Module 1 – core java

❖ Introduction to java

- Theory:
  1.histrory of java
  Ans:
  java history is interesting to know. The history of java start from green team. Java team member (also known as green team).

  initiated a revolutionary task to develop a language for digital devices such as set-top boxes, television etc.

  currently java is used in internet programming , mobile devices , games ,e-business solutions etc.

  james gosling , mike sheridan and partick naughton initiated the java language project in june 1991.

  originally developed by james gosling at sun miscrosystems (which is now a subsidiary of oracle corporation) and released in 1995.

2. Features of java (platform independent, object-oriented, etc.)

Ans:

following are the features of java:

i)simple:

According to sun, java language is simple because syntax is based on c++.

Removed many confusing or rarely used features. Like explicit pointers , operator overloading etc.

ii)object oriented:

Object oriented means we organize our software as a combination of different types of objects that incorporates both data and behaviour.

basic concept of oops are:

-object , class , inheritance polymorphism , abstraction , encapsulation.

Iii)platform independent:

a platform is the hardware or software environment in which a program runs.

3. Understanding jvm, jre, and jdk

Ans:

jvm(java virtual machine):-

Jvm is an abstract machine that enables your computer to run a java program.

When you run the java program , java complier first complies your java code to bytecode. Then the jvm translates bytecode into native machine code(set of instructions that a computer's cpu execute directly).

java is a platform independent language.

it's because when you write java code. It's ultimately written for jvm but not your physical machine(computer).

since , jvm execute the java bytecode which is platform independent.

Jre(java runtime environment) :-

jre is a software package that provides java class libraries along with java virtual machine(jvm) and other components to run applications written in java programming. Jre is the superset of jvm.

it is the implementation of jvm. It physically exists.

it contains a set of libraries + other files that jvm uses at runtime.

the implementation of jvm is also actively
released by other companies besides Sun microsytesm.

Jdk(java development kit):-

jdk is software development kit to develop application in java.

jdk is an implementation of any of the below given java platform released by oracle corporation:

-standard edition java platform
-enterprise edition java platform
-micro edition java platform

the jdk contains a private java virtual machine(jvm) and a few other resource such as an interpreter /loader(java), a complier(javac) , an archiver(jar), a documentation generator(javadoc) etc. To complete the development of a java application.

4. Java program structure (packages, classes, methods)

Ans:

package:-

A package is used to group related classes and interface.

Ex:-package mypackage;

Used to import classes from other packages(like java libraries).

Ex:-import java.util.scanner;

Class:-

every java program must have at least one **class**.the class is the blueprint for creating objects.

Ex:- public class abc {

// variables, methods, main() }

Main method:-

This is the entry point of every java application. The jvm starts execution from the main() method.

```
ex:-public static void main(string[] args) {
   system.out.println("hello, world!");
}
```

Methods:-

A method is a block of code that performs a specific task.

```
ex:- public static void greet() {
  system.out.println("welcome to java!");
}
```

- Lab exercise:

5. Install jdk and set up environment variables.

Ans:

step-1 : install jdk

step-2 : extract all file

step-3 : open windows

Step-4 : type edit the system environment variables in the prompt open it

Step-5 : then click environment variables

Step-6 : system variable go path then click edit button

Step-7 : click new button then

Step-8: write the path of we were install files

step-9 :all process done then click ok button

6. Write a simple "hello world" java program.

ans:

```
public class first {
Public static void main(string[] args) //main method of program
{
      system.out.print("hello world");//hello world
}
}
```

7. Compile and run the program using command-line tools (javac, java).

Ans:

Step-1 : open the command prompt

Step-2 : c:\users\mahek\eclipse-workspace\javaprac\src\javaprac>javac first.java

Step-3 : c:\users\mahek\eclipse-workspace\javaprac\src\javaprac>java first.java

❖ Data types, variables, and operators

- Theory

8. Primitive data types in java (int, float, char, etc.)
Ans:

following are the primitive data types in java:-
i)int:-
    it is store value of integer type.
    size: 32 bits
    range: -2,147,483,648 to 2,147,483,647.
    ex: int a=10

ii)float:-

it is store value of float type.

size: 32 bits

range:1.4e-45 to 3.4028235e38

ex: float f=10.10

iii)char:-

it is store one character of the string.

size: 16 bits

ex: char a='b';

iv)byte:-

It is useful when memory efficiency is crucial , as it occupies less space than other integer types like int or short

Size:8 bits (1 byte)

Range: -128 to 127.

