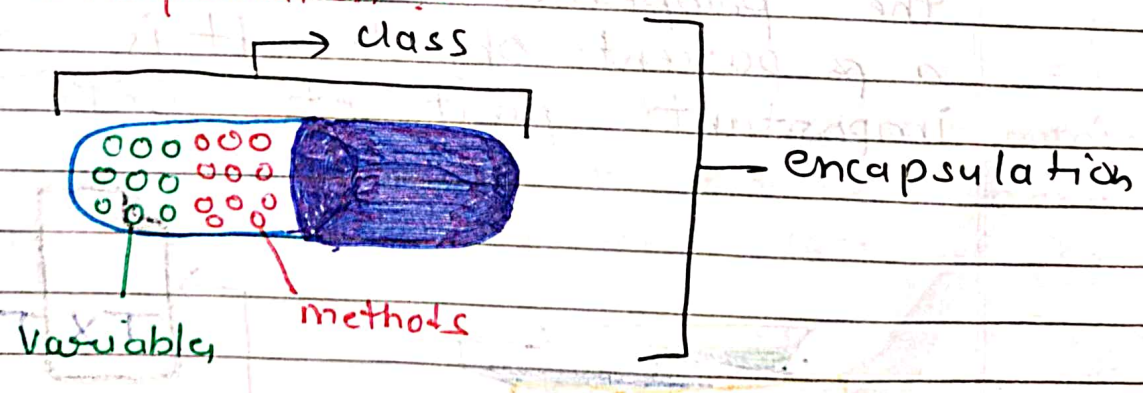
Introduction

Java programming paradigms :- Java is based on the concepts of object-oriented programming. As the name suggests, at the center of it all is an object. Objects contain both data and the functionality that operates on that data. This is controlled by the following four paradigms.

* Encapsulation
* Inheritance
* Information hiding
* Polymorphism

* Encapsulation :-



Encapsulation in Java is a process of wrapping code and data together into a single unit, for example

a capsule which is mixed of serval medicines

example

Book (Object)

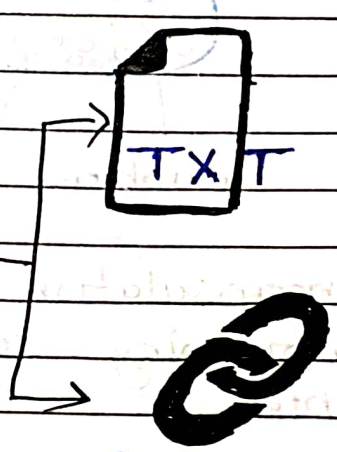
data
- Title
- Author
- Cost
- Borrow DOT

methods
- Issue()
- fine()
- Return()
- Open()
- Close()

* Inheritance in Java :-

Inheritane in Java is a mechanism in which one object acquires all the properties and behaviors of a parent object. It is an important part of OOPs.

Books



* Information hiding :-

Public
- Title
- Author

Protected
- Account No

Private
- Cost

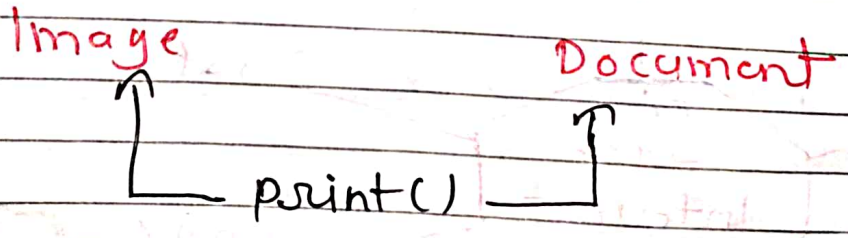
Public
- Issue ()
- Returns ()

Protected
- Resave ()

Private
- Open ()
- close ()

* polymorphism :-

Polymorphism refers to a programming language's ability to process objects depending on their class.

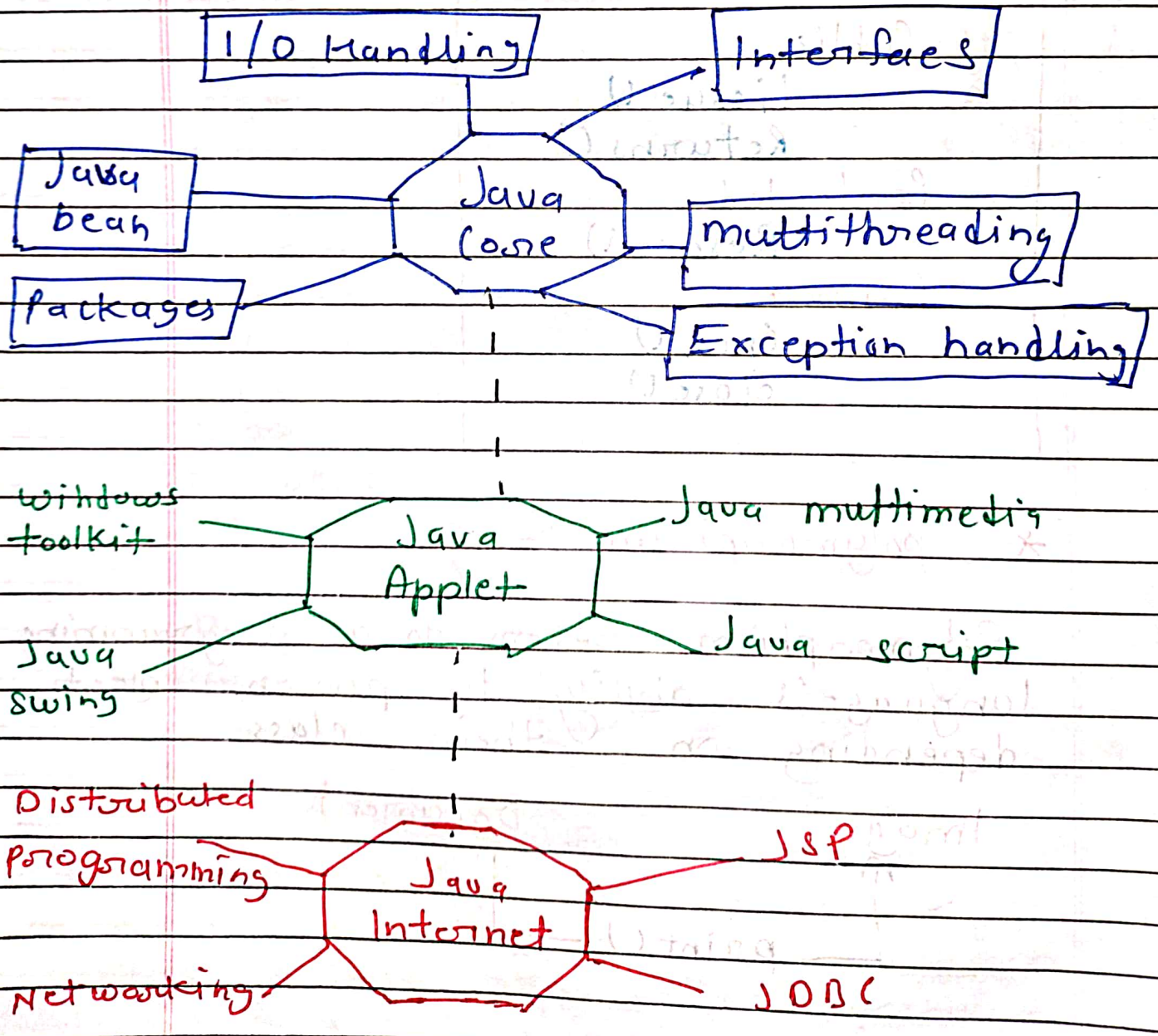


Add (x, y) : 2+34 (Add two numbers)

