Week5:

Write a Java program that implements a multi-thread application that has three threads. First thread generates random integer every 1 second and if the value is even, second thread computes the square of the number and prints. If the value is odd, the third thread will print the value of cube of the number.

import java.util.Random;

class RandNumThrd extends Thread

{

public void run()

{

Random r = new Random();

for (int i = 0; i < 10; i++)

{

int randInt = r.nextInt(100);

System.out.println("Random Integer generated : " + randInt);

if((randInt%2) == 0)

{

SquareThread sThread = new SquareThread(randInt);

sThread.start();

}

else {

CubeThread cThread = new CubeThread(randInt);

cThread.start();

}

try

{

Thread.sleep(1000);

}

catch (InterruptedException ex)

{

System.out.println(ex);

}

}

}

}

class SquareThread extends Thread

{

int number;

SquareThread(int randomNumbern)

{

number = randomNumbern;

}

public void run() {

System.out.println("Square of " + number + " = " + (number \* number));

}

}

class CubeThread extends Thread

{

int number;

CubeThread(int randomNumber)

{

number = randomNumber;

}

public void run() {

System.out.println("Cube of " + number + " = " + number \* number \* number);

}

}

public class MultiThreadingTest

{

public static void main(String args[])

{

RandNumThrd rnThread = new RandNumThrd();

rnThread.start();

}

}

Output:

Week6:

Write a Java program for the following:

Create a doubly linked list of elements.

Delete a given element from the above list.

Display the contents of the list after deletion.

import java.util.\*;

public class DoubleLinkedList

{

public static void main(String[] args)

{

int i,ch,element,position;

LinkedList<Integer> dblList = new LinkedList<Integer>();

System.out.println("1.Insert element at begining");

System.out.println("2.Insert element at end");

System.out.println("3.Insert element at position");

System.out.println("4.Delete a given element");

System.out.println("5.Display elements in the list");

System.out.println("6.Exit");

Scanner sc=new Scanner(System.in);

do {

System.out.print("Choose your choice(1 - 6) :");

ch=sc.nextInt();

switch(ch) {

case 1: // To read element form the user

System.out.print("Enter an element to insert at begining : ");

element=sc.nextInt();

// to add element to doubly linked list at begining

dblList.addFirst(element);

System.out.println("Successfully Inserted");

break;

case 2: // To read element form the user

System.out.print("Enter an element to insert at end : ");

element=sc.nextInt();

// to add element to doubly linked list at end

dblList.addLast(element);

System.out.println("Successfully Inserted");

break;

case 3: // To read position form the user

System.out.print("Enter position to insert element : ");

position=sc.nextInt();

// checks if the position is lessthan or equal to list size.

if(position<=dblList.size()) {

// To read element

System.out.print("Enter element : ");

element=sc.nextInt();

// to add element to doubly linked list at given position

dblList.add(position,element);

System.out.println("Successfully Inserted");

}

else {

System.out.println("Enter the size between 0 to"+dblList.size());

}

break;

case 4: // To read element form the user to remove

System.out.print("Enter element to remove : ");

Integer ele\_rm;

ele\_rm=sc.nextInt();

if (dblList.contains(ele\_rm)){

dblList.remove(ele\_rm);

System.out.println("Successfully Deleted");

Iterator itr=dblList.iterator();

System.out.println("Elements after deleting :"+ele\_rm);

while(itr.hasNext()) {

System.out.print(itr.next()+"<->");

}

System.out.println("NULL");

}

else {

System.out.println("Element not found");

}

break;

case 5: // To Display elements in the list

Iterator itr=dblList.iterator();

System.out.println("Elements in the list :");

while(itr.hasNext()) {

System.out.print(itr.next()+"<->");

}

System.out.println("NULL");

break;

case 6: System.out.println("Program terminated");

break;

default:System.out.println("Invalid choice");

}

}

while(ch!=6);

}}

**Week 9:**

Suppose that a table named Table.txt is stored in a text file. The first line in the file is the header, and the remaining lines correspond to rows in the table. The elements are separated by commas. Write a java program to display the table using Labels in Grid Layout.

import java.io.\*;

import java.util.\*;

import java.awt.\*;

import java.awt.event.\*;

import javax.swing.\*;

import javax.swing.event.\*;

class Text\_To\_Table extends JFrame

{

public void convertTexttotable()

{

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

setSize(400,300);

GridLayout g = new GridLayout(0, 4);

setLayout(g);

try

{

FileInputStream fis = new FileInputStream("./Table.txt");

Scanner sc = new Scanner(fis);

String[] arrayList;

String str;

while (sc.hasNextLine())

