**Java 2**

**Abstract Class**

//1. Create an abstract class with abstract and non-abstract methods.

//02. Create a subclass for an abstract class. Create an object in the child class for the

//abstract class and access the non-abstract methods

//03. Create an instance for the child class in child class and call abstract methods

//04. Create an instance for the child class in child class and call non-abstract methods

package Intern;

abstract class Animal\_9

{

//An abstract method doesn't have its body

abstract void sound();

//non-abstract method

void eat() {

System.out.println("Animal is eating");

}

}

//subclass for an abstract class

class Lion extends Animal\_9 {

//provide implementation of abstract method

void sound() {

System.out.println("Lion roars");

}

public static void main(String[] args) {

Animal\_9 A = new Lion();

//accessing the non-abstract methods

A.eat();

//instance for the child class

Lion L = new Lion();

//calling abstract methods

L.sound();

//calling non-abstract methods

L.eat();

}

}

**Interfaces**

//1. Create an interface with only one method and implement it in a class. Call the method

//implemented.

package Intern;

interface Jala\_01 {

void oneMethod();

}

public class InterfaceOneMethod\_10\_1 implements Jala\_01

{

public void oneMethod()

{

System.out.println("This is a method");

}

public static void main(String[] args)

{

InterfaceOneMethod\_10\_1 if1 = new InterfaceOneMethod\_10\_1();

if1.oneMethod();

}

}

//2. Create an interface with two methods, but implement only one in a class. Call the

//method implemented.

package Intern;

interface Jala\_02 {

void methodOne();

void methodTwo();

}

public class InterfaceTwoMethod\_10\_2 implements Jala\_02

{

public void methodOne() {

System.out.println("First Method");

}

public void methodTwo() {

System.out.println("Second Method");

}

public static void main(String[] args)

{

InterfaceTwoMethod\_10\_2 if2 = new InterfaceTwoMethod\_10\_2();

if2.methodOne();

if2.methodTwo();

}

}

//4.Create two interfaces with one method each. Implement these two interfaces in one class.

package Intern;

//interface 1

interface Jala\_4

{

void methodOne();

}

//interface 2

interface Jala\_04

{

void methodTwo();

}

public class TwoInterfaceOneMethod\_10\_4 implements Jala\_4, Jala\_04

{

public void methodOne()

{

System.out.println("This is Method one");

}

public void methodTwo()

{

System.out.println("This is Method two");

}

public static void main(String[] args)

{

TwoInterfaceOneMethod\_10\_4 if4 = new TwoInterfaceOneMethod\_10\_4();

if4.methodOne();

if4.methodTwo();

}

}

//5. Create two interfaces with the same method (same signature) in both the interfaces.

//Implement these two interfaces in one class. Call the method.

package Intern;

interface Jala\_5 {

void sameMethod();

}

interface Jala\_05 {

void sameMethod();

}

class TwoInterfaceSameMethod\_10\_5 implements Jala\_5, Jala\_05

{

public void sameMethod() {

System.out.println("This is same method in both the interfaces");

}

public static void main(String[] args) {

TwoInterfaceSameMethod\_10\_5 if5 = new TwoInterfaceSameMethod\_10\_5();

if5.sameMethod();

}

}

//6. Create an interface with a default method and implement it in a class. Do not provide

//implementation to the default method and call the method.

package Intern;

interface Jala\_06 {

default void defaultMethod() {

System.out.println("This is a default method");

}

}

class InterfaceDefaultMethod\_10\_6 implements Jala\_06

{

public static void main(String[] args) {

InterfaceDefaultMethod\_10\_6 if6 = new InterfaceDefaultMethod\_10\_6();

if6.defaultMethod();

}

}

//7. Create an interface and inherit it from the other interface.

package Intern;

//interface

interface Jala\_7 {

void methodOne();

}

//inherited interface

interface Jala\_07 extends Jala\_7 {

void methodTwo();

}

class InheritedInterface\_10\_7 implements Jala\_07

{

//override method of interface Jala\_7

public void methodOne() {

System.out.println("This is first method");

}

//override method of inherited interface Jala\_07

public void methodTwo() {

System.out.println("This is second method");

}

public static void main(String[] args) {

//creating obj of InheritedInterface

InheritedInterface\_10\_7 if7 = new InheritedInterface\_10\_7();

//Calling the methods implemented

if7.methodOne();

if7.methodTwo();

}

}

//8. Create a PUBLIC interface with fields and methods, fields should have values assigned.

