Guru Nanak Dev Engineering College , Ludhiana

Java Programming Laboratory

(LPCIT - 109)

Submitted by :
Chandan Kumar
D3 IT-A1
C.R.N : 2021022
U.R.N : 2004899

Submitted to :
Pf. Kamaljeet Kaur Dhillon

`

Lab 1: Datatypes

Data Types in Java

Data types specify the different sizes and values that can be stored in the variable. There are two types of data types in Java:

1. **Primitive data types:** The primitive data types include boolean, char, byte, short, int, long, float and double.
2. **Non-primitive data types:** The non-primitive data types include Classes, Interfaces, and Arrays.

Java Primitive Data Types

- boolean data type
- byte data type
- char data type
- short data type
- int data type
- long data type
- float data type
- double data type

Java Non-Primitive Data Types

- String
- Array

Program:

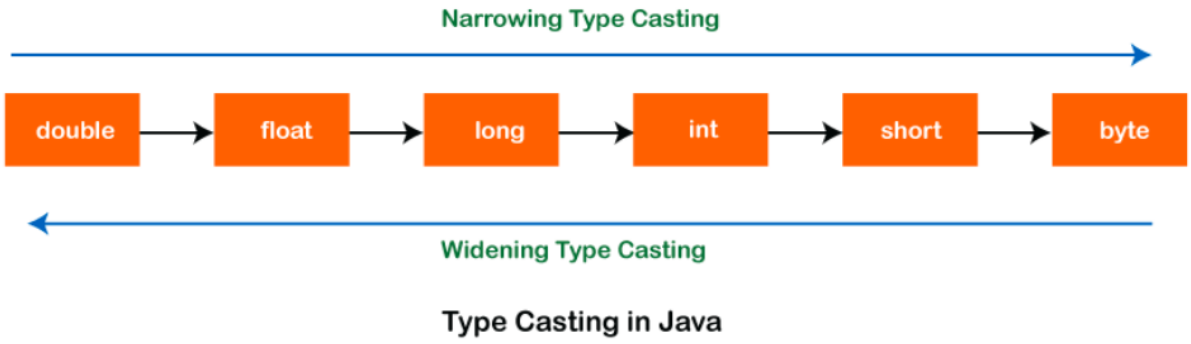
```
public class dataTypes {
    public static void main(String chndn[]) {
        int a = 9;
        char b = 'h';
        String s = "chandan";
        float _abc = 87.5856f;

        double have = 2.8569;
        System.out.println("Type of a is :"+
((Object)a).getClass().getSimpleName());
        System.out.println("Type of b is :"+
((Object)b).getClass().getSimpleName());
        System.out.println("Type of s is
"+s.getClass().getSimpleName());
        System.out.println("Type of abc is :"+
((Object)_abc).getClass().getSimpleName());


    }
}
```

Output:

```
cd "/home/chandan/Programming/JavaProgramming/" && javac dataTypes.java && java dataTypes
chandan@kumar:~/Programming/JavaProgramming$ cd "/home/chandan/Programming/JavaProgramming/" &&
Type of a is :Integer
Type of b is :Character
Type of s is String
Type of abc is :Float
chandan@kumar:~/Programming/JavaProgramming$
```

Practical No:- 2

Type casting:- In Java, type casting is a method or process that converts a data type into another data type in both ways manually and automatically. The automatic conversion is done by the compiler and manual conversion performed by the programmer.



Type Casting in Java

Code:-

```
public class typeCasting {
public static void main(String chndn[]) {
int a = 78;
double b = a;// Automatic casting int to double
System.out.println(b);

// manually casting double to int

double d = 98.25;
int e = (int) d;
System.out.println(e);

char ch = 'A';
int cha = (int) ch;
System.out.println(cha);
}

}
```

Output:-

```
chandan@kumar:~/Programming/JavaProgramming/javaLaboratory$
78.0
98
65
```

Practical No:-3

Arrays:-In Java, all arrays are dynamically allocated. (discussed below)

- Arrays are stored in contagious memory [consecutive memory locations].
- Since arrays are objects in Java, we can find their length using the object property *length*. This is different from C/C++, where we find length using sizeof.
- A Java array variable can also be declared like other variables with [] after the data type.
- The variables in the array are ordered, and each has an index beginning from 0.
- Java array can also be used as a static field, a local variable, or a method parameter.
- The size of an array must be specified by int or short value and not long.
- The direct superclass of an array type is **Object**.
- Every array type implements the interfaces Cloneable and **java.io.Serializable**.
- This storage of arrays helps us in randomly accessing the elements of an array [Support Random Access].
- The size of the array cannot be altered(once initialized). However, an array reference can be made to point to another array.