{

str = sc.nextLine();

arrayList = str.split(",");

for (String i : arrayList)

{

add(new Label(i));

}

}

}

catch (Exception ex) {

ex.printStackTrace();

}

setVisible(true);

setTitle("Display Data in Table");

}

}

public class TableText

{

public static void main(String[] args)

{

Text\_To\_Table tt = new Text\_To\_Table();

tt.convertTexttotable();

}

}

Table.txt

NAME,NUMBER,MARKS,RESULT

RAJU,501,544,PASS

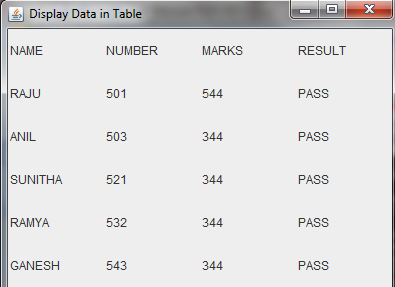
ANIL,503,344,PASS

SUNITHA,521,344,PASS

RAMYA,532,344,PASS

GANESH,543,344,PASS

OUTPUT



**Week – 13**

Write a Java program to list all the files in a directory including the files present in all its subdirectories.

import java.util.Scanner;

import java.io.\*;

public class ListingFiles

{

public static void main(String[] args)

{

String path = null;

Scanner read = new Scanner(System.in);

System.out.print("Enter the root directory name: ");

path = read.next() + ":\\";

File f\_ref = new File(path);

if (!f\_ref.exists()) {

printLine();

System.out.println("Root directory does not exists!");

printLine();

} else {

String ch = "y";

while (ch.equalsIgnoreCase("y")) {

printFiles(path);

System.out.print("Do you want to open any sub-directory (Y/N): ");

ch = read.next().toLowerCase();

if (ch.equalsIgnoreCase("y"))

{

System.out.print("Enter the sub-directory name: ");

path = path + "\\\\" + read.next();

File f\_ref\_2 = new File(path);

if (!f\_ref\_2.exists())

{

printLine();

System.out.println("The sub-directory does not exists!");

printLine();

int lastIndex = path.lastIndexOf("\\");

path = path.substring(0, lastIndex);

}

}

}

}

System.out.println("\*\*\*\*\* Program Closed \*\*\*\*\*");

}

public static void printFiles(String path)

{

System.out.println("Current Location: " + path);

File f\_ref = new File(path);

File[] filesList = f\_ref.listFiles();

for (File file : filesList)

{

if (file.isFile())

System.out.println("- " + file.getName());

else

System.out.println("> " + file.getName());

}

}

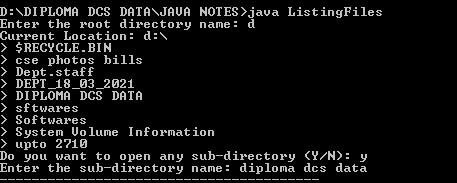
public static void printLine() {

System.out.println("----------------------------------------");

}

}

Output:



Week – 12

Write a Java program that correctly implements the producer – consumer problem using the concept of interthread communication.

class ItemQueue

{

int item;

boolean valueSet = false;

synchronized int getItem()

{

while (!valueSet)

try {

wait();

}

catch (InterruptedException e)

{

System.out.println("InterruptedException caught");

}

System.out.println("Consummed:" + item);

valueSet = false;

try

{

Thread.sleep(1000);

} catch (InterruptedException e)

{

System.out.println("InterruptedException caught");

}

notify();

return item;

}

synchronized void putItem(int item)

{

while (valueSet)

try {

wait();

} catch (InterruptedException e)

{

System.out.println("InterruptedException caught");

}

this.item = item;

valueSet = true;

System.out.println("Produced: " + item);

try {

Thread.sleep(1000);

}

catch (InterruptedException e)

{

System.out.println("InterruptedException caught");

}

notify();

}

}

class Producer implements Runnable

{

ItemQueue itemQueue;

Producer(ItemQueue itemQueue)

{

this.itemQueue = itemQueue;

new Thread(this, "Producer").start();

}

public void run()

{

int i = 0;

while(true)

{

itemQueue.putItem(i++);

}

}

}

class Consumer implements Runnable

{

ItemQueue itemQueue;

Consumer(ItemQueue itemQueue)

{

this.itemQueue = itemQueue;

new Thread(this, "Consumer").start();

}

public void run() {

while(true) {

itemQueue.getItem();

}

}

}

class ProducerConsumer

{

public static void main(String args[])

{

ItemQueue itemQueue = new ItemQueue();

new Producer(itemQueue);

new Consumer(itemQueue);

}

}

Output:

