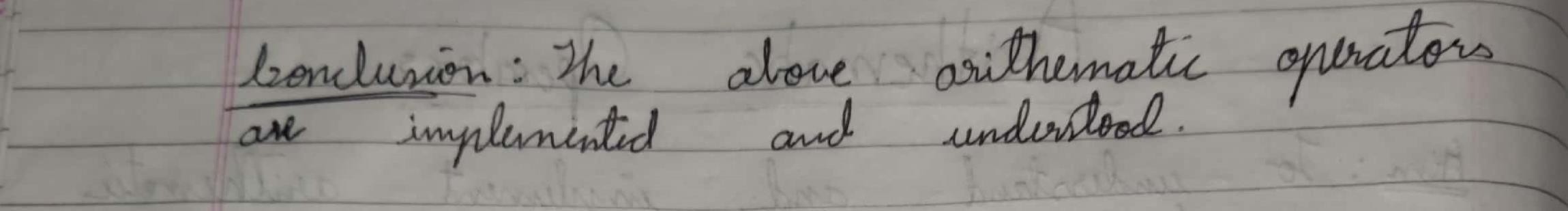


Scanned with



## Codes

public static void main(String args[])

float a, b, c, d ;

import java.util.Scanner;

public class Arithmetic

{

{

Scanner sc = new Scanner(System.in);

System.out.print("Enter First Number: "); a = sc.nextInt();

System.out.print("Enter Second Number: "); b = sc.nextInt();

System.out.println("A + B = " + (a+b) );

System.out.println("A - B = " + (a-b) ); System.out.println("A \* B = " + (a\*b) ); System.out.println("A / B = " + (a/b) ); System.out.println("A % B = " + (a%b) );

a++;

--b;

System.out.println("A++ = " + a );

System.out.println("--B = " + b );

}

}

/\*

Output

C:\JavaProg>java Arithmetic Enter First Number: 24 Enter Second Number: 55

A + B = 79.0 A - B = -31.0

A \* B = 1320.0

A / B = 0.43636364

A % B = 24.0 A++ = 25.0 --B = 54.0

\*/

import java.util.Scanner ; public class Bitwise

{ public static void main(String args[])

{ int a, b, c;

System.out.println(" Please Enter two integer Value: ");

Scanner sc = new Scanner(System.in); a = sc.nextInt(); b = sc.nextInt();

System.out.println("a&b = " + (a & b));

System.out.println("a|b = " + (a | b)); System.out.println("a^b = " + (a ^ b));

System.out.println("~a = " + ~a);

c = a >> 1;

System.out.println("a >> 1 = " + c);

c = b >> 1;

System.out.println("b >> 1 = " + c );