//Implement this interface to some class and print the values of the interface fields and

//call the interface methods

package Intern;

public interface InterfacePublicFieldMethod\_10\_8

{

int num = 111; //values assigned

public void myMethod();

}

class Jala\_08 implements InterfacePublicFieldMethod\_10\_8

{

public void myMethod() {

System.out.println("This is a method");

}

public static void main(String[] args) {

Jala\_08 j = new Jala\_08();

System.out.println(num);

j.myMethod();

}

}

//9. Create a PRIVATE or PROTECTED interface and print the values as above scenario

//10. Create an interface with private, public and protected fields

//11. Create an interface with static final variable

package Intern;

interface Jala\_09 {

public int num1 = 11;

static final int num2 = 22;

final int num3 = 33;

// private int num4 = 44; - error

// protected int num5 = 55; - error

}

class InterfacePrivateProtected\_10\_9\_10\_11 implements Jala\_09

{

public static void main(String[] args) {

InterfacePrivateProtected\_10\_9\_10\_11 if9 = new InterfacePrivateProtected\_10\_9\_10\_11();

System.out.println(if9.num1 + " " + if9.num2);

}

}

**This and super**

//1. Print the fields/instance members of the current class using this and without using object

//3. Call constructor of the current class using this()

//4. Call argument constructor of current class using this()

package Intern;

public class CallUsingThis\_11\_1\_3\_4

{

int num;

String name;

CallUsingThis\_11\_1\_3\_4() {

this(20, "Souma");

}

CallUsingThis\_11\_1\_3\_4(int num, String name) {

this.num = num;

this.name = name;

System.out.println(num + " " + name);

}

public static void main(String[] args) {

CallUsingThis\_11\_1\_3\_4 ts1 = new CallUsingThis\_11\_1\_3\_4();

CallUsingThis\_11\_1\_3\_4 ts = new CallUsingThis\_11\_1\_3\_4(21, "Paul");

}

}

//2. Print the fields/instance members of the parent class using super

//5. Call constructor of the parent class using super()

package Intern;

//SuperClass

class Parent\_02\_05 {

String name = "Souma";

Parent\_02\_05() {

System.out.println("This is parent class constructor");

}

}

//SubClass

class Child\_02\_05 extends Parent\_02\_05

{

String name = "Paul";

Child\_02\_05()

{

//5. Calling constructor of the parent class using super()

super();

System.out.println("This is child class constructor");

}

void myMethod()

{

//Printing fields of the parent class using super

System.out.println("My name is " + super.name);

}

}

public class CallUsingSuper\_11\_2\_5

{

public static void main(String[] args)

{

Child\_02\_05 c = new Child\_02\_05();

c.myMethod();

}

}

//6. Use this() and super() in methods not in constructor

package Intern;

//SuperClass

class Parent\_06

{

void method1()

{

System.out.println("My name is Souma Paul");

}

void method2()

{

//using this() to call method1

this.method1();

}

}

//SubClass

class Child\_06 extends Parent\_06

{

void method3()

{

//using super() to call parent method

super.method2();

}

}

public class ThisSuperInMethods\_11\_6

{

public static void main(String[] args)

{

Child\_06 c = new Child\_06();

//Calling method3

c.method3();

}

}

**Constructors**

//1. Write a class with a default constructor, one argument constructor and two argument

//constructors. Instantiate the class to call all the constructors of that class from

//a main class

package Intern;

class ParentClass\_01

{

int rollNo;

String name;

String branch;

//default constructor

ParentClass\_01() {

System.out.println("Student Details");

}

//one argument constructor

ParentClass\_01(int rollNo) {

this.rollNo = rollNo;

System.out.println("Roll No : " + this.rollNo);

}

//two argument constructor

ParentClass\_01(String name, String branch)

{

this.name = name;

this.branch = branch;

System.out.println("Name : " + this.name);

System.out.println("Branch : " + this.branch);

