***Java Exercises:***

1. **Write an abstract class named Arithmetic and an abstract method named add are declared . Write a class named Adder that inherits from the Arithmetic class. The Adder class must implement the add method which has two integer parameters, a and b, and calculates and returns the sum of a and b. Your implementation of the add method must return the sum of its two parameters. The add method should print the name of the Adder class' superclass (i.e., the class that Adder inherits from).The sum of integers a and b as returned by calling the add method on an Adder object.Write a Demo class which create the instance of Adder class and call it's add method**

**Program:**

import java.util.Scanner;

abstract class Arithmetic

{

abstract int add(int a,int b);

}

class adder extends Arithmetic

{

@Override

int add(int a,int b) {

return a+b;

}

}

public class abstarctclass {

public static void main(String arg[])

{

Scanner sc=new Scanner(System.in);

Arithmetic ar=new adder();

System.out.println("Enter the two number to Add:");

int a=sc.nextInt(),b=sc.nextInt();

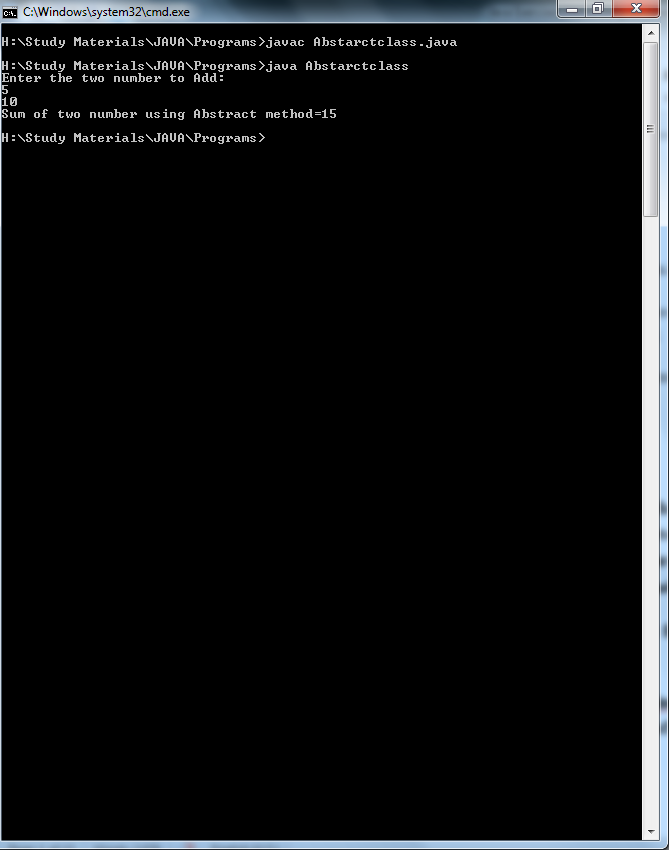
int sum=ar.add(a,b);

System.out.println("Sum of two number using Abstract method="+sum);

}

}

**Output:**



1. **Create class MyCalculator which has a single method: int power(int,int). This method should have two integer parameters, n and p, and must calculate and return np.If one or both arguments passed to it are negative, then the method must throw an exception with the following message: n and p should be non-negative.**

**Input Format : The parameters for your power method should be int n and int p, respectively.**

**Output Format: The power(int,int) method should return the result of np, or throw an exception in the event that one of the inputs was negative.**

**Program:**

import java.util.\*;

class MyCalculator{

public static void main(String ars[]){

int n,p;

Scanner scan=new Scanner(System.in);

System.out.println("Enter two numbers");

n=scan.nextInt();

p=scan.nextInt();

try{

int result=power(n,p);

System.out.println(result);

}

catch(IllegalArgumentException e){

System.out.println(e);

}

}

static int power(int n,int p)throws IllegalArgumentException {

if(n<0 || p< 0)

{

throw new IllegalArgumentException("n and p should be non negative");