}

}

/\* Output

C:\JavaProg>java Bitwise

Please Enter two integer Value: 24 88 a&b = 24

a|b = 88

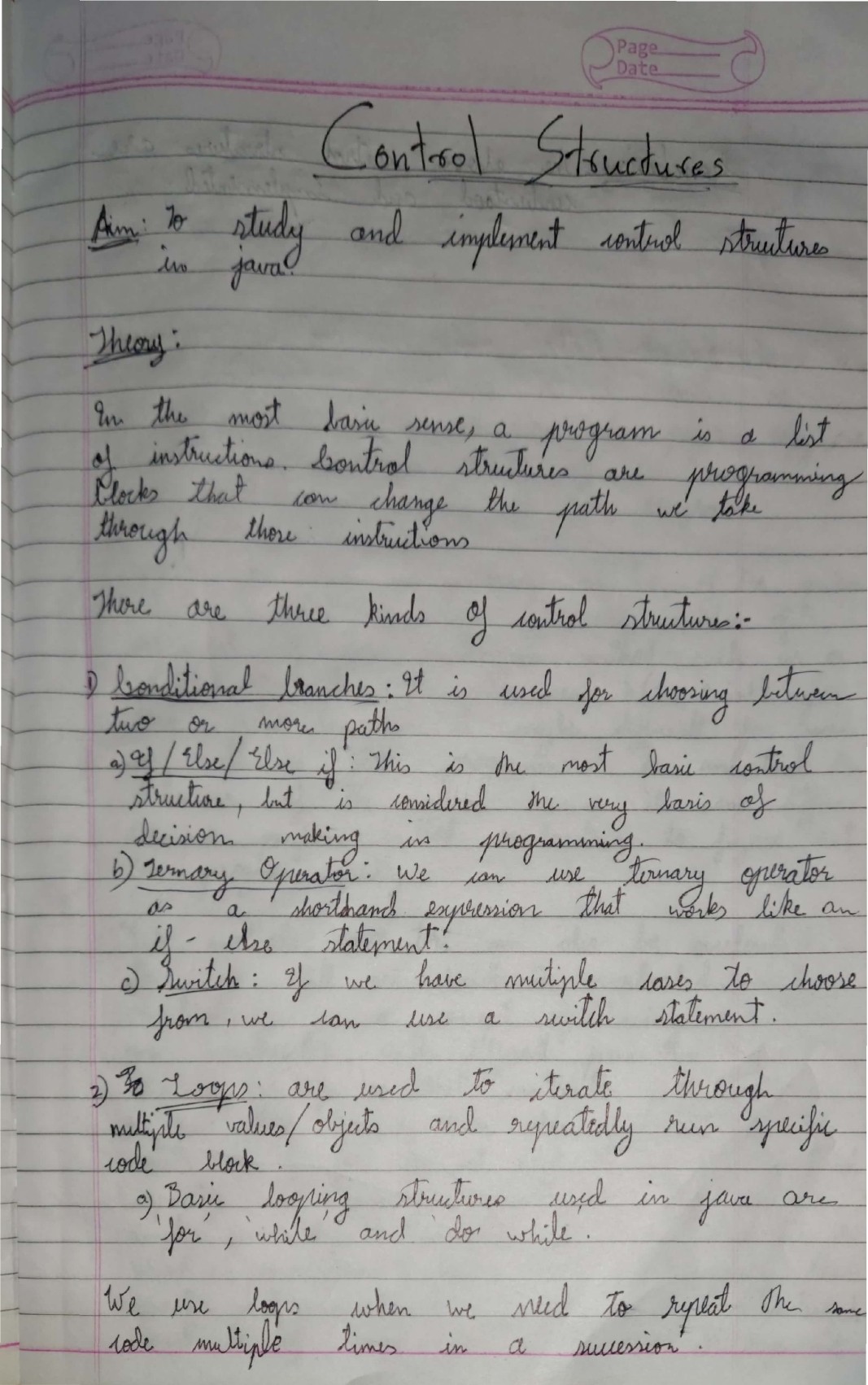
a^b = 64

~a = -25

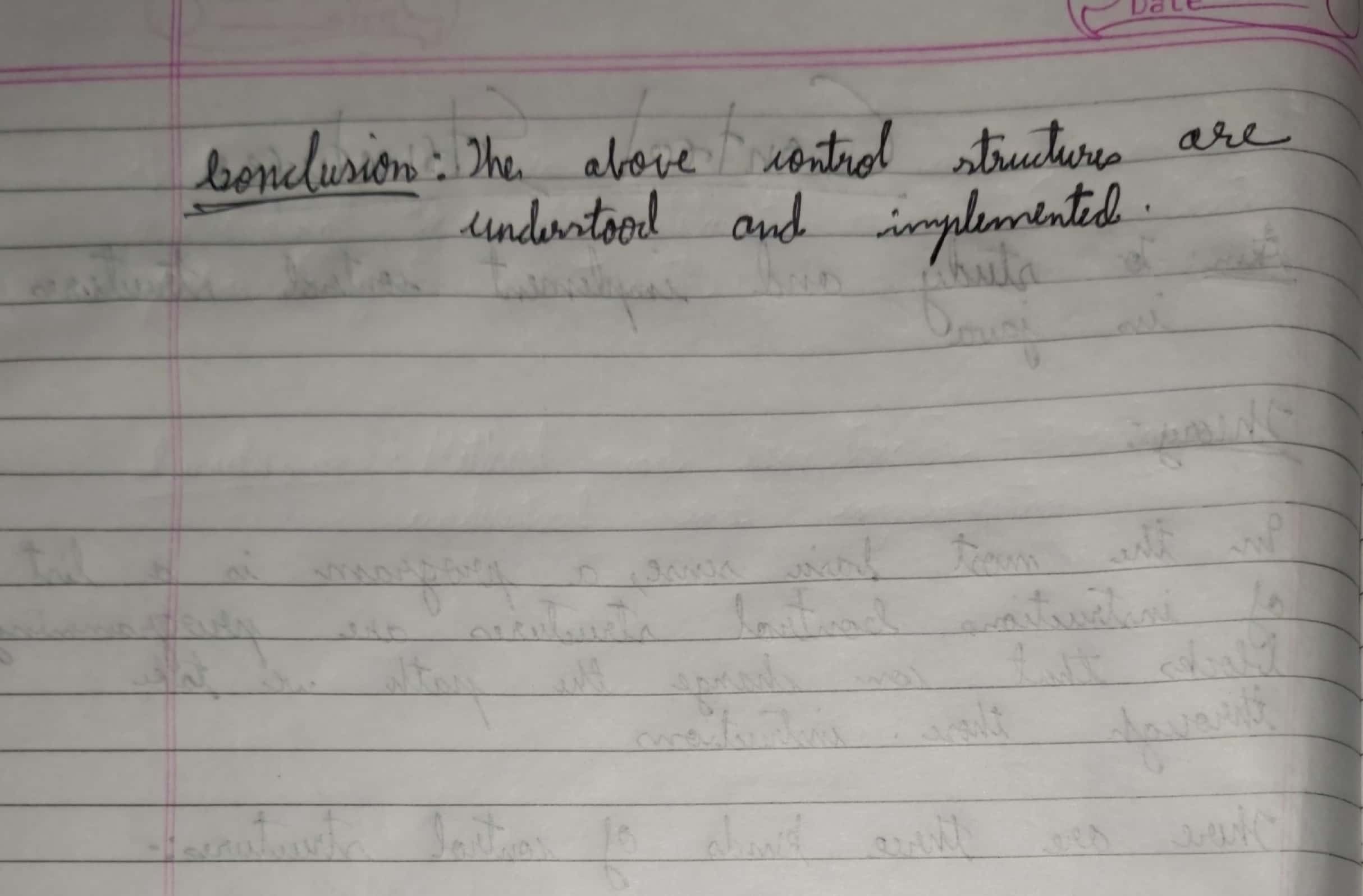
a >> 1 = 12

b >> 1 = 44

\*/



ScannedwithCamScanner





# Code

class DisplayArgs

{

public static void main(String[] args)

{

int length=args.length; if(args.length>0)

{

for(int i=0;i>length;i++)

{

System.out.println(args[i]);

}

}

else

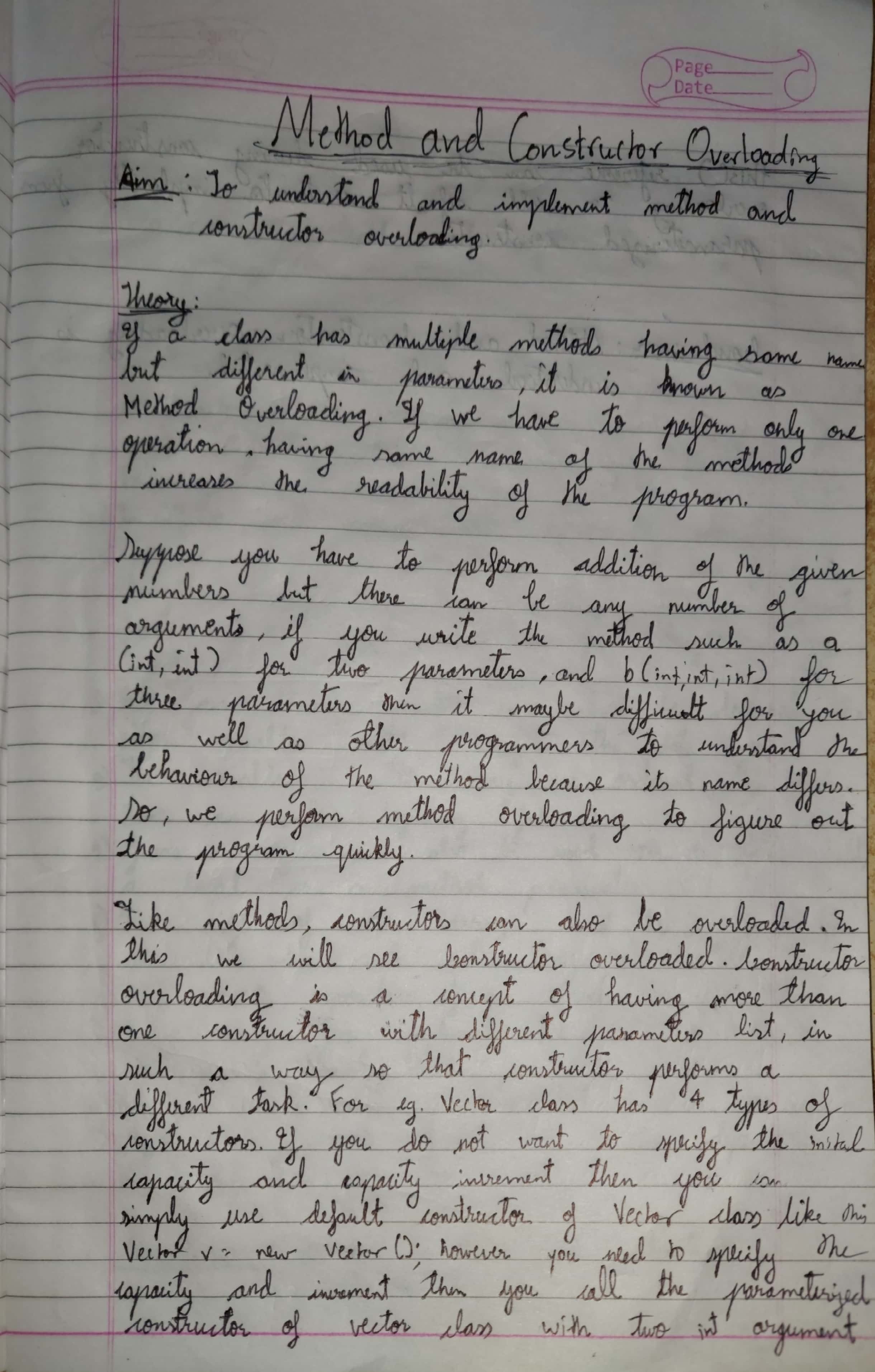
{

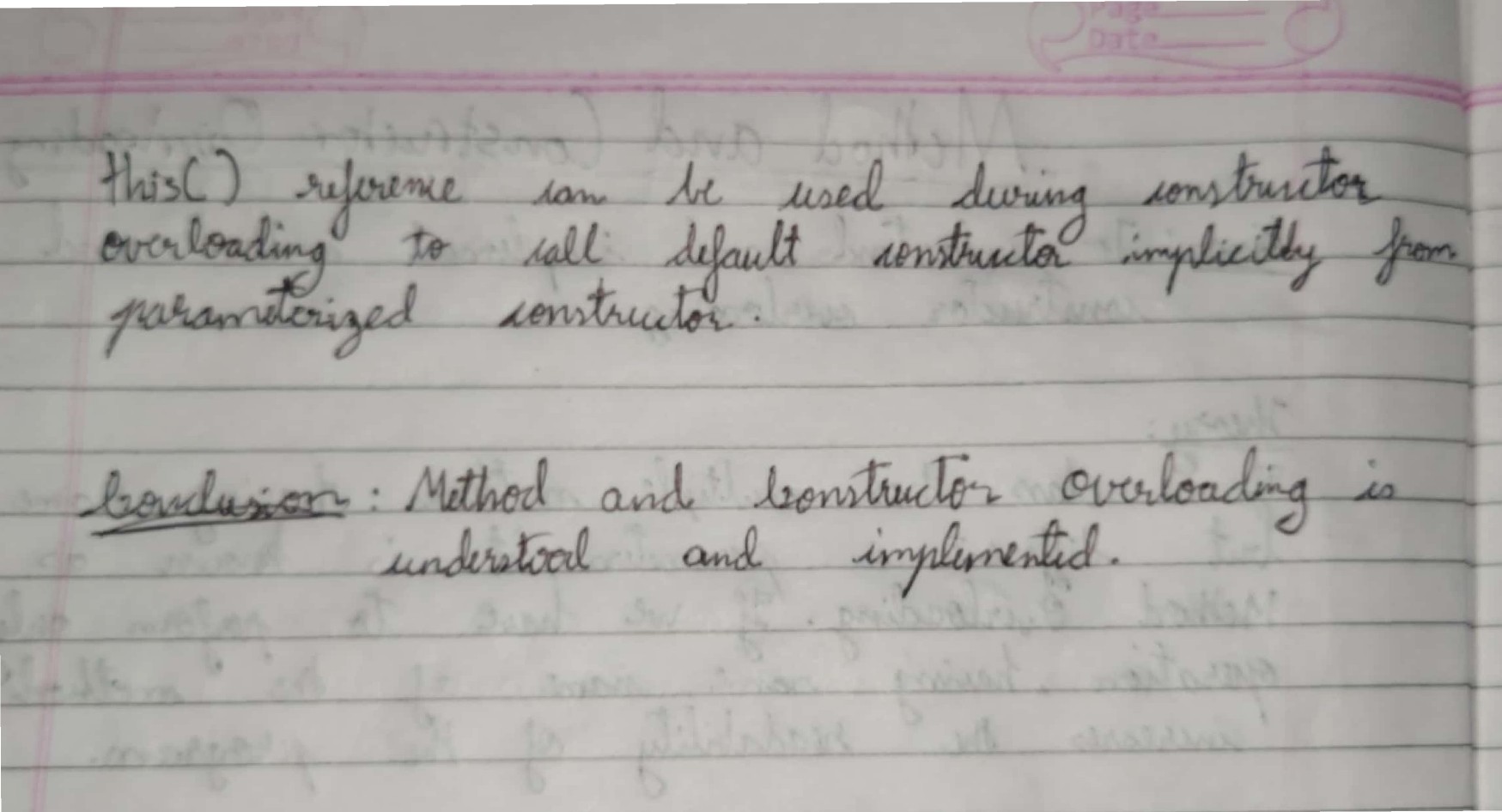
System.out.println("no arguments enter from commandline :");

