

CHAROTAR UNIVERSITY OF SCIENCE & TECHNOLOGY
FACULTY OF TECHNOLOGY AND ENGINEERING
U. & P. U. Patel Department of Computer Engineering

Subject Name: Java Programming **Semester: III**
Subject Code: CE251 **Academic year: 2019-20**

PART-I

Data Types, Variables, Arrays, Operators, Control Statements, String

Theory

Data type specifies the size and type of values that can be stored in an identifier.

Types of Variables are Instance Variables (Non-Static Fields), Class Variables (Static Fields), Local Variables, Parameters.

An array is a group of variables that share the same data type, and are referred to by a common name. Arrays of any type can be created and may have one or more dimensions.

An operator is a symbol that operates on one or more operands to produce a result.

It is a control statement to execute a single statement or a block of code when the given condition is true, and if it is false, then it skips if block and rest code of a program executed.

Strings are defined as an array of characters. The difference between a character array and a string is the string is terminated with a special character '\0'.

Practical 1

Introduction to Object Oriented Concepts, comparison of Java with other object oriented programming languages. Introduction to JDK, JRE, JVM, javadoc, command line argument

Theory

Object Oriented Programming is a programming style that is associated with the concept of

- **Object**
This is the basic unit of object oriented programming. That is both data and function that operate on data are bundled as a unit called as object.
- **Class**
When you define a class, you define a blueprint for an object. This doesn't actually define any data, but it does define what the class name means, that is, what an object of the class will consist of and what operations can be performed on such an object.

- **Abstraction**

Data abstraction refers to, providing only essential information to the outside world and hiding their background details, i.e., to represent the needed information in program without presenting the details.

- **Encapsulation**

Encapsulation is placing the data and the functions that work on that data in the same place. While working with procedural languages, it is not always clear which functions work on which variables but object-oriented programming provides you framework to place the data and the relevant functions together in the same object.

- **Inheritance**

As the name suggests Inheritance is the process of forming a new class from an existing class that is from the existing class called as base class, new class is formed called as derived class. This is a very important concept of object-oriented programming since this feature helps to reduce the code size.

- **Polymorphism**

The ability to use an operator or function in different ways in other words giving different meaning or functions to the operators or functions is called polymorphism. That is a single function or an operator functioning in many ways different upon the usage is called polymorphism.

Comparison of Java with other object oriented programming languages

JAVA VS C++

BASIS FOR COMPARISON	C++	JAVA
Platform dependency	Dependent	Independent
Memory management	Accessible to programmer	System controlled
Inheritance model	Single and multiple inheritance,	Single inheritance with abstract interfaces.
Portability	Source must be recompiled for platform, hence	Bytecode classes portable to

	code is not portable.	platform specific JVM's.
Libraries	Predominantly low-level functionality	Massive, classes for high-level services
Runtime error detection	Programmer responsibility	System responsibility
Supports	Pointers, structure and union.	Threads and interfaces.

JAVA VS C#

POINT	JAVA	C#
Development	Sun Microsystem	Microsoft
Development Year	1995	2000
Data Types	Less Primitive DT	More Primitive DT
Struct Concept	Not Supported	Supported
Switch Case	String in Switch Not Allowed	String in Switch Allowed
Delegates	Absent	Supported

JAVA VS PHP

FEATURE	JAVA	PHP
Garbage Collection	Native Support	None
Object Caching	Native Support	None

Security	Native Support	None
Packaging and deployment	Native Support	None
Editors	Many	Primitive

Definition:

JDK: The Java Development Kit (JDK) is a software development environment used for developing Java applications and applets. It includes the Java Runtime Environment (JRE), an interpreter/loader (java), a compiler (javac), an archiver (jar), a documentation generator (javadoc) and other tools needed in Java development.

JRE: The Java Runtime Environment (JRE) is a set of software tools for development of Java applications. It combines the Java Virtual Machine (JVM), platform core classes and supporting libraries.

JVM: It is a specification that provides runtime environment in which java bytecode can be executed. A specification where working of Java Virtual Machine is specified. An implementation is known as JRE (Java Runtime Environment).

Javadoc: Javadoc (originally cased JavaDoc) is a documentation generator created by Sun Microsystems for the Java language (now owned by Oracle Corporation) for generating API documentation in HTML format from Java source code.

Byte code: Java byte code is the result of the compilation of a Java program, an intermediate representation of that program which is machine independent. The Java byte code gets processed by the Java virtual machine (JVM) instead of the processor.

Compiler: A program that converts instructions into a machine-code or lower-level form so that they can be read and executed by a computer.

Interpreter: A program that can analyze and execute a program line by line.

Scripting language: A scripting or script language is a programming language that supports scripts: programs written for a special run-time environment that automate the execution of tasks that could alternatively be executed one-by-one by a human operator.

Programming language: A programming language is a formal language that specifies a set

of instructions that can be used to produce various kinds of output. Programming languages generally consist of instructions for a computer. Programming languages can be used to create programs that implement specific algorithms.

Hypertext language: Stands for "Hypertext Markup Language." HTML is the language used to create web pages. "Hypertext" refers to the hyperlinks that an HTML page may contain.

"Markup language" refers to the way tags are used to define the page layout and elements within the page.

Command line argument: The command line argument is the argument passed to a program at the time when you run it. To access the command-line argument inside a java program is quite easy, they are stored as string in String array passed to the args parameter of main() method.

