Segundo Certamen

En este certamen usted no podrá hacer preguntas. Si algo no está claro, haga una suposición razonable, anótela en su respuesta y resuelva conforme a ello.

Primera parte, sin apuntes (32 minutos; 32 puntos):

1.- Responda brevemente y entregue en hoja con su nombre.

a) ¿Qué diferencias existe entre crear un archivo .jar de un applet y aquel de una aplicación Java?.
El archivo .jar de una aplicación obliga especificar un archivo manifiesto que indica la clase que incluye el método main.

b) Mencione dos restricciones o limitaciones de las applets introducidas por razones de seguridad.
Las applets no pueden escribir o leer datos del disco de la máquina local.
Applets no pueden establecer conexiones con máquinas distintas del servidor origen del applet.
Applets no pueden correr programas en la máquina local.

c) Mencione dos ventajas de la metodología de desarrollo iterativa e incremental.
Es posible mostrar al cliente versiones parciales del sistema y así validar el desarrollo.
Cada etapa permite entender mejor la naturaleza del problema y hacer las mejoras o correcciones necesarias a tiempo.

d) Alguien dice que los actores de un caso de uso siempre son personas. Diga si es verdadero o falso y justifique.
Falso, actores de un sistema también pueden ser sensores o módulos de software externos al sistema bajo desarrollo.

e) En java usamos super(...) para invocar un constructor de la clase base. ¿Cómo se hace esto en C++?
En la lista de inicialización se debe pone el nombre de la clase con los parámetros del constructor que se desea invocar.

f) En java usamos super.metodo(..) para invocar un método de la clase base que hemos redefinido en la clase derivada. ¿Cómo se hace esta invocación en C++? Dé un ejemplo.
Se usa el nombre de la clase base seguida del método como en:
ClaseBase::metodo(...)

g) Se tiene una clase con prototipo foo(ClaseA **a**); Señale qué método(s) se invoca(n) para que el argumento **a** tome el valor del parámetro **"objeto"** usado en la invocación foo(objeto).
Para inicializar a, se usa el constructor como en: ClaseA a(objeto);

h) ¿Qué es un método virtual puro? ¿Para qué se sirven?
Es un método virtual sin implementación en la clase base, lo cual se indica con =0 al final del prototipo del método.
Sirven para listar métodos en la clase base que sólo tiene sentido implementar en las clases derivadas. Es necesario listarlos en la clase base para crear código donde usamos punteros o referencias de la clase base para invocar esos métodos y así crear código reusable.

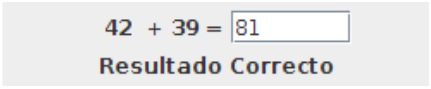
Segunda Parte, con apuntes (68 minutos)

2.- (24 puntos) En la documentación de la clase Math se tiene:

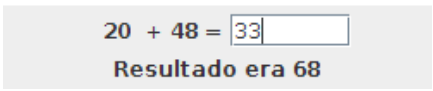
static double random()	Returns a double value with a positive sign, greater than or equal to 0.0 and less than 1.0.
static long round(double a)	Returns the closest long to the argument.

Haga un applet que muestre dos números aleatorios como muestra la imagen a) y espere por una respuesta del usuario. Ante una suma correcta, el applet responde como muestra la imagen b). Ante una respuesta errada, el applet responde como se muestra la imagen c). El rango para los números aleatorias varia desde 0 a Rango, donde Rango es un parámetro definido en el archivo html. Use el "layout manager" por omisión.

a)



b)



c)

```
import java.awt.event.*;
import javax.swing.*;

public class SumaApplet extends JApplet
{
  private JLabel op1,op2,msg;
  private long i1,i2;
  private JTextField suma;
  public void init()
  {
    op1 = new JLabel("");
    op2 = new JLabel("");
    suma = new JTextField(6);
    msg = new JLabel("");
    JPanel panel = new JPanel();
    panel.add(op1);
    panel.add(op2);
    panel.add(suma);
    panel.add(msg);
    getContentPane().add(panel);
    suma.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent event)
        {
          int isuma=Integer.parseInt(suma.getText());
          if ( isuma == (i1+i2))
            msg.setText("Resultado Correcto");
          else
            msg.setText("Resultado era " + (i1+i2));
        }
      });
  }
  public void start(){
    i1=Math.round(Math.random()*Integer.parseInt(getParameter("Rango")));
    i2=Math.round(Math.random()*Integer.parseInt(getParameter("Rango")));
    op1.setText("   "+ i1);
    op2.setText(" + "+ i2 + " =");
    suma.setText("");
    msg.setText("");
  }
}
```