}

}

}



Scanned with CamScanner

# Codes

import java.util.\* ;

class Shapes

{

static double ans ;

static double Area(int a, int b)

{

return(a \* b);

}

static double Area(float b)

{

ans = (22/7)\*b\*b ;

return(ans);

}

static double Area(float a,double b)

{

ans = 0.5 \* a \* b ;

return(ans);

}

static double Area(int r , float b)

{

ans = (22/7)\*r\*r\*b ;

return(ans);

}

static double Volume(int a, int b,int c)

{

return(a \* b \* c);

}

static double Volume(float r,float h)

{

ans = (22/7)\*r\*r \*h ; return ans;

}

public static void main(String args[])

{

int i, m,n,o,p ; float a,b,c ; double ans ;

System.out.println("1)Square"); System.out.println("2)Circle"); System.out.println("3)Rectangle"); System.out.println("4)Parallelogram"); System.out.println("5)Rombus"); System.out.println("6)Triangle"); System.out.println("7)Semi-circe"); System.out.println(); System.out.println("8)Cubiod"); System.out.println("9)Cube"); System.out.println("10)Cylinder");

Scanner sc = new Scanner(System.in); System.out.println("Enter the option");

i = sc.nextInt();

switch(i)

{

case 1 :

System.out.println("Enter the length of the side"); m = sc.nextInt();

ans = Area(m,m) ; System.out.println(ans);

break ;

case 2 :

System.out.println("Enter the Radius"); a = sc.nextFloat();

ans = Area(a) ;

System.out.println(ans);

break ;

case 3 :

System.out.println("Enter the length"); m = sc.nextInt();

System.out.println("Enter the Breath");

n = sc.nextInt();

ans = Area(m,n) ;

System.out.println(ans);

break ;

case 4 :

System.out.println("Enter the side 1"); n = sc.nextInt();

System.out.println("Enter the side 2"); m = sc.nextInt();

ans = Area(m,n) ;

System.out.println(ans);

break ;

case 5 :

System.out.println("Enter the length side"); n = sc.nextInt();

ans = Area(n,n) ;

System.out.println(ans);

break ;

case 6 :

System.out.println("Enter the base"); a = sc.nextFloat();

System.out.println("Enter the height"); b = sc.nextFloat();

ans = Area(a,b) ;

System.out.println(ans);

break ;

case 7 :

System.out.println("Enter the radius"); m = sc.nextInt();

ans = Area(m,0.5) ;

System.out.println(ans);

break ;

case 8 :

System.out.println("Enter the length"); m = sc.nextInt();

System.out.println("Enter the length"); n = sc.nextInt();

System.out.println("Enter the length"); o = sc.nextInt();

ans = Volume(m,n,o) ;

System.out.println(ans);

break ;

case 9 :

System.out.println("Enter the length"); m = sc.nextInt();

ans = Volume(m,m,m) ;

System.out.println(ans);

break ;

case 10 :

System.out.println("Enter the radius"); a = sc.nextFloat();

System.out.println("Enter the radius"); b = sc.nextFloat();

ans = Area(a,b) ;

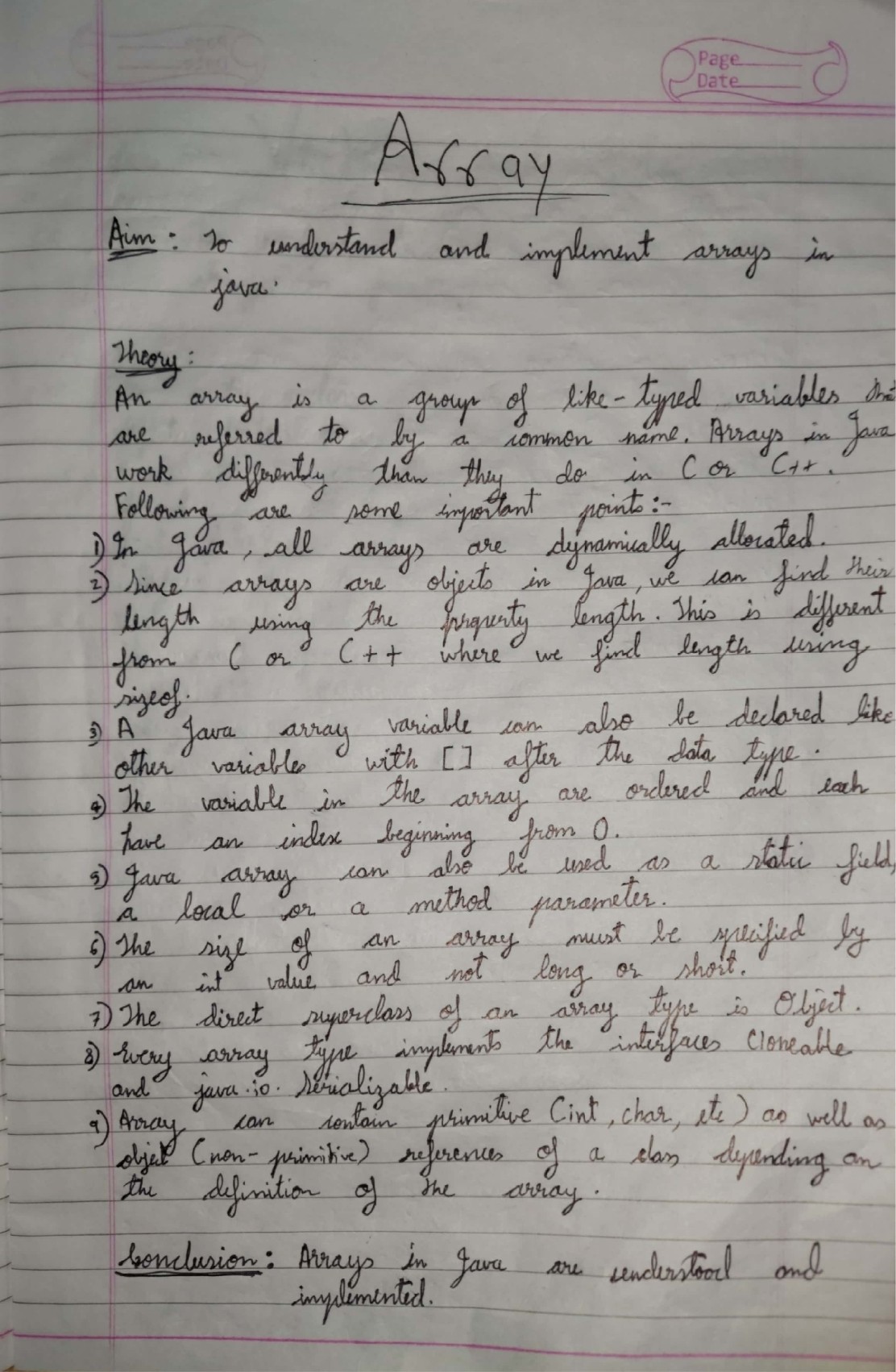
System.out.println(ans);

break ;

}

}

}



Scanned with CamScanner

# Codes

import java.util.Scanner ;

public class Arr\_operation

{

public static void main(String args[])

{

int i,j,n,m ;

System.out.println("Enter the number elements in rows and columns"); Scanner sc = new Scanner(System.in);

m = sc.nextInt(); n = sc.nextInt();

int a[][] = new int [m][n] ;

int b[][] = new int [m][n] ;

int c[][] = new int [m][n] ;

int d[][] = new int [m][n] ;

System.out.println("Enter the elements of first array");

for(i=0; i < m; i++)

{

for(j=0; j < n; j++)

{

a[i][j] = sc.nextInt();

}

}

System.out.println("Enter the elements of second array");

for(i=0; i < m; i++)

{

for(j=0; j < n; j++)

{

b[i][j] = sc.nextInt();

}

}

for(i=0; i < m; i++)

{

for(j=0; j < n; j++)

{

c[i][j] = a[i][j] + b[i][j];

}

}

for(i=0; i < m; i++)

{

for(j=0; j < n; j++)

{

d[i][j] = a[i][j] - b[i][j];

}

}

System.out.println("Added array"); for(i=0; i < m; i++)

{

for(j=0; j < n; j++)

{

System.out.print(c[i][j]+" ");

}

System.out.println();

}

System.out.println("Subtracted array"); for(i=0; i < m; i++)

{

for(j=0; j < n; j++)

{

System.out.print(d[i][j]+" ");

}

System.out.println();