Practical 2

Given a string, return a string made of the first 2 chars (if present), however include first char only if it is 'o' and include the second only if it is 'z', so "ozymandias" yields "oz".

startOz("ozymandias") → "oz"

startOz("bzoo") → "z"

startOz("oxx") → "o"

Input

```
package my;
public class Prac1 {
    public Prac1() {
        // TODO Auto-generated constructor stub
    }
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        A a = new A();
        a.startOz("ozymandias");
        a.startOz("bzoo");
        a.startOz("oxx");
    }
}
class A
{
    public
        char c1,c2;
    void startOz(String str)
    {

        c1 = str.charAt(0);
        c2 = str.charAt(1);
        System.out.print(str + " -> ");
        if(c1 == 'o' && c2 == 'z')
        {
            System.out.println(c1 + "" + c2);
        }
        else if(c1 == 'o')
        {
            System.out.println(c1);
        }
        else if(c2 == 'z')
```

```
    {  
        System.out.println(c2);  
    }  
    else  
    {  
        System.out.println("the initials of the word is not o or z ");  
    }  
}
```

Output

```
ozymandias -> oz  
bzoo -> z  
oxx -> o
```

Practical 3

Given two non-negative int values, return true if they have the same last digit, such as with 27 and 57. Note that the % "mod" operator computes remainders, so 17 % 10 is 7.

lastDigit(7, 17) → true

lastDigit(6, 17) → false

lastDigit(3, 113) → true

Input

```
package my;
public class Prac3 {
    public Prac3() {}
    public static void main(String[] args) {
        No n = new No();
        n.lastDigit(7, 17);
        n.lastDigit(6, 17);
        n.lastDigit(3, 113);
    }
}
class No{
    public
    void lastDigit(int n1,int n2){
        int r1,r2;
        System.out.print(n1 + " and " + n2 + " -> ");
        r1 = n1%10;
        r2 = n2%10;
        if(r1==r2){
            System.out.println("true");
        }
        else{
            System.out.println("false");
        }
    }
}
```

Output

```
7 and 17 -> true
6 and 17 -> false
3 and 113 -> true
```


Practical 4

Given an array of ints, return true if the sequence of numbers 1, 2, 3 appears in the array somewhere.

array123([1, 1, 2, 3, 1]) → true

array123([1, 1, 2, 4, 1]) → false

array123([1, 1, 2, 1, 2, 3]) → true

Input

```
package my;
import java.util.*;
public class Prac4 {
    public Prac4() {}
    public static void main(String[] args) {
        int i,j=0;
        System.out.println("Enter Array");
        Scanner obj = new Scanner(System.in);
        int [] arr = new int[6];
        for(i=0;i<6;i++){
            arr[i] = obj.nextInt();
        }
        for(i=0;i<3;i++){
            if(arr[i]==1 && arr[i+1]==2 && arr[i+2]==3){
                j++;
            }
        }
        if(j>0){
            System.out.println("true");
        }
        else{
            System.out.println("false");
        }
    }
}
```

Output

```
Enter Array
1 0 1 2 3 0
true
```

Practical 5

Given 2 strings, a and b, return the number of the positions where they contain the same length 2 substring. So "xxcaazz" and "xxbaaz" yields 3, since the "xx", "aa", and "az" substrings appear in the same place in both strings.

stringMatch("xxcaazz", "xxbaaz") → 3

stringMatch("abc", "abc") → 2

stringMatch("abc", "axc") → 0

Input

```
package my;
import java.lang.*;
public class Prac5 {
    public Prac5() { }
    public static void main(String[] args){
        System.out.print("xxcaazz and xxbaaz -> ");
        stringMatch("xxcaazz", "xxbaaz");
        System.out.print("abc and abc -> ");
        stringMatch("abc", "abc");
        System.out.print("abc and axc -> ");
        stringMatch("abc", "axc");
    }
    public static int stringMatch(String str1,String str2){
        int output = 0;
        int n = Math.min(str1.length(),str2.length());
        for(int i=0;i<n-1;i++){
            if(str1.substring(i, i+2).equals(str2.substring(i,i+2))){
                output++;
            }
        }
        System.out.println(output);
        return output;
    }
}
```

Output

```
xxcaazz and xxbaaz -> 3
abc and abc -> 2
abc and axc -> 0
```

Practical 6

Given an array of strings, return a new array without the strings that are equal to the target string. One approach is to count the occurrences of the target string, make a new array of the correct length, and then copy over the correct strings.

wordsWithout(["a", "b", "c", "a"], "a") → ["b", "c"]

wordsWithout(["a", "b", "c", "a"], "b") → ["a", "c", "a"]

wordsWithout(["a", "b", "c", "a"], "c") → ["a", "b", "a"]

Input

```
import java.util.*;
class Prac6{
public static void main(String args[])
{
    System.out.println("enter string: ");
    Scanner S=new Scanner(System.in);
    String a[]=new String[5];
    for(int i=0;i<5;i++){
        a[i]=S.nextLine();
    }
    System.out.print("enter character for skip: ");
    Scanner S1=new Scanner(System.in);
    String c=S1.nextLine();
    for(int i=0;i<5;i++){
        if(a[i].equals(c))
            continue;
        else
            System.out.println(a[i]);
    }
}
```

Output

```
enter string:
a
b
c
d
e
enter character for skip: b
a
c
d
e
```

Practical 7

Display numbers in a pyramid pattern.

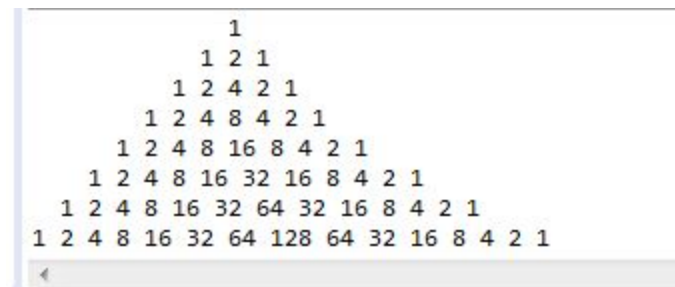
```

      1
    1 2 1
  1 2 4 2 1
1 2 4 8 4 2 1
1 2 4 8 16 8 4 2 1
1 2 4 8 16 32 16 8 4 2 1
1 2 4 8 16 32 64 32 16 8 4 2 1
1 2 4 8 16 32 64 128 64 32 16 8 4 2 1
  
```

Input

```

package my;
public class Pract7{
    public static void main(String[] args){
        for (int i = 1; i <=8; i++){
            for (int j = 8; j > i; j--){
                System.out.print(" ");
            }
            int val1 = 1;
            for (int k = 1; k <= i; k++){
                System.out.print(val1 + " ");
                val1 = val1 * 2;
            }
            val1 = val1 / 4;
            for (int l = i - 1; l >= 1; l--){
                System.out.print(val1 + " ");
                val1 = val1 / 2;
            }
            System.out.println();
        }
    }
}
  
```

Output


```

      1
    1 2 1
  1 2 4 2 1
1 2 4 8 4 2 1
1 2 4 8 16 8 4 2 1
1 2 4 8 16 32 16 8 4 2 1
1 2 4 8 16 32 64 32 16 8 4 2 1
1 2 4 8 16 32 64 128 64 32 16 8 4 2 1
  
```

Practical 8

The problem is to write a program that will grade multiple-choice tests. Assume there are eight students and ten questions, and the answers are stored in a two-dimensional array. Each row records a student's answers to the questions, as shown in the following array.

