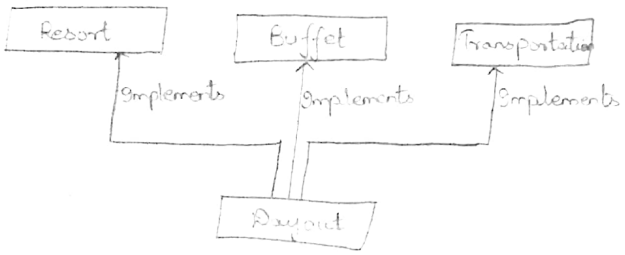
10 output

Leonia
Available
Good
2500/-
Tomato
Fish
French Fries
300/-
Vachibowli
Resort
Car
700/-




```
            t1.getTimings();
            t1.getTicketDetails();
            t1.getfood();
            t1.getorder();
        break;
    Default: System.out.println("No such entry");
    }
  }
}
```

10 Implement a class from three interfaces.

```
interface resort
{   void resortname();
    void waterpark();
    void DJ();
    void packagePrice(){ System.out.println("2,500/-");}
}
interface buffet
{   void veg();
    void non-veg();
    void snacks();
    void price(){ System.out.println("300/-");
}
interface transportation
{   void pickup();
    void drop();
    void type();
    void pricetrans{ System.out.println("700/-");
}
```

Like a class, an interface can have methods and variables, but the methods declared in interface are by default abstract. It is the blueprint of the class.

→ It specifies a set of methods that the class has to implement.

→ If a class implements an interface and does not provide method bodies for all functions specified in interface, the the class must be declared abstract.

→ All fields are public, static and final by default.

Sheet. 14 Exp: 10

```
class Dayout implements resort, buffet, transportation
{
void resortname(){ System.out.println("Leonia");}
void waterpark(){ System.out.println("Available");}
void DJ() { System.out.println("Good");}
void veg(){ System.out.println("Tomato");}
void non-veg(){System.out.println("Fish");}
void snacks(){ System.out.println("French Fries");}
void pickup(){System.out.println("Getitbewb");}
void drop(){System.out.println("Resort");}
void type(){System.out.println("car");}
public static void main(String[] args)
{
    Dayout d= new Dayout();
    d.resortname();
    d.waterpark();
    d.DJ();
    d.packagePolice();
    d.veg();
    d.non-veg();
    d.snacks();
    d.price();
    d.pickup();
    d.drop();
    d.type();
    d.pricetrans();
}
}
```

11. output:

(i) print
(ii) print
(iii) Inner

The class written within another class is called the nested class and the class that holds the inner class is called the outer class.

Nested classes are divided into 2 types

① static Nested classes • ② Non-stated nested class (Inner class).

Inner classes are of 3 types depending on how and where they are defined. They are

(i) Inner class
(ii) Method-local Inner class
(iii) Anonymous Inner class

Laboratory Record

of

Sheet No 15
Experiment No 11
Date

11. Write a program to implement all inner classes

(i) Non-Static inner class

```
class outer
{
    void inner()
    {
        Inner i = new Inner();
        i.print();
    }
    class Inner
    {
        void print(){ System.out.println("print"); }
    }
}
class Main
{ public static void main (String[] args)
    {   outer o = new outer();
        o.inner(); }}
```

(ii) Local inner class:

```
class outer
{   void inner()
    {   class Inner
        {   void print(){System.out.print("print");}}
        Inner i=new Inner();
        i.print();
    }}
```

Roll No. CBIT

Roll No. CBIT

12. output:

15 → Stored

15

15

A wrapper class is a class whose object wraps or contains a primitive data type. When we create an object to a wrapper class, it contains a field and in this field, we can store primitive data types.

→ We can wrap a primitive value into a wrapper class object.