3.- (24 puntos) Considere la clase Java Vector2D adjunta. Muestre los contenidos de los archivos Vector2D.h y Vector2D.cpp correspondientes a la versión C++ de la misma clase. Cambie los argumentos de los métodos si ello mejora la eficiencia de su versión C++ de la clase.

```
public class Vector2D {
  private double x, y;
  public Vector2D () {
    x = y = 0;
  }
  public Vector2D (double x, double y) {
    this.x = x;      this.y = y;
  }
  public Vector2D plus(Vector2D v) {  // suma de vectores. Ej. en C++: v3=v1+v2;
     if (v==null)   return new Vector2D(x,y);
     else   return new Vector2D(x+v.x, y+v.y);
  }
  public Vector2D times(double escalar) {  // multiplicación de escalar por vector Ej. en C++ v2= 5*v1;
    return new Vector2D(x*scalar, y*scalar);
  }
  public double module() {  // retorna la magnitud de un vector.  Ej. en C++ double m=v.module();
     return Math.sqrt(x*x+y*y);
  }
  public Vector2D unitary() {  // retorna un vector de magnitud 1 e igual dirección.  Ej. en C++ v2=v1.unitary();
    return times(1/module());
  }
}
```

En Vector2D.h

```
#ifndef VECTOR2D_H
#define VECTOR2D_H

class Vector2D {
private:
    double x, y;
public:
    Vector2D () { x = y = 0;};  // constructores simples los pongo inline
    Vector2D (double x, double y) {  // constructores simples los pongo inline
      this->x = x;      this->y = y;
    };
    Vector2D operator+(const Vector2D &v) const; // suma de vectores.
    double module() const; // retorna la magnitud de un vector.
    Vector2D unitary() const;  // retorna un vector de magnitud 1 e igual dirección.
    friend Vector2D operator*(double scalar, const Vector2D &v); // multiplicación escalar por vector
};
#endif
```

En Vector2D.cpp

```
#include "Vector2D.h"
#include <math.h>
Vector2D Vector2D::operator+(const Vector2D &v) const {  // suma de vectores.
  Vector2D suma(x+v.x, y+v.y);
  return suma;
}
double Vector2D::module() const {  // retorna la magnitud de un vector.
```

```
  return sqrt(x*x+y*y);
}
Vector2D Vector2D::unitary() const {  // retorna un vector de magnitud 1 e igual dirección.
  Vector2D u(x/module(), y/module());
  return u;
}
//Alternativamante:
/*
Vector2D Vector2D::unitary() const {  // retorna un vector de magnitud 1 e igual dirección.
  return (1/module())*(*this);
}
*/
// Función global
Vector2D operator* (double scalar, const Vector2D &v){ // multiplicación escalar por vector
  Vector2D r(scalar*(v.x), scalar*(v.y));
  return r;
}
```

4.- (20 puntos) Es común que debamos buscar el valor promedio entre los valores de un vector de elementos. En este caso con vector nos referimos a un contenedor de la biblioteca estándar de plantillas de C++.
a) Cree función plantilla (o template) ***promedio(..)*** en C++ para obtener el promedio de los elementos de un vector. Esta función recibe como parámetro un vector no nulo y como valor retornado entrega el promedio de los elementos del vector. Por ejemplo, podemos usar:
vector<double> vec; // y luego de ingresar varios doubles al vector, podemos hacer:
double p = promedio(vec);

En promedio.h podríamos tener:

```
#include <vector>
using namespace std;
template <class T>
T promedio(const vector<T> &v) {
  T sum=v[0];  // sabemos que el vector es no nulo
  for(int i=1; i<v.size(); i++)
    sum= sum+v[i];
  return sum/v.size();
}
```

Nota: Ésta no es la única solución.

b) Implemente las posibles modificaciones de la clase Vector2D de la pregunta 3 para obtener el promedio de un vector de Vectores2D usando su plantilla.
De esta implementación de la plantilla, se desprende que los elementos del vector deben admitir los siguientes métodos:
i) Constructor copia para permitir Vector2D sum=v[0]; En este caso el constructor por omisión, aquel con copia baja, es suficiente, luego no requiere adicional en Vector2D.
ii) sum= sum+v[i]; Vector2D debería sobrecargar el operador +, cosa que ya hace en pregunta 3.
iii) return suma/v.size(); Se requiere sobrecarga de operador / con argumento del mismo tipo retornado por size() de vector. El valor retornado usa el constructor copia, éste no requiere se implementado.
Es así que el único cambio se muestra en negrita.

```
#ifndef VECTOR2D_P4_H
```

```
#define VECTOR2D_P4_H
class Vector2D {
private:
    double x, y;
public:
    // aquí va lo mismo que en pregunta 3
    Vector2D operator/ (double f);   // es el único método para la implementación de promedio dada en a)
};
#endif
```

En archivo Vector2D.cpp se debe incluir el método:

```
Vector2D Vector2D::operator/(double f){
   Vector2D c(x/f,y/f);
   return c;
}
```