Code:-

```
import java.util.*;
public class arrays {
public static void main(String args[]){
int arr[] = {10,20,43,85,56,23};
for(int i=0;i<arr.length;i++){
System.out.print(arr[i]+" ");
}
System.out.print("\n");
}

}
```

Output:-

```
chandan@kumar:~/Programming/Java_lab$ cd "/home/chandan/Programmi
c/" && javac arrays.java && java arrays
10 20 43 85 56 23
```

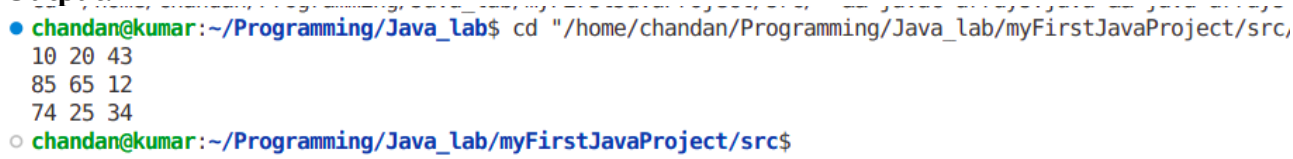
2-D array

Code:-

```
import java.util.*;

public class arrays {
public static void main(String args[]) {
int arr[][] = { { 10, 20, 43 }, { 85, 65, 12 }, { 74, 25, 34 } };
for (int i = 0; i < 3; i++) {
for (int j = 0; j < 3; j++) {
```

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

}
System.out.print("\n");

}

}

}
```

Output:-



Practical No:-4

Various control structures

1. for Loop:- A for loop is a control flow statement for specifying iteration, which allows code to be executed repeatedly. A for loop has two parts: a header specifying the iteration, and a body which is executed once per iteration.

Code->

```
import java.util.*;

public class forLoop {

public static void main(String[] args){
Scanner scn = new Scanner(System.in);
System.out.println("Enter a number.....");
int num = scn.nextInt ();
for(int i=1;i<=10;i++){
System.out.println(num+" * "+i + " = "+num*i );
}
}
}
```

Output:-

2.for each loop:- In computer programming, foreach loop (or for each loop) is a control flow statement for traversing items in a collection. foreach is usually used in place of a standard for loop statement.

```
chandan@kumar:~/Programming/JavaProgramming$ cd "/home/chandan/Pr
Enter a number.....
2
2 * 1 = 2
2 * 2 = 4
2 * 3 = 6
2 * 4 = 8
2 * 5 = 10
2 * 6 = 12
2 * 7 = 14
2 * 8 = 16
2 * 9 = 18
2 * 10 = 20
```

Code->

```
public class forEach {
public static void main(String args []){
String str[] = {"chandan","harsh","ajit","ankit","aditya"};
for(String v:str){
System.out.println(v);

}

}
}
```

Output:-

3.

```
chandan@kumar:~/Programming/JavaProgramming$ cd "/home/chandan/Pr
chandan
harsh
ajit
ankit
aditya
```

while Loop:- The while loop is used to repeat a section of code an unknown number of times until a specific condition is met

Code->

```
import java.util.*;

public class whileLoop {
public static void main(String args[]) {
Scanner scn = new Scanner(System.in);
System.out.println("Enter a number.....");
int num = scn.nextInt();
int a = 1;
while (num!=0) {
System.out.println(a++);
num--;

}

}
}
```

Output:-

```
chandan@kumar:~/Programming/JavaProgramming$ cd
Enter a number.....
5
1
2
3
4
5
```

Practical No- 5

Various decision structures

1. continue:- Continue is also a loop control statement just like the break statement. *continue* statement is opposite to that of break *statement*, instead of terminating the loop, it forces to execute the next iteration of the loop.

Code:-

```
public class Continue {
public static void main(String args[]){
for(int i=1;i<=10;i++){
if(i==5)
continue;
System.out.println(i);
}
}
}
```

Output:-

2.break :-The break in C or C++ is a loop control statement which is used to terminate the loop. As soon as the break statement is encountered from within a loop, the loop iterations stops there and control returns from the loop immediately to the first statement after the loop.

Code:-

```
public class Continue {
public static void main(String args[]){
for(int i=1;i<=10;i++){
if(i==5)
break;
System.out.println(i);
}
}
}
```

Output:-

3.if statement:-It is the most basic statement among all control flow statements in Java. It evaluates a Boolean expression and enables the program to enter a block of code if the expression evaluates to true.

Code:-

```
public class Continue {
public static void main(String args[]) {
for (int i = 1; i <= 10;i++) {
if (i == 5){
System.out.println("the entered no is 5");
break;}
}