}

}

}

/\* OUTPUT

C:\JavaProg>java Arr\_operation

Enter the number elements in rows and columns 2

2

Enter the elements of first array

-1

-2

-3

-4

Enter the elements of second array 5

6

7

8

Added array 4 4

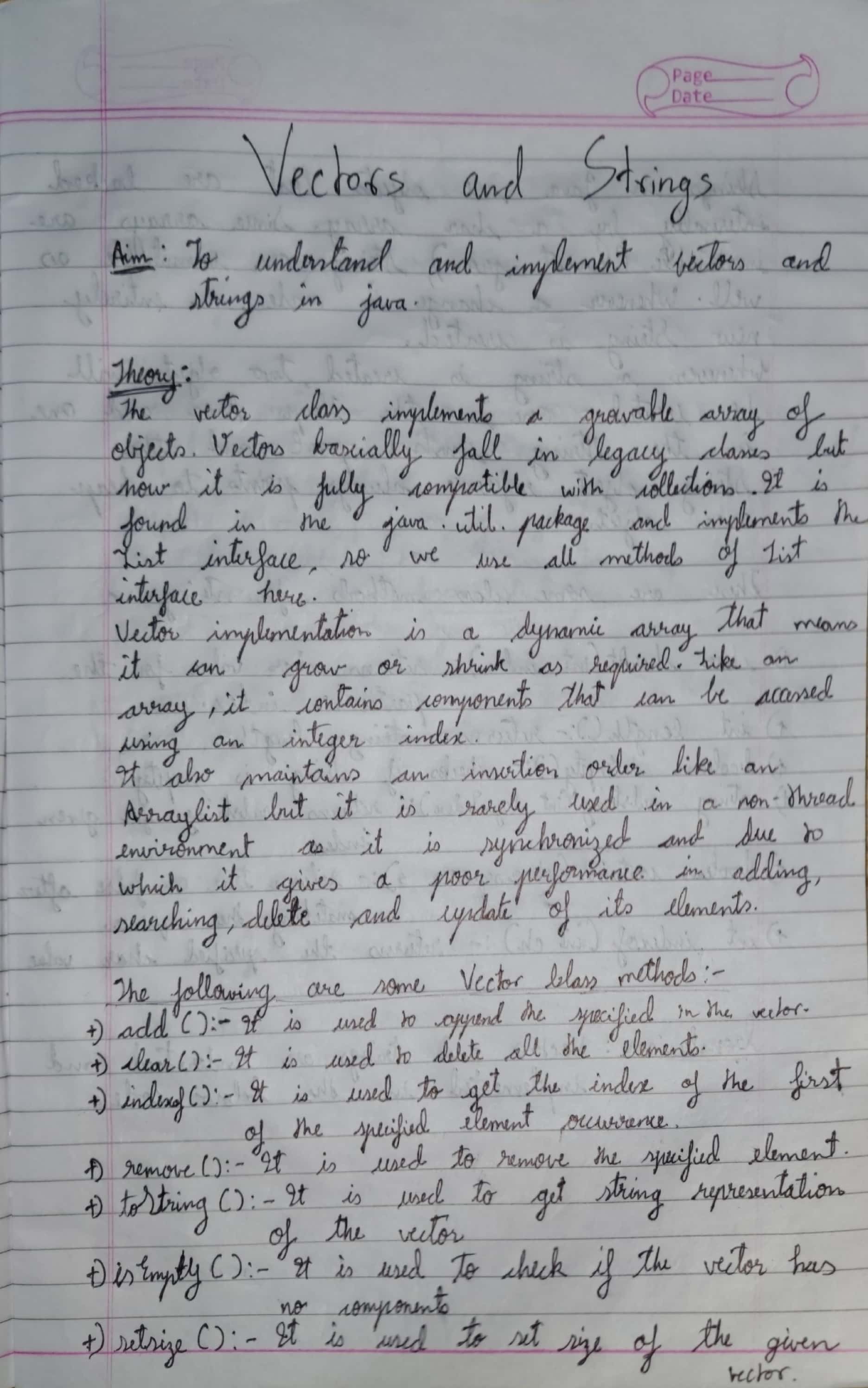
4 4

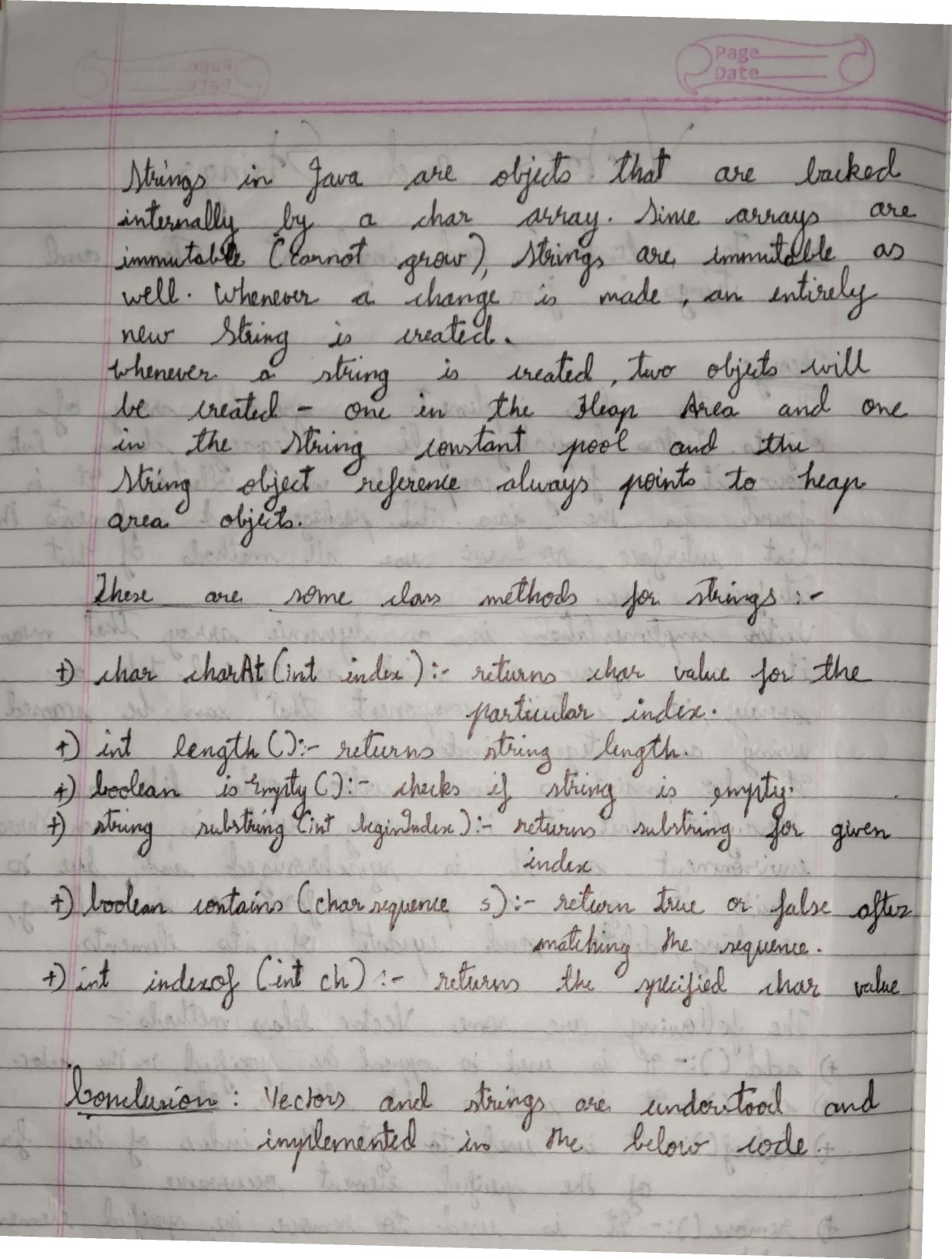
Subtracted array

-6 -8

-10 -12

\*/





Scanned withCamScanner

Example 1

# Codes

import java.util.\*;

public class VectorExample1

{

public static void main(String args[])

{

Vector<String> vec = new Vector<String>(4);

vec.add("Tiger");

vec.add("Lion");

vec.add("Dog"); vec.add("Elephant");

System.out.println("Size is: "+vec.size()); System.out.println("Default capacity is: "+vec.capacity());

System.out.println("Vector element is: "+vec); vec.addElement("Rat");

vec.addElement("Cat"); vec.addElement("Deer");

System.out.println("Size after addition: "+vec.size()); System.out.println("Capacity after addition is: "+vec.capacity());

System.out.println("Elements are: "+vec);

if(vec.contains("Tiger"))

{

System.out.println("Tiger is present at the index " +vec.indexOf("Tiger"));

}

else

{

System.out.println("Tiger is not present in the list.");

}

System.out.println("The first animal of the vector is = "+vec.firstElement());

System.out.println("The last animal of the vector is = "+vec.lastElement());

}

}

Example 2

import java.util.\*;

public class VectorExample2

{

public static void main(String args[])

{

Vector<Integer> in = new Vector<>(); in.add(100);

in.add(200);

in.add(300);

in.add(200);

in.add(400);

in.add(500);

in.add(600);

in.add(700);