Students' Answers to the Questions:

	0	1	2	3	4	5	6	7	8	9
Student 0	A	B	A	C	C	D	E	E	A	D
Student 1	D	B	A	B	C	A	E	E	A	D
Student 2	E	D	D	A	C	B	E	E	A	D
Student 3	C	B	A	E	D	C	E	E	A	D
Student 4	A	B	D	C	C	D	E	E	A	D
Student 5	B	B	E	C	C	D	E	E	A	D
Student 6	B	B	A	C	C	D	E	E	A	D
Student 7	E	B	E	C	C	D	E	E	A	D

The key is stored in a one-dimensional array:

Key to the Questions:

0 1 2 3 4 5 6 7 8 9

Key D B D C C D A E A D

Your program grades the test and displays the result. It compares each student's answers with the key, counts the number of correct answers, and displays it.

Input

```
import java.util.*;
public class Prac8 {
    public static void main(String[] args)
    {
        int i,j;
        int[] marks = new int[8];
        Scanner input = new Scanner(System.in);
        char[][] num = new char[8][10];
        System.out.println("");
        for(i=0;i<8;i++)
        {
            System.out.println(i);
            for(j=0;j<10;j++)
            {
                num[i][j] = input.next().charAt(0);
            }
        }
        for(i=0;i<8;i++)
        {
```

```
        if(num[i][0]=='D')
        {
            marks[i]++;
        }
        if(num[i][1]=='B')
        {
            marks[i]++;
        }
        if(num[i][2]=='D')
        {
            marks[i]++;
        }
        if(num[i][3]=='C')
        {
            marks[i]++;
        }
        if(num[i][4]=='C')
        {
            marks[i]++;
        }
        if(num[i][5]=='D')
        {
            marks[i]++;
        }
        if(num[i][6]=='A')
        {
            marks[i]++;
        }
        if(num[i][7]=='E')
        {
            marks[i]++;
        }
        if(num[i][8]=='A')
        {
            marks[i]++;
        }
        if(num[i][9]=='D')
        {
            marks[i]++;
        }
    }
    for(i=0;i<8;i++)
    {
```

```
        System.out.println("The marks of student " + i + " is " + marks[i]);
    }
}
```

Output

```
0
A B A C C D E E A D
1
D B A B C A E E A D
2
E D D A C B E E A D
3
C B A E D C E E A D
4
A B D C C D E E A D
5
B B E C C D E E A D
6
B B A C C D E E A D
7
E B E C C D E E A D
The marks of student 0 is 7
The marks of student 1 is 6
The marks of student 2 is 5
The marks of student 3 is 4
The marks of student 4 is 8
The marks of student 5 is 7
The marks of student 6 is 7
The marks of student 7 is 7
```

Practical 9

The problem is to check whether a given Sudoku solution is correct.

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6							
			4	1	9			5
				8			7	9

(a) Puzzle

Solution →

5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

(b) Solution

Input

```
public class Prac9
```

```
{
```

```
static int[][] sMatrix={
```

```
    {5,3,4,6,7,8,9,1,2},
```

```
    {6,7,2,1,9,5,3,4,8},
```

```
    {1,9,8,3,4,2,5,6,7},
```

```
    {8,5,9,7,6,1,4,2,3},
```

```
    {4,2,6,8,5,3,7,9,1},
```

```
    {7,1,3,9,2,4,8,5,6},
```

```
    {9,6,1,5,3,7,2,8,4},
```

```
    {2,8,7,4,1,9,6,3,5},
```

```
    {3,4,5,2,8,6,1,7,9}
```

```
};
```

```
static int rSum=0;
```

```
static int cSum=0;
```

```
static int[] rSumArray=new int[9];
```

```
static int[] cSumArray=new int[9];
```

```
static int[] boxSumArray=new int[9];
```

```
static boolean checkArrayStatus(int[] rSumArray,int[] cSumArray,int[] boxSumArray)
```

```
{
```

```
    int i=0;
```

```
    boolean sudukoStatus=true;
```

```
    while(i<9){
```

```
        if(rSumArray[i]!=45&&cSumArray[i]!=45&&rSumArray[i]!=45)
```

```
        {
```

```
            sudukoStatus=false;
```

```
            break;
```



```

    }
    i++;
}
return sudukoStatus;
}

public static void main(String[] args) {
    for(int i=0 ; i<sMatrix.length ; i++){
        for(int j=0 ; j<sMatrix.length ; j++){
            rSum+=sMatrix[i][j];
            cSum+=sMatrix[j][i];
        }
        rSumArray[i]=rSum;
        cSumArray[i]=cSum;
        rSum=0;
        cSum=0;
    }
    for(int i=0 ; i< sMatrix.length ; i++){
        for(int j=0 ; j<sMatrix.length ; j++){
            if(i<=2&&j<=2)
            {
                boxSumArray[0]+=sMatrix[i][j];
            }
            if(i<=2&&(j>=3&&j<=5))
            {
                boxSumArray[1]+=sMatrix[i][j];
            }
            if(i<=2&&(j>=6&&j<=8))
            {
                boxSumArray[2]+=sMatrix[i][j];
            }
            if((i>=3&&i<=5)&&(j<=2))
            {
                boxSumArray[3]+=sMatrix[i][j];
            }
            if((i>=3&&i<=5)&&(j>=3&&j<=5))
            {
                boxSumArray[4]+=sMatrix[i][j];
            }
            if((i>=3&&i<=5)&&(j>=6&&j<=8))
            {
                boxSumArray[5]+=sMatrix[i][j];
            }
            if((i>=6)&&(j<=2))

```

```

        {
            boxSumArray[6]+=sMatrix[i][j];
        }
        if((i>=6)&&(j>=3&&j<=5))
        {
            boxSumArray[7]+=sMatrix[i][j];
        }
        if((i>=6)&&(j>=6))
        {
            boxSumArray[8]+=sMatrix[i][j];
        }
    }
}
for(int i=0;i<9;i++)
{
    for(int j=0;j<9;j++)
    {
        System.out.print(sMatrix[i][j] + " ");
    }
    System.out.println();
}

if(checkArrayStatus(rSumArray,cSumArray,boxSumArray))
{
    System.out.println("The matrix is sudoku compliant");
}
else
{
    System.out.println("The matrix is not sudoku compliant");
}
}
}

```

Output

```

run:
5 3 4 6 7 8 9 1 2
6 7 2 1 9 5 3 4 8
1 9 8 3 4 2 5 6 7
8 5 9 7 6 1 4 2 3
4 2 6 8 5 3 7 9 1
7 1 3 9 2 4 8 5 6
9 6 1 5 3 7 2 8 4
2 8 7 4 1 9 6 3 5
3 4 5 2 8 6 1 7 9
The matrix is sudoku compliant

```

Practical 10

Implement Caesar Cipher.