Primitive Datatype	Wrapper class
char	Character
byte	Byte
short	Short
long	Integer
float	Float
double	Double
boolean	Boolean
int	Integer

Laboratory Record

Sheet No. 12

Laboratory Record

of

Sheet No. 16

Experiment No. 12

Date

```
class Main
{   public static void main(String[] args)
    {   Outer o = new Outer();
        o.Inner();
    }}
```

(iii) Static inner class

```
class outer
{   static int data=30;
    static class Inner
    {   void print(){System.out.println("Inner");}
    }}
class Main
{   public static void main(String[] args)
    {   Outer.Inner r = new Outer.Inner();
        r.print();
    }}
```

12. Write a program to implement wrapper class of all types and write a program to accept input to wrapper class by using Scanner class.

```
import java.util.Scanner;
class Wrapper
{   public static void main (String[] args)
    {   Scanner s = new Scanner (System.in);
```

Roll No.

CBIT

13. output:

12. c

print
display
in main display
in main print

f

a Anonymous Inner class

d → An inner class declared without a class name is
th known as an anonymous inner class

→ → In case of anonymous inner classes,
at we declare and instantiate them at the same time.
Generally they are used whenever you need to override
the method of a class or an interface

→ we can override the methods of concrete class
as well as interface using anonymous inner class.

→ The syntax of anonymous class expression is like
the invocation of a constructor, except that there
is a class definition contained in a block of code.

Laboratory Record

Sheet No 18

Laboratory Record

of

Sheet No 17

Experiment No 12, 13

Date

```
        int a = s.nextInt();
        Integer i = Integer.valueOf(a);
        System.out.println(i);
        int j = i.intValue();
        System.out.println(j);
    }}
```

Write a program to create anonymous inner class by extending class

```
class Realbase
{   void print(){ System.out.println("print"); }
}
class Anonymous
{   public static void main(String[] args)
    {   Real r1 = new Real();
        r1.print(); r1.display();
        Real r2 = new Real(){
        void display(){ System.out.println("in r2 display");}
        void print(){ System.out.println("in main print");}
        };
        r2.display();
        r2.print();
    }
}
```

Roll No.

CBIT

13 Output

14. Output:

overridden.

Laboratory Record

of

Sheet No. 18

Experiment No. 13, 14, 15

Date

```
class Real extends Realbase
{   void display()
    { System.out.println("display"); }}
```

14. Write a program to create anonymous inner class by implanting interface

```
interface Real
{ void print();}
class Anonymous
{   public static void main(String[] args)
    {
        Real r= new Real{
          void print(){System.out.println("overridden");}
    };
    r.print();
}}
```

15. Write a program to restrict inheritance creating final class and make constructor private

```
final class Final
{   void write(){System.out.println("write");}
}
class PrivateCons
{   private PrivateCons()
```

Roll No.

CBIT

15. output:

Error.

There are 2 ways to prevent inheritance in Java.

① By using final class

② By using private constructor in a class.

→ Using final keyword before class declaration, we can stop a class to be inherited by another classes.

→ If we make the class constructor private, we'll not be able to create the object of this class from outside of this class. But, our purpose is just to prevent class to be inherited and not stop creation. Hence, we need a method that can create an object of this class and return it.

Laboratory Record

of

Sheet No. 19

Experiment No. 15

Date

```
    {  System.out.println ("Constructor"); }
    static void print()
    {  System.out.println ("Print"); }
}
class Inheritance
{
    public static void main (String[] args)
    {  Final f = new Final();
       f.create();
       Private Constructor print();
}  }
```

14/02/18

Roll No.

CBIT

16 Write a program to implement exception handling and implement the program for some default exceptions

```
public class exceptionHand{
    public static void main (String[] args)
    {   try {
            try {
                int a;
                a= 50/0;
                int a[] = new int [5];
                a[6] = 20;
            } catch (ArrayIndexoutofBoundException f) {
                System.out.println("because of array
                                    f);
            }
        }catch (ArithmeticException e){
            System.out.println ("because of dividing
                                by 0" +e);
        }
    try {
            object arr[]= new Integer [10];
            String s1 = "abc";
            arr[1] = s1;
        }
    catch (ArrayStoreException e)
    {   System.out.println("store exception"+
                                e);
        }
    }
}
```

(17) output:

Arithmetic Exception Handled
End of program

Laboratory Record
of

Sheet No.
Experiment No. 17
Date

(17) write a program to implement throws for default exceptions

```
class Example1 {
    void method1() throws ArithmeticException{
        throw new ArithmeticException("Calculation Error");
    }
    void method2() throws ArithmeticException{
        method1();
    }
    void method3(){
        try{
            method2();
        }
        catch (ArithmeticException e){
            System.out.println("Arithmetic Exception Handled");
        }
    }
    public static void main(String[] args){
        Example1 obj = new Example1();
        obj.method3();
        System.out.println("End of program");
    }
}
```

Laboratory Record

of

Sheet No.

Experiment No. 18

Date

18 Write a program for throw statement by creating user defined class and handle expression using try and catch ?

```
import java.util.Scanner;
public class exceptionHand extends Throwable {
    public static void main(String[] args){
        System.out.println("enter amount");
        Scanner s = new Scanner(System.in);
        int a= s.nextInt();
        try{
            if (a>10,000) {
                throw new exceptionHand();
            } else {
                System.out.println("amount is okay");
            }
        } catch (exceptionHand)
        {
            System.out.println("amount exceeded");
        }
    }
}
```

19. Write a program to implement bank exception?

```
package exceptions;
public class InsufficientFundException extends Throwable{
    public InsufficientFundException() {
        super();
    } public InsufficientFundException(String msg){
        super(msg);
    }
}
```

→
```
package exceptions;
public class InvalidAmountException extends Throwable{
    public InvalidAmountException() {
        super();
    } public InsufficientFundException(String msg){
        super(msg);
    }
}
```

→
```
package exceptions;
public class InvalidTransactionException extends Throwable{
    public InvalidTransactionException() {
        super();
    } public InvalidTransactionExample(String msg){
        super(msg);
    }
}
```

Laboratory Record

of _____

Sheet No. _____

Experiment No. _____

Date _____

→ package exceptions;

```
public class limitExceededException extends Throwable{
    public limitExceededException() {
        super();
    } public limitExceededException(String msg){
        super(msg);
    }
}
```

⇒ package package_2;

```
import exceptions.*;
public interface bank{
    public void deposit(double amount) throws
                InvalidAmountException;
    public void withdraw(double amount) throws
                InsufficientFundException;
    public void RTGS(double amount) throws
                InvalidTransactionException;
    public void NEFT(double amount) throws
                limitExceededException;
    public void balance_enquiry();
}
```

CBIT

Roll No.

⇒
```
package package-2;
import package-2.bank.*;
import package-2.clerk.*;
import exceptions.*;
public class SBIbank implements bank{
    static double balance;
    public void deposit(double amount) throws
        InvalidAmountException{
            if (amount<=0){
                throw new InvalidAmountException("invalid
            }else{
                balance = balance+amount;
    }   }
public void withdraw (double amount) throws
    InsufficientFundException {
        if (amount>balance){
            throw new InsufficientFundException(amount+"ura
                                                ab
        }else{
            this.balance= this.balance-amount.
}   }
public void RTGis(double amount) throws InvalidTransactio
                                        Exception{
    throw new InvalidTransactionException(amount+
                    " is less than 200000");
```

```
public void NEFT(double amount) throws limitExceededException
                                                        {
    throw new limitExceededException(amount + "is
                                        more than 200000");
    }
    public void balance_enquiry(){
        System.out.println("balance:" + balance);
    }
}
```