Add (S1+S2) : Debasish Samanta (Merge two strings)

Add (Img, Doc) : Image + Document (past image to document)

Features of Java programming



C versus Java

Aspects	C	Java
(1) Paradigms	Procedual	Object-oriented
(2) Platform Dependency	Dependent	Independent
(3) Datatypes: union structure	Supported	Not-supported
(4) Pre-processor directives	Supported (#include, #define)	Not supporte
(5) Header file	supported	Use packages. (import)
(6) Storage class	Supported	Not supported
(7) Inheritance	No inheritance	Support simple inheritance
(8) Pointers	support	No-pointers
(9) Code translation	Compiled	Interpreted
(10) Multi-threading and Interface	Not - supported	Supported
(11) Exception-handling	No-exception handling	Supported
(12) Database Connectivity	Not - supported	Supported

	Features	C++	Java
(1)	Data abstraction and encapsulation	✓	✓
(2)	Polymorphism	✓	✓
(3)	Binding (a) Static	✓	✓
	(b) Dynamic	✓	✓
(4)	Inheritance (a) single	✓	✓
	(b) multiple	✓	✗
(5)	Operator overloading	✓	✗
(6)	Templete class	✓	✗
(7)	Global variable	✓	✗
(8)	Header file	✓	✗
(9)	Pointers	✓	✗
(10)	Interface and packages	✗	✓
(11)	~~App~~ API (Application Programming Interface)	✗	✓

Date
Page

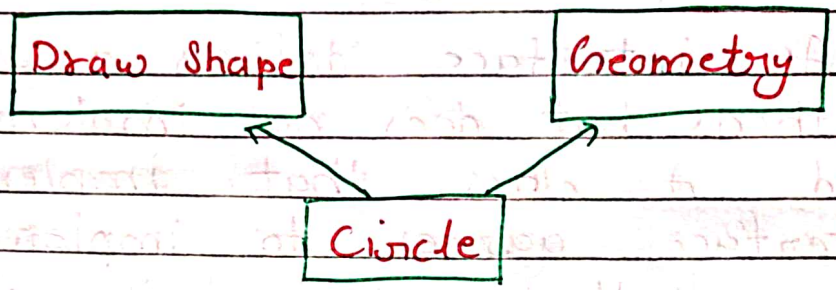
Tools

There are seven main programs in SDK

- Javac the Java Compiler
- Java the Java Interpreter
- Javadoc generates documentation in HTML
- Appletviwer the Java Interpreter to execute Java applets.
- Jdb the Java Debugger to find and fix bugs in Java programs
- Javap the Java Disassembler to displays the accessible function and data in a compiled class; It also displays the meaning of byte code
- Javah to create interface between Java and C routines.

Interface

* Multiple inheritance in Java



* However, this is not possible, as Java does not support multiple inheritane.
* Java's solution to this problem is called interface.

What is an interface?

* An interface is basically a kind of a class. Like classes, an interface contains members and methods; unlike classes, in interface, all members are final and all methods are abstract.

* for both interface and abstract class object instanb are not created.

* As an abstract class may contain both abstract methods and normal methods.

* An interface define a protocol of behavior that can be implemented by any class anywhere in the class hierarchy.

* An interface defines a set of methods but does not implement them and A class that implement the interface agrees to implement all the methods defined in the interface thereby agreeing to certain behavior.

* Interface reserve behaviors for classes that implement them.

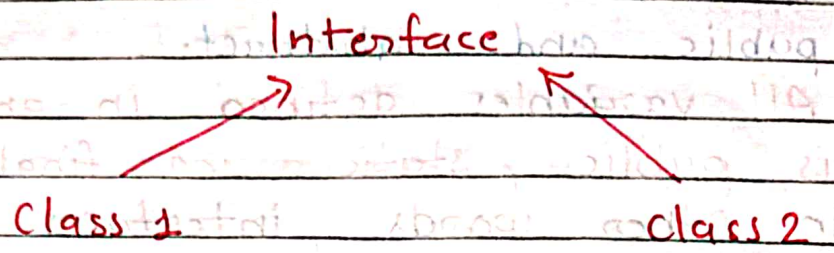
* Methods declared in an interface are always public and abstract,

* Static methods cannot be declared in the interfaces - these methods are never abstract and do not express behavior of objects.

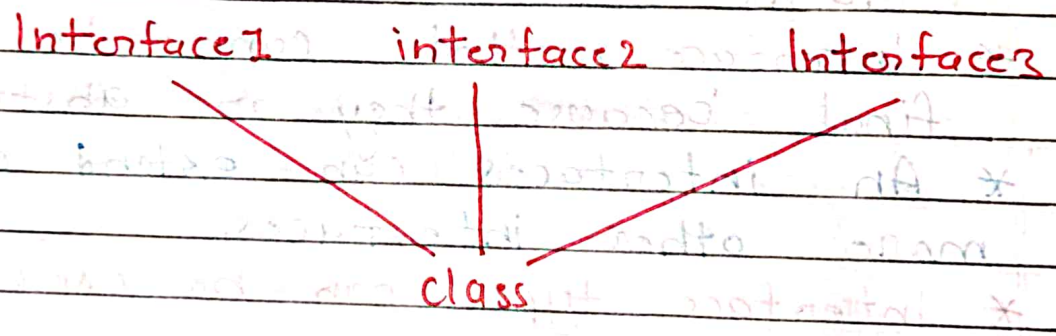
* How to define interfaces ?

```
interface callMe {
    void call(int p);
}
```

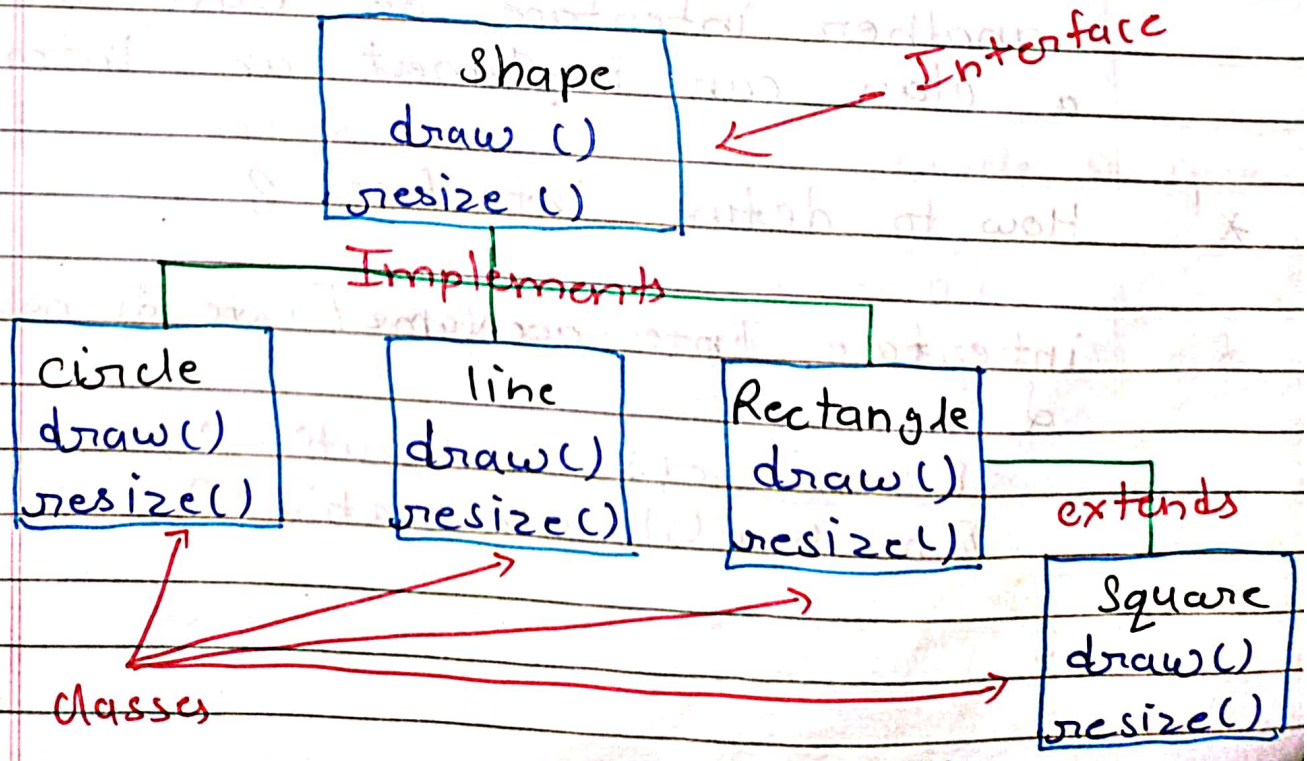
* Once an interface is defined, any number of classes can implement an interface.