}

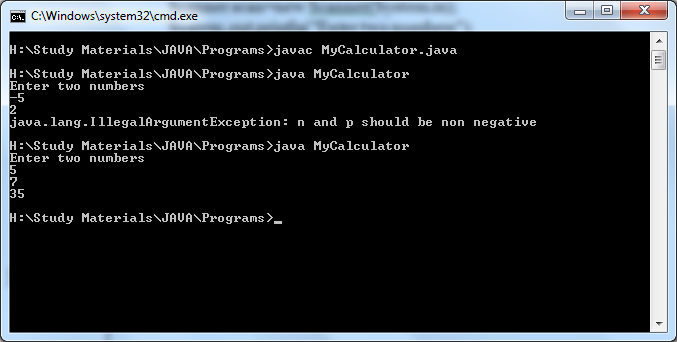
else

return (n\*p);

}

}

**Output:**



1. **Write the interface called GeometricObject, which declares two abstract methods: getParameter() and getArea().Write the implementation class Circle, with a protected variable radius, which implements the interface GeometricObject.Write a test program called TestCircle to test the methods defined in Circle**

**Program:**

import java.util.\*;

interface GeometricObject

{

void getParameter();

void getArea();

}

class Circle implements GeometricObject

{

int rad;

float area;

Scanner sc=new Scanner(System.in);

public void getParameter()

{

System.out.println("Enter Radius");

rad=sc.nextInt();

}

public void getArea()

{

area=(3.14f\*rad\*rad);

System.out.println("Area= "+area);

}

}

class TestCircle

{

public static void main(String arg[])

{

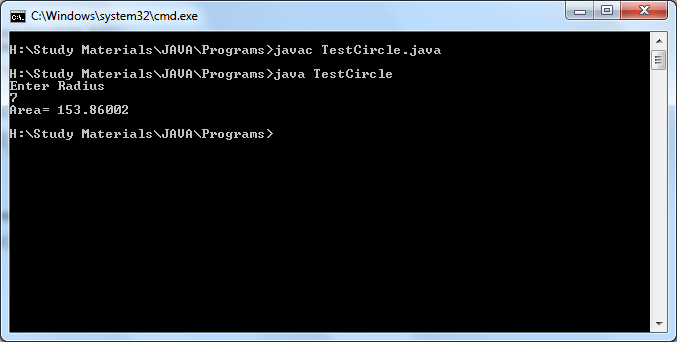
Circle c=new Circle();

c.getParameter();

c.getArea();

}

}

**Output:**

1. **The class ResizableCircle is defined as a subclass of the class Circle, which also implements an interface called Resizable. The interface Resizable declares an abstract method resize(), which modifies the dimension (such as radius) by the given percentage. Write the interface Resizable and the class ResizableCircle.Write a test program called TestResizableCircle to test the methods defined in ResizableCircle**

**Program:**

import java.util.\*;

interface GeometricObject

{

void getParameter();

void getArea();

}

interface Resizable

{

void resize(int perc);

}

class Circle implements GeometricObject

{

int rad;

float area;

Scanner sc=new Scanner(System.in);

public void getParameter()

{

System.out.println("Enter Radius");

rad=sc.nextInt();

}

public void getArea()

{

area=(3.14f\*rad\*rad);

System.out.println("Area= "+area);

}

}

class ResizableCircle extends Circle implements Resizable

{

public void resize(int perc)

{

rad+=(rad\*perc)/100;

System.out.println("Increased Radius= "+rad);

}

}

class TestResizable

{

public static void main(String arg[])

{

int per;

ResizableCircle r=new ResizableCircle();

r.getParameter();

r.getArea();

System.out.println("Enter percent");

Scanner sc=new Scanner(System.in);

per=sc.nextInt();

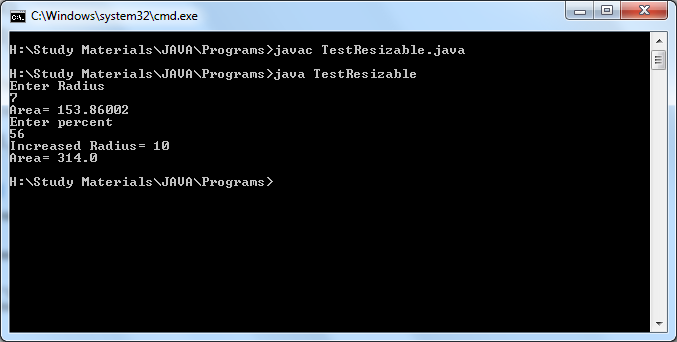
r.resize(per);

r.getArea();