Input

```

import java.util.Scanner;
public class prac {
public static void main(String[] args){
Scanner y=new Scanner(System.in);
System.out.println("Encrypting ");
String plainmsg;
int key;
System.out.print("Enter your message to be encrypted : ");
plainmsg=y.next();
System.out.print("Enter the key : ");
key=y.nextInt();
Test T=new Test();
System.out.println("Your encrpted message is : " +T.ecipher(plainmsg,key));
System.out.println("Decrypting ");
String encrypmsg;
int dkey;
System.out.print("Enter your message to be decrypted : ");
encrypmsg=y.next();
System.out.print("Enter the key : ");
dkey=y.nextInt();
System.out.println("Your encrpted message is : " +T.dcipher(encrypmsg,dkey));
}}
class Test{
String ecipher(String s,int k){
String emsg="";
char c;
for(int i=0;i<s.length();i++){
c=s.charAt(i);
if(c>='a' && c<='z'){
c=(char)(c+k);
if(c>'z')
{c=(char)(c - 'z' + 'a' -1 );}
emsg+=c;
}
else if(c>='A' && c<='Z'){
c=(char)(c+k);
if(c>'Z')
{c=(char)(c - 'Z' + 'A' - 1);}
emsg+=c;
}
}
}
}

```

```
}
else{
    emsg+=c;
}
return emsg;
}
String dcipher(String s,int k){
    String dmsg="";
    char c;
    for(int i=0;i<s.length();i++){
        c=s.charAt(i);
        if(c>='a' && c<='z'){
            c=(char)(c-k);
            if(c<'a')
                {c=(char)(c + 'z' - 'a' + 1 );}
            dmsg+=c;
        }
        else if(c>='A' && c<='Z'){
            c=(char)(c-k);
            if(c<'A')
                {c=(char)(c + 'Z' - 'A' + 1);}
            dmsg+=c;
        }
        else{
            dmsg+=c;
        }
    }
    return dmsg;
}
}
```

Output

```
run:
Encrypting
Enter your message to be encrypted : world
Enter the key : 5
Your encrypted message is : btwqi
Decrypting
Enter your message to be decrypted : btwqi
Enter the key : 5
Your encrypted message is : world
BUILD SUCCESSFUL (total time: 23 seconds)|
```

PART-II

Object Oriented Programming : Classes, Methods, Inheritance

Theory

Class Methods in Java. Class methods are methods that are called on the class itself, not on a specific object instance. The static modifier ensures implementation is the same across all class instances.

Inheritance in Java is a mechanism in which one object acquires all the properties and behaviors of a parent object. It is an important part of OOPs (Object Oriented programming system). The idea behind inheritance in Java is that you can create new classes that are built upon existing classes.

Practical 1

Design a class named Circle containing following attributes and behavior.

- One double data field named radius. The default value is 1.
- A no-argument constructor that creates a default circle.
- A Single argument constructor that creates a Circle with the specified radius.
- A method named getArea() that returns area of the Circle.
- A method named getPerimeter() that returns perimeter of it.

Input

```
package javaapplication18;
```

```
public class Prac11 {
    public static void main(String [] args)
    {
        double Area,Perimeter;
        Circle c = new Circle();
        Area = c.getArea();
        System.out.println("Area of circle is " + Area);
        Perimeter = c.getPerimeter();
        System.out.println("Perimeter of circle is " + Perimeter);
        Circle c1 = new Circle(10);
        Area = c1.getArea();
        System.out.println("Area of circle is " + Area);
        Perimeter = c1.getPerimeter();
        System.out.println("Perimeter of circle is " + Perimeter);
    }
}
class Circle
```

```
{
    double radius = 1;
    Circle()
    {
        System.out.println("default circle created");
    }
    Circle(double r)
    {
        radius=r;
        System.out.println("cirlce with radius " + r);
    }
    double getArea()
    {
        double area ;
        area = Math.PI*radius*radius;
        return area;
    }
    double getPerimeter()
    {
        double perimeter;
        perimeter = 2*Math.PI*radius;
        return perimeter;
    }
}
```

Output

```
default circle created
Area of circle is 3.141592653589793
Perimeter of circle is 6.283185307179586
cirlce with radius 10.0
Area of circle is 314.1592653589793
Perimeter of circle is 62.83185307179586
```

Practical 2

Design a class named Account that contains:

- A private int data field named id for the account (default 0).
- A private double data field named balance for the account (default 500₹).
- A private double data field named annualInterestRate that stores the current interest rate (default 7%). Assume all accounts have the same interest rate.
- A private Date data field named dateCreated that stores the date when the account was created.
- A no-arg constructor that creates a default account.
- A constructor that creates an account with the specified id and initial balance.
- The accessor and mutator methods for id, balance, and annualInterestRate.
- The accessor method for dateCreated.
- A method named getMonthlyInterestRate() that returns the monthly interest rate.
- A method named getMonthlyInterest() that returns the monthly interest.
- A method named withdraw that withdraws a specified amount from the account.
- A method named deposit that deposits a specified amount to the account.

