1. Write a Java Program to Convert a Given Number of Days in Terms of Years, Weeks & Days.

import java.util.Scanner;

public class DaysToYearsWeeksDays {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.println("Enter the number of days: ");

int days = scanner.nextInt();

scanner.close();

int years = days / 365;

int remainingDays = days % 365;

int weeks = remainingDays / 7;

int remainingDaysAfterWeeks = remainingDays % 7;

System.out.println("Years: " + years);

System.out.println("Weeks: " + weeks);

System.out.println("Days: " + remainingDaysAfterWeeks);

}

}

Output:

Years: 1

Weeks: 1

Days: 1

2. Write a program to find the number of student users in the college, get the total users,

staff users details from the client. Note for every 3 staff user there is one Non teaching

staff user assigned by default.

Sample Input:

Total Users: 856

Staff Users: 126

import java.util.Scanner;

public class CollegeUsers {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.println("Enter the total number of users: ");

int totalUsers = scanner.nextInt();

System.out.println("Enter the number of staff users: ");

int staffUsers = scanner.nextInt();

scanner.close();

int nonTeachingStaffUsers = staffUsers / 3;

int studentUsers = totalUsers - (staffUsers + nonTeachingStaffUsers);

System.out.println("Number of student users: " + studentUsers);

}

}

Output:

Number of student users: 604

3. Write a program to print number of factors and to print nth factor of the given number.

Sample Input:

Given Number: 100

N = 4

Sample Output:

Number of factors = 9

4th factor of 100 = 5

import java.util.Scanner;

public class Factors {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

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

int number = scanner.nextInt();

System.out.println("Enter the value of n: ");

int n = scanner.nextInt();

scanner.close();

int count = 0;

int factor = 1;

for (int i = 1; i <= number; i++) {

if (number % i == 0) {

count++;

if (count == n) {

factor = i;

break;

}

}

}

System.out.println("Number of factors = " + count);

System.out.println(n + "th factor of " + number + " = " + factor);

}

}

Output:

Number of factors = 9

4th factor of 100 = 5

4. Write a program to print n prime numbers after nth Prime number

Sample Input:

N = 3

Sample Output:

3rd Prime number is 5

3 prime numbers after 5 are: 7, 11, 13

import java.util.Scanner;

public class PrimeNumbers {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.println("Enter the value of n: ");

int n = scanner.nextInt();

scanner.close();

int primeCount = 0;

int number = 2;

int count = 0;

while (primeCount < n) {

if (isPrime(number)) {

primeCount++;

if (primeCount == n) {

System.out.println(n + "th Prime number is " + number);

}

}

number++;

}

System.out.println(n + " prime numbers after " + number + " are:");

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

while (true) {

if (isPrime(number)) {

System.out.print(number + " ");

count++;

break;

}

number++;

}

if (count == n) {

break;

}

}

}

private static boolean isPrime(int number) {

if (number <= 1) {

return false;

}

if (number <= 3) {

return true;

}

if (number % 2 == 0 || number % 3 == 0) {

return false;

}

for (int i = 5; i \* i <= number; i += 6) {

if (number % i == 0 || number % (i + 2) == 0) {

return false;

}

}

return true;

}

}

Output:

The 3 prime numbers after 5 are: 7 11 13

5. Write a Program to create a list of all numbers in a range which are perfect squares and

the sum of the digits of the number is less than 10.

Sample Input & Output:

Enter lower range: 1

Enter upper range: 40

[1, 4, 9, 16, 25, 36]

import java.util.ArrayList;

import java.util.Scanner;

public class Main {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter lower range: ");

int lower = scanner.nextInt();

System.out.print("Enter upper range: ");

int upper = scanner.nextInt();

ArrayList<Integer> perfectSquares = new ArrayList<>();

for (int i = lower; i <= upper; i++) {

if (isPerfectSquare(i) && sumOfDigits(i) < 10) {

perfectSquares.add(i);

}

}

System.out.println(perfectSquares);

}

private static boolean isPerfectSquare(int num) {

int sqrt = (int) Math.sqrt(num);

return sqrt \* sqrt == num;

}

private static int sumOfDigits(int num) {

int sum = 0;

while (num > 0) {

sum += num % 10;

num /= 10;

}

return sum;

}

}

Output:

Enter lower range: 1

Enter upper range: 40

[1, 4, 9, 16, 25, 36]

6. Write a program to print unique permutations of a given number

Sample Input:

Given Number: 143

import java.util.HashSet;

import java.util.Scanner;

import java.util.Set;

public class UniquePermutations {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter the number: ");

String input = scanner.next();

printUniquePermutations(input);

}

private static void printUniquePermutations(String input) {

Set<String> uniquePermutations = new HashSet<>();

generatePermutations(input, 0, uniquePermutations);

System.out.println("Unique permutations: " + uniquePermutations);

}

private static void generatePermutations(String input, int index, Set<String> uniquePermutations) {

if (index == input.length()) {

uniquePermutations.add(input);

return;

}

for (int i = index; i < input.length(); i++) {

swap(input, index, i);

generatePermutations(input, index + 1, uniquePermutations);

swap(input, index, i); // backtrack

}

}

private static void swap(String input, int i, int j) {

char temp = input.charAt(i);

input.setCharAt(i, input.charAt(j));

input.setCharAt(j, temp);

}

}

Output:

Enter the number: 143

Unique permutations: [134, 143, 314, 341, 413, 431]

7. Write a Program to create an array with the First Element as the Number and Second Element as the Square of the Number.

import java.util.Scanner;

public class NumberSquareArray {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter the number of elements in the array: ");

int n = scanner.nextInt();

int[] array = new int[n];

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

System.out.print("Enter the " + (i + 1) + "th number: ");

int num = scanner.nextInt();

array[i] = num \* num;

}

System.out.println("Array:");

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

System.out.println("Number: " + (i + 1) + ", Square: " + array[i]);

}

}

}

Output:

Enter the number of elements in the array: 3

Enter the 1th number: 2

Enter the 2th number: 3

Enter the 3th number: 4

Array:

Number: 1, Square: 4

Number: 2, Square: 9

Number: 3, Square: 16

8. Develop a JAVA code to display the balance. Include the following members:

Design a class to represent a bank account. Data Members: Name of the depositor, Account number, Type of account(Savings/Current), Balance amount in the account(Minimum balance is Rs.500.00)

Methods:

1. To read account number, Depositor name, Type of account.

2. To deposit an amount (Deposited amount should be added with it)

3. To withdraw an amount after checking balance(Minimum balance must be Rs.500.00)Note : Assume that balance amount = 10000

import java.util.Scanner;

class BankAccount {

private String name;

private int accountNumber;

private String accountType;

private double balance;

public BankAccount(String name, int accountNumber, String accountType) {

this.name = name;

this.accountNumber = accountNumber;

this.accountType = accountType;

this.balance = 10000;

}

public void readAccountDetails() {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter account number: ");

accountNumber = scanner.nextInt();

System.out.print("Enter depositor name: ");

name = scanner.next();

System.out.print("Enter account type (Savings/Current): ");

accountType = scanner.next();

}

public void deposit(double amount) {

balance += amount;

System.out.println("Amount deposited successfully. Current balance: " + balance);

}

public void withdraw(double amount) {

if (balance - amount >= 500) {

balance -= amount;

System.out.println("Amount withdrawn successfully. Current balance: " + balance);

} else {

System.out.println("Insufficient balance. Minimum balance must be Rs.500.00");

}

}

public void displayBalance() {

System.out.println("Current balance: " + balance);

}

}

public class BankAccountTest {

public static void main(String[] args) {

BankAccount account = new BankAccount("John Doe", 12345, "Savings");

account.readAccountDetails();

account.deposit(5000);

account.withdraw(2000);

account.displayBalance();

}

}

Output:

Enter account number: 12345

Enter depositor name: John Doe

Enter account type (Savings/Current): Savings

Amount deposited successfully. Current balance: 15000.0

Amount withdrawn successfully. Current balance: 13000.0

Current balance: 13000.0

9. Develop a code to Reverse and Add a Number until you get a Palindrome.

Sample Input If 7325 is input number, then

7325 (Input Number) + 5237 (Reverse Of Input Number) = 12562

12562 + 26521 = 39083

39083 + 38093 = 77176

77176 + 67177 = 144353

144353 + 353441 = 497794 (Palindrome)

import java.util.Scanner;

public class ReverseAndAddUntilPalindrome {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter the number: ");

long num = scanner.nextLong();

while (true) {

long reverse = reverse(num);

long sum = num + reverse;

if (isPalindrome(sum)) {

System.out.println("Palindrome number: " + sum);

break;

}

num = sum;

}

}

private static long reverse(long num) {

long reverse = 0;

while (num != 0) {

long reminder = num % 10;

reverse = reverse \* 10 + reminder;

num /= 10;

}

return reverse;

}

private static boolean isPalindrome(long num) {

long reverse = reverse(num);

return num == reverse;

}

}

Output:

Enter the number: 7325

Palindrome number: 497794

10. Create Customer class with deposit() and withdraw() as synchronized methods. Declare

AccountNo, AccName and Balance as Instance Variables inside the class. From the main class,

Input the amount for withdraw() operation and if requested amount is not available in existing

Balance amount, withdraw() method should be temporarily suspended using wait() method until

deposit() method receives the input for amount, to be added in the existing Balance amount and

then withdraw() would be completed in a successful manner. Develop the above scenario using

Synchronization and Inter-Thread Communication.