}

}

**Output:**



1. **Consider a class BankAcc with the following members:**
   * + **Accno**
     + **Name**
     + **Address**
     + **Balance**
     + **display()**
     + **withdraw (long amount)**
     + **deposit(long amount).**
   * **Create a subclass DepositAcc override the display () method in class DepositAcc. It only displays a message: Account type is Deposit with the statement:**
   * **The display() method from DepositAcc is not as useful as the one of its superclass or parent class. So make the display class to perform the BankAcc’s display() function and display the message.**
   * **Override the withdraw() method in class DepositAcc so that it first checks to see if the amount to be withdrawn is less than or equal to the balance. If it is, then it calls the withdraw() method of its parent class, otherwise it generates an error message.**
   * **Create a subclass of BankAcc called CurrentAcc which will represent a current account. It will need 3 extra attributes: withdrawal limit, interest rate charged on overdrafts, number of transactions. Each time a transaction is made, the number of transactions should be increased by 1**
   * **Write/Override methods in CurrentAcc**
     + **So that when a withdrawal or deposit is made, the transaction counter is incremented.**
     + **So that the display method displays the account type.**
     + **To subtract the interest due from the balance with the number of months as a parameter.**
     + **To withdraw money from the account, this should generate an error message if the overdraft limit is about to be exceeded.**
     + **To calculate the account fees owed and subtract them from the balance. The fee is 25p per transaction.**

**Program:**

class bankAcc

{

       int accno;

       String  name="";

       String address="";

       double  balance;

       void display()

       {

              System.*out*.println("Account Number="+accno);

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

              System.*out*.println("Address="+address);

              System.*out*.println("Balance="+balance);

       }

       void withdraw(long amount)

       {

              balance-=0.25;

              balance=balance-amount;

       }

       void deposit(long amount)

       {

              balance=balance+amount;

       }

       void  pass(int i, String string, String string2, double d) {

              accno=i;

              name=string;

              address=string2;

              balance=d;

       }

}

class DepositACC extends bankAcc

{

void withdraw(long amount)

{

       if(balance>amount)

       {

              super.withdraw(amount);

       }

       else

       {

              System.*out*.println("Insufficient Balance"+balance);

       }

}

}

class CurrentAcc extends DepositACC

{

       long limit;

       int trans=0,interest;

       DepositACC da=new DepositACC();

       public void CurrentAcc(long limit,int interest) {

       this.limit=limit;

       this.interest=interest;

       }

       void withdraw(long amount)

       {

              if(amount<=limit)

              {

                     trans++;

                     super.withdraw(amount);

              }

              else

              {

                     System.*out*.println("Withdrawn Amount exceeds the limit");

              }

       }

       void deposit(long amount)

       {

              super.deposit(amount);

       }

       void display()

       {

              super.display();

       }

}

public class bankingAssignment {

       public static void main(String Arg[])

       {

CurrentAcc Ca=new CurrentAcc();

bankAcc ba=new bankAcc();

Ca.pass(12345,"New Customer","Chennai",100000.0);

Ca.CurrentAcc(50000,4);

Ca.withdraw(300);

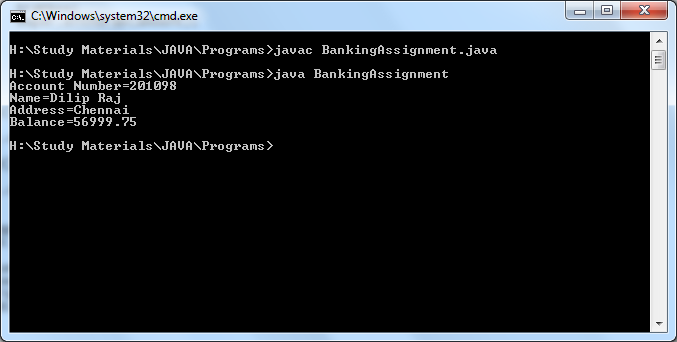
Ca.deposit(4500);