Input

```
import java.util.*;
public class prac2 {
    public static void main(String [] args){
        Account a = new Account();
        Account a1;
        a1 = new Account(112,1000);
        double with,dep ;
        System.out.println("id is : " + a.getId() );
        System.out.println("balance is : " + a.getBalance());
        System.out.println("annualInterestRate is : " + a.getannualInterestRate());
        System.out.println("date is : " + a.getDate());
        System.out.println("monthly interest rate is : " + a.getMonthlyInterestRate());
        System.out.println("monthly interest is : " + a.getMonthlyInterest());
        System.out.println("id is : " + a1.getId() );
        System.out.println("balance is : " + a1.getBalance());
        System.out.println("annualInterestRate is : " + a1.getannualInterestRate());
        System.out.println("date is : " + a1.setDate());
        System.out.println("monthly interest rate is : " + a1.getMonthlyInterestRate());
        System.out.println("monthly interest is : " + a1.getMonthlyInterest());
        System.out.println("enter the amount to withdraw : " );
        Scanner obj = new Scanner(System.in);
```

```
        with = obj.nextDouble();
        System.out.println("the amount to withdraw : " + a1.withdraw(with));
        System.out.println("enter the amount to deposit : " );
        dep = obj.nextDouble();
        System.out.println("the amount to deposit : " + a1.deposit(dep));
    }
}

class Account{
    private int id = 0;
    private double balance = 500;
    private double annualInterestRate = 7;
    private Date dateCreated;
    Account(){
        System.out.println("default created ");
    }
    Account(int i,double b){
        id = i;
        balance = b;
    }
    public int getId() {
        return id;
    }
    public double getBalance() {
        return balance;
    }
    public double getannualInterestRate() {
        return annualInterestRate;
    }
    public Date getDate(){
        return this.dateCreated;
    }
    public void setId(int idnew) {
        this.id = idnew;
    }
    public void setBalance(double balancenew) {
        this.balance = balancenew;
    }
    public void setannualInterestRate(double annualInterestRatenew) {
        this.annualInterestRate = annualInterestRatenew;
    }
    public Date setDate(){
        Date t;
        t = this.dateCreated = new Date();
    }
}
```



```

        return t;
    }
    double getMonthlyInterestRate() {
        double t;
        t = annualInterestRate/12;
        return t;
    }
    double getMonthlyInterest(){
        double t;
        t=annualInterestRate*balance/100;
        return t;
    }
    public double withdraw(double amount) {
        if(amount<balance){
            double t;
            t = balance -= amount;
            return t;
        }
        else {
            System.out.println("the withdraw amount is higher then your balance");
        }
        return 0;
    }
    public double deposit(double amount) {
        double t;
        t = balance += amount;
        return t;
    }
}

```

Output

```

run:
default created
id is : 0
balance is : 500.0
annualInterestRate is : 7.0
date is : null
monthly interest rate is : 0.5833333333333334
monthly interest is : 35.0
id is : 112
balance is : 1000.0
annualInterestRate is : 7.0
date is : Mon Aug 19 11:01:40 IST 2019
monthly interest rate is : 0.5833333333333334
monthly interest is : 70.0
enter the amount to withdraw :
100
the amount to withdraw : 900.0
enter the amount to deposit :
500
the amount to deposit : 1400.0

```

Practical 3

Use the Account class created as above to simulate an ATM machine. Create 10 accounts with id AC001.....AC010 with initial balance 300₹. The system prompts the users to enter an id. If the id is entered incorrectly, ask the user to enter a correct id. Once an id is accepted, display menu with multiple choices.

1. Balance inquiry
2. Withdraw money [Maintain minimum balance 300₹]
3. Deposit money
4. Money Transfer
5. Create Account
6. Deactivate Account
7. Exit

Input

```
import java.util.Scanner;
class Account{
    private String id;
    private double balance,annualInterestRate;
    private String datecreated;
    Account()
    { id="";
      balance=500;
      annualInterestRate=7;
    }
    Account(String i, double b)
    {
      id=i;
      balance=b;
    }
    String getId()
    {return id;}
    double getBalance()
    {return balance;}
    double getAnnualInterestRate()
    {return annualInterestRate;}
    String dateCreated()
    {return datecreated;}
    void setId(String i)
    {id=i;}
    void setBalance(double b)
    {balance=b;}
    void setAnnualInterestRate(double i)
```

```
{annualInterestRate=i;}
double getMonthlyInterestRate()
{return (annualInterestRate/12);}
double getMonthlyInterest()
{return (getMonthlyInterestRate()*balance);}
void withdraw(double w)
{balance-=w;}
void deposit(double d)
{balance+=d;}
}
public class pract3
{
@SuppressWarnings("empty-statement")
public static void main(String[] args)
{
Scanner sc = new Scanner(System.in);
String temp[] =
{"AC001","AC002","AC003","AC004","AC005","AC006","AC007","AC008","AC009","AC010
"};
Accoun []ac = new Accoun[10];
for(int i=0;i<10;i++)
ac[i] = new Accoun();
for(int i=0;i<10;i++)
{
ac[i].setId(temp[i]);
ac[i].setBalance(1000);
}
boolean f =true;
int i = 0,accnumber = 0;
String accId = new String();
System.out.println("A simple ATM Machine");
while(f)
{
System.out.print("\nEnter your account ID : ");
accId = sc.nextLine();
for(i=0;i<10;i++)
{
if(accId.equals(ac[i].getId()))
{
f=false;
accnumber = i;
i=10;
break;
}
```

```
}  
}  
if(i!=10)  
    System.out.println("Enter correct account Id");  
}  
f = true;  
int t;  
while(f)  
{  
    System.out.println("\n\nPress ...");  
    System.out.println("1. Balance inquiry");  
    System.out.println("2. Withdraw money");  
    System.out.println("3. Deposit money");  
    System.out.println("4. Money Transfer");  
    System.out.println("5. Create Account");  
    System.out.println("6. Deactivate Account");  
    System.out.println("7. Exit\n");  
    t = sc.nextInt();  
    switch(t)  
    {  
        case 1:  
        {  
            System.out.println("Your account balance is : " + ac[accnumber].getBalance());  
            break;  
        }  
        case 2:  
        {  
            System.out.print("Enter amount to withdraw : ");  
            double bal = sc.nextDouble();  
            if(bal<=ac[accnumber].getBalance())  
            {  
                ac[accnumber].withdraw(bal);  
                System.out.println("Your account's updated balance is : "+ac[accnumber].getBalance());  
            }  
            else  
                System.out.println("Sorry... Amount is greater then available balance");  
            break;  
        }  
        case 3:  
        {  
            System.out.print("Enter amount to be deposited : ");  
            double bal = sc.nextDouble();  
            ac[accnumber].deposit(bal);
```

```
System.out.println("Your account's updated balance is : "+ac[accnumber].getBalance());
break;
}
case 4:
{
System.out.print("Enter amount to transfer : ");
double bal = sc.nextDouble();
if(bal<=ac[accnumber].getBalance())
{
System.out.print("In which account ID want to transfer money : ");
accId = sc.nextLine();
accId = sc.nextLine();
for(int j=0;j<i;j++)
{
if(accId.equals(ac[j].getId()))
{
ac[accnumber].withdraw(bal);
ac[j].deposit(bal);
System.out.println("Your account's updated balance is : "+ac[accnumber].getBalance());
break;
}
}
if(i==j)
System.out.println("Enter correct account Id");
}
}
else
System.out.println("Sorry... Amount is greater then available balance");
break;
}
case 5:
{
System.out.print("Your new account ID is AC011\nDeposit Money to open account");
System.out.print("Enter amount to be deposited : ");
double bal = sc.nextDouble();
System.out.println("Your account's updated balance is : "+bal);
break;
}
case 6:
{
System.out.print("Your account is deactivated\n\n");
f=false;
break;
}
```

```
case 7:
{
f=false;
break;
}
default:
{
System.out.println("Invalid entry please choose valid number");
}
}
}
}
}
```