* Also, one can implement any number of interfaces using a class.



example



Date
Page

Properties of interfaces

* Declared with keyword interface.
* All interface methods are implicity public and abstract.
* All variables defined in an interface is public, static and final.
  in other words, interfaces can declare only constants, no instance variables.
* Interface methods must not be static.
* Interface methods cannot be marked final because they are abstract.
* An interfaces can extend one or more other interfaces.
* Interface type can be used polymorphically.
* An interface cannot implement another interface or class while a class can implement an interface.

*' How to define interface ?

```
interface InterfaceName [extends name1,...]
{
    [variable(s) declaration;]
    [methods(s) declaration;]
}
```

Variable declaration

```
static final type varName = value;
```

Method declaration

```
Return_type methodName(parameters);
```

example 1

```
interface anItem {

static final int code = 101;
static final String itemName = "Come";
void recordEntry();

}
```

example 2

```
interface Curves extends Circle, Ellipse
{
  static final float pi = 3.142F;
  float area (int a, int b);
}
```

example

```
interface GeoAnalyzer {
    final static float pi = 3.142F;
    float area();
    float perimeter();
}

class Circle implements GeoAnalyzer {
    float radius;
    Circle( float r ) {
        radius = r;
    }
    public float area () {
        return (pi * radius * radius);
    }
    public float perimeter() {
        return (2 * pi * radius);
    }
}

class Ellipse implements GeoAnalyzer {
    float major;
    float minor;
    Ellipse ( float m, float n) {
        major = m;
        minor = n;
    }
    public float area () {
        return (pi * major * minor);
    }
```

```
public float perimeter() {
    return (pi * (major + minor));
    }
}
```

Some of Java's most used interfaces:-

(1) Iterator :-

To run through a collection of objects without knowing how the objects are stored, for example, in array, list, bag or set.

(2) Cloneable :-

To make a copy of an existing object via the clone() method on the class object.

(3) Serializable :-

Pack a web of objects such that it can be used a network or stored to disk. A naturally later be restored as a wed of objects comparable

(4) Comparable :-

To make a total

order of objects, for example
2, 56, 67, 879, 3422, 34234

(i) Iterator interface :-

↳ It is in the pakage java.util
↳ It works on collections

example :-

```
package java.util.*;
public interface Iterator {
    public abstract boolean hasNext();
    Object next();
    void remove();
}
// use an Iterator
myShapes = getSomeCollectionofShapes();

Iterator Itor = myshapes.Iterator();
while ( itor.hasNext() ) {

    Shape s = (Shape) Itor.next();
    s.draw();
```

(2) Cloneable interface :-

↳ The interface is empty, that is, It has no method.
↳ returns an identical copy of

an object.
↳ A shallow copy, by default
↳ A deep copy is often performable

↳ Preventions of cloning.
↳ Necessary if unique attribute, for example, data base lock or open file reference.
↳ Not sufficient to omit to implement cloneable
→ Sub class might implement it
↳ clone method should throw an exception.
→ CloneNotSupportedException.

Example

```
public class Car implements Cloneable {
    private String make;
    private String model;
    private double price;
    public Car(); {
  // default constructor
    this(" ", " ", 0.0);
  }
// gives reasonable values to instance variable
  public Car(String make, String model
      double price) {
    this.make = make;
    this.model = model;
```

```
        this.price = price;
    }
    public Object clone () {
        // the Cloneable interface
    return new Car (this.make, this.model,
        this.price);
    }
}
```

(3) Serializable interface :-

```
public class car implements Serializable {
    // rest of class unaltered
}
// write to and read from disk

import java.io.*;

public class SerializeDemo {

Car myToyota, anotherToyota;
myToyota = new Car("Toyota", "carina",
                42312);
ObjectOutputStream out = getOutput();
out.writeObject(myToyota);
ObjectInputStream in = getInput();
anotherToyota = (car) in.readObject();
}
```

(4) Comparable interface

example

```
public class IPAddress implements Comparable{
private int[] n;
// here IP stored, eg. 125.555.201.123
/** The comparable interface */
public int CompareTo (object o){
IPAdress other = (IPAddress) o;
int result = 0;
for(int i=0; i<n.length; i++){

if( this.getNum(i) < other.getNum(i){
    result = -1;
    break;
}
if (this.getNum(i) > other.getNum(i){
    result = 1;
    break;
}
}
    return result;
}
}
```

Encapsulation

Concept of encapsulation :-

- the object is at the core of Java programming.
- Java provides the concepts of class to buit objects
- A class defines the shape and working of an object.
- The concept of class is the logical construct upon which the entire Java is built.

what is a class ?

- A class is a group of objects, which have common properties.
- It is a template or blueprint for which objects are created.
- It is a logical entity.

A class in Java can contain :

- Fields
- Methods
- Constructors
- Blocks
- Nested class(es) and interface(s)

Declaring object of type Circle class

```
class Circle {
    double x,y; double r;
```

```
    double circumference () {
        return 2*3.14159* r ;
    }

    double area () {
        return (22/7) * r * r ;
    }
}

Class CircleDemo1 {
  public static void main (String args[])
  {
    Circle c = new Circle ();
      c.x = 0.0;
      c.y = 0.0;
      c.r = 5.0;

    System.out.println("Circumference"+
            c.circumference ());
    System.out.println("Area " + c.area());
  }
}
```

* We can create multiple objects of a class.
* We can create multiple classes.

Important notes:-

* There should be a class which contains a method main ().

This class is called main class

* There should be only one main class.

* The name of the program file should be same as the name of the main class followed by .java as an extension.

* If there is no main class, then there should be compilation error.

Constructor :-

Constructor for automatic object initialization :-

* It can be tedious to initialize all of the variables in a class each time an object is instantiated
* Java allows objects to initialize themselves when they are created / declared.
* This automatic initialization is performed through the concept of constructor

properties :-

* A constructor initializes an object immediately upon creation.

* Constructor in Java is a method.
* This method has the same name as the class in which it resides.
* Once defined, the constructor is automatically called immediately after object is created.
* Constructor is a method which has no return type.
* In fact, the implicit return type of a class constructor is the class type itself.
* Constructor initialize the internal state of an object

~~This~~ this Keyword concept:-

* this is used to reduce name-space collisions
* Sometimes a method will need to refer to the object that invoked it.
* To allow this Java define this keyword.

