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

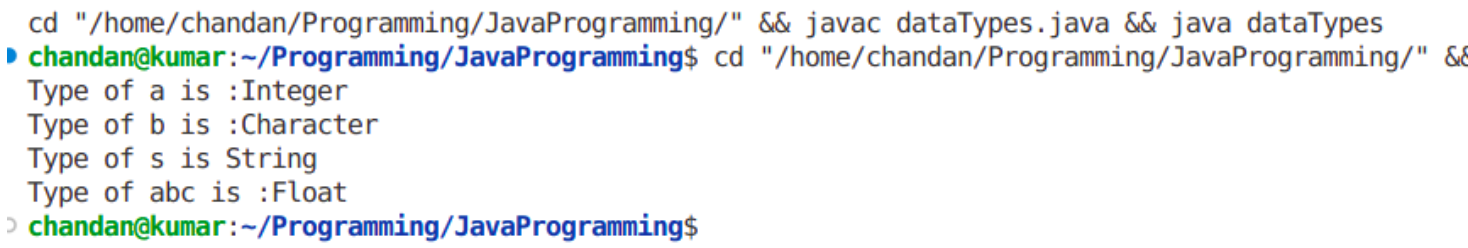
Java Non-Primitive Data Types

- 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());


    }
}
```

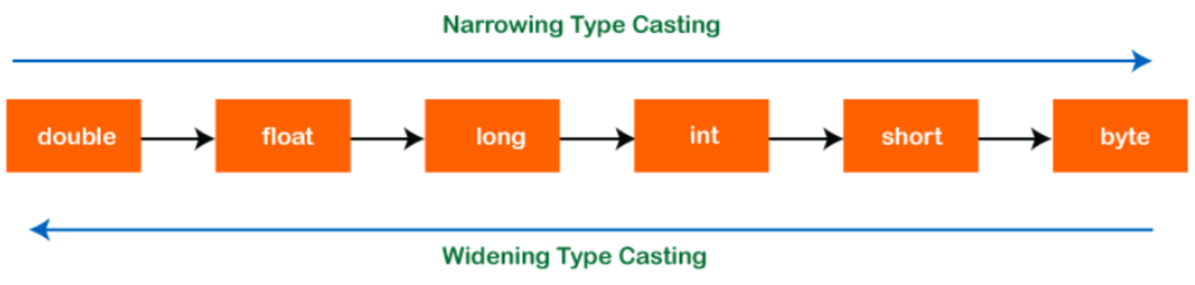
Output:



```
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.

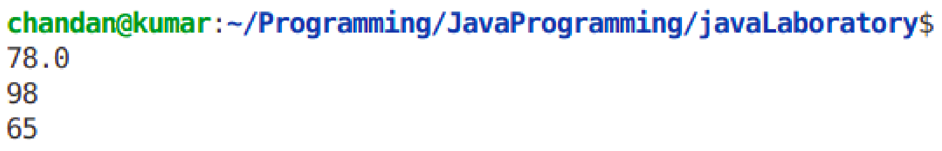


Type Casting in Java

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

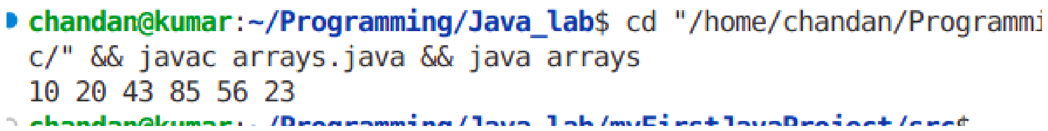
Arrays:-In Java, all arrays are dynamically allocated. (discussed below)

- 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
chandan@kumar:~/Programming/Java_lab/myFirstJavaProject/src$
```

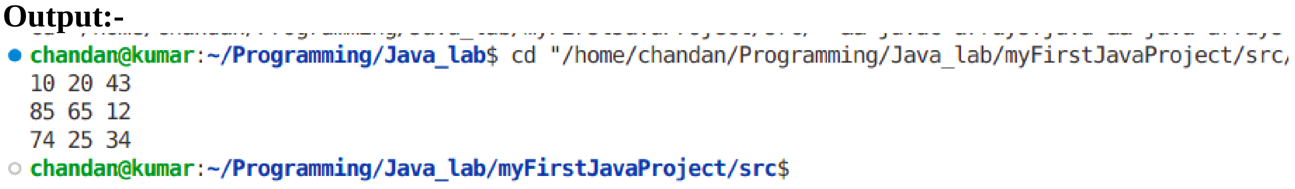
2-D array

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:-