}

}
```

Output:-

```
cd "/home/chandan/Programming/JavaProgramming/javaLabo
chandan@kumar:~/Programming/JavaProgramming$ cd "/home
the entered no is 5
chandan@kumar:~/Programming/JavaProgramming/javaLabora
```

4.if-else The if-else statement is an extension to the if-statement, which uses another block of code, i.e., else block. The else block is executed if the condition of the if-block is evaluated as false.

Code:-

```
public class Continue {
public static void main(String args[]) {
int n = 9;
if(n%2 ==0){
System.out.println(n+" is even no.");
}
else
System.out.println(n+" is odd no");
}

}
```

Output:-

```
chandan@kumar:~/Programming/JavaProgramming/javaLaboratory/src$ cd "/home/chandan/Prog
9is odd no
chandan@kumar:~/Programming/JavaProgramming/javaLaboratory/src$
```

5.nested if else:-In nested if-statements, the if statement can contain a **if** or **if-else** statement inside another if or else-if statement.

Code:-

Code:-

```
public class Continue {
```

```
public static void main(String[] args) {
String address = "Delhi, India";

if (address.endsWith("India")) {
if (address.contains("Meerut")) {
System.out.println("Your city is Meerut");
} else if (address.contains("Noida")) {
System.out.println("Your city is Noida");
} else {
System.out.println(address.split(",")[0]);
}
} else {
System.out.println("You are not living in India");
}
}
}
```

Ouput:-

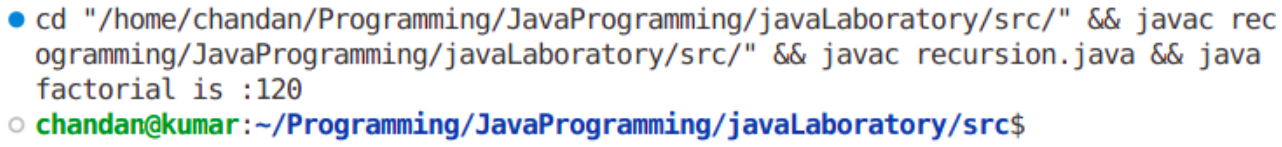
Practical No:-6

Recursion:-The process in which a function calls itself directly or indirectly is called recursion and the corresponding function is called as recursive function. Using recursive algorithm, certain problems can be solved quite easily. Examples of such problems are Towers of Hanoi (TOH), Inorder/Preorder/Postorder Tree Traversals, DFS of Graph, etc. What is base condition in recursion? In the recursive program, the solution to the base case is provided and the solution of the bigger problem is expressed in terms of smaller problems.

Code->

```
public class recursion {
public static void main(String args[]){
int n = 5;
System.out.println("factorial is :"+factorial(n));

}
public static int factorial(int n){
if(n == 0){
return 1;
}
else return n*factorial(n-1);
}
}
```

Output:-



Practical No:-7

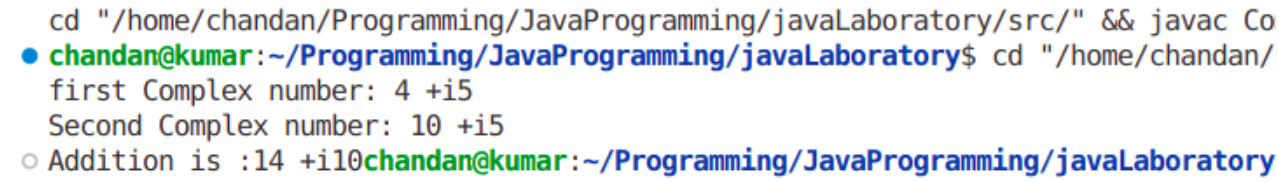
Method Overloading:-Method Overloading allows different methods to have the same name, but different signatures where the signature can differ by the number of input parameters or type of input parameters, or a mixture of both.

Method overloading is also known as ***Compile-time Polymorphism***, *Static Polymorphism, or* ***Early binding*** in Java. In Method overloading compared to parent argument, child argument will get the highest priority.

Code->

```
public class ComplexNumber {
int real, image;

public ComplexNumber(int r, int i) {
this.real = r;
this.image = i;
}

public void showC() {
System.out.print(this.real + " +i" + this.image);
}

public static ComplexNumber add(ComplexNumber n1, ComplexNumber n2) {
ComplexNumber res = new ComplexNumber(0, 0);

res.real = n1.real + n2.real;

res.image = n1.image + n2.image;

return res;
}

public static void main(String arg[]) {

// creating two complex numbers
ComplexNumber c1 = new ComplexNumber(4, 5);
ComplexNumber c2 = new ComplexNumber(10, 5);

// printing complex numbers
System.out.print("first Complex number: ");
c1.showC();

System.out.print("\nSecond Complex number: ");
c2.showC();

// calling add() to perform addition
ComplexNumber res = add(c1, c2);