* this can be used inside any method to refer to the current object
* That is, this is always a reference to the object on which the method is invoked

Java

Introduction to Java

What is Java ?

* HLPL
* Developed by James Gosling at Sun microsystems. (1995).
* First Name → Oak
  After change → Java

Use

* Android Apps Development.
* Desktop & Web Applications.
* Games Development.
* Database Connection.

Features of Java

* OOP
* Platform independent

C, C++ Program
Demo.cpp
Run → Windows

Java program → Demo.Java
Demo.class
Run → Linux → mac

Types of
(i) Instance
(ii) local
(iii) static

types of data types
(i) Primitive
(ii) non-primitive

Widening
Typecasting
Overflow

* Simple
* Portable (Bitcode → Run)
* Secure
* Robust (strong memory managment)
* multi Threading.

Basic structure

* Package declaration. (If any).
* Import statment (To add any specify classes to use in code).
* class defination (class name, variables and methods).
* A main method.
* Additional class and methods definitions (If needed).

example

```
package com.example;

import java.util.Scanner;

Public class Main {

Public static void main (String[] args) {
Scanner Input = new Scanner (System.in

System.out.println ("Enter a number:")
```

```
int number = input.nextInt();
System.out.println(" You entered : " +
                number);
}
}
```

↓

Demo.java → Compile → Demo.class
(Bitcode or semi-executable)

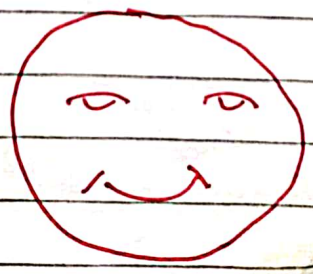
That's why it is platform independent

Simple code to Add two number

```
public class Demo {
    public static void main(String[] args)
    {
        int a = 5;
        int b = 6;
        int c = a + b;
        System.out.println(" This is the sum of
            a and b " + c);
    }
}
```

main function

class, object, function



* Keep growing.
* Work hard.
* let's start and see.

User Input

1. Using Scanner class

Let's start with a class Demo.

```
import java.util.Scanner;
        Or
import java.util.*;

public class Demo {

  public static void main(String[] args
  {
    Scanner object = new Scanner(System
                                .in);
    int a = object.nextInt();
    int b = object.nextInt();

    int c = a+b;

System.out.println("This is the
    sum of a and b" + c);

  }
}
```

```
int num = 6;
float num = 3.14f;
double num = 3.14;
String name = "Sandeep";
```

→ Variables.

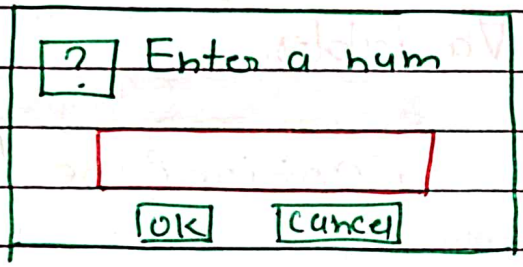
2. Using JOptionPane

```
import javax.swing.JOptionPane;
Public class Demo {
public static void main(String[] args){
int a = Integer.parseInt(JOptionPane
        ShowInputDialog(null," Enter first
        number "));
int b = Integer.parseInt(JOptionPane.
        showInputDialog(null," Enter second num"));
int c = a+b;
JOptionPane.ShowMessageDialog(null," Result
        is = " + c);
}}
```

explanation of ↓

```
int a = Integer.parseInt(JOptionPane.showInputDialog(null, "Enter a num"));
```

Here we are going to take a user input by showing an Dialog Box



* This Dialog Box will recieve input in string.
  So parseInt function is used to convert it into integer

like

Integer.parseInt() (In short Typecasting)

↓ class ↓ function (method)

* To show the Dialog Box we used of showInputDialog function

like

JOptionPane.showInputDialog();

↓ Class ↓ function (method)

Java Decision Making

* If, If-else, & Nested If

If statment :

```
If ( condition ) {
    // Code ;
}
```

example

```
int x = 5;
If ( x > 0 ) {
System.out.println(" x is positive ");
}
```

If-else statment

```
If( condition ) {
        // code ;
}
else {
        // code ;
}
```

example

```
int x = 4;
If( x ≥ 0 ) {
System.out.println(" x is positive ");
}
else {
System.out.println(" x is negative ");
}
```

* Nested If Statment.

```
If( condition1) {
    // code
}
If( condition2 ) {
    // code
} }
```

example

```
int x = 5, y = 10;
If(x > 0) {
    If(y > 0) {
System.out.println("x and y are both
    positive");
} }
```

* Hense we can use of many if condition in anothe if condition.

Switch case

The 'switch' statement in Java is used to execute a block of code based on the value of an expression.

Statement :-

```
switch (expression) {

    case value 1;
      // code
      break;

    Case value 2;
      // code
      break;
      :
      :

    default;
      // code
      break;
}
```

example

```
int day = 3;

switch (Day) {

case 1;
  System.out.println ("Monday");
  break;

case 2;
System.out.println ("Tuesday");
  break;

Case 3;
System.out.println ("Wednesday");
break;
```

```
        |       |       |
        |       |       |
        |       |       |

    defaut ;
      System.out.println(" Invalid day!"
        break;

}
```

Loops

* while loop

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

example

```
int i = 1;
while ( i <= 10) {
System.out.println( i );
    i++;
}
```

output

```
1
2
3
4
5
6
7
:
10
```

Do-while

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

For loop

```
for ( int i=0 ; i<10 ; i++) {
    // statments
}
```

initialization — condition — update

example

```
for (int i=2 ; i<=10 ; i++) {
System.out.println( 2*i );
}
```

Output

2
4
6
8
10
.
.
.
20

If we use an another for loop in current for loop than it will called Nested for loop.

imp

Array in Java :-

* Arrays are used to store multiple values in a single variable.
* But values should be same type.

Syntax: dataType[] arrayName = New type[size];

keyword

Example :- int Array[] = new int[5];

Note: declaration at compile time
definition at run time.

Input in Array :-

```
for(int i=0; i<array.length; i++)
{
    array[i] = sc.nextInt();
}
```

object of Scanner class

Input function

Two dimentional array :-

```
int array[][] = new int[i][j];
```

row

column

Exception Handling

What is Exception ?

Exception is an abnormal condition in a program. The class Exception Represents exceptions that a program faces due to abnormal or special conditions during execution.

Syntex:

```
try {
    // Code
}
catch ( ExceptionName object)
{
    // Message.
}
```

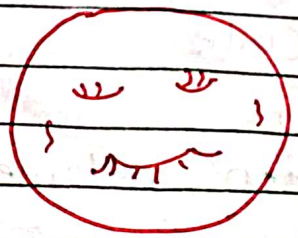
* By using catch keyword we were showing exception which java understant.
* There are also some errors or we can say exception which are not exceptions for java, so to show error message for these type of conditions we use of throw keyword.

Syntex

```
throw new ExceptionName
        (" Message : ");
```

example

```
int balance = 4000;
int withdrawAmount = 6000;

If( balane < withdrawAmount )
throw new ArithmeticException (" Insufficient balane ");

balance = balance - withdrawlAmount;
System.out.print ln (" Transaction Successfully");
```



ha ha ha - - -

Package

User define package

Packages are nothing more than the way we organize files into different directories according to their functionality, usability as well as category they should belong to.

Syntex : package foulderName;

* while creating a ~~w~~ package we don't use of ~~any~~ main function.

How to st create :-

1 → Go to terminal
2 → write → javac -d . ClassName
3 → Enter

space

Now we can use the methods of the classes which are created in this package

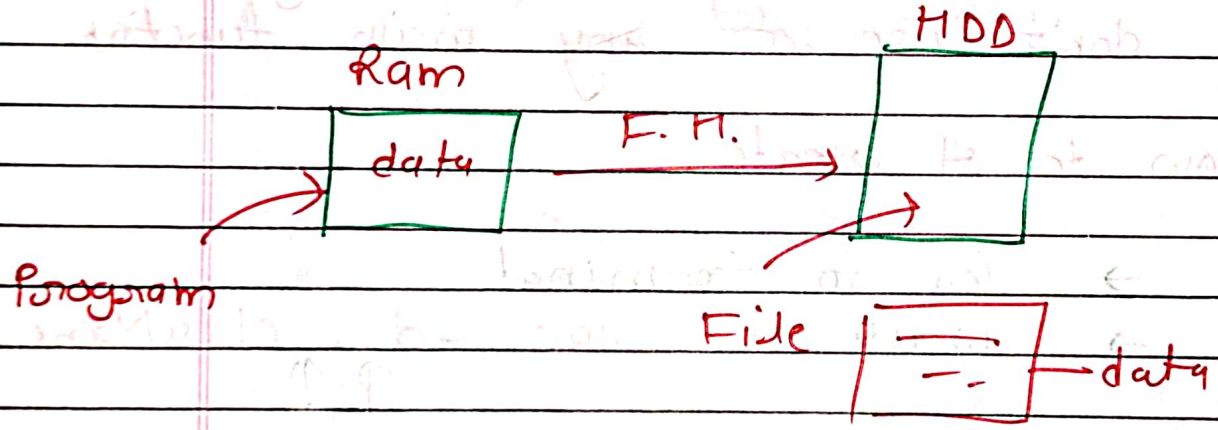
* first we will import the package.

Syntex. import packageName.ClassName;
Now create the object of this class and use the methods.

File Handling

Java has several methods for creating, reading, updating, and deleting files

* By using this method we will save the data in HDD from Ram.



Write:

* Import Java.io.FileOutputStream;
* Create object of this class.
* Store data in a variable and convert it into the byte code
* Write the byte code in the file

Example

```
try {
FileOutputStream fout = new FileOutput
stream("E:\\Test\\testout.txt");
```

Folder Name — File name

```
String s = " Hey I am Sandeep ";
byte b[] = s.getBytes();
fout.write(b);
fout.close();
System.out.println(" success- -");
}
catch (Exception e) {
  System.out.print(e);
}
```

* **Read**

example

```
import java.io.File;
File fin = new File("E:\\testout.
                    txt");
Scanner sc = new Scanner(fin);
while( sc.hasNextLine() ) {
  String data = sc.nextline();
  System.out.println(data);
}
sc.close();
```

Collection framework

→ Collections of classes and interfases.

Iterable
↑
Collection
↑
List | Queue | set

Methods on Collection

add, size, remove, Iterate
addAll, removeAll, clear.

List

- → ArrayList
- → LinkedList
- → Vector → Stack

* Queue FIFO → First in First Out

- → PriorityQueue
- → Linked List
- → Deque → Double ended queue
  - → ArrayDeque

* Set

- → HashSet
- → LinkedHashSet
- → SortedSet
  - → TreeSet

* Map

- → Hashmap
  - → LinkedHashmap
- → Sorted map
  - → Hashtable
    - → Treemap

Array List

↳ Java.util.ArrayList;

Note:

* In Array we have continuous memory allocation and fixed size so now we will study Array list.

Array	Array List
* Continuous memory	non Continuous memo
* primitive data typs, object (stores)	Objects (stores)
*	stores in heap memory

Define

ArrayList<Integers list = new Arraylist<>();

↓ Data type

To add:-

list.add(data);

To get:

List.get();

↳ Index

* To add in mid or at any place

```
list.add ( Index, data) ;
```

* To chance any data.

```
list.set ( Indext, newData);
```

* To delete any element

```
list.remove ( Index) ;
```

* To count size or array list.

```
list.size ();
```

* ~~to~~ Itrate / Loop

```
for ( int i=0; i<list.size() ; i++) {
    System.out.print ( list.get (i)) ;
}
```

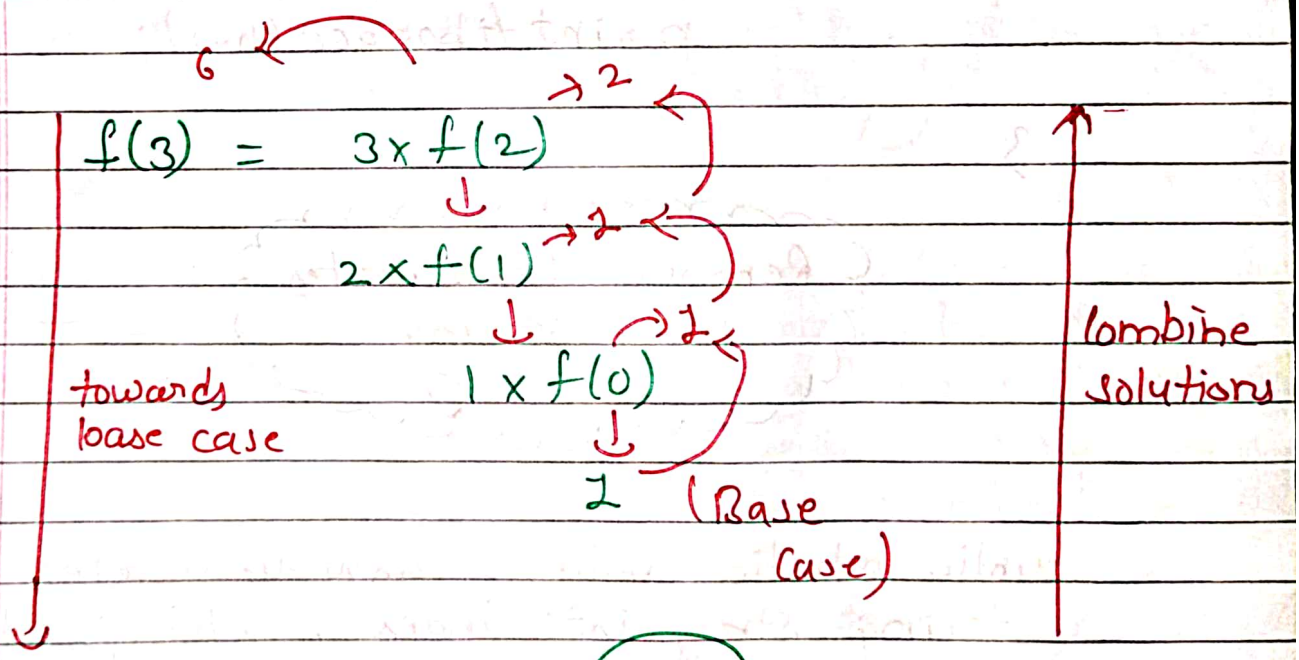
* Sorting → java.util.collection;

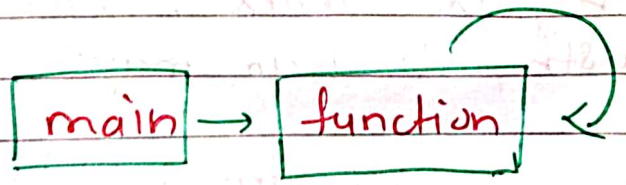
```
Collections.sort (list) ;
```

Recursion

* A function call itself again and again
* Recursion is a method of solving a computational problem where the solution depends on solutions to smaller instances of the sam problem.

* Base Case → where to stop recursion
* What to do → at every level
* Inner call → which will answer





Stack Overflow

↳ * To many calls
* creating large arry at every call.

* Print Fibonacci series

0 1 1 2 3 5 8 13 21 34 — —

Code

```
int printfibonacci( int n) {
  if ( n==0 || n==1) {
      return n;
  }

  return printfibonacci (n-1) +
         printfibonacc (n-2);
}
```

Remove Duplicates in a string

```
public static void removDuplicates (
  String str , int index , StringBuilder
      newstr , boolean map[] ) {

  if ( index == str.length()) {
    Systemout.println ( newstr);
      return;
  }
```

जय माँ सरस्वती

Anuj Bhaiya

Kaden's Algorithm

Used to find Maximum Sum Subarray

ex $a = [-5, 4, 6, -3, 4, -1]$

11

Code :-

```
int maxSumSubArry (int a[])
{
    int maxSum = 0;
    int curSum = 0;

for(int i=0; i<a.length; i++) {
    curSum = curSum + a[i];

if( curSum > maxSum) {
    maxSum = curSum;
}

if( curSum < 0) {
    curSum = 0;
}}

return maxSum;
}
```

Stocks Buys & Sell - 1

By using this teq. we can buy and sell one stock.

Code:-

ex $a = [2, 3, 9, 4, 7, 8, 6]$

$\underbrace{2, 3, 9}_{7}$

```
int maxProfit (int [] a ) {

    int maxProfit = 0;
    int minSofar = a[0];

for (int i=0 ; i< a.length ; i++) {

    minSofar = Math.min (minSofar, a[i] );
    int Profit = a[i] - minSofar;

    maxProfit = Math.max(maxprofit, Profit);
}
    return maxProfit ;

}
```

Stock Buys and Sell -2

By Using this tq. we can ~~by~~ buy and sell multiple stock.

Note:- At one time we will buy and sell only a single stock.

a = [1, 3, 5, 7, 2, 9, 2], ~~1, 8~~]

a = [5, 2, 7, 3, 6, 1, 2, 4]

Code:

```
int maxProfit(int a[]){
  int Profit = 0;
 for (int i=1; i< a.length ; i++){
 if (a[i] > a[i-1]){
  Profit += (a[i]- a[i-1]);
  }
 }
 return profit;
}
```

Search an element in an Infinite sorted array

exp a[] = [1, 3, 5, 10, 26, 59, 70, ---- 98]
0 1 2 3 4 5 --

Here we will find an element from the infinite sorted array in o(nlogn).

Code:

```
SearchInfinite (int arry[], int key){
    int low = 0;
    int high = 1;
    while (arry[high] < key ) {
        low = high;
        high = 2 * high;
    }
    return binarySearch(arry, key, low,
                        high);
}
```

binarySearch code:-

```
if( low > high ) return -1;
int mid = (low+high)/2;
if ( arry[mid] == key) return mid;
if( key > arry[mid]) return binarySear---
if( key < arry[mid]) return binarySear
(arry, key, mid+1, high); (arry, key, low, mid-1);
```

Search an element in a sorted and Rotated Array

(i) 120 130 40 50 90 100 110

(ii) 50 60 90 100 20 30 40

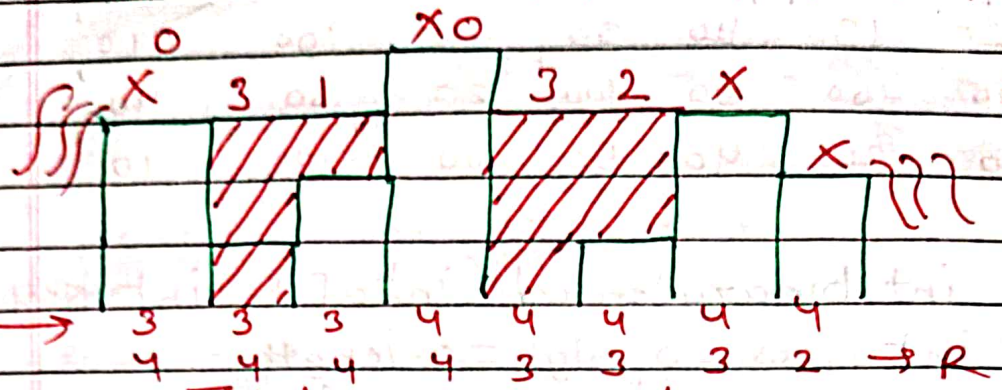
(iii) 20 30 40 50 60 5 10

Code

```
int binarySearch(int a[], int key){
    int low = 0, high = a.length -1;
    while( low <= high) {
    int mid = (low + high) /2;
    if( a[mid] == key) return mid;
    if( a[low] < a[mid]) {
    if( key >= a[low] && key < a[mid]) {
            high = mid-1;
    } else {
            low = mid +1;
    }}
    else {
    if( key > a[mid] && key <= a[high])
            low = mid+1;
    } else {
            high = mid -1;
    }
    return -1;
}
```

Rainwater Trapping Problem

int a[] = [3, 1, 2, 4, 0, 1, 3, 2]



⇒ Total stored water ⇒ 9

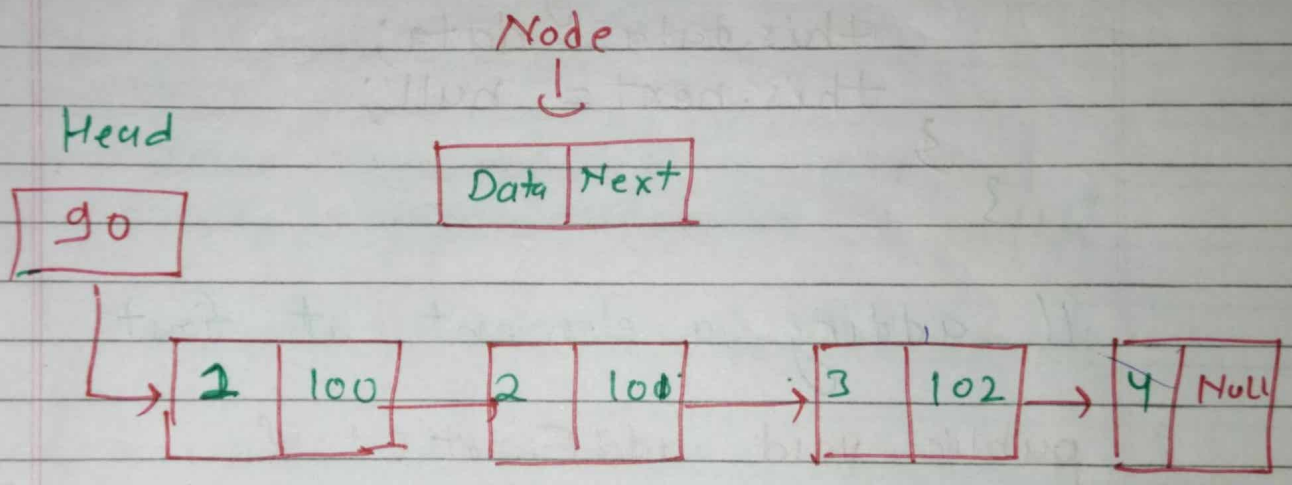
Code:-

```
Int rainwater (int a[]) {
    int n = a.length;
    int left[] = new int [n];
    int right[] = new int [n];
    left [0] = a [0];

for (int i= 1 ; i<n; i++) {
    left[i] = math.max(left[i-1], a[i]);
}

right [n-1] = a [n-1];
for (int i=n-2; i>=0; i--) {
    right (i) = math.max (right (i+1], a[i]);
}

int ans =0;
for (int i=0; i<n ; i++) {
ans += math.min(left[i], right [i]) -
                a[i];
} return ans; }
```

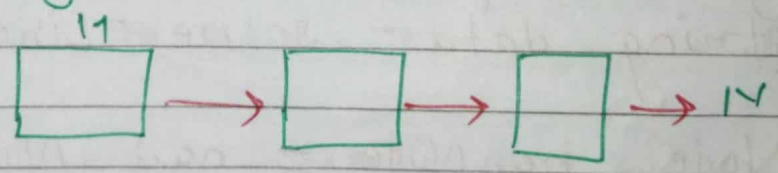
Linked List

Insert → O(1)
search → O(n)

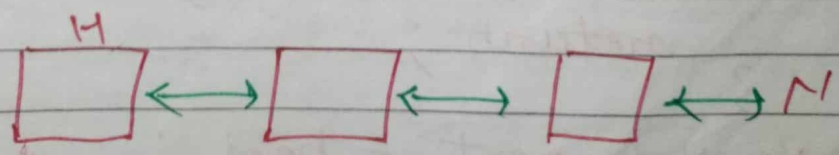
* Variable size
* Non-contiguous memory
* Insert O(1)
* search O(n)



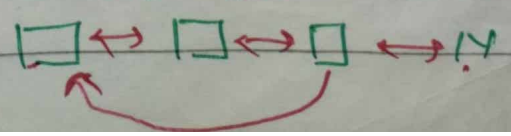
* singular linked list



* Double linked list



* Circular linked list



Code:

```
import java.util.* ;

class main {
Scanner sc = new Scanner (System.in);
Node head; class Node {
        String data;
        Node next;

        Node ( String data) {
          this.data = data;
          this.next = null;
        }
    }

    // adding a element at first

    public void addFirst ( ) {

    System.out.println(" Enter the String ");
    String data = sc.nextLine();

    Node newNode = new Node (data);
    if (head == null) {
          head = newnode;
          return ;
        }
      NewNode.next = head ;
      head = newNode;
    }
```

// ading an element at the last position.

```
public void addLast() {
Node newNode = new Node(data);
System.out.println("Enter a string to add");
String data = sc.nextLine();

If ( head == null) {
    head = newNode;
    return;
}
Node currentNode = head;
while ( currentNode.next != null){

currentNode = currentnode.next;
}
    currentNode.next = newNode;
}
```

// delete an element from the first

```
public void deleteFirst() {

If ( head == null) {
System.out.print ("The list is empty!");
return }

head = head.next;
}
```

```
// delete a node from last

public void deleteLast() {
  if( head ==null) {
 System.out.println(" list is empty! ");
  return;
  }
  Node current = head.next;
  Node previous = head;

if( head.next ==null){
        head = null;
   }

else {
    while (currentNode.next != null) {

  current = current.next;
  previous = previous.next;
  }
 }

      previous.next = null;
}

// Reverse the list using itrative
  method.

public void reverseList() {

if( head ==null || head.next ==null){
```

```
        return;
    }

    Node currentNode = head.next;
    Node previousNode = head;

while ( currentNode.next != null ) {

Node nextNode = currentNode.next;

    currentNode.next = previousNode;

    //update.

        previousNode = currentNode;
        currentNode. = nextNode;
    }
        head.next = null;
        head = previousNode;
    }

    // print list

public void printList() {

if ( head == null ) {
System.out.print(" list is empty ! ");
}
    Node currentNode = head;
while ( currentNode != null) {
        currentNode. = currentNode.next;
```

[redacted]

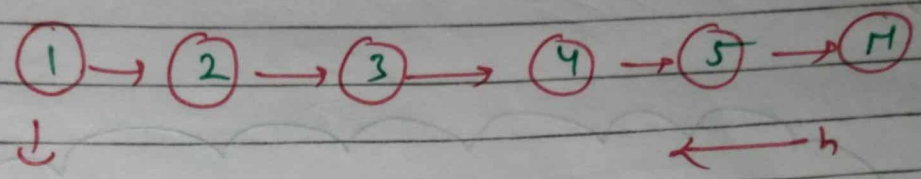
```
print ( currentNode.data + " → " );
  currentNode = currentNode.next;
}
  System.out.println( "Null" );
}
```

main function {

code

}

Find the nth node from last



head

(size-n+1)

if n=2, size=5
then
$\longrightarrow$ 5-2+1=4

```
fu( LinkedList head, int key, int size)
{
  Linkdlist prev = head ;
  int destination = size-key

if( head == null) {
        return null;
  }

if ( b. size == n) {
        return head.next ;
}
      int i=0;
while ( i< destination ) {
```

```
        prev = prev.next;
        i++;
    }
    prev.next = prev.next.next;
    return head;
}
}
```

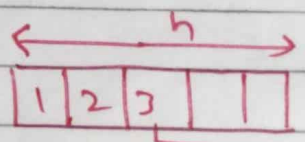
2, 3, 4, 5

Stack

* It is a linear data structure where elements are add and removed from the same end (Top) in a LIFO (Last in First out)
  - ↳ java.util.Stack

* Implementation ↴

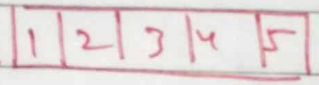
Arrays



↳ Fixed ↳ Top

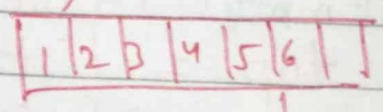
ArrayList

↳ Variable type



If size full

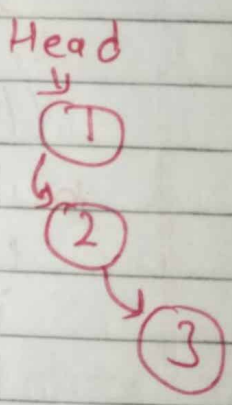
↳ copy



↓ Top

LinkedList

↳ Variable typ



most efficient

* Operation
  - * push → add
  - * pop → remove
  - → peek → return top

stack using arrayList

```
static class Stack {

static ArrayList<Integer> list = new
            ArrayList<>();
public static boolean isEmpty() {
        return list.size() == 0;
    }

    // push
  public static void push (int data) {
        list.add(data);
    }
  }

  // pop
  public static int pop() {
    if(isEmpty()) { return -1; }
    int top = list.get(list.size()-1);
    list.remove(list.size()-1);
    return top;
  }

  // peek

public static int peek() {
 if(isEmpty()) { return -1; }
return list.get(list.size()-1);
}
```

Stack using linked list

in main class

```
public static class Node {

    int data;
    Node next;

    Node ( int data ) {
        this.data = data;
        this.next = null;
    }
}

public static boolean isEmpty() {

    return head == null;
}

public static void push( int data) {

    Node newNode = new Node(data);

    If ( isEmpty()) { ~~return -1;~~ }
    head = newNode; return; }
    newNode.next = head;
    head = newNode;
}
```

```
public static ~~void~~ int pop(){
    if (isEmpty){
        return -1;
    }

    int top = head.data;
    head.next = head;
    return top;
}

public static int peek(){
    if (isEmpty){
        return -1;
    }

    return head.data;
}
```

Stack using collection framwark

Steps :- import java.util.*;

creat object of stack class

Stack<Integer> s = new Stack<>();

available functions :-

push()
pop()
peek()
isEmpty()

Code

```
public static void main (String args[])
{
  stack<Integer> s = new Stack<>();

    s.push(1);
    s.push(2);

  while ( ! s.isEmpty()) {

    System.out.println(s.peek());
      s.pop();
  }
}
```

Question's of stack

(1) Push an element at the bottom of the stack.