System.out.println("Values in vector: " +in);

System.out.println("Remove first occourence of element 200: "+in.remove((Integer)200)); System.out.println("Values in vector: " +in);

System.out.println("Remove element at index 4: " +in.remove(4)); System.out.println("New Value list in vector: " +in);

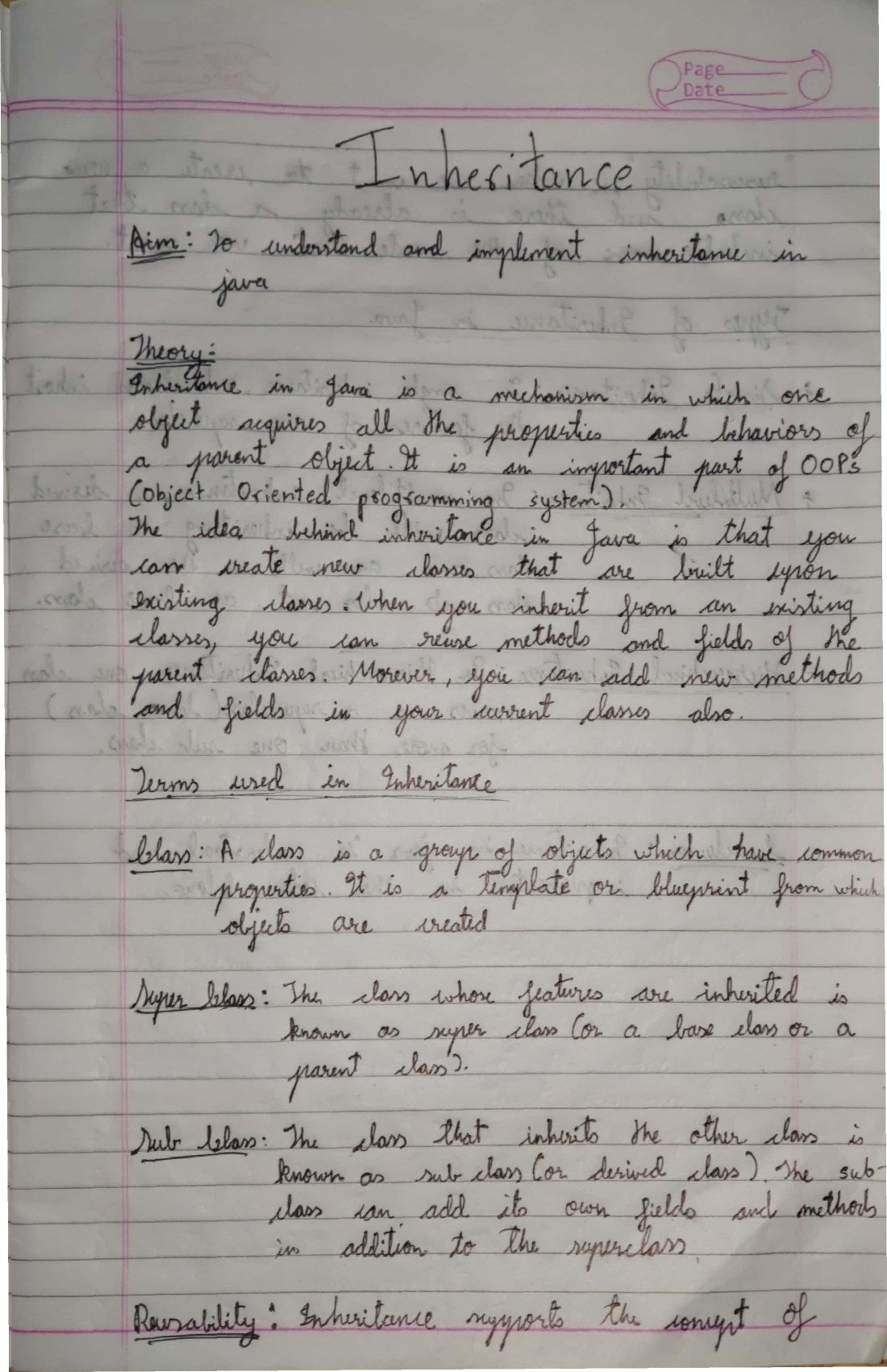
in.removeElementAt(5);

System.out.println("Vector element after removal: " +in); System.out.println("Hash code of this vector = "+in.hashCode());

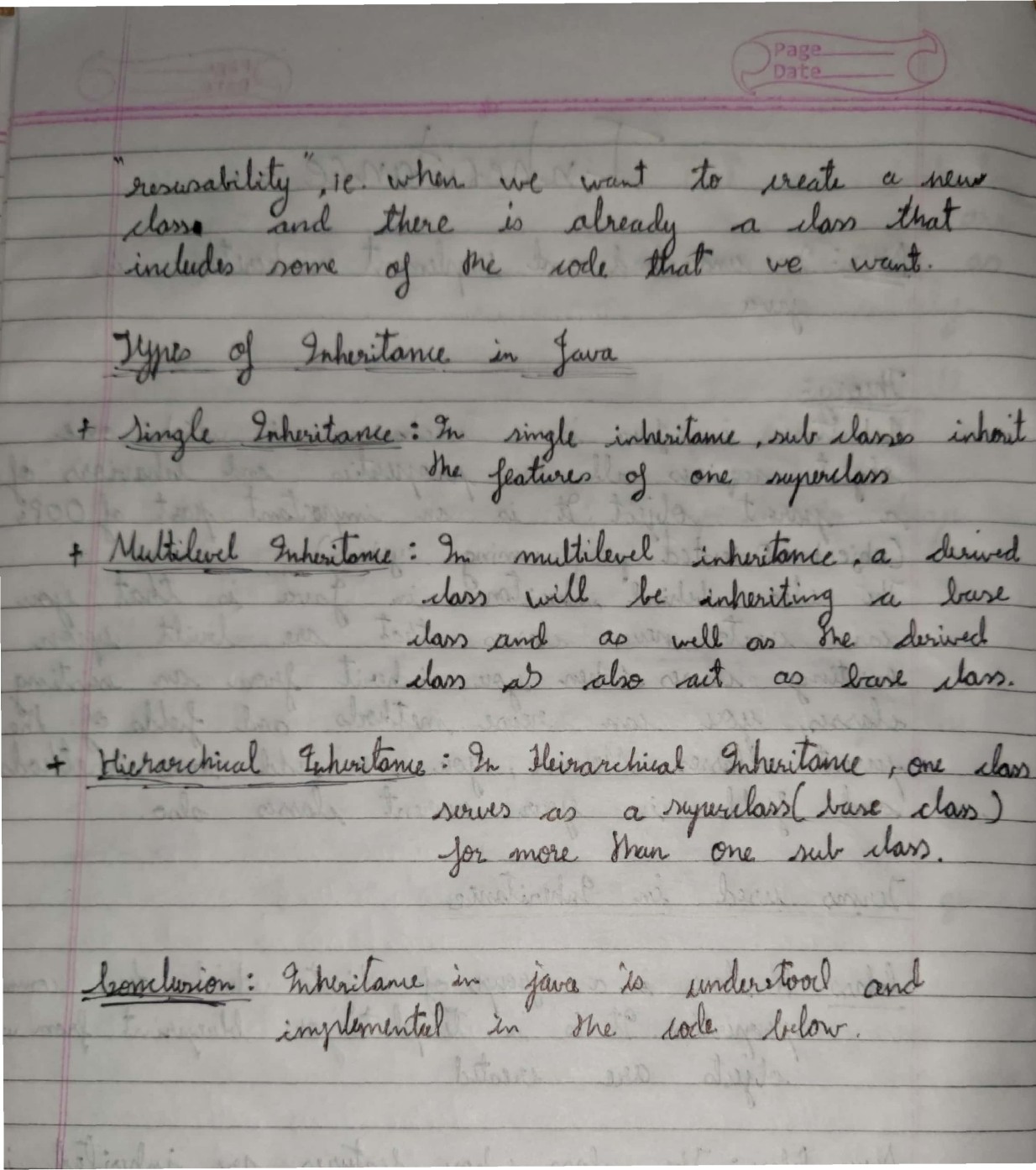
System.out.println("Element at index 1 is = "+in.get(1));

}

}



Scanned with CamScanner



Scanned with CamScanner

# Code

class Animal

{

void eat()

{

System.out.println("eating");

}

}

class Dog extends Animal

{

void bark()

{

System.out.println("barking");

}

}

class BabyDog extends Dog

{

void weep()

{

System.out.println("weeping");

}

}

class TestInheritance2

{

public static void main(String args[])

{

BabyDog d = new BabyDog();

d.weep();

d.bark();

d.eat();

}

}

## Java Applet

Applet is a special type of program that is embedded in the webpage to generate the dynamic content. It runs inside the browser and works at client side.

##### Advantage of Applet

There are many advantages of applet. They are as follows:

* It works at client side so less response time.

o Secured

* It can be executed by browsers running under many plateforms, including Linux, Windows, Mac Os etc.