// displaying addition
System.out.print("\nAddition is :");
res.showC();
}
}
```

Output:-


cd "/home/chandan/Programming/JavaProgramming/javaLaboratory/src/" && javac Co
chandan@kumar:~/Programming/JavaProgramming/javaLaboratory$ cd "/home/chandan/
first Complex number: 4 +i5
Second Complex number: 10 +i5
Addition is :14 +i10chandan@kumar:~/Programming/JavaProgramming/javaLaboratory

Practical No:- 8

Constructor Overloading by passing objects as arguments

Code->

```
import java.util.*;

class area {
double length, breadth;

area() {
length = 0;
breadth = 0;
}

area(int l, int b) {
length = l;
breadth = b;
}

area(int l) {
length = breadth = l;
}

double area() {
return length * breadth;

}
}

public class Rectangle {

public static void main(String args[]) {
area r1 = new area();
area r2 = new area(5);
area r3 = new area(2, 6);
System.out.println("area of rectangle 1 is: " + r1.area());
System.out.println("area of rectangle 2 is: " + r2.area());
System.out.println("area of rectangle 3 is: " + r3.area());

}
}
```

Output:-

```
chandan@kumar:~/Programming/JavaProgramming$ cd "/home/chandan/Programming/JavaProgramming/ja
area of rectangle 1 is: 0.0
area of rectangle 2 is: 25.0
area of rectangle 3 is: 12.0
```

Practical No:- 9

Various access control and usage of static, final and finalize ( )

final:

code->

```
public class MyClass {
final int age = 18;

void showAge() {
age = 55;
}

public static void main(String[] args) {

MyClass student = new MyClass();

student.showAge();
}
}
```

Output:->

```
chandan@kumar:~/Programming/JavaProgramming/javaLaboratory$ cd "/home
MyClass.java:6: error: cannot assign a value to final variable age
        age = 55;
        ^
1 error
```

finalize:

Code->

```
public class finalize {
public static void main(String[] args) {
finalize str2 = new finalize();
str2 = null;

System.gc();
System.out.println("output of main method");
}

protected void finalize() {
System.out.println("output of finalize method");
}
}
```

Output->

```
chandan@kumar:~/Programming/JavaProgramming/javaLaboratory$ cd "/ho
Note: finalize.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
output of finalize method
output of main method
```

Static:

Code->

```
public class static1 {
// static method
static void show() {
System.out.println("Calling method without creating any object of class");
}

public static void main(String[] args) {
show();
}
}
```

Output:-

```
chandan@kumar:~/Programming/JavaProgramming/javaLaboratory$ cd "/home/chand
Calling method without creating any object of class
chandan@kumar:~/Programming/JavaProgramming/javaLaboratory/src$
```

Practical No :- 10

Command line arguments

Code->

```
public class Main {
public static void main(String args[]) {
for (int i = 0; i < args.length; i++) {
System.out.println("args[" + i + "]: " + args[i]);
}
}
}
```

Output->

```
chandan@kumar:~/Programming/JavaProgramming/javaLaboratory/src$ javac Main.java
chandan@kumar:~/Programming/JavaProgramming/javaLaboratory/src$ java Main Chandan Ankit Ajit Harsh
args[0]: Chandan
args[1]: Ankit
args[2]: Ajit
args[3]: Harsh
```

Practical No :- 11

Inheritance in Java

Code->

```
class Worker {
public int salary = 20000;
}

class Foreman extends Worker {
public int bonus = 5000;
}

public class Myclass1 {
public static void main(String args[]) {
Foreman f = new Foreman();
System.out.println("Worker salary is:" + f.salary);
System.out.println("Bonus of Worker(Foreman) is:" + f.bonus);
}
}
```

Output->

```
chandan@kumar:~/Programming/JavaProgramming/javaLaboratory$
Worker salary is:20000
Bonus of Worker(Foreman) is:5000
```

Practical No :- 12

Method Overriding

Code->

```
class Parent {
void fun() {
System.out.println("parent's fun");
}
}

class Child extends Parent {
@Override
void fun() {
System.out.println("child's fun");
}
}

class GrandChild extends Child {
@Override
void fun() {
System.out.println("Grandchild's fun");
}
}

public class overriding {
public static void main(String args[]) {
Parent p = new Child();
Parent ch = new GrandChild();
p.fun();
ch.fun();

}

}
```

Output:-

```
chandan@kumar:~/Programming/JavaProgramming/javaLaboratory$ c
child's fun
Grandchild's fun
```

Practical No :-13

Abstract classes

Code->

```
abstract class parent {
abstract void write();
}

class child extends parent {
@Override
void write() {
System.out.println("Writing.....");
}

