SampleClass();
}
```

Create Thread → extending Thread class

```
class MyThread1 extends Thread
{
    public void run()
    { int i = 1;
      while (i<=10)
      { sout(" - - - ");
        i++;
}}}
```

```
clars MyThread2 extends Thread
{
    @overridd
    public void run()
    {
        int i = 1;
        while (i<=10)
        {
            sout(" --- ");
        }
        i++;
        }
    }
}

public clars simple
{
    main()
    {
        Mythread t1 = new MyThread1();
        Mythread2 t2 = new MyThread2();

            t1.start();
            t2.start();
    }
}
```

Create Thread → Runnable Interface

```
clars MyThread1 implements Runnable
{
    public void run()
    { int i = 0;
      while (i<500)
      { sout(" ... ");
        i++;
    } } }
```

```
class MyThread2 implements Runnable
{
    public void run()
    {
        =
    }
}

main()
{
    MyThread1 b1 = new MyThread1();
    Thread t1 = new Thread(b1);
        t1.start();
    MyThread2 b2 = new MyThread2();
    Thread t2 = new Thread(b2);
        t2.start();
}
```

Constructor in Thread

```
class MyTh extends Thread
{
    public MyTh(String name)
    {
        super(name);
    }
    public void run()
    {

    }
}
```

```
Main()
{
    MyTh t1 = new MyTh("Ram");
        t1.start();
print ( t1.getId(); )
 print ( t1.getName(); )
```

Thread – Methods

```
class MyTh1 extends Thread
  {
      public void run()
        {
          int i=1;
          while(i<=10)
            {
              try
                {
                  Thread.sleep(200);
                }
                catch(Exception e)
                  {
                    sout(e);
                  }
                i++;
            }//while
        }//run
    }//class

class MyTh2 extends Thread
  {
    -
    -
  }
```

```
main()
{
    t1.start();
    t2.start();
}
```
→ Both started concurrently

* to start t2 only after t1 ends

```
t1.start();
try
{
    t1.join();
}
catch(Exception e)
{
    sout(e);
}
t2.start();
```

* setting priorities :⇒

```
t2.setPriority(Thread.MAX_PRIORITY);
```
or
```
t2.setPriority(Thread.MIN_PRIORITY);
```

try - catch

```
try
{
    int c=a/b;
    sout(c);
}
Catch(Exception e)
{
    sout(e);
}
```

Nested - try - catch

```
try{
    try
    {
    }
    catch(Exception e)
    {
    }
}
Catch(Exception e)
{
}
```

Handling Specific Exception

```
try {
    =
}
Ceatch(Arithmatic Exception e)
{=
}
Ceatch(ArrayIndexOutofBoundException e)
{
  =
}
catch(Exception e)
{
  ==
}
```

finally - block

```
try {
    =
}
Catch(Exception e)
{
  =
}
```

```
finally
{
  ==
}
```

- finally → when errors occurs in exceptions
- try - catch together → no error
- try - finally together → we get error → although statements of finally block will get executed in the output

Throw - throws Keyword

```
class MyException extends Exception
  {
     @Override
      public String toString()
        {
          return "----";
        }
     @Override
      public String getMessage()
        {
          return ".....";
        }
  }

main()
 {
   public static int divide(int a, int b) throws
                                ArithematicException
     {
     }

   public static double area(int r) throws MyException
     {
        if(r<0)
          {
            throw new MyException();
          }
```

Advanced - Java

File Handling

```
import java.util.Scanner;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.io.FileNotFoundException;

main()
{
    File f = new File("rk.txt");
    // file creation
    try
    {
        f.createNewFile();
        sout("file created sucessfully");
    }
    catch(IOException e)
    {
        e.printStackTrace();
    }
    // file writing
    try
    {
        FileWriter fw = new FileWriter("rk.txt");
        fw.write("This is my 1st file");
        fw.close();
    }
    catch(IOException e)
    {
        e.printStackTrace();
    }
```

//reading content of file

```
try
{
    Scanner sc = new Scanner(f);
    while(sc.hasNextLine())
    {
        String st = sc.nextLine();
        sout(st);
    }
    sc.close();
}

catch(FileNotFoundException e)
{
    sout("file not found");
    e.printStackTrace();
}
```

//delete a file

```
if(f.delete())
{
  sout(" Deleted ! ");
}
else
{
  sout(" Error ! ");
}
```

Annotations

↓

provide extra infor. to program

(1) @Override → when method of same name exit in 2 classes.