##### Drawback of Applet

* Plugin is required at client browser to execute applet.

##### Lifecycle of Java Applet

1. Applet is initialized.
2. Applet is started.
3. Applet is painted.
4. Applet is stopped.
5. Applet is destroyed.

##### Lifecycle methods for Applet:

The java.applet.Applet class 4 life cycle methods and java.awt.Component class provides 1 life cycle methods for an applet.

##### java.applet.Applet class

For creating any applet java.applet.Applet class must be inherited. It provides 4 life cycle methods of applet.

1. **public void init():** is used to initialized the Applet. It is invoked only once.
2. **public void start():** is invoked after the init() method or browser is maximized. It is used to start the Applet.
3. **public void stop():** is used to stop the Applet. It is invoked when Applet is stop or browser is minimized.
4. **public void destroy():** is used to destroy the Applet. It is invoked only once.

##### java.awt.Component class

The Component class provides 1 life cycle method of applet.

1. **public void paint(Graphics g):** is used to paint the Applet. It provides Graphics class object that can be used for drawing oval, rectangle, arc etc.

##### Who is responsible to manage the life cycle of an applet?

Java Plug-in software.

##### How to run an Applet?

There are two ways to run an applet

1. By html file.
2. By appletViewer tool (for testing purpose).

##### Hierarchy of Applet

Lifecycle of Java Applet

* 1. Applet is initialized.
  2. Applet is started.
  3. Applet is painted.
  4. Applet is stopped.
  5. Applet is destroyed.

##### Simple example of Applet by html file:

To execute the applet by html file, create an applet and compile it.

After that create an html file and place the applet code in html file. Now click the html file.

## Codes

import java.awt.\*; import java.applet.\*; import java.awt.event.\*; import javax.swing.\*;

/\*

<applet code="Applet2" width="500" height="200">

</applet>

\*/

public class Applet2 extends JApplet

{

public void init()

{

add(new B()); //Adding a JPanel to this Swing applet

}

}

class B extends JPanel implements ActionListener

{

JLabel jb;

JButton Box1, box2, box3, box4;

String str;

B()

{

jb= new JLabel("Welcome, please click on any button to unbox some interesting knowledge -");

Box1 = new JButton("Box1"); box2 = new JButton("Box2"); box3 = new JButton("Box3"); box4 = new JButton("Box4");

str ="";

setLayout(new FlowLayout());

add(jb); add(Box1); add(box2); add(box3); add(box4);

Box1.addActionListener(this); box2.addActionListener(this); box3.addActionListener(this); box4.addActionListener(this);

}

public void actionPerformed(ActionEvent ae)

{

if(ae.getActionCommand().equals("Box1"))

{

str="Amazon is the largest tropical rain forest, covering 40% of the South Ame rica Continent.";

repaint();

}

if(ae.getActionCommand().equals("Box2"))

{

str="The Mariana Trench is the deepest point in Earth's ocean, with depth of o ver 10,994 metres.";

repaint();

}

if(ae.getActionCommand().equals("Box3"))

{

str="The coldest temperature ever recorded was - 128.6F in Antarctica, on July 21, 1983."; repaint();

}

if(ae.getActionCommand().equals("Box4"))

{

str="The oldest person to climb Mt. Everent was Japanese Miura Yiuchiro, at th e age of 80.";

repaint();

}

}

public void paintComponent(Graphics g)

{

super.paintComponent(g); g.drawString(str, 2, 170);

}

public static void main(String args[])

{

new Applet2();

}

}

#### **Exception Handling**

An exception is an unwanted or unexpected event, which occurs during the execution of a program i.e at run time, that disrupts the normal flow of the program’s instructions. The Exception Handling in Java is one of the powerful mechanisms to handle the runtime errors so that normal flow of the application can be maintained. The core advantage of exception handling is **to maintain the normal flow of the application**. An exception normally disrupts the normal flow of the application that is why we use exception handling.

In this page, we will learn about Java exceptions, its type and the difference between checked and unchecked exceptions.

Error: An Error indicates serious problem that a reasonable application should not try to catch.

Exception: Exception indicates conditions that a reasonable application might try to catch.

All exception and errors types are sub classes of

class Throwable, which is base class of hierarchy. One branch is headed by Exception. This class is used for exceptional conditions that user programs should catch.

NullPointerException is an example of such an exception. Another branch , Error are used by the Java run-time system[(JVM)](https://www.geeksforgeeks.org/jvm-works-jvm-architecture/) to indicate errors having to do with the run-time environment itself(JRE). StackOverflowError is an example of such an error.

Default Exception Handling: Whenever inside a method, if an exception has occurred, the method creates an Object known as Exception Object and hands it off to the run-time system (JVM). The exception object contains name and description of the exception, and current state of the program where exception has occurred. Creating the Exception Object and handling it to the run-time system is called throwing an Exception. There might be the list of the methods that had been called to get to the method where exception was occurred. This ordered list of the methods is called Call Stack. Now the following procedure will happen.

* The run-time system searches the call stack to find the method that contains block of code that can handle the

occurred exception. The block of the code is called Exception handler.

* The run-time system starts searching from the method in which exception occurred, proceeds through call stack in the reverse order in which methods were called.
* If it finds appropriate handler then it passes the occurred exception to it. Appropriate handler means the type of the exception object thrown matches the type of the exception object it can handle.
* If run-time system searches all the methods on call stack and couldn’t have found the appropriate handler then run-time system handover the Exception Object to default exception handler, which is part of run-time system. This handler prints the exception information in the following format and terminates program abnormally.

**Customized Exception Handling** : Java exception handling is managed via five keywords: try, catch, [throw, throws,](https://www.geeksforgeeks.org/throw-throws-java/) and finally. Briefly, here is how they work. Program statements that you think can raise exceptions are contained within a try block. If an exception occurs within the try block, it is thrown. Your code can catch this exception (using catch block) and handle it in some rational manner. System-generated exceptions are automatically thrown by the Java run-time system. To manually throw an exception, use the keyword [throw.](https://www.geeksforgeeks.org/throw-throws-java/) Any exception that is thrown out of a method must be specified as such by a [throws](https://www.geeksforgeeks.org/throw-throws-java/) clause. Any code that absolutely must be executed after a try block completes is put in a finally block.

* In a method, there can be more than one statements that might throw exception, So put all these statements within its own try block and provide separate exception handler within own catch block for each of them.
* If an exception occurs within the try block, that exception is handled by the exception handler associated with it. To associate exception handler, we must put catch block after it.

There can be more than one exception handlers. Each catch block is a exception handler that handles the exception of the type indicated by its argument. The argument, ExceptionType declares the type of the exception that it can handle and must be the name of the class that inherits from Throwable class.

* For each try block there can be zero or more catch blocks, but only one finally block.
* The finally block is optional. It always gets executed whether an exception occurred in try block or not . If exception occurs, then it will be executed after try and catch blocks. And if exception does not occur then it will be executed after the try block. The finally block in java is used to put important codes such as clean up code e.g. closing the file or closing the connection.

**Codes**

*Example 1*

import java.util.Scanner;

public class Except2

{

public static void main(String arg[])

{

int a=1, data ;

Scanner sc = new Scanner(System.in);

try

{

System.out.println("Enter the value of a ");

a = sc.nextInt();

}

catch(Exception e)

{

System.out.println("Invalid Input");

}

try

{

data=100/a;

System.out.println("Answer = "+ data);

}

catch(ArithmeticException m)

{

System.out.println("Cannot divide by zero");

}

}

}

/\* OUTPUT

C:\JavaProg>java Execpt2 Enter the value of a

3

Answer = 33

C:\JavaProg>java Execpt2 Enter the value of a

r

Invalid Input Answer = 100

\*/

*Example 2*

import java.util.Scanner;

public class Except3

{

public static void main(String arg[])

{

int a=0, data ;

Scanner sc = new Scanner(System.in);

try

{

System.out.println("Enter the value of a "); a = sc.nextInt();

try

{

data=100/a;

System.out.println("Answer = "+ data);

}

catch(ArithmeticException m)

{

System.out.println("Cannot divide by zero");

}

}

catch(Exception e)

{

System.out.println("Invalid Input");

}

}

}

/\*

OUTPUT

C:\JavaProg>java Except3 Enter the value of a

f

Invalid Input

C:\JavaProg>java Except3 Enter the value of a

0

Cannot divide by zero

\*/

**Multithreading**

Before we talk about multithreading, let’s discuss threads. A thread is a light-weight smallest part of a process that can run concurrently with the other parts (other threads) of the same process. Threads are independent because they all have separate path of execution that’s the reason if an exception occurs in one thread, it doesn’t affect the execution of other threads. All threads of a process share the common memory. The process of executing multiple threads simultaneously is known as multithreading.

Let’s summarize the discussion in points:

1. The main purpose of multithreading is to provide simultaneous execution of two or more parts of a program to maximum utilize the CPU time. A multithreaded program contains two or more parts that can run concurrently. Each such part of a program called thread.
2. Threads are lightweight sub-processes, they share the common memory space. In Multithreaded environment, programs that are benefited from multithreading, utilize the maximum CPU time so that the idle time can be kept to minimum.
3. A thread can be in one of the following states:

NEW – A thread that has not yet started is in this state. RUNNABLE – A thread executing in the Java virtual machine is in this state.