```
● chandan@kumar:~/Programming/Java_lab$ cd "/home/chandan/Programming/Java_lab/myFirstJavaProject/src,
  10 20 43
  85 65 12
  74 25 34
○ chandan@kumar:~/Programming/Java_lab/myFirstJavaProject/src$
```

Practical No:-4

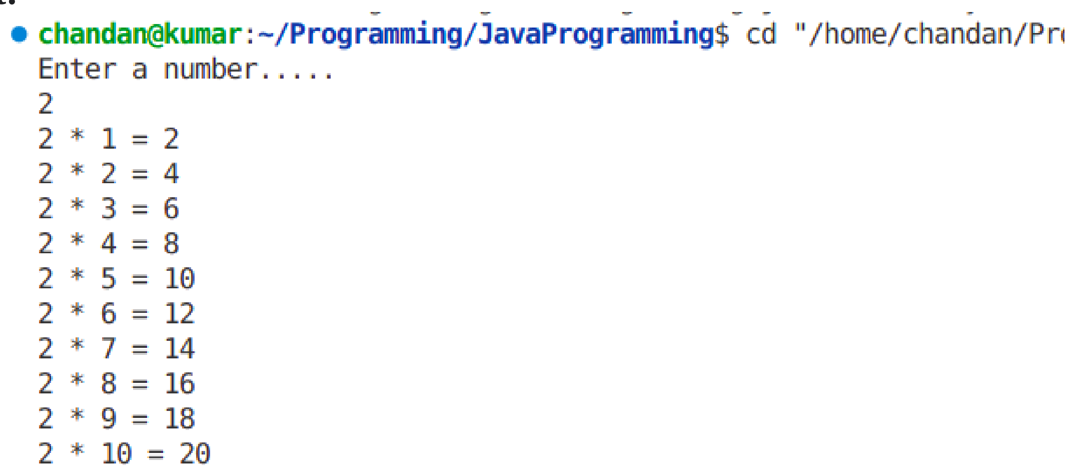
Various control structures

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:-



```
● 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
```

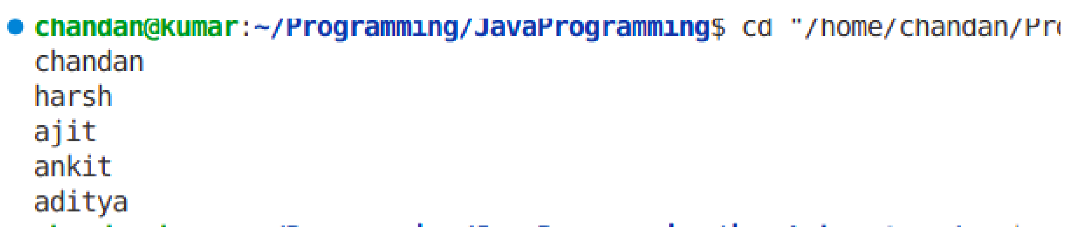
2.for each loop:- In computer programming, foreach loop (or for each loop) is a control flow statement for traversing items in a collection. foreach is usually used in place of a standard for loop statement.

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
```

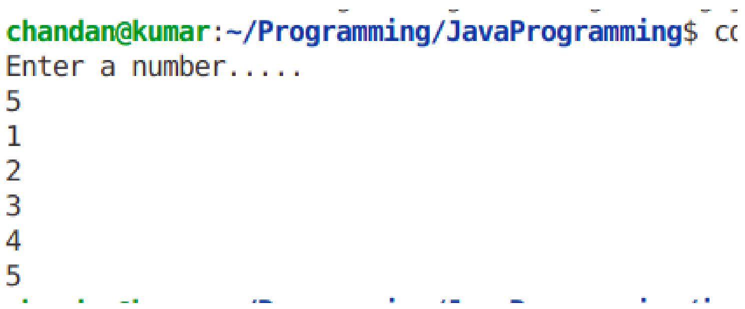
while Loop:- The while loop is used to repeat a section of code an unknown number of times until a specific condition is met