}
public class Abstract{

public static void main(String args[]) {
child a = new child();
a.write();
}
}
```

Output:-

Practical No :-14

Nested class

Code->

```
class outer {
int x = 9;

class nested {
int y = 4;
}
}

public class nestedClass {
public static void main(String[] args) {

outer otr = new outer();

outer.nested nstd = otr.new nested();

System.out.println("Variable of outer class : " + otr.x);
System.out.println("Variable of nested class : " + nstd.y);
}
}
```

Output:-

```
chandan@kumar:~/Programming/JavaProgramming/javaLaboratory$
Variable of outer class : 9
Variable of nested class : 4
```

Practical No:- 15

Constructor chaining

Code->

```
public class ConstructorChaining {
ConstructorChaining() {
this(5);
System.out.println("The Default constructor");
}

ConstructorChaining(int x) {
this(5, 15);
System.out.println(x);
}

ConstructorChaining(int x, int y) {
System.out.println(x * y);
}

public static void main(String args[]) {
new ConstructorChaining();
}
}
```

Output:-

```
chandan@kumar:~/Programming/JavaProgramming/javaLaboratory$ cd "/hc
tructorChaining
75
5
The Default constructor
```

Practical No:- 16

Importing classes from user defined package and creating packages using access protection

Code->

```
package p1;


class A {
void print() {
System.out.println("In class A ");
}
}

public class Packages {
public static void main(String[] args) {
A a = new A();
a.print();

}

}
```

Output:-

```
chandan@kumar:~/Programming/JavaProgramming/java_packages/Introduction_packages/src$ javac -d . Packages.java
chandan@kumar:~/Programming/JavaProgramming/java_packages/Introduction_packages/src$ java p1.Packages
In class A
```

Practical No:-17

Interfaces, nested interfaces and use of extending interfaces

Interfaces:-

Code->

```
interface Gne{
public void print();
}
class It implements Gne{
@Override
public void print(){
System.out.println("Guru Nanak Dev Engineering College");
}
}

public class Interfaces {
public static void main(String[] args) {
It a = new It();
a.print();
}
}
```

Output:-

```
chandan@kumar:~/Programming/JavaProgramming/java_packages/Introduction_packages/src$ javac Interfaces.java
chandan@kumar:~/Programming/JavaProgramming/java_packages/Introduction_packages/src$ java Interfaces
Guru Nanak Dev Engineering College
```

Nested Interfaces:-

Code->

```
interface Mahindra{
interface Thar{
public void print();
}
}
class Testing implements Mahindra.Thar{
@Override
public void print(){
System.out.println("Thar is a car model launched by Mahindra");
}
}

public class NestedInterface {
public static void main(String[] args) {
Mahindra.Thar obj;
Testing t = new Testing();
obj = t;
obj.print();

}
```

```
}
```

Output->

```
chandan@kumar:~/Programming/JavaProgramming/java_packages/Introduction_packages/src$ javac NestedInterface.java
chandan@kumar:~/Programming/JavaProgramming/java_packages/Introduction_packages/src$ java NestedInterface
Thar is a car model launched by Mahindra
```

Practical No:-18

Exception Handling - using predefined exception

Code->

```
public class PredefinedException {
public static void main(String args[]) {
try {
int a = 30, b = 0;
int c = a / b; // cannot divide by zero
System.out.println("Result = " + c);
} catch (ArithmeticException e) {
System.out.println("Can't divide a number by 0");
}
}
}
```

Output:-

```
chandan@kumar:~/Programming/JavaProgramming/java_packages/Introduction_packages$ cd src
chandan@kumar:~/Programming/JavaProgramming/java_packages/Introduction_packages/src$ javac PredefinedException.java
chandan@kumar:~/Programming/JavaProgramming/java_packages/Introduction_packages/src$ java PredefinedException
Can't divide a number by 0
```

Practical No:-19

Exception Handling - creating user defined exceptions

Code->

```
// Java program to demonstrate user defined exception
// This program throws an exception whenever balance
// amount is below Rs 1000
public class Myexceptions extends Exception {
// store account information
private static int accno[] = { 1001, 1002, 1003, 1004 };

private static String name[] = { "Nish", "Shubh", "Sush", "Abhi", "Akash" };

private static double bal[] = { 10000.00, 12000.00, 5600.0, 999.00, 1100.55 };

// default constructor
Myexceptions() {
}

// parameterized constructor
Myexceptions(String str) {
super(str);
}

// write main()
public static void main(String[] args) {
try {
// display the heading for the table
System.out.println("ACCNO" + "\t" + "CUSTOMER" +
"\t" + "BALANCE");

// display the actual account information
for (int i = 0; i < 5; i++) {
System.out.println(accno[i] + "\t" + name[i] +
"\t" + bal[i]);