⇒

```
package package_clerk;
import java.util.Scanner;
import exceptions.*;
import package_2.SBIbank;
public class Bankclerk{
    public static void main(String[] args) {
        while(2>1){
            try {
                Scanner ch=new Scanner(System.in);
                SBI bank bc = new SBIbank();
                System.out.println("1.deposit 2.withdraw
                    3.RTGS 4.NEFT 5.balance enquiry");
                System.out.println("enter your choice");
                int a=ch.nextInt();
                switch(a){
                    case 1: System.out.println("enter amount");
                        double d= ch.nextDouble();
                        bc.deposit(d);
                        break;
```

```
        case 2: System.out.println("enter amount");
                double e= ch.nextDouble();
                bc.withdraw(e);
                break;
        case 3: System.out.println("enter amount");
                double f= ch.nextDouble();
                if (f<200000){
                    bc.RTGS(f);
                }else{
                    bc.withdraw(f);
                } break;
        case 4: System.out.println("enter amount");
                double g= ch.nextDouble();
                if (f>200000){
                    bc.NEFT(g);
                }else{
                    bc.withdraw(g);
                } break;
        case 5: bc.balance_enquiry();
                break;
        default : System.exit(0);
        }
    } catch(InvalidAmountException ob){
        System.out.println(ob.getMessage());
    } catch(InsufficientFundException ie){
        System.out.println(ie.getMessage());
    }
```

Laboratory Record

of _____

Sheet No _____

Experiment No _____

Date _____

```
        catch (InvalidTransactionException it) {
            System.out.println(it.getMessage());
        } catch (LimitExceededException le) {
            System.out.println(le.getMessage());
        }
    }
}
```

Laboratory Record

of _____

Sheet No. _____

Experiment No. 20

Date _____

(20) Write a program to implement Collections class and implement methods of that class.

```
import java.util.ArrayList;
import java.util.Collections;
public class CollectionsDemo{
    public static void main(String[] args){
        ArrayList a1 = new ArrayList();
        a1.add(11);
        a1.add(2);
        a1.add(5);
        a1.add(8);
        a1.add(2);
        a1.add("cbit");
        System.out.println(a1);
        Collections.sort(a1);
        System.out.println("Ascending order" + a1);
        System.out.println(Collections.binarySearch(a1,11));
        Collections.reverse(a1);
        System.out.println("reverse:" + a1);
        System.out.println("no of times: " +
                Collections.frequency(a1,2));
    }
}
```

Roll No.

CBIT

(22) Arraylist of Arraylist. Use listiterator and print elements in forward and reverse direction.

```
import java.util.*;
public class Arraylist1{
    public static void main(String[] args){
        Arraylist<Arraylist<String>> arr1 = new
                                Arraylist<Arraylist<String>>();
        Arraylist<String> ar[] = new Arraylist<String>[5];
        for(int i=0; i<5; i++){
            ar[i] = new Arraylist<String>();
            System.out.println("Enter Student Name");
            ar[i].add(new Scanner(System.in).next());
            System.out.println("Enter roll no.");
            ar[i].add(new Scanner(System.in).next());
            System.out.println("Enter cgpa");
            ar[i].add(new Scanner(System.in).next());
            arr1.add(ar[i]);
        }
        System.out.println(arr1);
        ListIterator ai = arr1.listIterator();
        int i=1;
        while(ai.hasNext()){
            System.out.println("Student"+i+"Details");
```

```
        ArrayList<String> x = (ArrayList) ai.next();
        ListIterator ai1 = x.listIterator();
        while(ai1.hasNext()){
            System.out.println(ai1.next());
        }
    }
while(ai.hasPrevious()){
    System.out.println("Student"+i+"Details");
    ~~ArrayList<~~
    ~~while(ai1.hasPrevious~~
    ArrayList<String> x = (ArrayList) ai.next();
    ListIterator ai2 = x.listIterator();
    while(ai2.hasNext()){
        System.out.println(ai2.next());
    }
}
}
```

(23)

Write program to implement linkedlist.

```
import java.util.*;
class linkedlist1 {
    public static void main (String[] args){
    linkedlist<Integer> ll = new linkedlist<Integ
    ll.add(1);
    ll.add(2);
    ll.add(3);
    ll.add(4);
    ll.add(5);
    ll.add (3,25);
    System.out.println (ll);
    ll.remove(4);
    System.out.println(ll);
    }
}
```