}

}

public class DefaultOneTwoArgument\_12\_1

{

public static void main(String... args)

{

//calling constructors from main class

//creating objects and passing values

new ParentClass\_01();

new ParentClass\_01(67);

new ParentClass\_01("Souma", "CBSE");

}

}

//2. Call the constructors(both default and argument constructors) of super class from a child

//class

package Intern;

//child subclass extending superclass

class ChildClass\_02 extends ParentClass\_01 {

String clg;

String city;

//calling Parent class constructor using super() keyword

ChildClass\_02() {

super();

}

//calling Parent class argument constructor using super() keyword

ChildClass\_02(int rollNo) {

super(rollNo);

}

//calling Parent class argument constructor using super() keyword

ChildClass\_02(String name, String branch, String clg, String city)

{

super(name, branch);

this.clg = clg;

this.city = city;

System.out.println("College : " + this.clg);

System.out.println("City : " + this.city);

}

}

public class ClassSuperFromChild\_12\_2

{

public static void main(String... args)

{

//creating objects and passing values

new ChildClass\_02();

new ChildClass\_02(67);

new ChildClass\_02("Souma", "CBSE", "AIITK","Kolkata");

}

}

//3. Apply private, public, protected and default access modifiers to the constructor

//4. Write constructors with return type int and String

//5. Try to call the constructor multiple times with the same object

package Intern;

public class PublicPrivateProtectedDefault\_12\_3\_4\_5

{

int rollNo;

String name;

String clg;

String city;

{System.out.println("Student Details");}

//default access modifier

PublicPrivateProtectedDefault\_12\_3\_4\_5()

{

this(67);

}

//public access modifier

public PublicPrivateProtectedDefault\_12\_3\_4\_5(int rollNo)

{

this("Souma");

this.rollNo = rollNo;

System.out.println("rollNo : " + rollNo);

}

//private access modifiers

private PublicPrivateProtectedDefault\_12\_3\_4\_5(String name)

{

this("AIITK", "Kolkata");

this.name = name;

System.out.println("Name : " + name);

}

//protected access modifiers

protected PublicPrivateProtectedDefault\_12\_3\_4\_5(String clg, String city)

{

this.clg = clg;

this.city = city;

System.out.println("college : " + clg);

System.out.println("City : " + city);

}

public static void main(String[] args) {

//calling the constructor multiple times with the same object

PublicPrivateProtectedDefault\_12\_3\_4\_5 c = new PublicPrivateProtectedDefault\_12\_3\_4\_5();