Ex:byte my=100

v)short:-

It is used to store whole numbers within a specific range

Size: 16 bits

Range:-32,768 to 32,767

Ex: short age=20

vi)long:-

it is used to store whole numbers.

size: 64 bits

range: -9,223,372,036,854,775,808 to

9,223,372,036,854,775,807

ex: long bignumber =9223372036854775807l

vii)double:-

It is used to store double-precision 64-bit floating-point numbers

Size:64 bits

Ex: double pi = 3.14159

viii)boolean:-

It is used for the boolean type data store.

Size:8 bits

Ex: boolean light= true

9.variable declaration and initialization.

ans:

variable declaration:

In java, variables must be **declared** before they are used.

ex: int age;

Float price;

Char grade;

variable initialization:

You are declaring the variable **and** giving it an **initial value**.

ex: int age = 25;

float price = 99.99f;

char grade = 'a'

10. Operators: arithmetic, relational, logical , assignment, unary, and bitwise

ans:

operator are special symbols or keywords used to perform operations on variables and values.

Following are the operator in java:-

(i)arithmetic operator:

It is used to perform basic mathematical operations.

Ex: addition(a+b),

substraction(a-b),

multiplication(a*b),

division(a/b),

module(a%b).

(ii)relational(comparison) operator:

it is used to compare two values.

Ex: equal to(a==b),

not equal to(a!=b),

greater than(a>b),

less than(a<b),

greater than or equal to(a>=b),

less than or equal to (a<=b).

(iii)logical operator:

It is used to combine multiple boolean expressions.

Ex: logical and (a>10 && b<5) ,

logical not (!(a>10)).

(iv)assignment operators:

It is used to assign values to variables.

Ex: assign(a=b),

add and assign (a += b),

substract and assign (a += b),

multiply and assign(a*=b),

division and assign(a/=b),

modulue and assign(a%=b).

(v) unary operators:

It is operate on a single operand.

ex: unary plus(positive number +a),

unary minus(negative -a),

increment(++a,a++),

decrement(--a,a--),

logical not(!flag).

(vi)bitwise operator:

It is operate on bits

And perform bit-by-bit operations.

Ex: bitwise and (a&b),

bitwise xor (a^b),

bitwise complement(~a),

left shift(a<<2),

right shift(a>>2),

unsigned right shift(a>>>2).

11.type conversion and type casting

Ans:

Type conversion and type casting are processes used to convert a variable from one data type to another.

(i)) type conversion (implicit casting):

in this type conversion compiler can automatically convert one datatype to another.

ex: int a = 10;

    double b = a; // int to double

(ii)type casting(explicit casting):

in this type casting compiler can not automatically convert one datatype to another. If you want cast so must be manually cast.

ex: double a = 10.5;

    int b = (int) a

- Lab exercise

12. Write a program to demonstrate the use of different data types.

Ans:

```
public static void main(string[] args) {
        // todo auto-generated method stub
        int a=10;
        float per=(float) 89.90;
        char c='a';
        Boolean light=true;
        system.out.println("integer="+a+"\nfloat="+p
        er+"\ncharacter="+c+"\nboolean="+light);
}
```