25 Stack collections and implement its methods

```
import java.util.*;
import java.lang.*;
class Stack{
    public static void main (String[] args){
        Stack s= new Stack();
        s.push(2);
        s.push(3);
        s.push(4);
        s.push(5);
        System.out.println("Elements in stack are"+s);
          s.pop();
          s.pop();
        System.out.println("top element" + s.peek());
        System.out.println("empty() " + s.empty());
        System.out.println("search " + s.search(5,3));
    }
}
```

26 Vector collection and implement its methods.

```
import java.util.Vector;
class Vector {
    public static void main(String[] args){
        Arraylist A= new Arraylist();
        A.add(1);
        A.add("chaitha");
        System.out.println("Arraylist values are"+A);
        Vector v= new Vector();
        v.add("2");
        v.addAll(A);
        v.add(4);
        System.out.println("After Add All"+v);
        System.out.println("first element is"+
                                v.firstElement());
        System.out.println("last element is"+
                                v.lastElement());
        System.out.println("element at index 2 is"+
                                v.get(2));
        System.out.println("capacity is"+v.capacity());
        System.out.println("cleared is"+v.clear()+v);
    }
}
```

Laboratory Record

of _____

Sheet No. _____

Experiment No. 27

Date _____

(27) Write a program to implement Treeset using Comparable (A-Z)

```
import java.util.*; {
    class Name implements Comparable<Name> {
        String ka;
        Name (String ka) {
            this.ka = ka;
        }
        public int compareTo (Name s) {
            return this.ka.CompareTo(s.ka);
        }
        public String toString() {
            return String.format("\n[Name: %s]",
                                 ka.toString());
        }
}
public class treesetcomparable {
    public static void main (String[] args) {
        Treeset <Name> d = new Treeset <Name>();
        d.add (new Name ("cherry"));
        d.add (new Name ("hyn"));
        System.out.println(d);
    }
}
```

Roll No.

CBIT

(28) Write a program to Implement Treeset using Comparator. (7-A)

```
import java.util.*
public class ittreeset1 {
    public static void main(String[] args){
        Treeset <String> ts= new Treeset<String>
                                (new MyComp());
        ts.add("RED");
        ts.add("ORANGE");
        ts.add("BLUE");
        ts.add("GREEN");
        ts.add("PINK");
        System.out.println(ts);
    }
}
class MyComp implements Comparator<String>{
    public int compare(String str1, String str2){
        return str2.compareTo(str1);
    }
}
```

(21) Write a program to implement HashMap

```
import java.util.*;
public class hashmap1 {
    public static void main(String[] args) {
        HashMap<Integer, String> hm = new
                HashMap<Integer, String>();
        hm.put(100, "Amit");
        hm.put(101, "Vijay");
        hm.put(102, "Rahul");
        System.out.println("contains key 100: "+hm.containsKey
                                                    (100));
        System.out.println("contains value Amit: " +
                            hm.containsKey("Amit"));
        System.out.println("102 : "+hm.get(102));
        System.out.println("set is empty?"+hm.isEmpty());
        hm.put(103, "Hasi");
        hm.remove(100);
        System.out.println("no.of values = "+hm.size());
        System.out.println("values in hashmap: "+hm.values());
        System.out.println("entry set : "+hm.entrySet());
        HashMap<Integer, String> nm = new HashMap<Integer,
                                                String>();
        nm.put(101, "Hyn");
        nm.put(102, "nik");
        nm.put(103, "vani");
        hm.putAll(nm);
        System.out.println("entry set: "+hm.entrySet());
    }
}
```

30) Write program to implement TreeMap.

```
import java.util.*
public class treemap1 {
    public static void main(String[] args) {
        TreeMap<String, TreeMap> tm1 = new
                    TreeMap<String, TreeMap>();
        TreeMap<String Integer, String> tm2 =
            new TreeMap<Integer, String>();
        tm2.put(150, "Hyderabad");
        tm1.put("Rajesh", tm2);
        Set set = tm1.entrySet();
        Iterator itr = set.iterator();
        while (itr.hasNext()){
            Map.Entry me = (Map.Entry) itr.next();
            System.out.print("Key is: "+me.getKey()+
                            "&");
            System.out.println("Value is: "+ me.getValue()
    }
}
}
```