}

}

**Method Overloading**

**//1. Write two methods with the same name but different number of parameters of same type**

**//and call the methods from main method.**

**//2. Write two methods with the same name but different number of parameters of different**

**//data type and call the methods from main method.**

**//3. Write two methods with the same name and same number of parameters of same type**

**//and call from main method.**

**//4. Write two methods with the same name and same number of parameters of different**

**//type and call from main method.**

**//5. Write two methods with the same name, number of parameters and data type but**

**//different return Type.**

**package Intern;**

**public class MethodOverloading\_13**

**{**

**int num1;**

**int num2;**

**int rollNo;**

**String name;**

**// 01.Two methods with the same name but different number of parameters of same type**

**void method1(int num1) {**

**this.num1 = num1;**

**System.out.println("1.First number : " + num1);**

**}**

**void method1(int num1, int num2) {**

**this.num1 = num1;**

**this.num2 = num2;**

**System.out.println("1.Sum of two numbers : " + (num1 + num2));**

**}**

**// 02.Two methods with the same name but different number of parameters of different data**

**// type same method name one parameter**

**void method2(int rollNo) {**

**this.rollNo = rollNo; //Integer**

**System.out.println("2.Roll number : " + rollNo);**

**}**

**//same method name two parameters**

**void method2(int rollNo, String name) {**

**this.rollNo = rollNo; //Integer**

**this.name = name; //String**

**System.out.println("2.Roll number : " + rollNo + " ; " + "Name : " + name);**

**}**

**// 04.Two methods with the same name and same number of parameters of different type**

**//same method name different parameter type**

**void method4(int rollNo) {**

**this.rollNo = rollNo; //Integer data type**

**System.out.println("4.Roll number : " + rollNo);**

**}**

**//same method name different parameter type**

**void method4(String name) {**

**this.name = name; //String**

**System.out.println("4.Name : " + name);**

**}**

**public static void main(String[] args) {**

**MethodOverloading\_13 mo = new MethodOverloading\_13();**

**// 01.Calling the methods from main method**

**mo.method1(11);**

**mo.method1(10, 20);**

**// 02.Calling the methods from main method**

**mo.method2(67);**

**mo.method2(67, "Souma");**

**// 03.Calling the methods from main method**

**mo.method4(67);**

**mo.method4("Souma");**

**}**

**}**

**Exceptions**

//1. Write a program to generate Arithmetic Exception without exception handling

//2. Handle the Arithmetic exception using try-catch block

package Intern;

public class ArithmeticExceptionTryCatch\_14\_1\_2

{

public static void main(String[] args)

{

int a = 10, b = 0, c;

try

{

System.out.println("Inside try block");

//below code throws divide by zero exception

c = a / b;

System.out.println("c = " + c);

}

//2. handles the Arithmetic Exception

catch (java.lang.ArithmeticException e)

{

System.out.println("Handling the Arithmetic exception using try-catch block");

}

//1. generating Arithmetic Exception without exception handling

c = a / b;

System.out.println("without exception handling");

}

}

// 3.Write a method which throws exception, Call that method in main class without try block

package Intern;

public class MethodThrowsException\_14\_3

{

static void throwException() {

//throw keyword is used to invoke an exception explicitly.

throw new RuntimeException("Throwing exception");

}

public static void main(String[] args) {

//Calling method in main class without try block

throwException();

}

}

//4. Write a program with multiple catch blocks

//5. Write a program to throw exception with your own message

package Intern;

public class MultipleCatchBlocks\_14\_4\_5

{

public static void main(String[] args) {

//try block can be followed by one or more catch blocks.

try {

int[] a = new int[6];

//Arithmetic Exception

a[6] = 10 / 0;

//ArrayIndexOutOfBounds Exception

System.out.println(a[10]);

}

catch (ArithmeticException e) {

//throwing exception with your own message

System.out.println("Arithmetic Exception occurs");

e.getStackTrace();

} catch (ArrayIndexOutOfBoundsException e) {

System.out.println("ArrayIndexOutOfBounds Exception occurs");

e.getStackTrace();

} catch (Exception e) {

System.out.println("Parent Exception occurs");

e.getStackTrace();

}

System.out.println("Main method ended");

}

}

//6. Write a program to create your own exception

package Intern;

class InvalidExamException extends Exception {

InvalidExamException(String msg) {

System.out.println(msg);

}

}

public class CreateOwnException\_14\_6

{

static void exam(int marks) throws InvalidExamException {

if (marks < 40) {

throw new InvalidExamException("Failed in exam");

} else {

System.out.println("Passed in exam");

}

}

public static void main(String[] args) {

try {

exam(39);

} catch (Exception i) {

i.printStackTrace();

}

}

}

//7.Write a program with finally block

package Intern;

public class FinallyBlock\_14\_7

{

public static void main(String args[]) {

try {

//below code throws divide by zero exception

int data = 10 / 0;

System.out.println(data);

}

//handles the Arithmetic Exception / Divide by zero exception

catch (ArithmeticException e) {

System.out.println("Exception handled");

e.printStackTrace();

}

//finally block executes regardless of exception occurred or not

finally {

System.out.println("finally block is always executed");

}

System.out.println("Main method ended");

}

}

//8. Write a program to generate Arithmetic Exception

package Intern;

public class GenerateArithmeticException\_14\_8

{

public static void main(String[] args) {

int a = 10, b = 0, c;

try {

c = a / b;

} catch (ArithmeticException e) {

System.err.println("ArithmeticException caught!");

e.printStackTrace();

}

}

//9. Write a program to generate ArrayIndexOutOfBoundException

package Intern;

public class ArrayIndexOutOfBound\_14\_9

{

public static void main(String[] args) {

String[] arr = {"Souma", "Sam", "Abhik", "Akash"};

try {

System.out.println(arr[5]);

}

catch (ArrayIndexOutOfBoundsException e){

System.err.println("ArrayIndexOutOfBoundsException caught");

e.printStackTrace();