13. Create a calculator using arithmetic and relational operators

Ans:

```
        public static void main(string[] args) {
        scanner sc = new scanner(system.in);
                system.out.print("enter value of a=");
                int a = sc.nextint();
                system.out.print("enter value of b=");
                int b = sc.nextint();
                System.out.println("enter 1 for
                addition");
                System.out.println("enter 2 for
                subtraction");
                System.out.println("enter 3 for
                multiplication");
                system.out.println("enter 4 for division");
```

```
system.out.println("enter 5 for module");
system.err.print("enter the number=");
int c = sc.nextint();

if (c == 1) {
System.out.println("addition of " + a + "
and " + b + " is= " + (a + b));
} else if (c == 2) {
System.out.println("substration of " + a +
" and " + b + " is= " + (a - b));
}
else if (c == 3) {
System.out.println("multiplication of " + a
+ " and " + b + " is= " + (a * b));
}
else if(c==4) {
System.out.println("division of " + a + "
and " + b + " is= " + (a / b));
}
else if(c==5) {
```

```
            System.out.println("modulue of " + a + "
            and " + b + " is= " + (a % b));
            }
            else {
                system.out.println("invalid input");
            }
        }
```
14. Demonstrate type casting (explicit and implicit).

ans:

```
        public static void main(string[] args) {
        Int a = 10;
        double b = a;
        system.out.println("type conversion");
       system.out.println("before="+a+"\nafter="+b);
      //int to double

        system.out.println("type casting");
        double c = 10.5;
        int d = (int)c;
        system.out.println("before="+c+"\nafter="+d);
        //double to int

        }
```

❖ Control flow statements

- theory:

15. If-else statements

Ans:

Executes one block if the condition is true, another if it's false.

16. Switch case statements

ans:

Switch case statements in java provide a way to execute different code blocks based on the value of a single variable or expression.

17. Loops (for, while, do-while)

Ans:

A loop in java is a way to repeat a block of code multiple times as long as a specified condition is true.

following are the types of loop:

(i)for loop:

For loop is used when the number of iterations is known.

(ii)while loop:

It is used when the number of iterations is unknown and depends on a condition.

(iii)do-while loop:

Similar to while, but the block runs at least once before the condition is checked.

18. Break and continue keywords

ans:

break keywords:

Break exits the entire loop or switch statement.

continue keywords:

Continue skips the current iteration and moves to the next.

- Lab exercise:

19. Write a program to find if a number is even or odd using an if-else statement

Ans:

```
public static void main(string[] args) {
            scanner sc = new scanner(system.in);
            system.out.print("enter number=");
            int n = sc.nextint();

            if(n%2==0)
                  System.out.println(n+" number is
                  even");
            else
                  System.out.println(n+" number is
                  odd");
            }
```

20. Implement a simple menu-driven program using a switch-case.
Ans:

21. Write a program to display the fibonacci series using a loop.
Ans:

```
      public static void main(string[] args) {
      scanner sc = new scanner(system.in);
            system.out.print("enter number=");
            int n = sc.nextint();


            int a=0,b=1,c=0;
```

```
        for(int i=0;i<n;i++)
        {
                system.out.println(c);
                a=b;
                b=c;
                c=a+b;

        }

}
```

22. Defining a class and object in java

ans:

class:

Collation of variable, datatype , constructor,

Methods and objects.

Object:

instance of class is called object.

23. Constructors and overloading

ans:

constructors:

Class name and constructors name same is called constructors.

overloading:

One class more than one constructor name same and parameter different is called overloading.

24. Object creation, accessing members of the class

Ans:

object creation:

To create an object, you use the new keyword along with the constructor of the class.

accessing members of the class:

You access variables and methods of a class using the object with the dot (.) Operator.

25. This keyword

ans:

This keyword used for the access child class method.

