\*\*\*\*\*Logical interview programs\*\*\*\*\*

Q.1.wtite a program fabonici series.

//Fibonacci Series using Recursion

class fibonacci

{

static int fib(int n)

{

if (n <= 1)

return n;

return fib(n-1) + fib(n-2);

}

public static void main (String args[])

{

int n = 9;

System.out.println(fib(n));

}

}

/\* This code is contributed by Pratik By Java coffee \*/

Output

34

Q.2.how to convert binary to decimal and decimal to binary conversion.

package com.java2novice.algos;

public class BinaryToDecimal {

public int getDecimalFromBinary(int binary){

int decimal = 0;

int power = 0;

while(true){

if(binary == 0){

break;

} else {

int tmp = binary%10;

decimal += tmp\*Math.pow(2, power);

binary = binary/10;

power++;

}

}

return decimal;

}

public static void main(String a[]){

BinaryToDecimal bd = new BinaryToDecimal();

System.out.println("11 ===> "+bd.getDecimalFromBinary(11));

System.out.println("110 ===> "+bd.getDecimalFromBinary(110));

System.out.println("100110 ===> "+bd.getDecimalFromBinary(100110));

}

}

Output:

11 ===> 3

110 ===> 6

100110 ===> 38

\*\*\*\*\*DecimalToBinary\*\*\*\*\*

package com.java2novice.algos;

public class DecimalToBinary {

public void printBinaryFormat(int number){

int binary[] = new int[25];

int index = 0;

while(number > 0){

binary[index++] = number%2;

number = number/2;

}

for(int i = index-1;i >= 0;i--){

System.out.print(binary[i]);

}

}

public static void main(String a[]){

DecimalToBinary dtb = new DecimalToBinary();

dtb.printBinaryFormat(25);

}

}

Output:

11001

Q.3.write a program to swap two numbers without using third variable.

import java.util.\*;

class Swap

{

public static void main(String a[])

{

System.out.println("Enter the value of x and y");

Scanner sc = new Scanner(System.in);

/\*Define variables\*/

int x = sc.nextInt();

int y = sc.nextInt();

System.out.println("before swapping numbers: "+x +" "+ y);

/\*Swapping\*/

x = x + y;

y = x - y;

x = x - y;

System.out.println("After swapping: "+x +" " + y);

}

}

Output:

Enter the value of x and y

23 43

before swapping numbers: 23 43

After swapping: 43 23

Q.4.write a program factorial number.

// Java program to find factorial of given number

class Test

{

// method to find factorial of given number

static int factorial(int n)

{

if (n == 0)

return 1;

return n\*factorial(n-1);

}

// Driver method

public static void main(String[] args)

{

int num = 5;

System.out.println("Factorial of "+ num + " is " + factorial(5));

}

}

Output:

Factorial of 5 is 120

Q.5. write a program to check the given number is palindrome or not.

// Java program to check whether a number

// is Palindrome or not.

class GFG

{

/\* Iterative function to reverse digits of num\*/

static int reverseDigits(int num)

{

int rev\_num = 0;

while (num > 0) {

rev\_num = rev\_num \* 10 + num % 10;

num = num / 10;

}

return rev\_num;

}

/\* Function to check if n is Palindrome\*/

static int isPalindrome(int n)

{

// get the reverse of n

int rev\_n = reverseDigits(n);

// Check if rev\_n and n are same or not.

if (rev\_n == n)

return 1;

else

return 0;

}

/\*Driver program to test reversDigits\*/

public static void main(String []args)

{

int n = 4562;

System.out.println("Is" + n + "a Palindrome number? -> " +

(isPalindrome(n) == 1 ? "true" : "false"));

n = 2002;

System.out.println("Is" + n + "a Palindrome number? -> " +

(isPalindrome(n) == 1 ? "true" : "false"));

}

}

// This code is contributed

// by Hritik Raj ( ihritik )

Output:

Is 4562 a Palindrome number? -> false

Is 2002 a Palindrome number? -> true

6.> write a program a given number is prime or not.

package com.java2novice.algos;

public class MyPrimeNumCheck {

public boolean isPrimeNumber(int number){

for(int i=2; i<=number/2; i++){

if(number % i == 0){

return false;

}

}

return true;

}

public static void main(String a[]){

MyPrimeNumCheck mpc = new MyPrimeNumCheck();

System.out.println("Is 17 prime number? "+mpc.isPrimeNumber(17));