}

}

}

//10.Write a program to generate ClassNotFoundException

package Intern;

public class ClassNotFound\_14\_10

{

public static void main(String[] args) {

try {

Class.forName("com.exceptions.jala");

ClassLoader.getSystemClassLoader().loadClass("com.exceptions.jala");

} catch (ClassNotFoundException e) {

e.printStackTrace();

}

}

}

//11.Write a program to generate FileNotFoundException

package Intern;

import java.io.\*;

public class FileNotFound\_14\_11

{

public static void main(String[] args) {

try {

BufferedReader reader = new BufferedReader(new FileReader(new File("/invalid/file/location")));

}

catch (FileNotFoundException e) {

System.err.println("FileNotFoundException caught!");

e.printStackTrace();

}

}

}

//12. Write a program to generate IOException

package Intern;

import java.io.\*;

//IOExceptions are thrown when there is any input / output file operation issues

public class GenerateIOException\_14\_12

{

public static void main(String[] args) {

File file = new File("test.txt");

FileInputStream fis = null;

try {

fis = new FileInputStream(file);

fis.read();

fis.close();

}

//FileNotFoundException is a subclass of IOException.

catch (IOException e)

{

e.printStackTrace();

}

}

}

//14.Write a program to generate NoSuchMethodException

package Intern;

import java.lang.reflect.Method;

//creating a class

class E\_14 {

//creating a method which receives two integer values as parameters

public void addition(int a, int b) {

int c;

c = a + b;

System.out.println(c);

}

}

public class NoSuchMetodException\_14\_14

{

public static void main(String[] args) {

try {

Class c = Class.forName("com.jala.exceptions.E\_14");

//There is no subtraction() method in class E\_14

Method m = c.getDeclaredMethod("subtraction", int.class, int.class);

}

//Throws NoSuchMethodException

catch (NoSuchMethodException | ClassNotFoundException e)

{

e.printStackTrace();

}

}

}

//15.Write a program to generate NullPointerException

package Intern;

public class NullPointerException\_14\_15

{

//we are trying to perform the length() operation on str which is null

static void simpleNullCheck(String str) {

System.out.println(str.length());

}

public static void main(String args[]) {

String input = null;

try {

simpleNullCheck(input);

}

catch (NullPointerException e)

{

System.out.println("NullPointerException caught!");

e.printStackTrace();

}

}

}

//16. Write a program to generate NumberFormatException

package Intern;

public class NumberFormatException\_14\_16

{

public static void main(String[] args) {

String str = "111ABC";

try {

// Converting string with inappropriate format

int x = Integer.parseInt(str);

int y = Integer.valueOf(str);

} catch (NumberFormatException e) {

System.err.println("NumberFormatException caught!");

e.printStackTrace();

}

}

}

//17. Write a program to generate StringIndexOutOfBoundsException

package Intern;

public class StringIndexOutOfBound\_14\_17

{

public static void main(String[] args) {

String str = "Sharan";

try {

// Trying to access at negative index

char charAtNegativeIndex = str.charAt(-1);

// Trying to access at index equal to size of the string

char charAtLengthIndex = str.charAt(11);

} catch (StringIndexOutOfBoundsException e)

{

System.err.println("StringIndexOutOfBoundsException caught");

e.printStackTrace();

}

}

}

**Java IO**

// 1.Write a program to read text from .txt file using InputStream

package Intern;

import java.io.FileInputStream;

import java.io.InputStream;

public class ReadUsingInputStream\_15\_1

{

public static void main(String[] args)

{

try

{

InputStream fis = new FileInputStream("os.txt");

int i;

while ((i = fis.read()) != -1)

{

System.out.print((char) i);

}

fis.close();

} catch (Exception e)

{

e.getStackTrace();

}

}

}

//2.Write a program to write text to .txt file using OutputStream

package Intern;

import java.io.FileOutputStream;

import java.io.OutputStream;

public class WriteUsingOutputStream\_15\_2

{

public static void main(String args[]) {

String data = "This text is written using OutputStream.";

try {

OutputStream fos = new FileOutputStream("os.txt");

byte[] db = data.getBytes();

fos.write(db);