BLOCKED – A thread that is blocked waiting for a monitor lock is in this state.

WAITING – A thread that is waiting indefinitely for another thread to perform a particular action is in this state.

TIMED\_WAITING – A thread that is waiting for another thread to perform an action for up to a specified waiting time is in this state.

TERMINATED – A thread that has exited is in this state. A thread can be in only one state at a given point in time.

There are two ways to create a thread in Java:

1. By extending Thread class.
2. By implementing Runnable interface.

Before we begin with the programs(code) of creating threads, let’s have a look at these methods of Thread class. We have used few of these methods in the example below.

* + getName(): It is used for Obtaining a thread’s name
  + getPriority(): Obtain a thread’s priority
  + isAlive(): Determine if a thread is still running
  + join(): Wait for a thread to terminate
  + run(): Entry point for the thread
  + sleep(): Suspend a thread for a period of time
  + start(): Start a thread by calling its run() method

**Thread priorities**

* + Thread priorities are the integers which decide how one thread should be treated with respect to the others.
  + Thread priority decides when to switch from one running thread to another, process is called context switching
  + A thread can voluntarily release control and the highest priority thread that is ready to run is given the CPU.
  + A thread can be preempted by a higher priority thread no matter what the lower priority thread is doing. Whenever a higher priority thread wants to run it does.
  + To set the priority of the thread setPriority() method is used which is a method of the class Thread Class.
  + In place of defining the priority in integers, we can use

MIN\_PRIORITY, NORM\_PRIORITY or MAX\_PRIORITY.

**Methods: isAlive() and join()**

* + In all the practical situations main thread should finish last else other threads which have spawned from the main thread will also finish.
  + To know whether the thread has finished we can call isAlive() on the thread which returns true if the thread is not finished.
  + Another way to achieve this by using join() method, this method when called from the parent thread makes parent thread wait till child thread terminates.
  + These methods are defined in the Thread class.
  + We have used isAlive() method in the above examples too.

**Methods: isAlive() and join()**

* + In all the practical situations main thread should finish last else other threads which have spawned from the main thread will also finish.
  + To know whether the thread has finished we can call isAlive() on the thread which returns true if the thread is not finished.
  + Another way to achieve this by using join() method, this method when called from the parent thread makes parent thread wait till child thread terminates.
  + These methods are defined in the Thread class.

**Codes**

*Example 1:*

class TestSleepMethod1 extends Thread

{

public void run()

{

for(int i=1;i<5;i++)

{

try

{

Thread.sleep(500);

}

catch(InterruptedException e)

{

System.out.println(e);

}

System.out.println(i);

}

}

public static void main(String args[])

{

TestSleepMethod1 t1=new TestSleepMethod1(); TestSleepMethod1 t2=new TestSleepMethod1();

t1.start();

t2.start();

}

}

### **Java AWT**

AWT (Abstract Window Toolkit) is an API to develop GUI or window-based applications in java. Java AWT components are platform-dependent i.e. components are displayed according to the view of operating system. AWT is heavyweight i.e. its components are using the resources of OS. Java AWT calls native platform (Operating systems) subroutine for creating components such as textbox, checkbox, button etc.

For example, an AWT GUI having a button would have a different look and feel across platforms like windows, Mac OS & Unix, this is because these platforms have different look and feel for their native buttons and AWT directly calls their native subroutine that creates the button. In simple, an application build on AWT would look like a windows application when it runs on Windows, but the same application would look like a Mac application when runs on Mac OS.

The java.awt [package p](https://www.javatpoint.com/package)rovides [classes f](https://www.javatpoint.com/object-and-class-in-java)or AWT api such as [TextField](https://www.javatpoint.com/java-awt-textfield)[, Label](https://www.javatpoint.com/java-awt-label)[, TextArea,](https://www.javatpoint.com/java-awt-textarea)

RadioButt[on, CheckBox](https://www.javatpoint.com/java-awt-checkbox)[, Choice](https://www.javatpoint.com/java-awt-choice)[, List e](https://www.javatpoint.com/java-awt-list)tc.

**Components and containers**

All the elements like buttons, text fields, scrollbars etc are known as components. In AWT we have classes for each component as shown in the above diagram. To have everything placed on a screen to a particular position, we have to add them to a container. A container is like a screen wherein we are placing components like buttons, text fields, checkbox etc. In short, a container contains and controls the layout of components. A

container itself is a component (shown in the above hierarchy diagram) thus we can add a container inside container.

**Types of containers:**

As explained above, a container is a place wherein we add components like text field, button, checkbox etc. There are four types of containers available in AWT: Window, Frame, Dialog and Panel. As shown in the hierarchy diagram above, Frame and Dialog are subclasses of Window class.

**Window**

The window is the container that have no borders and menu bars. You must use frame, dialog or another window for creating a window.

**Panel**

The Panel is the container that doesn't contain title bar and menu bars. It can have other components like button, textfield etc.

**Frame**

The Frame is the container that contain title bar and can have menu bars. It can have other components like button, textfield etc.

**Java AWT Example**

To create simple awt example, you need a frame. There are two ways to create a frame in AWT. o By extending Frame class (inheritance) o By creating the object of Frame class (association)

# Code

import java.awt.\*; import java.awt.event.\*;

class Calculator implements ActionListener

{

Frame f=new Frame();

Label l1=new Label("First Number"); Label l2=new Label("Second Number"); Label l3=new Label("Result");

TextField t1=new TextField(); TextField t2=new TextField(); TextField t3=new TextField();

Button b1=new Button("Add"); Button b2=new Button("Sub"); Button b3=new Button("Mul"); Button b4=new Button("Div"); Button b5=new Button("Cancel");

Calculator()

{

l1.setBounds(50,100,100,20); l2.setBounds(50,140,100,20); l3.setBounds(50,180,100,20);

t1.setBounds(200,100,100,20); t2.setBounds(200,140,100,20); t3.setBounds(200,180,100,20);

b1.setBounds(50,250,50,20); b2.setBounds(110,250,50,20); b3.setBounds(170,250,50,20); b4.setBounds(230,250,50,20); b5.setBounds(290,250,50,20);

f.add(l1);

f.add(l2);

f.add(l3);

f.add(t1);

f.add(t2);

f.add(t3);

f.add(b1);

f.add(b2);

f.add(b3);

f.add(b4);

f.add(b5);

b1.addActionListener(this); b2.addActionListener(this); b3.addActionListener(this); b4.addActionListener(this); b5.addActionListener(this);

f.setLayout(null); f.setVisible(true);

f.setSize(400,350);

}

public void actionPerformed(ActionEvent e)

{

int n1=Integer.parseInt(t1.getText()); int n2=Integer.parseInt(t2.getText());

if(e.getSource()==b1)

{

t3.setText(String.valueOf(n1+n2));

}

if(e.getSource()==b2)

{

t3.setText(String.valueOf(n1-n2));

}

if(e.getSource()==b3)

{

t3.setText(String.valueOf(n1\*n2));

}

if(e.getSource()==b4)

{

t3.setText(String.valueOf(n1/n2));

}

if(e.getSource()==b5)

{

System.exit(0);

}

}

public static void main(String args[])

{

new Calculator();

}

}

**Java Swings**

Java Swing tutorial is a part of Java Foundation Classes (JFC) that is used to create window-based applications. It is built on the top of AWT (Abstract Windowing Toolkit) API and entirely written in java. Swing in Java is a Graphical User Interface (GUI) toolkit that includes the GUI components. Swing provides a rich set of widgets and packages to make sophisticated GUI components for Java applications. Swing is a part of Java Foundation Classes (JFC), which is an API for Java programs that provide GUI.

The Java Swing library is built on top of the Java Abstract Widget Toolkit (AWT), an older, platform dependent GUI toolkit. You can use the Java GUI programming components like button, textbox, etc. from the library and do not have to create the components from scratch.

**GUI (Graphical User Interface) in Java** is an easy-to-use visual experience builder for Java applications. It is mainly made of graphical components like buttons, labels, windows, etc. through which the user can interact with an application. GUI plays an important role to build easy interfaces for Java applications.

Unlike AWT, Java Swing provides platform-independent and lightweight components. The javax.swing package provides classes for java swing API such as JButton, JTextField, JTextArea, JRadioButton, JCheckbox, JMenu, JColorChooser etc.

Container classes are classes that can have other components on it. So for creating a Java GUI, we need at least one container object. There are 3 types of Java Swing containers.

1. Panel: It is a pure container and is not a window in itself. The sole purpose of a Panel is to organize the components on to a window.
2. Frame: It is a fully functioning window with its title and icons.
3. Dialog: It can be thought of like a pop-up window that pops out when a message has to be displayed. It is not a fully functioning window like the Frame.