```
class KeyPad
{
    void sendMess()
    {
    ----
    }
}

Class Android
{
    @Override
    void sendMess()
    {
    -----
    }
}
```

(2) @Deprecated

→ high chance of removal in future version
→ complier generates warning

```
class KeyPad
{
    @Deprecated
    void sendMess()
    {
    ------
    }
}

class Android
{
    @Override
    void sendMess()
    {
    ----
    }
}
```

③ @SuppressWarnings ("deprecation")

→ supress warning generated by compiler

```
Main()
{
   @SuppressWarnings ("deprecation")
    Android Sam = new Android();
     Sam.sendMess();
}
```

④ @FunctionalInterface → interface that contains only 1 abstract method

```
@FunctionalInterface
Interface myF
{
   void method1();
}

Main()
{
    sout ("----");
}
```

Anonymous Class

```
interface Demo
{
    void met1();
    void met2();
}
```

```
Main()
{
    Demo obj = new Demo()
    {
        @Override
        public void met1()
        {
            sout(" ----- ");
        }

        @Override
        public void met2()
        {
            sout(" --- ");
        }
    };

    obj.met1();
    obj.met2();
}
```

Lambda func:

```
@FunctionalInterface
interface My
{
    void met1(int a);
}

Main()
{
    My obj = (a) ->
    {
        sout(" ... " + a);
    };
    obj.met1(23);
} //main
```

Java Generic

```
Main()
{
    ArrayList<Integer> al = new ArrayList<>();   // Generic
    MyGeneric<String, Integer> g1 = new MyGeneric(23, "String");
    int a = (int) al.get(1); //typecasting
```

Array - list → modified Array

```
import java.util.*;                    => [ ,  ,  , ]
or import java.util.ArrayList;

Main()
{
    ArrayList<Integer> l1 = new ArrayList<>();
    ArrayList<Integer> l2 = new ArrayList<>();

    l1.add(6);  → adding elements
    l2.add(7);
    l1.add(0, 51);   element → adding elements at given index
                 index
    //to print its elements
    for(i=0; i<l1.size(); i++)
    {
        sout(l1.get(i) + "  ");
    }
    l1.size(); → length of l1 arraylist

    l1.addAll(l2); → add l2 at the end of l1

    l1.addAll(0, l2); → add l2 at the beg.

    l1.contains(7); → check whether 7 present or not
    l1.indexOf(5); → index of 5 in list
    l1.lastIndexOf(5); → last index of 5 in list
}
```

Linked - List

→ [, , ,]

```
main()
{
    LinkedList<Integer> l1 = new LinkedList<>();
    LinkedList<Integer> l2 = new LinkedList<>();
        l1.add(6);  → adding elements
        l1.add(7);
        l1.add(0, 51);
                ↑ index   ↑ element
    for(i=0; i<l1.size(); i++)
    {
        sout(l1.get(i) + "  ");
    }
        l2.add(45);
        l2.add(99);
        l1.size();  → length of l1
        l1.addAll(l2);  → add l2 at the end of l1
        l1.set(3, 567);  → index, element
        l1.addFirst(789);  → add to 1st
        l1.addLast(77);  → add to last
        l1.clear();  → clear all elements of l1
        l1.contains(7);  → check whether 7 present or not
        l1.indexOf(4);  → index of 4 in list
        l1.lastIndexOf(5);  → last index of 5 in list
}

    import java.util.LinkedList;
or  import java.util.*;
```

Hash-Set

import java.util.HashSet;
or import java.util.*;

