Segundo Certamen

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

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

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

a) Mencione una operación válida en una aplicación Java pero no en un applet (a menos que sea firmada).
Las aplicaciones Java pueden escribir en disco local, un applet no.

b) Las aplicaciones java reciben argumentos en la línea de comando al ser ejecutadas ¿Cómo es posible pasar valores a un applet?
Un applet puede recibir valores a través del rótulo html <param> del tipo:
<param name="año" value="2012"/ >
Luego pueden ser leídos con el método getParameter de la clase JApplet.

c) Mencione desventajas de la metodología de desarrollo de software en "cascada" frente al "iterativo e incremental".
"Cascada" no contempla obtener avances parciales ejecutables del sistema para ser validados por el cliente.
"Cascada" dificulta la identificación oportuna de problemas, por ejemplo derivados de una errada definición de requerimiento.
"Cascada" no facilita como "iterativo e incremental" el entendimiento del problema en la medida que el sistema se desarrolla.

d) Mencione dos objetivos buscados por las metodologías de desarrollo de software.
Ayuda: ¿Da lo mismo usar o no una metodología de desarrollo de software? Como no da lo mismo, ¿qué se busca con el uso de una metodología de desarrollo de software?
Se busca mantener los desarrollos de software dentro de los tiempo y costos acordados con el cliente y además cumplir adecuadamente con los requerimientos.

e) ¿Cuál es la razón de existencia o propósito de las certificaciones de software?
El propósito de las certificaciones de software es dar información a los clientes sobre cuán riguroso es el proceso seguido por una empresa para generar software de calidad.

f) ¿Qué es un caso de uso? ¿Para qué sirve?
Un caso de uso es una secuencia de acciones donde interactúa un usuario con el sistema a desarrollar para mostrar una funcionalidad específica del sistema.
Los casos de uso sirven para definir requerimientos del sistema y luego son usados en las etapas de análisis y al final para hacer las pruebas unitarias.

g) En C++ se tiene la función foo con el siguiente prototipo:
int foo(Student s);
Al invocar la función con int i=foo(juan); Indique el constructor invocado para crear el objeto s y cómo éste adopta el valor del parámetro actual.
Se invoca el constructor copia Student(const Student & s). Al invocar foo(juan), el objeto s se inicializa con s(juan);

h) ¿Es posible que el siguiente segmento de código se pueda compilar sin errores? Si no es posible, explique por qué. Si es posible, dé un ejemplo de declaración de foo.
….. // otras cosas sin errores de compilación.
int i;
foo(i) = 20;
….// otras cosas sin errores de compilación.
Sí es posible, una posible declaración es:
int & foo(int &i);
Nota: su código pudo ser:
int & foo(int &i) { return i}

Segunda Parte, con apuntes (68 minutos)

2.- a) (18 puntos) Complete el código adjunto y declaraciones e implementaciones para las clases Forma, Rectangulo y Circulo.

```
#include “Forma.h”
#include “Rectangulo.h”
#include “Circulo.h”
#include <vector>
#include <iostream>
using namespace std;
void main (int argc, char * argv[]) {
   double perimetro =0;
   vector<Forma*> formas;   /* por error puse pushBack en lugar de push_back */
   formas.push_back(new /* creamos un rectángulo en (1, 1) y lados 4 y 2 */ );
   formas.push_back(new /* creamos un círculo en (2, 1) y radio 1,5 */ );
   for(int i=0; i<formas.size(); i++)
     perimetro += formas[i]->getPerimetro();  /* por error puse . En lugar de ->*/
     cout << "Perimetro = " << perimetro;
   }
}

#ifndef FORMA_H          /*  Forma.h  */
#define FORMA_H
class Forma {
public:
     virtual float getPerimetro() const =0;
};
#endif

#ifndef RECTANGULO_H  /* Rectangulo.h */
#define RECTANGULO_H
#include "Forma.h"
class Rectangulo:public Forma {
public:
      Rectangulo();
      Rectangulo(int, int, int, int);
      virtual float getPerimetro() const;
private:
      int x, y;
      int width, hight;
};
#endif

#include "Rectangulo.h"                    /*  Rectangulo.cpp  */
Rectangulo::Rectangulo (int xx, int yy, int w, int h){
    x = xx;   y = yy;
    width = w; hight = h;
}
Rectangulo::Rectangulo (){
    x = y = width = hight =0;
}
float Rectangulo::getPerimetro() const{
   return 2*(width+hight);
}

#ifndef CIRCULO_H    /*  Circulo.h  */
#define CIRCULO_H
#include "Forma.h"
class Circulo : public Forma {
public:
```

```
      Circulo();
      Circulo(int xc, int yc, float r);
      virtual float getPerimetro() const ;
private:
      int x, y;
      float r;
};
#endif

#include "Circulo.h"           /*   Circulo.cpp   */
Circulo::Circulo () {
  x = y = 0;
  r = 0;
}
Circulo::Circulo(int xx, int yy, float radio){
  x = xx;   y = yy;   r = radio;
}
float Circulo::getPerimetro() const {
  return 2*3.1416*r;
}
```