Ca.display();

       }

}

**Output:**



# Create a class Point with the following attributes

* + - **x**
    - **y**
    - **Point()**
    - **Point(int, int)**
    - **getX**
    - **getY**
    - **getArea**
    - **translate(int dx, int dy) – moves the Point dx in the x direction and by dy in the y direction**

# Create a class Circle from the Point class with members

* + - **radius**
    - **Circle()**
    - **Circle(int, int, int)**
    - **getRadius()**
    - **getArea()**
    - **grow(int)**
  + **Create a class Square from the Point class with members**
    - **width**
    - **Square(int, int, int)**
    - **getWidth()**
    - **getArea()**
    - **grow(int)**
  + **Create a class Rectangle from the Square class with members**
    - **height**
    - **Rectangle(int, int, int, int)**
    - **getHeight()**
    - **getArea()**
    - **grow(int,int)**
  + **Instantiate all the classes in the main function and check for the functionalities.**

**Program:**

public class Square

extends Point {

private int width;

public Square() {

width = 1;

}

public Square(int x, int y, int w) {

super(x,y);

width = w;

}

public int getWidth() {

return width;

}

public double getArea() {

return width \* width;

}

public void grow(int gw) {

width = width + 2\*gw;

translate(-gw, -gw);

}

}

public class Circle

extends Point {

private int radius;

public Circle() {

radius = 1;

}

public Circle(int x, int y, int r) {

super(x, y);

radius = r;

}

public int getRadius() {

return radius;

}

public void grow(int gr) {

radius = radius + gr;

}

public double getArea() {

return Math.PI \* radius \* radius;

}

}

public class Point {

private int x;

private int y;

public Point() {

x = 0;

y = 0;

}

public Point(int x, int y) {

this.x = x;

this.y = y;

}

public void translate(int dx, int dy) {

x = x + dx;

y = y + dy;

}

public int getX() {

return x;

}

public int getY() {

return y;

}

public class Rectangle

extends Square {

private int height;

private int width;

public Rectangle() {

width=1;

height=1;

}

public Rectangle(int x, int y, int w, int h) {

super(x,y,w);

height = h;

}

public int getHeight() {

return height;

}

public double getArea() {

return height \* getWidth();

}

public void grow(int gw, int gh) {

height = height + 2\*gh;

width=width+2\*gw;

}

}

public double getArea() {

return 0;

}

}

public class DemoLen {

public static void main(String args[])

{

Point pnt=new Point(0,0);

Rectangle r=new Rectangle(3,4,5,6);

Circle c=new Circle(3,4,5);

Square s=new Square(5,6,8);

pnt.translate(3,4);

System.out.println("Point");

System.out.println("X-coordinate:"+pnt.getX());

System.out.println("Y-coordinate:"+pnt.getY());

System.out.println("Area:"+pnt.getArea());

System.out.println("Rectangle");

System.out.println("Area:"+r.getArea());

r.grow(1,2);

System.out.println("Height:"+r.getHeight());

System.out.println("Radius:"+c.getRadius());

c.grow(8);

System.out.println("Circle");

System.out.println("Area:"+c.getArea());

s.grow(4);

System.out.println("Square");

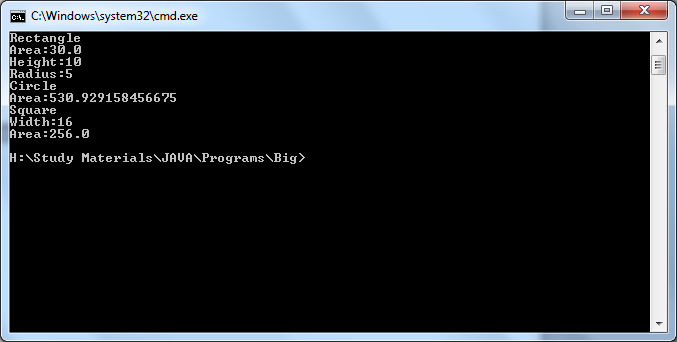
System.out.println("Width:"+s.getWidth());