// display own exception if balance < 1000
if (bal[i] < 1000) {
Myexceptions me = new Myexceptions("Balance is less than 1000");
throw me;
}
}
} // end of try

catch (Myexceptions e) {
e.printStackTrace();
}
}
}
```

Output:-

```
chandan@kumar:~/Programming/JavaProgramming/java_packages/Introduction_packages/src$ javac Myexceptions.java
chandan@kumar:~/Programming/JavaProgramming/java_packages/Introduction_packages/src$ java Myexceptions
ACCNO   CUSTOMER        BALANCE
1001    Nish    10000.0
1002    Shubh   12000.0
1003    Sush    5600.0
1004    Abhi    999.0
Myexceptions: Balance is less than 1000
        at Myexceptions.main(Myexceptions.java:38)
```

Practical No:- 20

Multithreading by extending Thread Class

Code->

```
class Test extends Thread {
public void run() {
System.out.println("inside run thread");
}
}

public class ExtendingThread {

public static void main(String[] args) {
Test t = new Test();
t.start();
System.out.println("inside main thread");

}
}
```

Output:-

```
chandan@kumar:~/Programming/JavaProgramming/java_packages/Introduction_packages/src$ javac ExtendingThread.java
chandan@kumar:~/Programming/JavaProgramming/java_packages/Introduction_packages/src$ java ExtendingThread
inside main thread
inside run thread
```

Practical No:-21

Multithreading by implementing Runnable Interface

Code->

```
class Test implements Runnable {
public void run() {
System.out.println("inside test thread");
}
}

public class UsingRunnable {
public static void main(String[] args) {
Test t = new Test();
Thread d = new Thread(t);
d.start();
System.out.println("inside main thread");
}
}
```

Output:-

```
chandan@kumar:~/Programming/JavaProgramming/java_packages/Introduction_packages/src$ javac UsingRunnable.java
chandan@kumar:~/Programming/JavaProgramming/java_packages/Introduction_packages/src$ java UsingRunnable
inside main thread
inside test thread
```

Practical No:-22

Thread life cycle

Code->

```
// Java program to demonstrate thread states
class thread implements Runnable {
public void run()
{
// moving thread2 to timed waiting state
try {
Thread.sleep(1500);
}
catch (InterruptedException e) {
e.printStackTrace();
}

System.out.println(
"State of thread1 while it called join() method on thread2 -"
+ ThreadLife.thread1.getState());
try {
Thread.sleep(200);
}
catch (InterruptedException e) {
e.printStackTrace();
}
}
}

public class ThreadLife implements Runnable {
public static Thread thread1;
public static ThreadLife obj;

public static void main(String[] args)
{
obj = new ThreadLife();
thread1 = new Thread(obj);

// thread1 created and is currently in the NEW
// state.
System.out.println(
"State of thread1 after creating it - "
+ thread1.getState());
thread1.start();

// thread1 moved to Runnable state
System.out.println(
"State of thread1 after calling .start() method on it - "
+ thread1.getState());
}

public void run()
{
thread myThread = new thread();
Thread thread2 = new Thread(myThread);
```

```
// thread1 created and is currently in the NEW
// state.
System.out.println(
"State of thread2 after creating it - "
+ thread2.getState());
thread2.start();

// thread2 moved to Runnable state
System.out.println(
"State of thread2 after calling .start() method on it - "
+ thread2.getState());

// moving thread1 to timed waiting state
try {
// moving thread1 to timed waiting state
Thread.sleep(200);
}
catch (InterruptedException e) {
e.printStackTrace();
}
System.out.println(
"State of thread2 after calling .sleep() method on it - "
+ thread2.getState());

try {
// waiting for thread2 to die
thread2.join();
}
catch (InterruptedException e) {
e.printStackTrace();
}
System.out.println(
"State of thread2 when it has finished it's execution - "
+ thread2.getState());
}
}
```

Output:-

```
chandan@kumar:~/Programming/JavaProgramming/java_packages/Introduction_packages/src$ javac ThreadLife.java
chandan@kumar:~/Programming/JavaProgramming/java_packages/Introduction_packages/src$ java ThreadLife
State of thread1 after creating it - NEW
State of thread1 after calling .start() method on it - RUNNABLE
State of thread2 after creating it - NEW
State of thread2 after calling .start() method on it - RUNNABLE
State of thread2 after calling .sleep() method on it - TIMED_WAITING
State of thread1 while it called join() method on thread2 -WAITING
State of thread2 when it has finished it's execution - TERMINATED
```

Practical No:-23

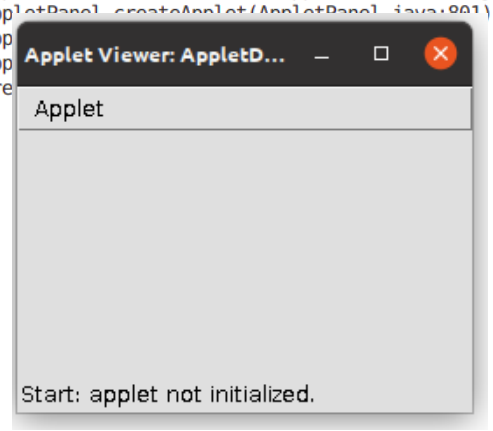
Applet life cycle

Code->