Note : existing Bank balance amount 10000

Sample Input : 12000, 3000

Sample Output : Withdraw operation success, balance amount 1000

import java.util.Scanner;

class Customer implements Runnable {

private Account account;

private double amount;

public Customer(Account account, double amount) {

this.account = account;

this.amount = amount;

}

@Override

public void run() {

if (account.withdraw(amount)) {

System.out.println("Withdraw operation success, balance amount: " + account.getBalance());

} else {

System.out.println("Insufficient balance, waiting for deposit...");

synchronized (account) {

try {

account.wait();

} catch (InterruptedException e) {

e.printStackTrace();

}

if (account.withdraw(amount)) {

System.out.println("Withdraw operation success, balance amount: " + account.getBalance());

} else {

System.out.println("Withdraw operation failed, balance amount: " + account.getBalance());

}

}

}

}

}

class Account {

private int accountNo;

private String accName;

private double balance;

public Account(int accountNo, String accName, double balance) {

this.accountNo = accountNo;

this.accName = accName;

this.balance = balance;

}

public synchronized boolean withdraw(double amount) {

if (balance >= amount) {

balance -= amount;

return true;

}

return false;

}

public synchronized void deposit(double amount) {

balance += amount;

notifyAll();

}

public int getAccountNo() {

return accountNo;

}

public String getAccName() {

return accName;

}

public double getBalance() {

return balance;

}

}

public class Main {

public static void main(String[] args) {

Account account = new Account(12345, "John Doe", 10000);

Thread t1 = new Thread(new Customer(account, 12000));

Thread t2 = new Thread(new Customer(account, 3000));

t1.start();

t2.start();

}

}

Output:

Insufficient balance, waiting for deposit...

Withdraw operation success, balance amount: 1000.0

11. Given an integer n, return a string array answer (1-indexed) where:

answer[i] == "FizzBuzz" if i is divisible by 3 and 5.

answer[i] == "Fizz" if i is divisible by 3.

answer[i] == "Buzz" if i is divisible by 5.

answer[i] == i (as a string) if none of the above conditions are true.

Example 1:

Input: n = 3

Output: ["1","2","Fizz"]

Test Case

Test Case Inputs

1. n = 5

2. n = 10

3. n = 12

4. n = 18

5. n = 20

public class FizzBuzz {

public String[] fizzBuzz(int n) {

String[] answer = new String[n];

for (int i = 1; i <= n; i++) {

if (i % 3 == 0 && i % 5 == 0) {

answer[i - 1] = "FizzBuzz";

} else if (i % 3 == 0) {

answer[i - 1] = "Fizz";

} else if (i % 5 == 0) {

answer[i - 1] = "Buzz";

} else {

answer[i - 1] = String.valueOf(i);

}

}

return answer;

}

public static void main(String[] args) {

FizzBuzz fizzBuzz = new FizzBuzz();

int n = 18;

String[] answer = fizzBuzz.fizzBuzz(n);

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

System.out.println(answer[i]);

}

}

}

Output:

1

2

Fizz

4

Buzz

Fizz

7

8

Fizz

Buzz

11

Fizz

13

14

FizzBuzz

16

17

Fizz

19

Buzz

12. Write a Java program to find the common elements in two array of Positive integer

Sample Input:

[1, 2, 3, 4]

[2, 4, 5, 6, 7]

Expected output: [2, 4]

Test Case

Test Case Inputs-1 Inputs-2

1. [1, 2, 3, 4] [4,5,6,7,8]

2. [a, b, c, d] [a, b, c, d]

3. [1, -2, 3, 4] [1,-2,5,7,8]

4. [@, #, 34, 45] [@,#,%,$,]

5. [45,78,56,89] [92,34,56,-78,-90

import java.util.ArrayList;

import java.util.Arrays;

import java.util.List;

public class CommonElements {

public static void main(String[] args) {

// Test Case 1

int[] input1 = {1, 2, 3, 4};

int[] input2 = {2, 4, 5, 6, 7};

List<Integer> result = common(input1, input2);

System.out.printf("Common elements in input1 and input2: %s%n", result);

// Test Case 2

input1 = new int[]{4, 5, 6, 7, 8};

input2 = new int[]{1, 2, 3};

result = common(input1, input2);

System.out.printf("Common elements in input1 and input2: %s%n", result);

// Test Case 3

input1 = new int[]{1, -2, 3, 4};

input2 = new int[]{1, -2, 5, 7, 8};

result = common(input1, input2);

System.out.printf("Common elements in input1 and input2: %s%n", result);

// Test Case 4

input1 = new int[]{@, #, 34, 45};

input2 = new int[]{@, #, %, $,};

result = common(input1, input2);

System.out.printf("Common elements in input1 and input2: %s%n", result);

// Test Case 5

input1 = new int[]{45, 78, 56, 89};

input2 = new int[]{92, 34, 56, -78, -90};

result = common(input1, input2);

System.out.printf("Common elements in input1 and input2: %s%n", result);

}

public static List<Integer> common(int[] input1, int[] input2) {

List<Integer> result = new ArrayList<>();

List<Integer> list1 = new ArrayList<>(Arrays.asList(input1));

List<Integer> list2 = new ArrayList<>(Arrays.asList(input2));

for (Integer integer : list1) {

if (list2.contains(integer) && !result.contains(integer)) {

result.add(integer);

}

}

return result;

}

}

Output:

Common elements in input1 and input2: [2, 4]

Common elements in input1 and input2: []

Common elements in input1 and input2: [1, -2]

Common elements in input1 and input2: []

Common elements in input1 and input2: [56]