• lab exercise:

26. Create a class student with attributes (name, age) and a method to display the details.

Ans:

```
public class student {


Public void student_detail(string name,int age)
{
        System.out.println("student
        name="+name+
                    "\nstudent age="+age);
        }
        public static void main(string[] args) {
        student s=new student();
        s.student_detail("abc", 20);
        }
}
```

27. Create multiple constructors in a class and demonstrate constructor overloading.

Ans:

```
public class student {
public student() {
        system.out.println("student details");
}
public  student(string name,int age) {
        system.out.println("student name="+name+
                    "\nstudent age="+age);
}
public static void main(string[] args) {
        student s=new student();
        student s1=new student("abc", 20);


}
}
```

28. Implement a simple class with getters and setters for encapsulation.

Ans:

pojo.java

```
public class pojo {


private int id;

private string email;

private string pass;


public int getid() {

	return id;

}

public void setid(int id) {

	this.id = id;

}

public string getemail() {

	return email;

}

public void setemail(string email) {

	this.email = email;

}

public string getpass() {

	return pass;
}
public void setpass(string pass) {
        this.pass = pass;
}
}
Methods.java
Public static void main(string[] args) {
                pojo p=new pojo();
                p.setid(1);
                p.setemail("mahek@gmail.com");
                p.setpass("mahek");


                        system.out.println(p.getid()+"
        "+p.getemail()+" "+p.getpass());
}
```

❖ Methods in java

- Theory:

29. Defining methods.

Ans:

collection of code that code we will use for particular task.

Types of method:

(i)default method

(ii)user defined method

30. Method parameters and return types.

Ans:

method parameter:

Parameters are like placeholders for the values you pass to a method.

return types:

The return type tells what type of value the method will return. Like int, string, boolean, void.

31. Method overloading.

Ans:

One class more than one method name same and parameter different is called method overloading.

32. Static methods and variables.

Ans:

static methods:

A static method can be called without creating an object. It can only access static variables directly. It belongs to the class, not to the object.

static variables:

A static variable is shared by all objects of a class. It is created once in memory. Accessed using the class name.

- lab exercise:

33. Write a program to find the maximum of three numbers using a method.

Ans:

```
Import java.util.scanner;
Public class maximum {


    public void maxi_num() {
        scanner sc=new scanner(system.in);
        system.out.print("enter number1=");
        int a=sc.nextint();
        system.out.print("enter number2=");
        int b=sc.nextint();
        system.out.print("enter number3=");
        int c=sc.nextint();


        if(a>b && a>c)
            System.out.println(a+" is greater
            number");
        else if(b>a && b>c)
            System.out.println(b+" is greater
            number");
        else
            System.out.println(c+" is greater
            number");


    }
    public static void main(string[] args) {
        maximum m=new maximum();
        m.maxi_num();
    }
}
```

34. Implement method overloading by creating methods for different data types.

Ans:

```
        public class polymorphism extends emp{



        public static void main(string[] args) {
                polymorphism p=new polymorphism();
                p.emp_details("abc");
                p.emp_details(20, 50000);
                p.emp_details("done","medical store");
        }

}
Class emp{
        public void emp_details(string name) {
                System.out.println("employee
                name="+name);
        }
        public void emp_details(int attendance,int salary) {
                System.out.println("employee
                attendace="+attendance+"\nsalary="+salary);
        }
        Public void emp_details(string task,string
        projectname) {
                System.out.println("task="+task+"\nproject
                name="+projectname);
        }
}
```

35. Create a class with static variables and methods to demonstrate their use.

Ans:

```
        public class statice {
        static int count=0;

        public static void static_method() {
                count++;
        }

        public static void main(string[] args) {
                // todo auto-generated method stub
                statice.static_method();
                system.out.println("counter="+statice.count)
```

```
            }
    }
```

❖. Object-oriented programming (oops) concepts.

- theory:

36.basics of oop: encapsulation, inheritance, polymorphism, abstraction.

Ans:

encapsulation:

Encapsulation is used for data wraping means data wrap in pojo(plain old java object) class.

inheritance:

Child class can use the functionality of parent class using extends keywords is called inheritance.

types of inheritance:

(i)simple inheritance

(ii)multiple inheritance

(iii)multilevel inheritance

(iv)hierarchical inheritance

polymorphism:

One interface and multiple implementation.

can achieve two types:

(i)method overloading

(ii)method overriding

abstraction:

Combination of abstract class and abstract method is called abstraction class. Abstraction can store sensitive data and data hide.

37. Inheritance: single, multilevel, hierarchical.

ans:

(i)single inheritance:

One child and one parent class is called single inheritance.