System.out.println("Data is written to the file.");

fos.close();

} catch (Exception e)

{

e.getStackTrace();

}

}

}

//3.Read text from a .txt file using BufferedInputStream

package Intern;

import java.io.\*;

public class ReadUsingBufferedInputStream\_15\_3

{

public static void main(String[] args) {

try

{

FileInputStream fis = new FileInputStream("bos.txt");

BufferedInputStream bis = new BufferedInputStream(fis);

int i;

while ((i = bis.read()) != -1) {

System.out.print((char) i);

}

//closing Streams

bis.close();

fis.close();

}

catch (Exception e)

{

e.getStackTrace();

}

}

}

//4.Write text to a .txt file using BufferedOutputStream

package Intern;

import java.io.\*;

public class WriteUsingBufferedOutputStream\_15\_4

{

public static void main(String[] args) {

String s = "This text is written using BufferedOutputStream.";

try {

FileOutputStream fos = new FileOutputStream("bos.txt");

//the internal buffer has the default size of 8192 bytes.

BufferedOutputStream bos = new BufferedOutputStream(fos);

byte[] arr = s.getBytes();

// write() method writes data to the output stream

bos.write(arr);

//closing streams

bos.close();

fos.close();

System.out.println("Data is written to the file.");

}

catch (Exception e)

{

e.printStackTrace();

}

}

}

//5.Write a program to read text from .txt file using FileReader

package Intern;

import java.io.FileReader;

public class ReadUsingFileReader\_15\_5

{

public static void main(String[] args) {

try {

FileReader fr = new FileReader("fw.txt");

int i;

while ((i = fr.read()) != -1) {

System.out.print((char) i);

}

//closing FileReader

fr.close();

} catch (Exception e)

{

e.printStackTrace();

}

}

}

//6.Write a program to write text to .txt file using FileWriter

package Intern;

import java.io.FileWriter;

// FileWriter class is used to write character-oriented data to a file.

public class WriteUsingFileWriter\_15\_6

{

public static void main(String[] args) {

String s = "This text is written using FileWriter.";

try {

// Creates a FileWriter

//file path passed as parameter to the FileWriter constructor.

FileWriter fw = new FileWriter("fw.txt");

//provides method to write string directly.

fw.write(s);

//closing FileWriter

fw.close();

System.out.println("Data is written to the file.");

}

catch (Exception e)

{

e.printStackTrace();

}

}

}

//7.Read text from a .txt file using BufferedReader

package Intern;

import java.io.BufferedReader;

import java.io.FileReader;

public class ReadUsingBufferedReader\_15\_7

{

public static void main(String args[]) {

try {

FileReader fr = new FileReader("bw.txt");

BufferedReader br = new BufferedReader(fr);

int i;

while ((i = br.read()) != -1) {

System.out.print((char) i);

}

br.close();

fr.close();

}

catch (Exception e)

{

e.printStackTrace();

}

}

}

//8.Write text to a .txt file using BufferedWriter

package Intern;

import java.io.BufferedWriter;

import java.io.FileWriter;

//BufferedWriter class is used to write the text from a character-based input stream.

public class WriteUsingBufferedWriter\_15\_8

{

public static void main(String[] args) {

String s = "This text is written using BufferedWriter.";

try {

// Creates a FileWriter

//file path passed as parameter to the FileWriter constructor.

FileWriter fw = new FileWriter("bw.txt");

// Creates a BufferedWriter

BufferedWriter bw = new BufferedWriter(fw);

//provides method to write string directly.

bw.write(s);

//closing FileWriter and BufferedWriter

bw.close();

fw.close();

System.out.println("Data is written to the file.");

}

catch (Exception e)

{

e.printStackTrace();

}

}

}

//9.Write a program to read data from properties file

package Intern;

import java.io.FileInputStream;

import java.util.Properties;

public class ReadUsingPropertiesFile\_15\_9

{

public static void main(String args[]) {

Properties prop = readPropertiesFile("pf.txt");

System.out.println("Name: " + prop.getProperty("Name"));

System.out.println("EN.no: " + prop.getProperty("EN.no"));

System.out.println("College: " + prop.getProperty("College"));