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$ cc
Enter a number.....
5
1
2
3
4
5
```

Practical No- 5

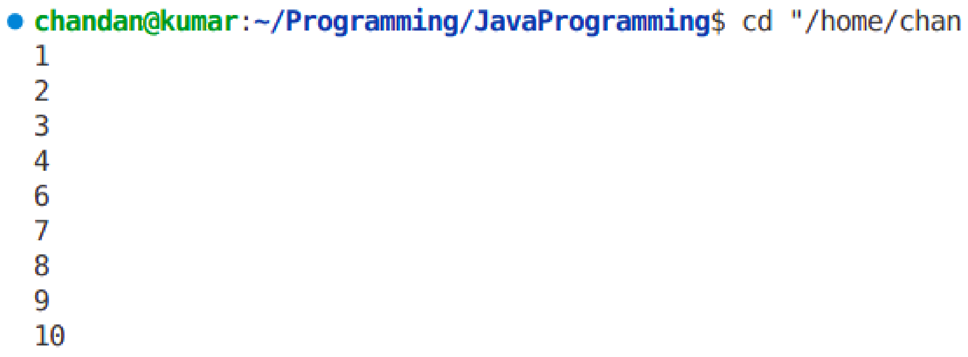
Various decision structures

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:-



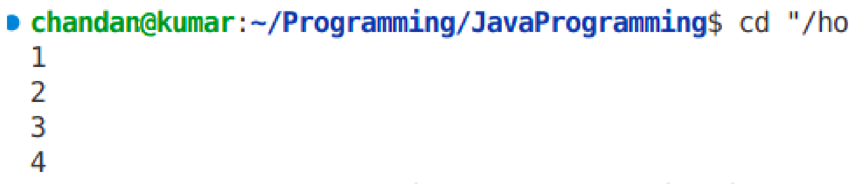
```
● chandan@kumar:~/Programming/JavaProgramming$ cd "/home/chan
  1
  2
  3
  4
  6
  7
  8
  9
  10
```

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:-



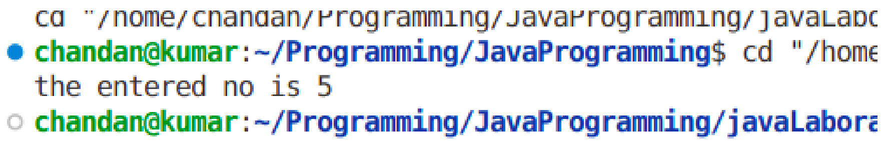
```
● chandan@kumar:~/Programming/JavaProgramming$ cd "/ho
  1
  2
  3
  4
```

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:-



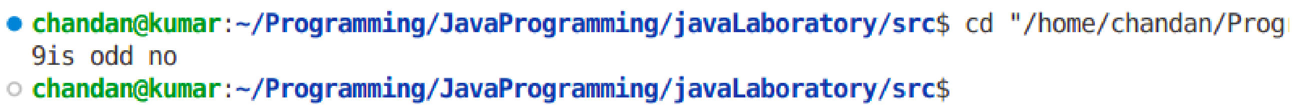
```
cd "/home/chandan/Programming/JavaProgramming/javaLabo
● chandan@kumar:~/Programming/JavaProgramming$ cd "/home
  the entered no is 5
○ chandan@kumar:~/Programming/JavaProgramming/javaLabora
```

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.

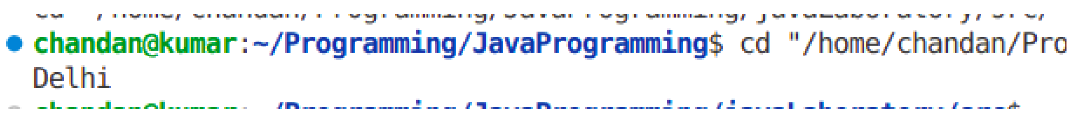
Code:-

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:-



```
● chandan@kumar:~/Programming/JavaProgramming$ cd "/home/chandan/Pro
  Delhi
```

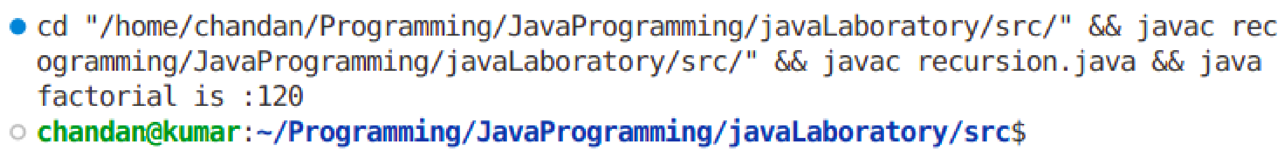
Practical No:-6

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:-