(ii)multilevel inheritance:

One and more child class and one and more parent class is called Multilevel inheritance. There are work as chain wise.

(iii) hierarchical inheritance:

one parent class and child more than one class is called Hierarchical inheritance.

38.method overriding and dynamic method dispatch.

Ans:

- lab exercise:

39. Write a program demonstrating single inheritance.

Ans:

```
public class inheritance extends cal {
public void add() {
        int a=6;
        int b=4;
        int c=a+b;
        system.out.println("addition is="+c);
}

public static void main(string[] args) {
inheritance i=new inheritance();
i.add();
```

```
        i.mul();
        }
}
Class cal{
        public void mul() {

                int a=6;
                int b=4;
                int c=a*b;

            system.out.println("mutliplication is="+c);

        }
}
```

40. Create a class hierarchy and demonstrate multilevel inheritance.

Ans:

```
        public class inheritance extends xyz {
        public void display() {
            system.out.println("hello");
```

```
	}

	public static void main(string[] args) {
		inheritance a = new inheritance();
		a.display();
		a.display1();
	}
}

Class xyz {
	public void display1() {
		system.out.println("hello1");
	}

}

Class abc1 extends xyz {
	public void display2() {
		system.out.println("hello2");
	}
}
```

41. Implement method overriding to show polymorphism in action.

Ans:

```
	public class polymorphism extends customer {
		public static void main(string[] args) {
		customer c=new customer();
		c.display();
	}
	}
	Class customer extends cust1{

		Public void customer_details(string
		custname) {
			System.out.println("customer
			name="+custname);
		}
		public void display() {
			this.customer_details("mahek");
			super.customer_details("sindhav");
```

```
        }

    }
    Class cust1{
        Public void customer_details(string
        custnam) {
                System.out.println("customer
                surname="+custnam);
        }

    }
```

❖Constructors and destructors

- theory:

42. Constructor types (default, parameterized).

Ans:

(i)default constructor:

Default constructor with no parameters. This type constructor java automatically provides one if you don’t create any constructor yourself.

(ii) parameterized constructor:

A parameterized constructor is a constructor that accepts arguments (parameters) to initialize an object with specific values.

43. Constructor overloading

ans:

Constructor overloading means having multiple constructors in the same class, but with different parameters.

- lab exercise:

44. Write a program to create and initialize an object using a parameterized constructor.

ans:

```
public class student {
        public student() {

    system.out.println("student details");
                }
    public  student(string name,int age) {
```

```
                        system.out.println("student
                        name="+name+
                             "\nstudent age="+age);
                                        }
              public static void main(string[] args) {
                                            student s=new
                                       student();
                                            student s1=new
                             student("abc", 20);
                                        }
               }
```

45. Demonstrate constructor overloading by passing different types of parameters.

Ans:

```
     public class student {
                              public student() {

               system.out.println("student details");
                                        }
              public  student(string name,int age) {
```

```
          system.out.println("student name="+name+
                             "\nstudent age="+age);
        }
       public student(string task,string projectname) {

            system.out.println("student task="+task+
                  "\nproject name="+projectname);
                                   }
              public static void main(string[] args) {
               student s=new student();
               student s1=new student("abc", 20);
                student s2=new student("done","medical
store");
                                   }
}
```

❖. Interfaces and abstract classes

- theory:

46. Abstract classes and methods

Ans:

abstract class:

an abstract class is a class that can’t be used to create objects directly. Abstract class is used as a base class for other classes.

Abstract methods:

an abstract method Has no body (just a name and parentheses). It is ends with a semicolon.it must be overridden (written) in the child class.

47. Interfaces: multiple inheritance in java.

ans:

using interfaces, we can achieve multiple inheritance in java. In an interface, we don't need to use the abstract keyword, because all methods are abstract by default.