```
Main()
{
    HashSet<Integer> hs = new HashSet<>(6, 0.5f);
```

load factor ↑

↓ initial capacity

```
    hs.add(6); → add elements
    hs.add(9);
    hs.add(16);
```

[, ,]

```
}
```

Array-Dequeue

import java.util.ArrayDeque;
or import java.util.*;

```
Main()
{
    ArrayDeque<Integer> ad = new ArrayDeque<>();
    ad.add(6); → add element
    ad.add(56)
    ad.addFirst(78); → add 1st
    ad.add(99);
    ad.getFirst(); → get 1st
    ad.getLast();
```

[, ,]

Date class in Java

- 1900 → start years of java
- holds the no of milisecond passed since 1 Jan 1970.
- Current time in mili seconds : → System.currentTimeMillis();
- " seconds : → System.currentTimeMillis()/1000;

#.
```
Date d = new Date();
Sout("Date current : " + d);

d.getDate();
d.getMonth();
d.getYear();
d.getHours();
d.getMinutes();
d.getSeconds();
```

Digit format time : ⇒
```
(d.getHours()+":" + d.getMinutes()+":"
    + d.getSeconds());
```

Calendar class

```
import java.util.*;

main()
{
    Calendar c = Calendar.getInstance();

    c.getCalendarType();
    c.getTimeZone();
    c.getTimeZone().getID();
    c.getTime();

    c.get(Calendar.HOUR);
    c.get(Calendar.MINUTE);
    c.get(Calendar.SECOND);

GregorianCalendar gc = new GregorianCalendar();

    gc.isLeapYear(2006);
    gc.getTime();
    gc.roll(Calendar.MONTH, true);   // +1
    gc.roll(Calendar.DATE, false);   // -1
    gc.roll(Calendar.YEAR, true);    // +1

    gc.hashCode();
```

Time - Class

```
import java.time.*;
import java.time.format.DateTimeFormatter;
Main()
{
    LocalDate d = LocalDate.now();          → current date
    Sout(d);

    LocalTime t = LocalTime.now();          → current time
    sout(t);

    LocalDateTime dt = LocalDateTime.now();
    sout(dt);                               → current (date + time)

    DateTimeFormatter df = DateTimeFormatter.ofPattern("dd/MM/yyyy");
    String st = dt.format(df);
    sout(st);

    Clock c1 = Clock.systemDefaultZone();
    c1.getZone();   ⇒ Asia/Calcutta
    c1.instant();   → current time
```

Arrays

* ways to create :-

1) int arr[] = {23, 54, 68};

2) int arr[] = new int[3];

3) int arr[];
   arr = new int[3];

* for each loop

```
for (double element : arr)
    {
        sout(element + " ");
    }
```

* array display

```
for (i=0; i<arr.length; i++)
    {
        sout (arr[i] + " ");
    }
```

* user input array

```
for (i=0; i<n; i++)
    {
        arr[i] = sc.nextInt();
    }
```

* Sum of 2 array

$[]_{2\times2} + []_{2\times2} = []_{2\times2}$

```
for (i=0; i<mat1.length; i++)
    {
        for (j=0; j<mat1[i].length; j++)
            {
```

```
        s[i][j] = mat1[i][j] + mat2[i][j] ;
        Sout(" ");
      }
      Sout("\n");
  }
```

* Reverse Array

```
n = Math.FloorDiv(arr.length, 2)
  ∴ n = (3,2) ⇒ 1   (GIF)
    ln = arr.length ;
for (i=0 ; i < n ; i++)
  {
      temp = arr[i] ;
      arr[i] = arr[ln - i - 1] ;
      arr[ln - i - 1] = temp ;
  }
```

* Multiplication Array

```
for (i=0 ; i < mat1.length ; i++)
  {
    for (j=0 ; j < mat1[i].length ; j++)
      {
        for (k=0 ; k < mat1.length ; k++)
          {
            mul[i][j] = mat1[i][j] * mat2[j][i];
          }
        Sout(mul[i][j]+" ");
      }
    Sout("\n");
  }
```