b) (16 puntos) Cree la clase GrupoForma. Esta clase almacena un conjunto de formas, cada una de las cuales puede ser un rectángulo, un círculo o un conjunto de formas (GrupoForma). En la clase GrupoForma considere algún método para incorporar formas al conjunto y los necesarios para que instancias de GrupoForma puedan ser incorporadas al vector del código dado en a).

```
#ifndef GRUPO_FORMA_H            /* GrupoForma.h*/
#define GRUPO_FORMA_H
#include "Forma.h"
#include <vector>
using namespace std;
class GrupoForma : public Forma {
public:
   GrupoForma();
   virtual float getPerimetro() const;
   void addForma(Forma * f);
private:
   vector<Forma*> grupo;
};
#endif

#include "GrupoForma.h"            /*GrupoForma.cpp */
GrupoForma::GrupoForma() {
}
float GrupoForma::getPerimetro() const {
   float perimetro=0;
   for (int i=0; i < grupo.size(); i++)
       perimetro += grupo[i]->getPerimetro();
   return perimetro;
}

void GrupoForma::addForma( Forma * f) {
   grupo.push_back(f);
}
```

3.- (34 puntos) Para el siguiente código:

```
#include “Complejo.h”
using namespace std;
void main (int argc, char * argv[]) {
   Complejo z1(1,2), z2(3,4), z3;  // z1 = 1+2i  , z2 = 3+4i  , z3=0
   z3 = z1 + z2; // z3 = (suma de partes reales)+(suma de partes imaginarias)i
   cout << “z1 + z2 = ” << z3 << endl;
   z3 = z1 * z2; // z3=Re(z1)*Re(z2)-Im(z1)*Im(z2)+[(Re(z1)*Im(z2)+Im(z1)*Re(z2)]i
   cout << “z1 * z2 = ” << z3 << endl;
}
```

a) (18 puntos) Cree la clase Complejo para números complejos de manera que podamos correr código dado.

```
#ifndef COMPLEJO_H        /* Complejo.h */
#define COMPLEJO_H
#include <ostream>
using namespace std;
class Complejo {    /* Indico puntajes por definición+implementación */
  public:
    Complejo () {r=i=0;};
    Complejo (float, float);
    Complejo operator + (const Complejo &) const;
    Complejo operator * (const Complejo &) const;
    friend ostream & operator<< (ostream &os, const Complejo &z);
  private:
    float r, i;
};
#endif

#include "Complejo.h"             /*Complejo.cpp*/
Complejo::Complejo (float x, float y) {
  r = x;
  i = y;
}
Complejo Complejo::operator+ (const Complejo &z) const {
  Complejo temp;
  temp.r = r + z.r;
  temp.i = i + z.i;
  return temp;
}
Complejo Complejo::operator* (const Complejo &z) const {
  Complejo temp;
  temp.r = r * z.r - i * z.i;
  temp.i = r * z.i + i * z.r;
  return temp;
}
ostream & operator << (ostream &os, const Complejo &z) {
  os << z.r << " + " << z.i << "i ";
  return os;
}
```

b) (16 puntos) Para evaluar polinomios de números reales, se recomienda usar un código del estilo:

// polinomio = $a[0]+a[1]x+a[2]x^2+a[3]x^3+a[4]x^4...+a[n-1]x^{n-1}$

// polinomio = a[0]+(a[1]+(a[2]+(a[3]+(a[4]+(.. +(a[n-1])*x) ..)*x)*x)*x)*x

```
double evaluePolinomio(vector<double> a, double x) { // a: vector de coeficientes
   double result = 0;                                 // x: punto de evaluación.
   for (int i=a.size()-1; i >= 0; i--)
      result = (a[i] + result*x); //a[0]+(a[1]+(a[2]+(..+(a[size-1])*x)...)*x)*x
   return result;
}
```

Desarrolle una función plantilla (templete) para evaluar polinomios con coeficientes y dominios de igual tipo pero arbitrario (enteros, flotantes, Complejos, etc).

```
#ifndef EVALUE_POLINOMIO_H
#define EVALUE_POLINOMIO_H
#include <vector>
using namespace std;
template <class T>
T evaluePolinomio(vector<T> a, T x) { // a: vector de coeficientes
   T result;                          // x: punto de evaluación.
   for (int i=a.size()-1; i >= 0; i--)
      result= (a[i] + result*x); //a[0]+(a[1]+(a[2]+(..+(a[size-1])*x)...)*x)*x
   return result;
}
#endif
```