- lab exercise:

48. Create an abstract class and implement its methods in a subclass.

Ans:

```
public class abst extends userdata{
    public static void main(string[] args) {
    abst a=new abst();
    a.sen1();
    a.sen2();
    a.usedata();
}
@override
public void sen1() {
       system.out.println("sensitive data1");
     }

@override
 public void sen2() {
   system.out.println("sensitive data2");
   }
  abstract class userdata{
    int a=10;
         Public void usedata() {

system.out.println("name and address");
```

```
                                        }
                        abstract public void sen1();
                        abstract public void sen2();
                             }
```

49. Write a program that implements multiple interfaces in a single class.

Ans:

```
        public class interfacecon extends hello
    implements test,test1{
    Public static void main(string[] args) {
                        // todo auto-generated
    method stub
                        interfacecon i=new
    interfacecon();
                        i.display();
                        i.display1();
                        i.display2();
                        i.display3();
                        }
Ublic void display() {
                             // **todo** auto-generated
method stub
                             system.*out*.println("hii
display");

                        }



                  @override
          public void display2() {
                             // **todo** auto-generated
method stub
                             system.*out*.println("hii
display2");

                        }

@override
```

```
              Public void display1() {

                 system.out.println("hii display1");

                                             }
              }
```

50. Implement an interface for a real-world example, such as a payment gateway.

Ans:

```
     public class payment extends upipayment {

              public static void main(string[] args) {
                                             payment p = new
payment();
                                             p.display();

                                             }

}

Interface paymentgateway {
        void pay(double amount); // abstract method
}

Class creditpayment implements paymentgateway {
                            @override
                public void pay(double amount) {
                                        system.out.println("paid
₹" + amount + " using creditcard.");
                                    }

}

Class upipayment extends creditpayment implements
paymentgateway {

                            @override
                public void pay(double amount) {
                                        system.out.println("paid
₹" + amount + " using upi.");
                                    }
                                public void display() {
                                        super.pay(2000);
                                        this.pay(500);

                                    }

}
```

- ❖ Exception handling

51. Try, catch, finally

Ans:

try block:

a try block is used to wrap code that might throw an exception during execution. It allows you to handle errors gracefully using catch, finally, or both.

catch block:

a catch block is used to handle errors that happen inside a try block.
If something goes wrong in the try block, the catch block runs and shows a message or takes action.

finally block:

a finally block is used to write code that should always run — no matter if an exception happens or not.

- lab exercise:

52. Write a program to demonstrate exception handling using try-catch-finally.

Ans:

```
                                                public static void
main(string[] args) {
                                                // **todo** auto-generated
method stub

                          system.***out***.println("java");

                         system.***out***.println("java1");

                         system.***out***.println("java2");
                                                int a=5;
                                                int b=0;

                                                try {
                                                      int c=a/b;
```

```
                    system.out.println(c);
                                    } catch (exception e) {
                                        // todo: handle
exception

        system.out.println("cant divided by zero");
                                    }finally {

                system.out.println("all ok");
                                    }

                system.out.println("java3");

                system.out.println("java4");


                                }
```

❖Collections framework

53. Introduction to collections framework

Ans:

the java collection framework is a set of classes and interfaces used to store, manage, and process groups of objects (data).

It provides ready-made data structures like list, set, map, queue, etc.

54.list, set, map interfaces
Ans:

list :

list is an interface in java that represents an ordered collection (sequence) of elements. You can store duplicate values and access them by index.types:
(i)arraylist
(ii)linkedlist
(iii)list

Set:

never allow duplicate data or value.

Map:

map is a interface store value as key and value(pairs).
Types:
(i)hashmap
(ii)treemap

55. Arraylist, linkedlist, hashset, treeset, hashmap,treemap

Ans:

arraylist is work base on dynamic array.arraylist is allow random access.it is used for search, sort, filter ,fetch(select).

linkedlist:

linked list is based on doubly linkelist. It is not allow random access. Linkedlist is used for insert, update and delete operation.

Hashset:

hashset is a class in java that is used to store a collection of unique elements.
It does not allow duplicates and does not maintain order.

hashmap:

hashmap is a class in java used to store data in the form of key-value pairs.
Each key must be unique, but values can be duplicate.