System.out.println("Area:"+s.getArea());

}

}

**Output:**



1. **Write a custom exception class InsufficientBalanceException. Write a class which has withDrawMoney method which takes amountToWithDraw as parameter.Inside the method, compare the balance with amountToWithdraw. If balance is less than amountToWithdraw then raise an exception InsufficientBalanceException.**

**Program:**

import java.util.\*;

class InsufficientBalanceException extends Exception

{

InsufficientBalanceException(String s)

{

super(s);

}

}

class WithDraw

{

double bal;

void getBal()

{

Scanner sc=new Scanner(System.in);

System.out.println("Enter balance");

bal=sc.nextDouble();

}

void withDrawMoney(double atwd)throws InsufficientBalanceException

{

if(bal<=atwd)

{

throw new InsufficientBalanceException("You Cannot WithDraw Amount ur balance is less than the amount u need!!!");

}

else

{

System.out.println(" "+(bal-atwd));

}

}

}

class ExceptionFirst

{

public static void main(String s[])

{

double amt;

Scanner sc=new Scanner(System.in);

WithDraw wd=new WithDraw();

wd.getBal();

System.out.println("Enter Withdraw");

amt=sc.nextDouble();

try

{

wd.withDrawMoney(amt);

}

catch(Exception m)

{

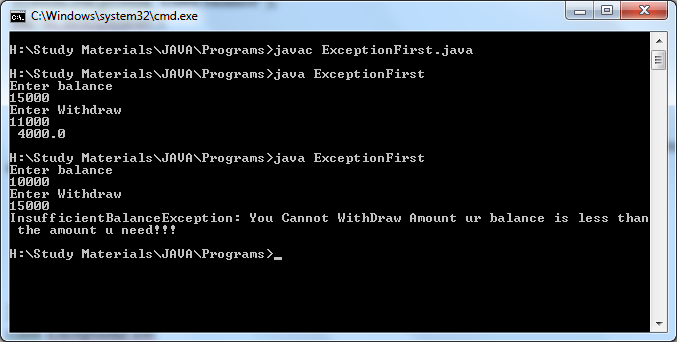
System.out.println(m);

}

}

}

**Output:**



1. **Write a program to have try catch and finally blocks for the following exceptions**

* **Divide by zero**
* **Accessing array with indices**
* **Call a method of using a null variable**

**Program:**

import java.util.\*;

class ExceptionTwo

{

public static void main(String s[])

{

try

{

int a[]={1,2};

Scanner sc=new Scanner(System.in);

System.out.print("Enter the value to divide");

int n=sc.nextInt();

a[0]=a[1]/n;

}

catch(ArithmeticException e)

{

System.out.println("ArithmeticException Caught:"+e);

}

try

{

int a[]={1,2};

a[1]=a[5];

}

catch(ArrayIndexOutOfBoundsException e)

{

System.out.println("ArrayIndexOutOfBound Caught:"+e);

}

try

{

String str=null;

if(str.equals("asdf"))

System.out.println("NO Null");

}

catch(NullPointerException e)

{

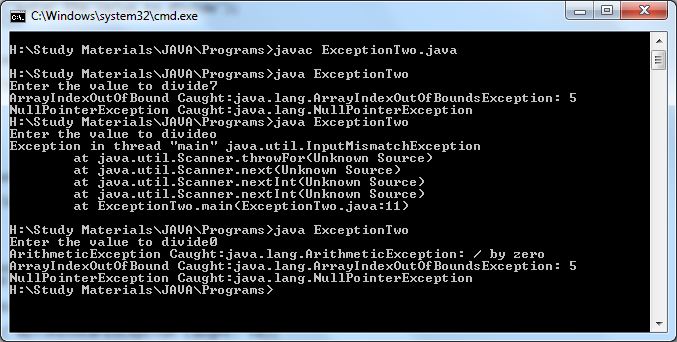
System.out.print("NullPointerException Caught:"+e);