Output

```
run:
A simple ATM Machine

Enter your account ID : AC002

Press ...
1. Balance inquiry
2. Withdraw money
3. Deposit money
4. Money Transfer
5. Create Account
6. Deactivate Account
7. Exit

1
Your account balance is :1000.0

Press ...
1. Balance inquiry
2. Withdraw money
3. Deposit money
4. Money Transfer
5. Create Account
6. Deactivate Account
7. Exit

2
Enter amount to withdraw : 500
Your account's updated balance is : 500.0
```

```
Press ...
1. Balance inquiry
2. Withdraw money
3. Deposit money
4. Money Transfer
5. Create Account
6. Deactivate Account
7. Exit

5
Your new account ID is AC011
Deposit Money to open accountEnter amount to be deposited : 500
Your account's updated balance is : 500.0
```

```
Press ...
1. Balance inquiry
2. Withdraw money
3. Deposit money
4. Money Transfer
5. Create Account
6. Deactivate Account
7. Exit
```

```
6
Your account is deactivated
```

```
Press ...
1. Balance inquiry
2. Withdraw money
3. Deposit money
4. Money Transfer
5. Create Account
6. Deactivate Account
7. Exit
```

```
3
Enter amount to be deposited : 1000
Your account's updated balance is : 1500.0
```

```
Press ...
1. Balance inquiry
2. Withdraw money
3. Deposit money
4. Money Transfer
5. Create Account
6. Deactivate Account
7. Exit
```

```
4
Enter amount to transfer : 500
In which account ID want to transfer money : AC001
Your account's updated balance is : 1000.0
```

Practical 4

(Subclasses of Account) In Programming Exercise 2, the Account class was defined to model a bank account. An account has the properties account number, balance, annual interest rate, and date created, and methods to deposit and withdraw funds. Create two subclasses for checking and savings accounts. A checking account has an overdraft limit, but a savings account cannot be overdrawn. Draw the UML diagram for the classes and then implement them. Write a test program that creates objects of Account, SavingsAccount, and CheckingAccount and invokes their toString() methods.

Input

```
import java.util.Scanner;

class Part2_4
{
    public static void main(String args[])
    {
        Scanner y=new Scanner(System.in);

        SavingsAccount SA=new SavingsAccount();

        System.out.println("\n***SAVINGS ACCOUNT***");

        System.out.print("\nEnter the initial balance to be in the account : ");

        double b=y.nextDouble();

        SA.setBalance(b);

        System.out.print("Enter the amount to be withdrawn from account :");

        double d=y.nextDouble();

        SA.checkWithdrawal(d);

        System.out.println("Your updated balance is : "+SA.getBalance());

        CheckingAccount CA=new CheckingAccount();

        System.out.println("\n\n***CHECKING ACCOUNT***");

        System.out.println("Overdraft limit for your checking account is 50000/-");
```



```
System.out.print("\nEnter the initial balance to be in the account : ");
double a=y.nextDouble();
CA.setBalance(a);
System.out.print("Enter the amount to be withdrawn from account :");
double x=y.nextDouble();
CA.overDraft(x);
System.out.println("Your updated balance is : "+CA.getBalance());
}
}
class Account
{
private int id;
private double balance,annualInterestRate;
private String datecreated;
Account(){
id=0;
balance=500;
annualInterestRate=7;
}
Account(int i, double b){
id=i;
balance=b;
}
int getId()
{return id;}
```

```
double getBalance()
{return balance;}

double getAnnualInterestRate()
{return annualInterestRate;}

String dateCreated()
{return datecreated;}

void setId(int i)
{id=i;}

void setBalance(double b)
{balance=b;}

void setAnnualInterestRate(double i)
{annualInterestRate=i;}

double getMonthlyInterestRate()
{return (annualInterestRate/12);}

double getMonthlyInterest()
{return (getMonthlyInterestRate()*balance);}

void withdraw(double w)
{balance-=w;}

void deposit(double d)
{balance+=d;}
}

class SavingsAccount extends Account
{
void checkWithdrawal(double wd)
{
```

```
if(wd>this.getBalance())
{
    System.out.println("\nYour withdrawal amount is more than your
balance");
    System.out.println("So you cannot withdraw any money...");
}
else if(this.getBalance()==0)
{
    System.out.println("Your account has zero balance");
    System.out.println("So you cannot withdraw any money...");
}
else{
    super.withdraw(wd);
}
}
}

class CheckingAccount extends Account
{
    int limit=50000;
    void overDraft(double w){
        if(w<getBalance())
            super.withdraw(w);
        else if(w>getBalance() && w<(limit+getBalance())){
            System.out.println("Your withdrawal amount is more than your account balance");
```

```
System.out.println("But still you can withdraw money as withdrawal amount is less than the overdraft limit");
```

```
super.withdraw(w);
```

```
}
```

```
else if(w>(limit+getBalance()))
```

```
System.out.println("You cannot withdraw any money as it is beyond your account balance and your overdraft limit...");
```

```
}
```

Output

```
run:
```

```
***SAVINGS ACCOUNT***
```

```
Enter the initial balance to be in the account : 36000
```

```
Enter the amount to be withdrawn from account :16000
```

```
Your updated balance is : 20000.0
```

```
***CHECKING ACCOUNT***
```

```
Overdraft limit for your checking account is 50000/-
```

```
Enter the initial balance to be in the account : 60000
```

```
Enter the amount to be withdrawn from account :12999
```

```
Your updated balance is : 47001.0
```

```
BUILD SUCCESSFUL (total time: 1 minute 2 seconds)
```

```
|
```

Practical 5

Develop a Program that illustrate method overloading concept.

