1. În următorul program Java vi se cere să determinați și să explicați:
   1. rezultatele afișate;
   2. rezultatele afișate, daca linia //sem\_synchr este stearsă;
   3. rezultatele afișate, daca linia //sem\_synchr este stearsă iar void display() se înlocuiește prin synchronized void display().

class Fir extends Thread{

public Fir (int i){id=i;}

public void run() {

display();

}

void display() {

synchronized(sem) // synchr sem

{ x=id;

System.out.println(x);

}

}

static int x=0;

static Object sem=new Object();

int id;

}

public class ExecFir{

public static void main (String[] args){

Fir f1, f2;

f1= new Fir(1); f1.start();

f2= new Fir(2); f2.start();

while(f1.isAlive() || f2.isAlive()){}

System.out.println("The end");

}

}

1. În programul următor,
   1. supradefiniți operatorul << astfel încât cout<<i să afișeze valoarea atributului i.x;
   2. precizați și explicați rezultatele afișate la executarea programului astfel obținut.

#include <iostream.h>

class C{

public:

C(int i=0){x=i;}

C& operator++(){++x; return \*this;}

C operator--(){--x; return \*this;}

private:

int x;

};

void main(){

C i;

cout<<i<<endl;

cout<<++(++i)<<endl<<i<<endl;

cout<<--(--i)<<endl<<i<<endl;

}

1. Explicați rezultatele afișate prin executarea următorului program C++:

class A{

public:

virtual void v(){

cout<<"A::v()"<<endl;

s(); w();

}

protected:

void s(){cout<<"A::s()"<<endl;}

virtual void w(){cout<<"A::w()"<<endl;}

}

class B:public A{

public:

virtual void v(){

cout<<"B::v()"<<endl;

s();w();

}

protected:

virtual void w(){cout<<"B::w()"<<endl;}

};

void main(){

cout << "Start" << endl;

A \*a; B \*b; b= new B(); a=b; a->v();

a=(A \*)b; a->v();

((A)(\*b)).v();

}

4. În următorul program Java, adăugați o clasă numită Impl astfel încât executarea acestui program să afișeze mesajele:

Valoare corecta, mai mica decat 100

Valoare incorectă, mai mare decat 100

(Observatie: Nu sunt permise modificări în clasa Main sau în interfața Interf)

public class Main{

public static void main(String[] args){

Interf ma= new Impl();

try{

ma.verify(99);

ma.verify(101);

}catch(Exception e){System.out.println("Valoare incorecta, mai mare decat 100");}

}

}

interface Interf{

public void verify(int i) throws Exception;

}

//class Impl

1. Descrieți arhitectura (design pattern) programului următor și explicați rezultatele afișate:

import java.util.\*;

class Obsv extends Observable{

public void changeData(String year){

data=year; setChanged();

notifyObservers(data);

}

private String data="2015";

}

class Observator implements Observer{

public Observator(String s){id=s;}

public void update(Observable o, Object arg){

System.out.println(id+": year ="+ arg);

}

private String id;

}

public class ObsrObsv{

public static void main(String[] args){

Obsv m= new Obsv();

Observator a,b,c;

a=new Observator("A");

b=new Observator("B");

c= new Observator("C");

m.addObserver(a);

m.addObserver(c);

m.changeData("2016");

m.deleteObserver(c);

m.changeData("2017");

}

}