}

}

}

**Output:**



1. **Given an array of ints, return the number of 9's in the array.**

**Examples**

**arrayCount9([1, 2, 9]) → 1**

**arrayCount9([1, 9, 9]) → 2**

**arrayCount9([1, 9, 9, 3, 9]) → 3**

**Program:**

import java.util.\*;

public class Solution{

public static void main(String arg[]){

int result=0;

Scanner s= new Scanner(System.in);

int len=s.nextInt();

int[] arr=new int[len];

for(int i=0;i<len;i++)

{

arr[i]=s.nextInt();

}

result=arrayCount9(arr);

System.out.println("count"+result);

}

public static int arrayCount9(int[] nums) {

int count = 0;

for(int i = 0; i < nums.length; i++) {

if(nums[i] == 9)

count++;

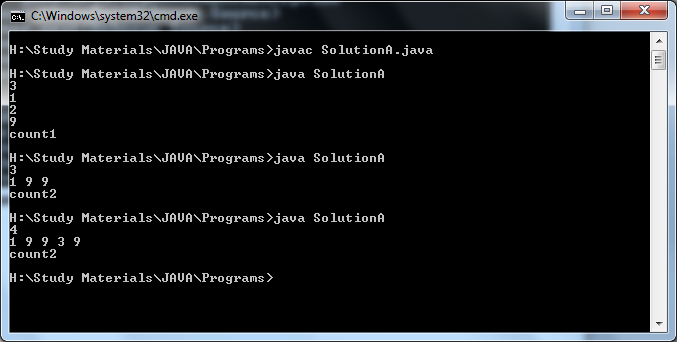
}

return count;

}

}

**Output:**



1. **Count the number of "xx" in the given string. We'll say that overlapping is allowed, so "xxx" contains 2 "xx".**

**Examples:**

**countXX("abcxx") → 1**

**countXX("xxx") → 2**

**countXX("xxxx") → 3**

**Program:**

import java.io.\*;

import java.util.\*;

public class Count{

static int countXX(String str)

{

int count = 0;

for(int i = 0; i < str.length() - 1; i++) {

if(str.substring(i, i + 2).equals("xx"))

count++;

}

return count;

}

public static void main(String[] args){

Scanner scan=new Scanner(System.in);

String str=scan.next();

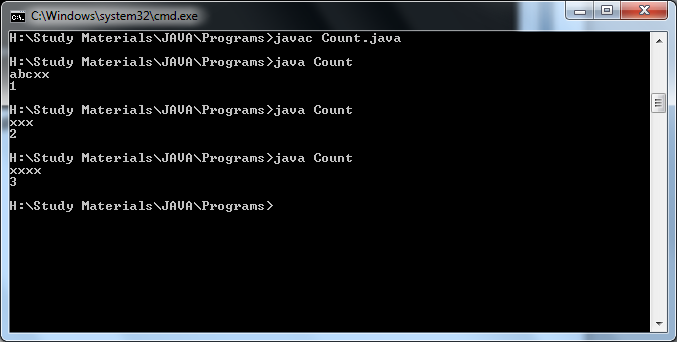
int k=countXX(str);

System.out.println(k);

}

}

**Output:**



1. **Given a string of even length, return the first half. So the string "WooHoo" yields "Woo".**

**Examples:**

**firstHalf("WooHoo") → "Woo"**

**firstHalf("HelloThere") → "Hello"**

**firstHalf("abcdef") → "abc"**

**Program:**

import java.io.\*;

import java.util.\*;

public class Solution

{

public static void main(String args[])

{

Scanner in=new Scanner(System.in);