**Java Layout Manager**

The Layout manager is used to layout (or arrange) the GUI java components inside a container.There are many layout managers, but the most frequently used are-

**Java BorderLayout**

A BorderLayout places components in up to five areas: top, bottom, left, right, and center. It is the default layout manager for every java JFrame.

**Java FlowLayout**

FlowLayout is the default layout manager for every JPanel. It simply lays out components in a single row one after the other.

**Java GridBagLayout**

It is the more sophisticated of all layouts. It aligns components by placing them within a grid of cells, allowing components to span more than one cell.

**Code**

import javax.swing.\*; import java.awt.event.\*;

public class bill\_generator extends JFrame implements ActionListener

{

JLabel l;

JCheckBox cb1,cb2,cb3,cb4,cb5; JButton b;

bill\_generator()

{ l=new JLabel("Food Ordering System");

l.setBounds(50,25,300,20); cb1=new JCheckBox("Pizza = 250"); cb1.setBounds(100,100,150,20); cb2=new JCheckBox("Burger = 150"); cb2.setBounds(100,125,150,20); cb3=new JCheckBox("Noddles = 120"); cb3.setBounds(100,150,150,20); cb4=new JCheckBox("Tea = 25"); cb4.setBounds(100,175,150,20);

cb5=new JCheckBox("Coffee = 25"); cb5.setBounds(100,200,150,20);

b=new JButton("Order");

b.setBounds(100,300,80,30); b.addActionListener(this);

add(l);add(cb1);add(cb2);add(cb3);add(b);add(cb4);add(cb5); setSize(400,400); setLayout(null); setVisible(true);

setDefaultCloseOperation(EXIT\_ON\_CLOSE);

} public void actionPerformed(ActionEvent e)

{

float amount=0; String msg=""; if(cb1.isSelected())

{

amount+=100;

msg="Pizza : 250\n";

}

if(cb2.isSelected())

{

amount+=150;

msg+="Burger : 150\n";

}

if(cb3.isSelected())

{

amount+=120; msg+="Noodles : 120\n";

}

if(cb4.isSelected())

{

amount+=25; msg+="Tea : 25\n";

}

if(cb5.isSelected())

{

amount+=25;

msg+="Coffee : 25\n";

}

}

args)

{

msg+=" \n";

JOptionPane.showMessageDialog(this,msg+"Total: "+amount); public static void main(String[]

new

bill\_generator();

}

}

**Menus and Menubar**

The JMenuBar class is used to display menubar on the window or frame. It may have several menus. The object of JMenu class is a pull-down menu component which is displayed from the menu bar. It inherits the JMenuItem class. The object of JMenuItem class adds a simple labeled menu item. The items used in a menu must belong to the JMenuItem or any of its subclass. JMenuBar, JMenu and JMenuItems are a part of Java Swing package.

JMenuBar is an implementation of menu bar . the JMenuBar contains one or more JMenu objects, when the JMenu objects are selected they display a popup showing one or more JMenuItems .

**JMenuBar class declaration**

1. public class JMenuBar extends JComponent implements MenuElement, Accessible
2. public class JMenu extends JMenuItem implements MenuElement, Accessible. JMenuItem class declaration
3. public class JMenuItem extends AbstractButton implements Accessible, MenuElement

Class Constructors

|  |  |
| --- | --- |
| **Sr.No.** | **Constructor & Description** |
| 1 | **JMenuBar()**  Creates a new menu bar. |

Class Methods

Here are some methods in Swing JMenuBar Control class.

|  |  |
| --- | --- |
| **Sr.No.** | **Method & Description** |
| 1 | **JMenu add(JMenu c)**  Appends the specified menu to the end of the menu bar. |

|  |  |
| --- | --- |
| 2 | **void addNotify()**  Overrides JComponent.addNotify to register this menu bar with the current keyboard manager. |
| 3 | **AccessibleContext getAccessibleContext()**  Gets the AccessibleContext associated with this JMenuBar. |
| 4 | **Component getComponent()**  Implemented to be a MenuElement. |
| 5 | **Component getComponentAtIndex(int i)**  Deprecated. Replaced by getComponent(int i) |
| 6 | **int getComponentIndex(Component c)**  Returns the index of the specified component. |
| 7 | **JMenu getHelpMenu()**  Gets the help menu for the menu bar. |
| 8 | **Insets getMargin()**  Returns the margin between the menubar's border and its menus. |
| 9 | **JMenu getMenu(int index)**  Returns the menu at the specified position in the menu bar. |
| 10 | **int getMenuCount()**  Returns the number of items in the menu bar. |
| 11 | **SingleSelectionModel getSelectionModel()**  Returns the model object that handles single selections. |
| 12 | **MenuElement[] getSubElements()**  Implemented to be a MenuElement. Returns the menus in this menu bar. |

|  |  |
| --- | --- |
| 13 | **MenuBarUI getUI()**  Returns the menubar's current UI. |
| 14 | **String getUIClassID()**  Returns the name of the L&F class that renders this component. |
| 15 | **boolean isBorderPainted()**  Returns true if the menu bars border should be painted. |

**Code**

import javax.swing.\*;

class Menu\_1

{

JMenu menu, submenu; JMenuItem i1, i2, i3, i4, i5; Menu\_1()

{

JFrame f = new JFrame(); JMenuBar mb = new JMenuBar(); menu = new JMenu("Menu"); submenu = new JMenu("Sub Menu");

i1 = new JMenuItem("1"); i2 = new JMenuItem("2"); i3 = new JMenuItem("3"); i4 = new JMenuItem("4"); i5 = new JMenuItem("5");

menu.add(i1);

menu.add(i2);

menu.add(i3);

submenu.add(i4); submenu.add(i5);

menu.add(submenu); mb.add(menu);

f.setJMenuBar(mb); f.setSize(400,400);

f.setLayout(null); f.setVisible(true);

}

public static void main(String args[])

{

new Menu\_1();

}

}

**Japplet**

JApplet is a java swing public class designed for developers usually written in Java. JApplet is generally in the form of Java bytecode that runs with the help of a Java virtual machine (JVM) or Applet viewer from Sun Microsystems. It was first introduced in 1995.

JApplet can also be written in other programming languages and can later be compiled to Java byte code.

Java applets can be executed on multiple platforms which include Microsoft Windows, UNIX, Mac OS and Linux. JApplet can also be run as an application, though this would require a little extra coding. The executable applet is made available on a domain from which it needs to be downloaded. The communication of the applet is restricted only to this particular domain.

JApplet extends the class in the form of java.applet.Applet. JApplets are executed in a tightly-controlled set of resources referred to as sandboxes. This prevents the JApplets from accessing local data like the clipboard or file system.

The first JApplet implementations were performed by downloading an applet class by class. Classes contain many small files and so applets were considered to be slow loading components. Since the introduction of the Java Archive (or simply JAR file), an applet is aggregated and sent as a single, but larger file.

The definition of JApplet provides a framework for building an applet. By itself, the class JApplet does little that is visible in the Web browser. (It does a great many things behind the scenes, however.) To build upon this framework,

you import javax.swing.JApplet and extend the JApplet class.

When you extend a class, you are making a new class by building upon a base class. This example defines a new class called JustOneCircle . The new class has everything in it that the class JApplet has. (This is

called inheritance. Inheritance is discussed at greater length in chapter 50.)

The class JApplet has a paint() method, but that method does little. Objects of class JustOneCircle have their own paint() method because the definition in JustOneCircle.java overrides the one in JApplet.

The Web browser calls the paint() method when it needs to "paint" the section of the monitor screen devoted to an applet. Each applet that you write has its own paint() method.

# Code

import java.awt.\*; import javax.swing.\*; import java.awt.event.\*;

public class Japplet1

{

JFrame f;

JButton b1, b2, b3, b4, b5;

Japplet1()

{

f = new JFrame("Direction picker"); b1 = new JButton("UP");

b1.addActionListener(new AbstractAction("UP")

{

@Override

public void actionPerformed(ActionEvent e)

{

System.out.println("You are at Up");

}

});

b2 = new JButton("Down"); b2.addActionListener(new AbstractAction("Down")

{

@Override

public void actionPerformed(ActionEvent e)

{

System.out.println("You are at Down");

}

});

b3 = new JButton("Right"); b3.addActionListener(new AbstractAction("Right")

{

@Override

public void actionPerformed(ActionEvent e)

{

System.out.println("You are at Right");

}

});

b4 = new JButton("Left"); b4.addActionListener(new AbstractAction("Left")

{

@Override

public void actionPerformed(ActionEvent e)

{

System.out.println("You are at Left");

}

});

b5 = new JButton("Centre"); b5.addActionListener(new AbstractAction("Centre")

{

@Override

public void actionPerformed(ActionEvent e)

{

System.out.println("You are at Centre");

}

});

f.add(b1,BorderLayout.NORTH); f.add(b2,BorderLayout.SOUTH); f.add(b3,BorderLayout.EAST); f.add(b4,BorderLayout.WEST); f.add(b5,BorderLayout.CENTER); f.setSize(500,500); f.setVisible(true);

}

public static void main(String[] args)

{

new Japplet1();

}

}