Code

```
public class check {

public static void addAtBottom (
        Stack<Integer> s, int data) {

 If( s.isEmpty() ) {
        s.push(data);
        return;
   }

  int top = s.pop();
  addAtBottom(s, data);
  s.push(data);
 }

public static void main (String args[])
 {
  stack<Integer> s = new stack<>();
   s.push(1);
   s.push(2);
   addAtBottom (s, 5);
   while (! s.isEmpty ()) {
   System.out.println( s.pop());
    }}}
```

2. Reverse a string using a stack.

Code

```
public static String reverseString (String s)
{
  Stack< charactor> s1 = new stack<>();
  int idx = 0;
  while ( idx < s.length()) {
     s1.push( s.charAt (idx));
     idx ;
   }

   StringBuilder str = new StringBuilder("")

  while( ! s1.isEmpty()) {

   char curr = s1.pop();
   str.append( curr );
  }

   return str.toString();

}
```

Now call this function with a string to get the reverse of that string.

3. Reverse a stack

```
public static string addAtBottom (
 stack<Integer> s, int data) {
     // written in 2nd Q.
 }

public static void reverseStack (
            Stack<Integer> s ) {

 if(s.isEmpty()) {
            return ;
    }

    int top = s.pop();
    reverseStack(s);
    pushAtBottom ( s, top);

 }
```

4. Stock Span Problem

↳ max. numbers of ~~cosy~~ consecutive days for which price ≤ today's price

function

```
Public static void spanStack( int stock[],
        int span[] ) {

 Stack<Integer> s = new stack<>();
```

```
span[0] = 1;
~~s.pop(0);~~ s.push(0);

for(int i=1; i<stock.length; i++){

 int current = stock[i];
 while(!s.isEmpty() && current >
                    stock[s.peek()]){
    s.pop();
  }
  if (s.isEmpty){
      span[i] = i+1;
  }
  else{
     int preHeigh = s.peek();
      span[i] = i-preHeigh;
    }
    s.push(i);
  }
}

public static void main (string args[])
{
int stock[] = {100, 80, 60, 70, 90, 100}
int span[] = new int[stock.length];

spanStack (stock, span);
for(int i=0; i<span.length; i++){
 System.out.print(span[i]+" ");
  }
  }}
```

Important

4. Next Greater Element

The next greater element of some element x in an array is the first greater element that is to the right of x in the same array.

arr = [6, 8, 0, 1, 3]

next Greater = [8, -1, 1, 3, -1]

→ We can solve this problem using nested loop but the time complexity will be $O(n^2)$ that is not good.

→ Algo for next method which ~~are~~ is using stack

(1) while (s.empty && stack[top] <= arr[i])

s.pop() ;

(2) If stack.empty

then nextGreater = -1

else

nextGreater = s.peek()

(3) s.push(i)

Code

```
public static void nextGreater( int nums[],
    int nextGreater[] ) {

  stack<Integer> s = new Stack<>();
        int current
  while ( !s.isEmpty  &&

  for( int i= num.length-1 ; i>=0 ; i--){

    int current = num[i] ;
  while ( !s.isEmpty && current >= num[
                                s.peck]) {
      s.pop();
    }

  if ( s.isEmpty ) {

      nextGreater[i] = -1 ;
    }

  else {
    nextGreater[i] = s.peck() ;
    }
    s.push(i) ;
  }
```

5. Valid Parentheses

Given a string s containing just the charactors '(', ')', '{', '}', '[' and ']', determine if the input string is valid.

An input string is valid If:

1. Open brackets must be closed by the same type of brackets
2. Open brackets must be closed in the correct order.
3. Every close bracket has a corresponding open bracket of the same type.

example ({[]}()) ✓
({(})) ✗

Code

```
public static boolean validParantheses (
                string s ) {

 Stack<Character> s1 = new Stack<>();

for( int i=0; i<s.length(); i++){

    char ch = s.charAt(i);
If( ch == '(' || ch == '{' || ch == '[')
 {          s1.push(ch); }
```

```
else {

    if (s1.isEmpty()){
            return false;
    }

    if( ( ch == ')' && s1.peek() == '(' ) ||
( ch == '}' && s1.peek() == '{' ) || ( ch==']'
    && s1.peek() == '[' ) ) {
        s1.pop();
    }

    else {
        return false;
    }
  }
}
  if( s1.isEmpty()) {
            return true;
    }
  else {
        return false;
    }
}
```

6. Duplicate Parentheses

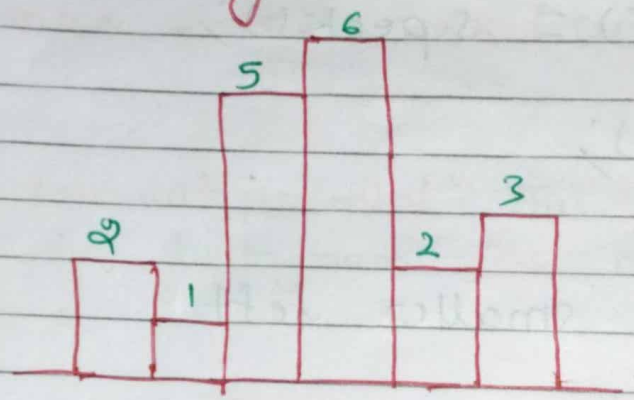
Given a balanced expression, find if it contains duplicate parentheses or not. A set of parentheses are duplicate if the same subexpression is surrounded by multiple parentheses.

```
public static boolean DoublePa(String str){

 Stack<Character> s = new stack<>();

for( int i=0; i< str.length(); i++ ) {

char ~~int~~ ch = str.charAt(i);
 If ( ch == '(' ){
        int count =0;
        while ( s.pop() != '(' ){
                Count ++;
        }
       If ( count < 1 ){
            return true;
       }
 }
 else{
        s.push(ch);
    } }
 return false;
}
```

* Max Area of Histogram (Leetcode, Hard)

Given an array of integers heights representing the histogram's bar height where the width of each bar is 1, return the area of the largest rectangle in the histogram.

heights = [2, 1, 5, 6, 2, 3]



Area 2 6 10 6 8 3

```
public static void maxArea( int array[] ){

    int nsr[] = new int[array.length];
    int nsl[] = new int[array.length];

    int maxArea = 0;

    Stack<Integer> s = new Stack<>();

    // Next smaller right
    for(int i = arr.length-1; i>=0; i--){
        int curr = arr[i];
```

```
while ( !s.isEmpty() && curr <= arr[s.
                                peek()]){
        s.pop();
    }

if (s.isEmpty()){
        nsr[i] = arr.length;
    }
else {
        nsr[i] = s.peek();
    }
    s.push(i);
}

    // next smaller left

    s = new Stack<>();

for( int i=0; i<arr.length; i++){

    int curr = arr[i];
while( !s.isEmpty() && curr <=
                arr[s.peek()]){
        s.pop();
    }
if ( s.isEmpty()) {
        nsl[i] = -1;
    }
else {
        nsl[i] = s.peek();
    }
```

```
    s.push(i);
}

    // Calculating area
for( int i=0; i<arr.length; i++){

    int height = arr[i];
    int width = nsr[i] - nsl[i] - 1;
    int currArea = height * width;
    max = Math.max (currArea , max);
}

System.out.println(" This the maximum area
    of histogram = " + max);
}
```