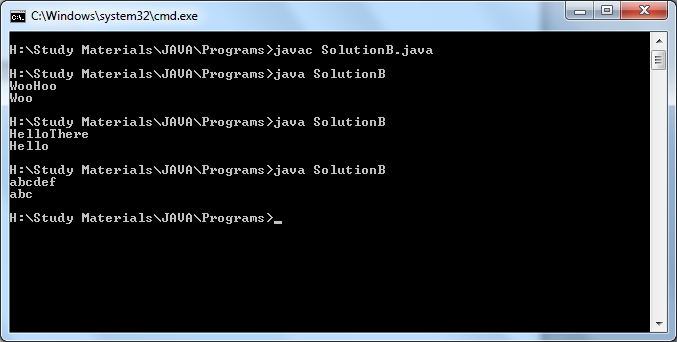
String str=in.next();

System.out.println(str.substring(0,str.length()/2));

}

}

**Output:**



1. **Given an array of ints length 3, return the sum of all the elements.**

**Examples:**

**sum3([1, 2, 3]) → 6**

**sum3([5, 11, 2]) → 18**

**sum3([7, 0, 0]) → 7**

**Program:**

import java.util.\*;

public class Solution{

public static void main(String arg[]){

int result=0;

Scanner s= new Scanner(System.in);

int[] arr=new int[4];

for(int i=0;i<3;i++)

{

arr[i]=s.nextInt();

}

result=arraysum(arr);

System.out.println("sum:"+result);

}

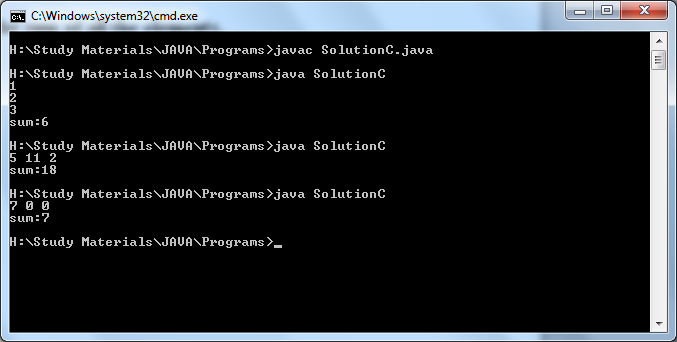
public static int arraysum(int[] nums) {

return nums[0]+nums[1]+nums[2];

}

}

**Output:**



1. **Given 2 ints, a and b, return their sum. However, "teen" values in the range 13..19 inclusive, are extra lucky. So if either value is a teen, just return 19.**

**Examples:**

**teenSum(3, 4) → 7**

**teenSum(10, 13) → 19**

**teenSum(13, 2) → 19**

**Program:**

import java.util.\*;

class Solution

{

public static void main(String args[])

{

Scanner sc=new Scanner(System.in);

int a=sc.nextInt();

int b=sc.nextInt();

int result;

result=teenSum(a,b,c);

System.out.println(result);

}

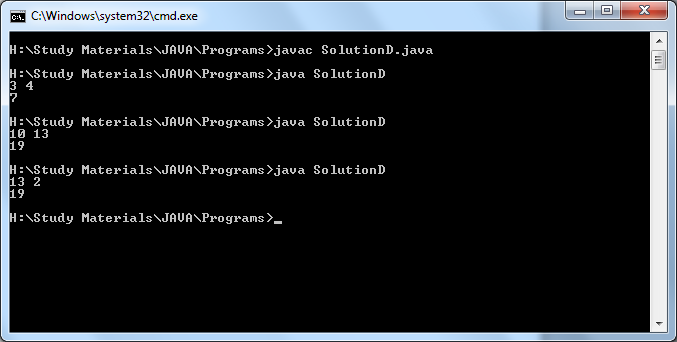
public static int teenSum(int a, int b) {

return (a >= 13 && a <= 19 || b >= 13 && b <= 19) ? 19 : a + b;

}

}

**Output:**



1. **Given two ints, each in the range 10..99, return true if there is a digit that appears in both numbers, such as the 2 in 12 and 23. (Note: division, e.g. n/10, gives the left digit while the % "mod" n%10 gives the right digit.)**

**Examples**

**shareDigit(12, 23) → true**

**shareDigit(12, 43) → false**

**shareDigit(12, 44) → false**

**Program:**

import java.util.\*;

class SolutionE

{

public static void main(String args[])