```
● cd "/home/chandan/Programming/JavaProgramming/javaLaboratory/src/" && javac rec
  ogramming/JavaProgramming/javaLaboratory/src/" && javac recursion.java && java
  factorial is :120
○ chandan@kumar:~/Programming/JavaProgramming/javaLaboratory/src$
```

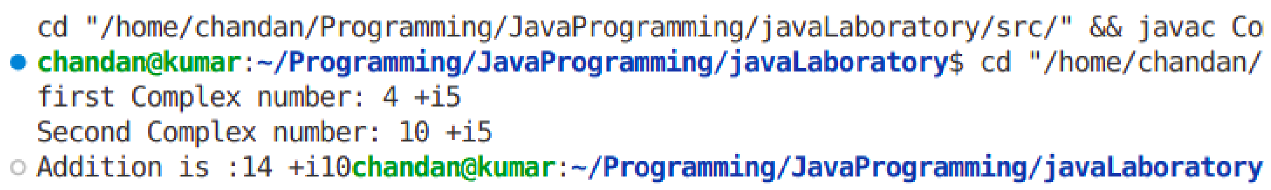
Practical No:-7

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
```

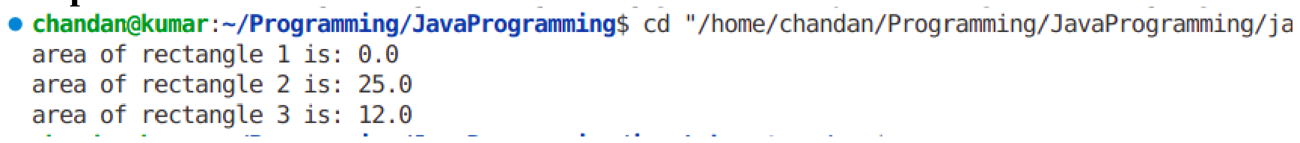
Practical No:- 8

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

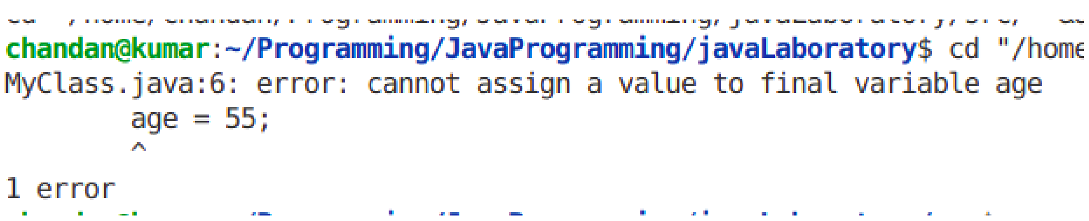
Various access control and usage of static, final and finalize ( )

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:->



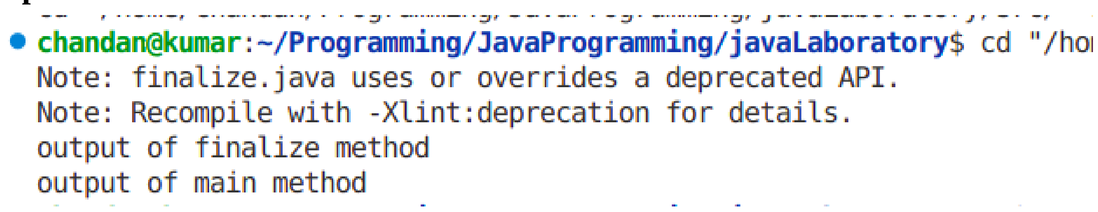
```
chandan@kumar:~/Programming/JavaProgramming/javaLaboratory$ cd "/home
MyClass.java:6: error: cannot assign a value to final variable age
        age = 55;
        ^
1 error
```