```
// Java Program to Illustrate Insertion of HTML File in
// Applet As Commands

// Importing required classes
import java.applet.*;
import java.awt.*;

// Note: Insertion of HTM:L file as comments

/* <applet code = AppletDemo width=400 height=500>
</applet>*/

// Java Program

// Class extending Applet
public class MyApplet extends Applet {
public void init()
{
setBackground(Color.black);
setForeground(Color.yellow);
}
public void paint(Graphics g)
{
g.drawString("Welcome to Applets", 50, 50);
}
}
```

Output:-

```
chandan@kumar:~/Programming/JavaProgramming/java_packages/Introduction_packages/src$ javac MyApplet.java
Note: MyApplet.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
chandan@kumar:~/Programming/JavaProgramming/java_packages/Introduction_packages/src$ appletviewer MyApplet.java
load: class AppletDemo not found.
java.lang.ClassNotFoundException: AppletDemo
        at sun.applet.AppletClassLoader.findClass(AppletClassLoader.java:220)
        at java.lang.ClassLoader.loadClass(ClassLoader.java:418)
        at sun.applet.AppletClassLoader.loadClass(AppletClassLoader.java:153)
        at java.lang.ClassLoader.loadClass(ClassLoader.java:351)
        at sun.applet.AppletClassLoader.loadCode(AppletClassLoader.java:637)
        at sun.applet.AppletPanel.createApplet(AppletPanel.java:801)
        at sun.applet.App
        at sun.applet.App
        at java.lang.Thre
```



Practical No:- 25

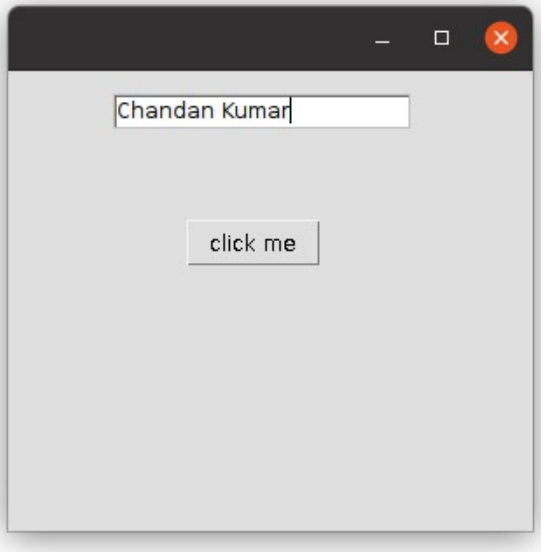
Event Handling:- Changing the state of an object is known as an event. For example, click on a button,dragging a mouse etc. The java.awt.event package provides many event classes and Listener interfaces for event handling.

Code->

```
import java.awt.*;
import java.awt.event.*;
class AEvent extends Frame implements ActionListener{
TextField tf;
AEvent(){
//create components
tf=new TextField();
tf.setBounds(60,50,170,20);
Button b=new Button("click me");
b.setBounds(100,120,80,30);
//register listener
b.addActionListener(this);//passing current instance
//add components and set size, layout and visibility
add(b);add(tf);
setSize(300,300);
setLayout(null);
setVisible(true);
}
public void actionPerformed(ActionEvent e){
tf.setText("Welcome");
}
public static void main(String args[]){
new AEvent();
}
}
```

Output:-

```
chandan@kumar:~/Programming/JavaProgramming/MyPackges$ cd "/home/chandan/P
```



Practical No:- 26

Reading and writing from a particular file.

Writing in a file:-

Code->

```
import java.io.FileWriter;
import java.io.IOException;
public class Write {
public static void main(String[] args) {
try{
FileWriter wr = new FileWriter("write.txt");
wr.write("java is a programing language");
wr.close();
System.out.println("Successfully wrote in file");
}
catch(IOException e){
System.out.println(e.toString());
}
}
}
```

Output:-

```
chandan@kumar:~/Programming/JavaProgramming/MyPackges/src$ javac Write.java
chandan@kumar:~/Programming/JavaProgramming/MyPackges/src$ java Write
Successfully wrote in file
```

src > write.txt