Input

```
public class NewClass {
    public static void main(String[] args) {
        Add value = new Add();
        value.sum(2,3);
        value.sum(2,3,4);
        value.sum(2,3,4,5);
        value.sum(2,3,4,5,6);
    }
}
class Add{
    int add = 0;
    public void sum(int a,int b){
        add = a + b;
        System.out.println("sum of 2 number is :" + add);
    }
    public void sum(int a,int b,int c){
        add = a + b + c;
        System.out.println("sum of 3 number is :" + add);
    }
    public void sum(int a,int b,int c,int d){
        add = a + b + c + d;
        System.out.println("sum of 4 number is :" + add);
    }
    public void sum(int a,int b,int c,int d,int e){
        add = a + b + c + d + e;
        System.out.println("sum of 5 number is :" + add);
    }
}
```

Output

```
sum of 2 number is :5
sum of 3 number is :9
sum of 4 number is :14
sum of 5 number is :20
```

PART-III

Package & Interface

Theory

A package in Java is used to group related classes. Think of it as a folder in a file directory. We use packages to avoid name conflicts, and to write a better maintainable code. Packages are divided into two categories:

- Built-in Packages (packages from the Java API)
- User-defined Packages (create your own packages)

An interface in java is a blueprint of a class. It has static constants and abstract methods. The interface in Java is a mechanism to achieve abstraction. There can be only abstract methods in the Java interface, not method body.

Practical 1

WAP that illustrate the use of interface reference. Interface Luminious Object has two method lightOn() and lightOff(). There is one class Solid extended by 2 classes Cube and Cone. There is one class LuminiousCone extends Cone and implements Luminoius Interface. LumminuiousCube extends Cube and implements Luminious Interface. Create a object of LuminiousCone and LuminousCube and use the concept of interface reference to invoke the methods of interface.

Input

```
public class Practical1 {
    public static void main(String[] args)
    {
        LumminuiousCube obj= new LumminuiousCube();
        obj.lightOn();
        obj.lightOff();
        LumminuiousCone obj1= new LumminuiousCone();
        obj1.lightOn();
        obj1.lightOff();
    }
}
interface Lumminuious
{
    public void lightOn();
    public void lightOff();
}
```

```
class Solid
{

}
class Cube extends Solid
{

}
class Cone extends Solid
{

}

class LumminuiousCube extends Cube implements Lumminuious
{
    public void lightOn()
    {
        System.out.println("on");
    }
    public void lightOff()
    {
        System.out.println("off");
    }
}
class LumminuiousCone extends Cone implements Lumminuious
{
    public void lightOn()
    {
        System.out.println("lights are on");
    }
    public void lightOff()
    {
        System.out.println("lights are off");
    }
}
```

Output

```
on
off
lights are on
lights are off
```

Practical 2

WAP that illustrate the interface inheritance. Interface P is extended by P1 and P2 interfaces. Interface P12 extends both P1 and P2. Each interface declares one method and one constant. Create one class that implements P12. By using the object of the class invokes each of its method and displays constant

Input

```
import java.util.*;

interface P{

    int a=10;

    void m1();

}

interface P1 extends P{

    int b=20;

    void m2();

}

interface P2 extends P{

    int c=30;

    void m3();

}

interface P12 extends P1,P2{

    int d=40;

    void m4();

}

class Imp{

    public void m1(){

        System.out.println("interface p: a=10");}
```



```
public void m2(){
    System.out.println("interface p1: b=20");}
public void m3(){
    System.out.println("interface p2: c=30");}
public void m4(){
    System.out.println("interface p12: d=10");
}
}

class Part3_2{
    public static void main(String args[]){
        Imp A=new Imp();
        A.m1();A.m2();A.m3();A.m4();
    }
}
```

Output

```
run:
interface p: a=10
interface p1: b=20
interface p2: c=30
interface p12: d=10
BUILD SUCCESSFUL (total time: 0 seconds)
```

Practical 3

Create an abstract class Robot that has the concrete subclasses, RobotA, RobotB, RobotC. Class RobotA1 extends RobotA, RobotB1 extends RobotB and RobotC1 extends RobotC. There is interface Motion that declares 3 methods forward(), reverse() and stop(), implemented by RobotB and RobotC. Sound interface declare method beep() implemented by RobotA1, RobotB1 and RobotC1. Create an instance method of each class and invoke beep() and stop() method by all objects.

Input

```
import java.util.*;

interface Motion
{
    void forward();
    void reverse();
    void stop();
}

interface Sound
{
    void beep();
}

abstract class Robot
{
}

class RobotA extends Robot
{
}

class RobotB extends Robot implements Motion
{
    public void forward()
```

```
{  
    System.out.println("Forward method of RobotB");  
}  
public void reverse()  
{  
    System.out.println("Reverse method of RobotB");  
}  
public void stop()  
{  
    System.out.println("Stop method of RobotB");  
}  
}  
class RobotC extends Robot implements Motion  
{  
    public void forward()  
    {  
        System.out.println("Forward method of RobotC");  
    }  
    public void reverse()  
    {  
        System.out.println("Reverse method of RobotC");  
    }  
    public void stop()  
    {  
        System.out.println("Stop method of RobotC");  
    }  
}
```

```
}  
}  
class RobotA1 extends RobotA implements Sound  
{  
    public void beep()  
    {  
        System.out.println("beep method of RobotA1");  
    }  
}  
class RobotB1 extends RobotB implements Sound  
{  
    public void beep()  
    {  
        System.out.println("beep method of RobotB1");  
    }  
}  
class RobotC1 extends RobotC implements Sound  
{  
    public void beep(){  
        System.out.println("beep method of RobotC1");  
    }  
}  
class Part3_3{  
    public static void main(String args[])  
    {
```

```
RobotA1 a=new RobotA1();  
a.beep();  
RobotB1 b=new RobotB1();  
b.beep();  
RobotC1 c=new RobotC1();  
c.beep();  
RobotB B=new RobotB();  
B.stop();  
RobotC C=new RobotC();  
C.stop();  
}  
}
```