}

public static Properties readPropertiesFile(String fileName) {

Properties prop = null;

try {

FileInputStream fis = new FileInputStream(fileName);

prop = new Properties();

prop.load(fis);

fis.close();

}

catch (Exception e)

{

e.printStackTrace();

}

return prop;

}

}

//10.Write a program to write data to properties file

package Intern;

import java.io.FileOutputStream;

import java.util.Properties;

public class WriteUsingPropertiesFile\_15\_10

{

public static void main(String args[]) {

try {

Properties props = new Properties();

props.put("Name", "Souma");

props.put("EN.no", "67");

props.put("College", "AIITK");

FileOutputStream outputStrem = new FileOutputStream("pf.txt");

props.store(outputStrem, "This is a sample properties file");

System.out.println("Properties file created");

}

catch (Exception e)

{

e.printStackTrace();

}

}

}

**Collections**

//1.Create an ArrayList of type String with 10 string elements. Add 10 string elements to

//ArrayList and perform the below operations

//a. Add an element to the ArrayList

//b. Iterate through the ArrayList by using Iterator object

//c. Add an element at a specific index

//d. Remove an element from the ArrayList, Remove at an index

//e. Update the element at a specific index

//f. Check the element is present at a particular index

//g. Get an element at a particular index

//h. Find out the size of the ArrayList

//i. Check the given element is present in the ArrayList

//j. Remove all elements of the ArrayList

package Intern;

import java.util.Arrays;

import java.util.Iterator;

public class ArrayList\_16\_1

{

public static void main(String[] args)

{

//passing an Array converted to List using the asList() method

java.util.ArrayList<String> jala = new java.util.ArrayList<>(

Arrays.asList("zero", "one", "two", "three", "four", "five"));

System.out.println("\njala = " + jala);

//a. adding elements to the ArrayList using add method using add() method

jala.add("six");

jala.add("seven");

jala.add("eight");

jala.add("nine");

System.out.println("\nAfter adding elements :");

System.out.println("jala = " + jala);

//b. Iterating through the ArrayList by using Iterator object

Iterator<String> itr = jala.iterator();

System.out.println("\nIterating through the ArrayList : ");

while (itr.hasNext()) {

System.out.print(itr.next() + " ");

}

System.out.println(" ");

//c. Adding an element at a specific index using add() method

jala.add(1, "AtIndex1");

System.out.println("\nAfter adding an element at index 1 :");

System.out.println("jala = " + jala);

//d. Removing an element at a specific index using remove() method

jala.remove(1);

System.out.println("\nAfter removing an element at index 1 :");

System.out.println("jala = " + jala);

//e. Updating the element at a specific index using set() method

jala.set(1, "ONE");

System.out.println("\nAfter updating the element at index 1 :");

System.out.println("jala = " + jala);

//f. Checking the element is present at a particular index using indexOf() method

System.out.println("\nThe element is present at index :");

System.out.println(jala.indexOf("three"));

//g. Getting an element at a particular index using get() method

System.out.println("\nThe element at index 3 :");

System.out.println(jala.get(3));

//h. Finding the size of the ArrayList using size() method

System.out.println("\nSize of the ArrayList : ");

System.out.println(jala.size());

//i. Checking the element is present in the ArrayList using contains() method

System.out.println("\n'two' is present in the ArrayList : ");

System.out.println(jala.contains("two")); //returns boolean value

//j. Removing all elements of the ArrayList using clear() method

System.out.println("\nAfter removing all elements of the ArrayList :");

jala.clear();

System.out.println("jala = " + jala);

}

}

//2. Create a HashMap with at least 10 key value pairs of the Student ID and Name

//a. Insert a Key value mapping into the map

//b. Fetch the value of a Key

//c. Create a clone/copy of HashMap

//d. Check if the given Key is in the Map

//e. Check if the value is in the Map

//f. Check if the map is empty

//g. Print the size of the Map to the console

//h. Print all the Keys of the map to the console

//j. Remove a specific Key-value pair

//k. Copy all the elements of the Map to another Map

package Intern;

/\* Java HashMap class implements the Map interface which allows us

to store key and value pair, where keys should be unique. \*/

public class HashMap\_16\_2

{

public static void main(String[] args) {

//storing key in Integer and value in String

java.util.HashMap<Integer, String> jala = new java.util.HashMap<>();