System.out.println("Is 19 prime number? "+mpc.isPrimeNumber(19));

System.out.println("Is 15 prime number? "+mpc.isPrimeNumber(15));

}

}

Output:

Is 17 prime number? true

Is 19 prime number? true

Is 15 prime number? false

7.> write a program to check number is armstrong or not

Description: Armstrong numbers are the sum of their own digits to the power of the number of digits. It is also known as narcissistic numbers.

package com.java2novice.algos;

public class MyArmstrongNumber {

public boolean isArmstrongNumber(int number){

int tmp = number;

int noOfDigits = String.valueOf(number).length();

int sum = 0;

int div = 0;

while(tmp > 0)

{

div = tmp % 10;

int temp = 1;

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

temp \*= div;

}

sum += temp;

tmp = tmp/10;

}

if(number == sum) {

return true;

} else {

return false;

}

}

public static void main(String a[]){

MyArmstrongNumber man = new MyArmstrongNumber();

System.out.println("Is 371 Armstrong number? "+man.isArmstrongNumber(371));

System.out.println("Is 523 Armstrong number? "+man.isArmstrongNumber(523));

System.out.println("Is 153 Armstrong number? "+man.isArmstrongNumber(153));

}

}

Output:

Is 371 Armstrong number? true

Is 523 Armstrong number? false

Is 153 Armstrong number? true

8.> how to write custom exception in java.

package com.myjava.exceptions;

public class MyOwnException {

public static void main(String[] a){

try{

MyOwnException.myTest(null);

} catch(MyAppException mae){

System.out.println("Inside catch block: "+mae.getMessage());

}

}

static void myTest(String str) throws MyAppException{

if(str == null){

throw new MyAppException("String val is null");

}

}

}

class MyAppException extends Exception {

private String message = null;

public MyAppException() {

super();

}

public MyAppException(String message) {

super(message);

this.message = message;

}

public MyAppException(Throwable cause) {

super(cause);

}

@Override

public String toString() {

return message;

}

@Override

public String getMessage() {

return message;

}

}

Example Output

Inside catch block: String val is null

9.> how to create immutable class.

Immutable class means that once an object is created, we cannot change its content. In Java, all the wrapper classes (like String, Boolean, Byte, Short) and String class is immutable. We can create our own immutable class as well.

Following are the requirements:

• Class must be declared as final (So that child classes can’t be created)

• Data members in the class must be declared as final (So that we can’t change the value of it after object creation)

• A parameterized constructor

• Getter method for all the variables in it

• No setters(To not have option to change the value of the instance variable)

Example to create Immutable class

// An immutable class

public final class Student

{

final String name;

final int regNo;

public Student(String name, int regNo)

{

this.name = name;

this.regNo = regNo;

}

public String getName()

{

return name;

}

public int getRegNo()

{

return regNo;

}

}

// Driver class

class Test

{

public static void main(String args[])

{

Student s = new Student("ABC", 101);

System.out.println(s.getName());

System.out.println(s.getRegNo());

// Uncommenting below line causes error

// s.regNo = 102;

}

}

Output :

ABC

101

10.> how to create singleton class.

Description:

Singleton class means you can create only one object for the given class. You can create a singleton class by making its constructor as private,

so that you can restrict the creation of the object. Provide a static method to get instance of the object,

wherein you can handle the object creation inside the class only.

In this example we are creating object by using static block.

package com.java2novice.algos;

public class MySingleton {

private static MySingleton myObj;

static{

myObj = new MySingleton();

}

private MySingleton(){

}

public static MySingleton getInstance(){

return myObj;

}

public void testMe(){

System.out.println("Hey.... it is working!!!");

}

public static void main(String a[]){

MySingleton ms = getInstance();

ms.testMe();

}

}

11.>how to create deadlock between two threads.

Description:

Deadlock describes a situation where two or more threads are blocked forever, waiting for each other. Deadlocks can occur in Java when the synchronized keyword causes the executing thread to block while waiting to get the lock, associated with the specified object. Since the thread might already hold locks associated with other objects, two threads could each be waiting for the other to release a lock. In such case, they will end up waiting forever.

package com.java2novice.algos;

public class MyDeadlock {

String str1 = "Java";

String str2 = "UNIX";

Thread trd1 = new Thread("My Thread 1"){

public void run(){

while(true){

synchronized(str1){

synchronized(str2){

System.out.println(str1 + str2);

}

}

}

}

};

Thread trd2 = new Thread("My Thread 2"){

public void run(){

while(true){

synchronized(str2){

synchronized(str1){

System.out.println(str2 + str1);

}

}

}

}

};

public static void main(String a[]){

MyDeadlock mdl = new MyDeadlock();

mdl.trd1.start();

mdl.trd2.start();

}

}

12.> write a program to implement hashcode() and equals().