Output

```
run:  
beep method of RobotA1  
beep method of RobotB1  
beep method of RobotC1  
Stop method of RobotB  
Stop method of RobotC  
BUILD SUCCESSFUL (total time: 0 seconds)
```

Practical 4

Develop a Program that illustrate method overriding concept

Input

```
class Parent{

private void m1() { System.out.println("From parent m1()");}

protected void m2() { System.out.println("From parent m2()"); }

}

class Child extends Parent{

private void m1() { System.out.println("From child m1()");}

public void m2() { System.out.println("From child m2()");}

}

class Main{

public static void main(String[] args)

{

    Parent obj1 = new Parent();

    obj1.m2();

    Parent obj2 = new Child();

    obj2.m2();

}

}
```

Output

```
run:
From parent m2 ()
From child m2 ()
BUILD SUCCESSFUL (total time: 0 seconds)
```

Practical 5

Write a java program which shows importing of classes from other user define packages.

Input

```
//User-defined package

package pkg1;

public class Demo{

public void print()

{System.out.println("\nPrint method of Demo class...under package named pkg1");}

public static void main(String args[]){

Demo d=new Demo();

d.print();

}}

//importing user-defined package

import pkg1.Demo;

class Part3_5{

public static void main(String args[]){

Demo D=new Demo();

D.print();

}}
```

Output

```
run:

Print method of Demo class...under package named pkg1
BUILD SUCCESSFUL (total time: 0 seconds)
|
```

Practical 6

Write a program that demonstrates use of packages & import statements

Input

//User-defined packages

```
package pkg;
```

```
public class Demo
```

```
{
```

```
public void print()
```

```
{System.out.println("\nPrint method of Demo class...under package named pkg");}
```

```
public static void main(String args[])
```

```
{
```

```
Demo d=new Demo();
```

```
d.print();
```

```
}
```

```
}
```

```
package MSDhoni;
```

```
public class Helicopter
```

```
{
```

```
public void sixer()
```

```
{System.out.println("Sixer method of Helicopter class under package named MSDhoni");}
```

```
public static void main(String args[])
```

```
{
```

```
Helicopter H=new Helicopter();
```

```
H.sixer();
```

```
}
```

```
}
```



```
//importing multiple user-defined packages
import pkg .Demo;
import MSDhoni.Helicopter;

class Part3_6
{
    public static void main(String args[])
    {
        Demo D=new Demo();
        D.print();
        Helicopter H=new Helicopter();
        H.sixer();
    }
}
```

Output

```
run:
Print method of Demo class... under package name pkg|
Sixer method of Helicopter class under package named MSDhoni
```

Practical 7

Write a program that illustrates the significance of interface default method.

Input

```
interface Normal{

void fun1();

default void fun2()

{System.out.println("Default method of the interface");}}

class Child implements Normal{

public void fun1()

{System.out.println("Child class's implemntation method of interface");}

}

public class Part3_7{

public static void main(String args[]){

Child C=new Child();

C.fun1();

C.fun2();

}

}
```

Output

```
run:
Child class's implemntation method of interface
Default method of the interface
BUILD SUCCESSFUL (total time: 0 seconds)
```

PART-IV

Exception Handling

Theory

An exception (or exceptional event) is a problem that arises during the execution of a program. When anException occurs the normal flow of the program is disrupted and the program/Application terminates abnormally, which is not recommended, therefore, these exceptions are to be handled.

Practical 1

WAP to show the try - catch block to catch the different types of exception.

Input

```
public class excep {
    public static void main(String[] args) {
        int[] myNumbers = {1, 2, 3};
        int n,m;
        try {
            System.out.println(myNumbers[10]);
        }
        catch(ArrayIndexOutOfBoundsException e){
            System.out.println("array Error occured");
        }
        try{
            n=0;
            m=12/n;
            System.out.println(m);
        }
        catch(ArithmeticException e) {
            System.out.println("arithmetic error occured");
        }
        finally {
            System.out.println("The 'try catch' is finished.");
        }
    }
}
```

Output

```
run:
array Error occured
arithmetic error occured
The 'try catch' is finished.
```

Practical 2

WAP to generate user defined exception using "throw" and "throws" keyword.

Input

```
package javaapplication1;
public class practical2 {
    static void method1() throws ArrayIndexOutOfBoundsException, ArithmeticException
    {
        System.out.println("Inside the method that throws exception\n");
        throw new ArithmeticException();
    }
}
class Part4_2
{
    public static void main(String args[])
    {
        practical2 D=new practical2();
        try
        {
            practical2.method1();
        }
        catch(ArithmeticException e)
        {System.out.println(e);}
        finally
        {System.out.println("Statements of finally block");
        }
    }
}
```

Output

```
run:
Inside the method that throws exception

java.lang.ArithmeticException
Statements of finally block
```

Practical 3

Write a program that raises two exceptions. Specify two 'catch' clauses for the two exceptions. Each 'catch' block handles a different type of exception. For example the exception could be 'ArithmeticException' and 'ArrayIndexOutOfBoundsException'. Display a message in the 'finally' block.

Input

```
package javaapplication1;

public class practical3
{

    void fun1()
    {
        int nums[]=new int[5];

        try
        {
            int num1=34;
            float ans=(num1/0);

            /*this shall throw Arithmetic exception and will be
            commented to check the exception in next line of code*/

            nums[10]=10;//this shall generate the ArrayIndexOutOfBoundsException
        }

        catch(ArithmeticException e)
        {
            System.out.println("Arithmetic exception : "+e);
        }

        catch(ArrayIndexOutOfBoundsException e)
        {
            System.out.println("Array index exception : "+e);
        }

        finally
        {
            System.out.println("\nFinally block executed...");
        }
    }
}
```

```
}  
class Part4_3  
{  
    public static void main(String[] args)  
    {  
        practical3 obj=new practical3();  
        obj.fun1();  
    }  
}
```

Output

```
run:  
Arithmetic exception : java.lang.ArithmeticException: / by zero  
  
Finally block executed...
```