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->



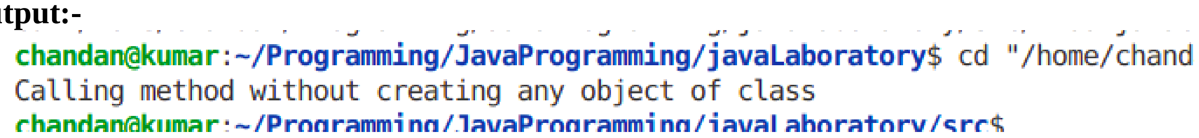
```
● chandan@kumar:~/Programming/JavaProgramming/javaLaboratory$ cd "/ho
  Note: finalize.java uses or overrides a deprecated API.
  Note: Recompile with -Xlint:deprecation for details.
  output of finalize method
  output of main method
```

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$
```

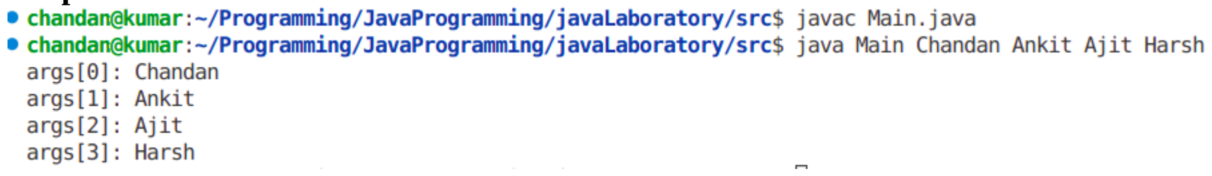
Practical No :- 10

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
```

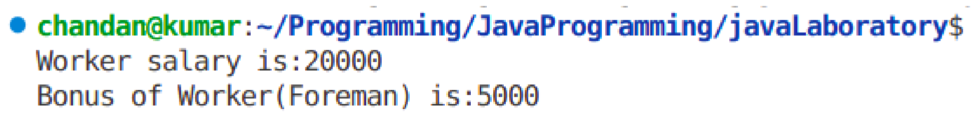
Practical No :- 11

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
```

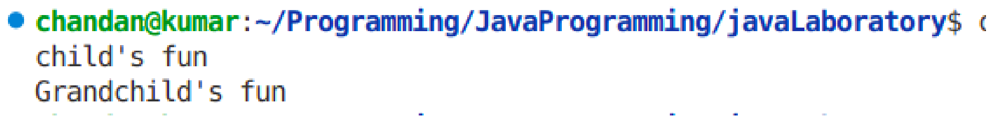
Practical No :- 12

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$
child's fun
Grandchild's fun
```

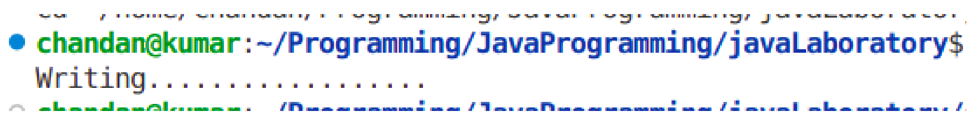
Practical No :-13

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:-



```
● chandan@kumar:~/Programming/JavaProgramming/javaLaboratory$
  Writing..................
```

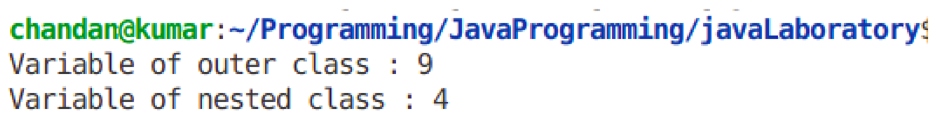
Practical No :-14

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
```

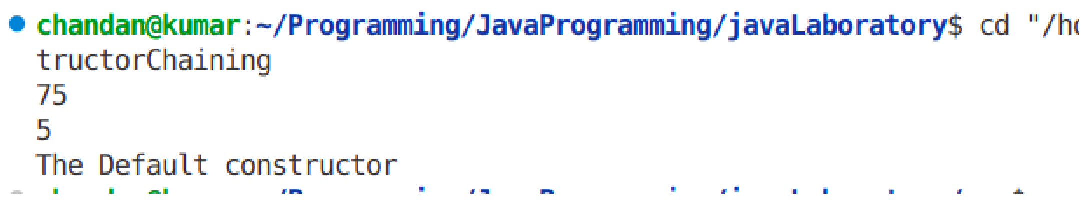
Practical No:- 15

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 "/h
  tructorChaining
  75
  5
  The Default constructor
```