Treemap:

treemap is a map in java that stores key-value pairs in sorted order based on the keys.

It does not allow duplicate keys, and the keys are automatically arranged in ascending order.

- Lab exercise:

  56. Write a program that demonstrates the use of an arraylist and linkedlist.

  Ans:

```
        public class arraylistex {

        public static void main(string[] args) {
//                                      // todo auto-generated
method stub
//                                      arraylist<string> al=new
arraylist<string>();//simple fix datatype
//                                      al.add("java");
//                                      al.add("php");
//
            al.add("java1");//insertion order
//
//                                      for(string s:al) {
//
                    system.out.println(s);
//                                      }

                                        /*
                                        * difference between
arraylist and linklist
                                        * 1.arraylist is work
base on dynamic array and linked list is based on
doubly linkelist
                                        *2.arraylist is allow
random access and linkedlist is not allow
                                        *3.arraylist is used for
search,sort,filter,fetch(select) and linkedlist
insert,update,delete
                                        *
                                        */

                                       beanclass b = new
beanclass(1, "java", "amd");
                                       beanclass b1 = new
beanclass(2, "java1", "amd1");
                                       beanclass b2 = new
beanclass(3, "java2", "amd2");

                                       arraylist<beanclass> al =
new arraylist<beanclass>();//using bean class
                                       al.add(b);
                                       al.add(b1);
                                       al.add(b2);
```

```
                                for(beanclass s:al) {

    system.out.println(s.sid+" "+s.saddress+"
"+s.sname);
                                }



                                linkedlist<beanclass> l =
new linkedlist<beanclass>();//using bean class
                                l.add(b);
                                l.add(b1);
                                l.add(b2);

                                for(beanclass s1:l) {

  system.out.println(s1.sid+" "+s1.saddress+"
"+s1.sname);
                                }


                            }

}
```

57. Implement a program using hashset to remove duplicate elements from a list.

Ans:

```
         import java.util.hashset;

Public class setex {

        public static void main(string[] args) {
                                    // todo auto-generated
method stub



                                    beanclass b = new
beanclass(1, "java", "amd");
                                    beanclass b1 = new
beanclass(2, "java1", "amd1");
                                    beanclass b2 = new
beanclass(3, "java2", "amd2");

                                    hashset<beanclass>
s=new hashset<beanclass>();
//                                  s.add("java");
//                                  s.add("java1");
//                                  s.add("java2");
                                    s.add(b);
                                    s.add(b1);
                                    s.add(b2);
```

```
                              for(beanclass v:s) {

    system.out.println(v.sid+" "+v.saddress+"
"+v.sname);
                              }

                         }

}
```

58. Create a hashmap to store and retrieve key-value pairs.

Ans:

```
 package javaprac;

Import java.util.hashmap;
Import java.util.iterator;
Import java.util.map;
Import java.util.treemap;

Public class mapprac {

       public static void main(string[] args) {
                              // todo auto-generated
method stub
```

```
                                        beanclass b = new
beanclass(1, "java", "amd");
                                        beanclass b1 = new
beanclass(2, "java1", "amd1");
                                        beanclass b2 = new
beanclass(3, "java2", "amd2");

//                                      hashmap<integer,
beanclass> h1=new hashmap<integer,
beanclass>();
//
//                                      h1.put(1, b);
//                                      h1.put(2, b1);
//                                      h1.put(3, b2);

                                        map<integer,beanclass>
e=new treemap<integer, beanclass>();
                                        e.put(1, b);
                                        e.put(2, b1);
                                        e.put(3, b2);




                                        for(map.entry<integer,
beanclass> m:e.entryset()) {
        system.out.println(m.getvalue().sid +"
"+m.getvalue().sname +" "+m.getvalue().saddress);
                                    }
//                             }

//                                  for(map.entry<integer,
string> e:m.entryset()) {
//
//
//
system.out.println(e.getkey()+" "+e.getvalue());
//
//                                  }
                               }

}
```