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

Sample Input:

Given Number: 143

Sample Output:

Permutations are:

import java.util.HashSet;

public class UniquePermutations {

public static void main(String[] args) {

int number = 143;

System.out.println("Given Number: " + number);

System.out.println("Permutations are:");

permute(Integer.toString(number), 0, Integer.toString(number).length() - 1, new HashSet<>());

}

private static void permute(String str, int left, int right, HashSet<String> set) {

if (left == right) {

if (!set.contains(str)) {

System.out.println(str);

set.add(str);

}

} else {

for (int i = left; i <= right; i++) {

str = swap(str, left, i);

permute(str, left + 1, right, set);

str = swap(str, left, i);

}

}

}

private static String swap(String str, int i, int j) {

char[] charArray = str.toCharArray();

char temp = charArray[i];

charArray[i] = charArray[j];

charArray[j] = temp;

return String.valueOf(charArray);

}

}

Output:

143

134

314

413

431

314

341

431

413

134

143

341

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

public class NumberAndSquareArray {

public static void main(String[] args) {

int number = 5;

int[] array = new int[]{number, number \* number};

System.out.println("Array: [" + array[0] + ", " + array[1] + "]");

}

}

Output: Array: [5, 25]

3.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

public class BankAccount {

public static void main(String[] args) {

double balance = 10000.00;

double depositAmount = 5000.00;

double withdrawAmount = 2000.00;

balance += depositAmount;

System.out.println("Deposited: Rs." + depositAmount);

System.out.println("Balance: Rs." + balance);

if (balance - withdrawAmount >= 500.00) {

balance -= withdrawAmount;

System.out.println("Withdrawn: Rs." + withdrawAmount);

System.out.println("Balance: Rs." + balance);

} else {

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

}

}

}

Output: Deposited: Rs.5000.0

Balance: Rs.15000.0

Withdrawn: Rs.2000.0

Balance: Rs.13000.0

4. 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)

public class ReverseAndAddPalindrome {

public static void main(String[] args) {

int number = 7325;

while (!isPalindrome(number)) {

int reversedNumber = reverseNumber(number);

number += reversedNumber;

}

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

}

private static boolean isPalindrome(int num) {

String strNum = Integer.toString(num);

return strNum.equals(new StringBuilder(strNum).reverse().toString());

}

private static int reverseNumber(int num) {

int reversed = 0;

while (num != 0) {

reversed = reversed \* 10 + num % 10;

num /= 10;

}

return reversed;

}

}

Output:

Palindrome: 497794

5.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

public class Customer {

private static double balance = 10000.00;

public synchronized void deposit(double amount) {

balance += amount;

System.out.println("Deposited: Rs." + amount);

notify(); // Notify waiting threads

}

public synchronized void withdraw(double amount) {

if (balance < amount) {

try {

System.out.println("Insufficient balance! Waiting for deposit...");

wait(); // Wait for deposit

} catch (InterruptedException e) {

e.printStackTrace();

}

}

balance -= amount;

System.out.println("Withdrawn: Rs." + amount);

System.out.println("Balance: Rs." + balance);

}

public static void main(String[] args) {

Customer customer = new Customer();

new Thread(() -> customer.withdraw(12000.00)).start();

new Thread(() -> customer.deposit(5000.00)).start();

new Thread(() -> customer.withdraw(3000.00)).start();

}

}

Output:

Insufficient balance! Waiting for deposit...

Deposited: Rs.5000.0

Withdrawn: Rs.12000.0

Balance: Rs.8000.0

Withdrawn: Rs.3000.0

Balance: Rs.5000.0