/\* If you try to insert the duplicate key, it will

replace the element of the corresponding key. \*/

//a. Inserting Student ID as Key & Name as value using put() method

jala.put(1, "Souma");

jala.put(2, "Sam");

jala.put(3, "Paul");

jala.put(4, "Abhik");

jala.put(5, "Aditya");

jala.put(6, "Akash");

jala.put(7, "Rik");

jala.put(8, "Randeep");

jala.put(9, "Sunil");

jala.put(10, "Rahul");

System.out.println("\nkey-value pairs of the Student ID and Name : ");

System.out.println("jala = " + jala);

//b. Fetching the value of a Key using get() method

System.out.println("\nName of the Student with ID no = 6 :");

System.out.println(jala.get(6));

//c. Creating a clone/copy of HashMap using clone() method

System.out.println("\nclone/copy of HashMap jala : ");

System.out.println(jala.clone());

//d. Checking if the given Key is in the Map using containsKey() method

System.out.println("\nID No = 3 is in the Map : ");

System.out.println(jala.containsKey(3));

//e. Checking if the value is in the Map using containsValue() method

System.out.println("\nStudent Name = 'Souma' is in the map :");

System.out.println(jala.containsValue("Sharan"));

//f. Checking if the map is empty using isEmpty() method

System.out.println("\nChecking if the map is empty :");

System.out.println(jala.isEmpty());

//g. Printing the size of the Map to the console using size() method

System.out.println("\nThe size of the Map is :");

System.out.println(jala.size());

//h. Printing all the Keys of the map to the console using keySet() method

System.out.println("\nAll ID No of the students :");

System.out.println(jala.keySet());

//j. Removing a specific Key-value pair using remove() method

System.out.println("\nRemoved ID no = 2 ,Student name = 'Sam' : ");

jala.remove(2, "Sam");

System.out.println(jala);

//k. Copying all the elements of the Map to another Map using puttAll() method

System.out.println("\nCopied all the elements of the jala to icfai :");

java.util.HashMap<Integer, String> icfai = new java.util.HashMap<>();

icfai.putAll(jala);

System.out.println("icfai = " + icfai);

}

}

//3.Create a HashSet with at least 10 elements of type String

//Write program covering all the operations of HashSet

package Intern;

import java.util.Iterator;

public class HashSet\_16\_3

{

public static void main(String[] args) {

java.util.HashSet<String> jala = new java.util.HashSet<>();

//a. adding elements to the HashSet using add() method

jala.add("zero");

jala.add("one");

jala.add("two");

jala.add("three");

jala.add("four");

jala.add("five");

jala.add("six");

jala.add("seven");

jala.add("eight");

jala.add("nine");

jala.add("ten");

//does not allow duplicate elements, Set contains unique elements only.

jala.add("nine");

System.out.println("\nAfter adding elements to HashSet :");

//HashSet doesn't maintain the insertion order

System.out.println("jala = " + jala);

//b. Iterating through the HashSet by using Iterator object

Iterator<String> itr = jala.iterator();

System.out.println("\nIterating through the HashSet : ");

while (itr.hasNext()) {

System.out.print(itr.next() + " ");

}

System.out.println(" ");

//c. Create a clone/copy of HashSet using clone() method

System.out.println("\nclone/copy of HashSet : ");

System.out.println(jala.clone());

//c. Removing a specific element using remove() method

jala.remove("zero");

System.out.println("\nAfter removing element 'zero' :");

System.out.println("jala = " + jala);

//f. Checking if the set is empty using isEmpty() method

System.out.println("\nChecking if the set is empty :");

System.out.println(jala.isEmpty());

//h. Finding the size of the HashSet using size() method

System.out.println("\nSize of the HashSet : ");

System.out.println(jala.size());

//d. Checking if element is present in the HashSet using contains() method

System.out.println("\nElement 'one' is present in the HashSet : ");

System.out.println(jala.contains("one")); //returns boolean value

//j. Removing all elements of the HashSet using clear() method

System.out.println("\nAfter removing all elements of the HashSet :");

jala.clear();

System.out.println("jala = " + jala);

}

}