```
1   java is a programing language
```

Reading file:-

Code->

```
import java.io.File;
import java.io.FileReader;
// import java.io.FileWriter;
import java.io.IOException;
import java.util.Scanner;

public class Read {
public static void main(String[] args) {
try{
File obj = new File("write.txt");
Scanner sc = new Scanner(obj);
while(sc.hasNextLine()){
String data = sc.nextLine();
System.out.println(data);
}
sc.close();
}
catch(Exception e){
System.out.println(e.toString());
}
}

}
```

Output:-

```
chandan@kumar:~/Programming/JavaProgramming/MyPackges/src$ javac Read.java
chandan@kumar:~/Programming/JavaProgramming/MyPackges/src$ java Read
java is a programing language
```

Practical No:- 28

String class and its methods

Code->

```
// Java code to illustrate different constructors and methods
// String class.

import java.io.*;
import java.util.*;

public class StringMethod {
public static void main(String[] args) {
String s = "GeeksforGeeks";
// or String s= new String ("GeeksforGeeks");

// Returns the number of characters in the String.
System.out.println("String length = " + s.length());

// Returns the character at ith index.
System.out.println("Character at 3rd position = "
+ s.charAt(3));

// Return the substring from the ith index character
// to end of string
System.out.println("Substring " + s.substring(3));

// Returns the substring from i to j-1 index.
System.out.println("Substring = " + s.substring(2, 5));

// Concatenates string2 to the end of string1.
String s1 = "Geeks";
String s2 = "forGeeks";
System.out.println("Concatenated string = " +
s1.concat(s2));

// Returns the index within the string
// of the first occurrence of the specified string.
String s4 = "Learn Share Learn";
System.out.println("Index of Share " +
s4.indexOf("Share"));

// Returns the index within the string of the
// first occurrence of the specified string,
// starting at the specified index.
System.out.println("Index of a = " +
s4.indexOf('a', 3));

// Checking equality of Strings
Boolean out = "Geeks".equals("geeks");
System.out.println("Checking Equality " + out);
out = "Geeks".equals("Geeks");
System.out.println("Checking Equality " + out);

out = "Geeks".equalsIgnoreCase("gEeks ");
System.out.println("Checking Equality " + out);
```

```
// If ASCII difference is zero then the two strings are similar
int out1 = s1.compareTo(s2);
System.out.println("the difference between ASCII value is=" + out1);
// Converting cases
String word1 = "GeeKyMe";
System.out.println("Changing to lower Case " +
word1.toLowerCase());

// Converting cases
String word2 = "GeekyME";
System.out.println("Changing to UPPER Case " +
word2.toUpperCase());

// Trimming the word
String word4 = " Learn Share Learn ";
System.out.println("Trim the word " + word4.trim());

// Replacing characters
String str1 = "feeksforfeeks";
System.out.println("Original String " + str1);
String str2 = "feeksforfeeks".replace('f', 'g');
System.out.println("Replaced f with g -> " + str2);
}
}
```

Output:-

```
chandan@kumar:~/Programming/JavaProgramming/MyPackges/src$ javac StringMethod.java
chandan@kumar:~/Programming/JavaProgramming/MyPackges/src$ java StringMethod
String length = 13
Character at 3rd position = k
Substring ksforGeeks
Substring = eks
Concatenated string = GeeksforGeeks
Index of Share 6
Index of a = 8
Checking Equality false
Checking Equality true
Checking Equality false
the difference between ASCII value is=-31
Changing to lower Case geekyme
Changing to UPPER Case GEEKYME
Trim the word Learn Share Learn
Original String feeksforfeeks
Replaced f with g -> geeksgorgeeks
```

Practical No:- 29

StringBuffer class and its methods

Code->

```
import java.io.*;
import java.lang.StringBuffer;

public class Stringbuffer {
public static void main(String args[]) {
StringBuffer sb = new StringBuffer("Hello ");
System.out.println(sb.capacity());
sb.append("Java"); // now original string is changed
System.out.println(sb);
sb.insert(1, "Java");
// Now original string is changed
System.out.println(sb);
sb.replace(1, 3, "Java");
System.out.println(sb);

sb.delete(1, 3);
System.out.println(sb);
sb.reverse();
System.out.println(sb);

System.out.println(sb.capacity()); // now 22
sb.append("java is my favourite language");
System.out.println(sb.capacity());
// Now (22*2)+2=46 i.e (oldcapacity*2)+2

}
}
```

Output:-

```
chandan@kumar:~/Programming/JavaProgramming/MyPackges/src$ javac Stringbuffer.java
chandan@kumar:~/Programming/JavaProgramming/MyPackges/src$ java Stringbuffer
22
Hello Java
HJavaello Java
HJavavaello Java
Hvavaello Java
avaJ olleavavH
22
46
```