{

Scanner sc=new Scanner(System.in);

int a=sc.nextInt();

int b=sc.nextInt();

boolean result;

result=shareDigit(a,b);

System.out.println(result);

}

public static boolean shareDigit(int a, int b) {

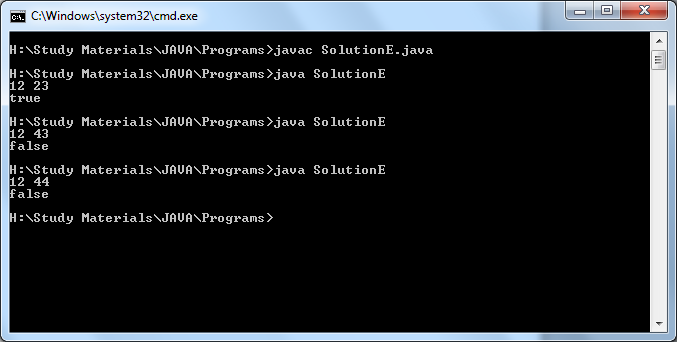
return (a % 10 == b % 10 || a / 10 == b / 10 ||

a % 10 == b / 10 || b % 10 == a / 10);

}

}

**Output:**



1. **Given 3 int values, a b c, return their sum. However, if one of the values is the same as another of the values, it does not count towards the sum.**

**Examples:**

**loneSum(1, 2, 3) → 6**

**loneSum(3, 2, 3) → 2**

**loneSum(3, 3, 3) → 0**

**Program:**

import java.util.\*;

import java.util.\*;

public class SolutionF{

public static void main(String arg[]){

Scanner scan=new Scanner(System.in);

int a=scan.nextInt();

int b=scan.nextInt();

int c=scan.nextInt();

int k=lonesum(a,b,c);

System.out.println(k);

}

public static int lonesum(int a,int b,int c)

{ if(a == b)

{

if(a == c)

return 0;

return c;

}

if(a == c)

return b;

if(b == c)

return a;

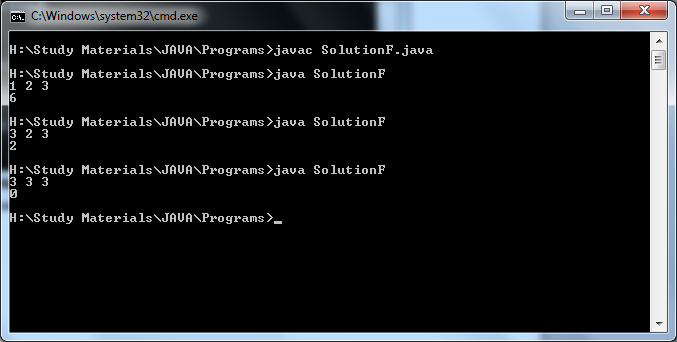
return(a + b + c);

}

}

}

**Output:**



1. **Modify and return the given map as follows: if the key "a" has a value, set the key "b" to have that value, and set the key "a" to have the value "". Basically "b" is a bully, taking the value and replacing it with the empty string.**

**Examples:**

**mapBully({"a": "candy", "b": "dirt"}) → {"a": "", "b": "candy"}**

**mapBully({"a": "candy"}) → {"a": "", "b": "candy"}**

**mapBully({"a": "candy", "b": "carrot", "c": "meh"}) → {"a": "", "b": "candy", "c": "meh"}**

**Program:**

import java.util.\*;

import java.io.\*;

class Map

{

public static void main(String args[])

{

Scanner s=new Scanner(System.in);

System.out.println("enter size");

int n=s.nextInt();

s.nextLine();

HashMap<String,String> map=new HashMap<String,String>();

for(int i=0;i<n;i++)

{

System.out.println("Enter key");

String z=s.next();

System.out.println("Enter value");

String z1=s.next();

map.put(z,z1);

}

map=mapBully(map);

System.out.println(map);

}

public static HashMap<String, String> mapBully(HashMap<String, String> map) {

if(map.containsKey("a"))

{

map.put("b",map.get("a"));

map.put("a","");

}return map;

}

}

**Output:**