Description: The hashcode of a Java Object is simply a number, it is 32-bit signed int, that allows an object to be managed by a hash-based data structure. We know that hash code is an unique id number allocated to an object by JVM. But actually speaking, Hash code is not an unique number for an object. If two objects are equals then these two objects should return same hash code. So we have to implement hashcode() method of a class in such way that if two objects are equals, ie compared by equal() method of that class, then those two objects must return same hash code. If you are overriding hashCode you need to override equals method also. The below example shows how to override equals and hashcode methods. The class Price overrides equals and hashcode. If you notice the hashcode implementation, it always generates unique hashcode for each object based on their state, ie if the object state is same, then you will get same hashcode. A HashMap is used in the example to store Price objects as keys. It shows though we generate different objects, but if state is same, still we can use this as key.

package com.java2novice.algos;

import java.util.HashMap;

public class MyHashcodeImpl {

public static void main(String a[]){

HashMap<Price, String> hm = new HashMap<Price, String>();

hm.put(new Price("Banana", 20), "Banana");

hm.put(new Price("Apple", 40), "Apple");

hm.put(new Price("Orange", 30), "Orange");

//creating new object to use as key to get value

Price key = new Price("Banana", 20);

System.out.println("Hashcode of the key: "+key.hashCode());

System.out.println("Value from map: "+hm.get(key));

}

}

class Price{

private String item;

private int price;

public Price(String itm, int pr){

this.item = itm;

this.price = pr;

}

public int hashCode(){

System.out.println("In hashcode");

int hashcode = 0;

hashcode = price\*20;

hashcode += item.hashCode();

return hashcode;

}

public boolean equals(Object obj){

System.out.println("In equals");

if (obj instanceof Price) {

Price pp = (Price) obj;

return (pp.item.equals(this.item) && pp.price == this.price);

} else {

return false;

}

}

public String getItem() {

return item;

}

public void setItem(String item) {

this.item = item;

}

public int getPrice() {

return price;

}

public void setPrice(int price) {

this.price = price;

}

public String toString(){

return "item: "+item+" price: "+price;

}

}

Output:

In hashcode

In hashcode

In hashcode

In hashcode

Hashcode of the key: 1982479637

In hashcode

In equals

Value from map: Banana

13.> how sort element using comparable and comparator.

public class BadExampleOfComparable {

public static void main(String... args) {

List<SimpsonCharacter> characters = new ArrayList<>();

SimpsonCharacter homer = new SimpsonCharacter("Homer") {

@Override

public int compareTo(SimpsonCharacter simpson) {

return this.name.length() - (simpson.name.length());

}

};

SimpsonCharacter moe = new SimpsonCharacter("Moe") {

@Override

public int compareTo(SimpsonCharacter simpson) {

return this.name.length() - (simpson.name.length());

}

};

characters.add(homer);

characters.add(moe);

Collections.sort(characters);

System.out.println(characters);

}

As you can see, this code is complicated and includes a lot of repetition. We had to override the compareTo() method twice for the same logic. If there were more elements we would have to replicate the logic for each object.

Fortunately, we have the Comparator interface, which lets us detach the compareTo() logic from Java classes. Consider the same example above rewritten using Comparator:

public class GoodExampleOfComparator {

public static void main(String... args) {

List<SimpsonCharacter> characters = new ArrayList<>();

SimpsonCharacter homer = new SimpsonCharacter("Homer");

SimpsonCharacter moe = new SimpsonCharacter("Moe");

characters.add(homer);

characters.add(moe);

Collections.sort(characters, (Comparator.<SimpsonCharacter>

comparingInt(character1 -> character1.name.length())

.thenComparingInt(character2 -> character2.name.length())));

System.out.println(characters);

}

}

These examples demonstrate the main difference between Comparable and Comparator.Use Comparable when there is a single, default comparison for your object. Use Comparatorwhen you need to work around an existing compareTo(), or when you need to use specific logic in a more flexible way. Comparator detaches the sorting logic from your object and contains the compareTo() logic within your sort() method.