Diseño y Programación Orientados a Objetos

1er. Semestre 2025

Certamen II ELO-329 Diseño y Programación Orientada a Objetos (1s2025)

Nombre: _____
Paralelo: _____

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** (30 minutos):

Primera pregunta (30 pts.)

A. (5pts) Considere la siguiente clase:

```
class Empleado {
  double sueldo;
public:
  Empleado(double s){sueldo = s;}
  double getSalary() const {
    sueldo += 1000;
    return sueldo;
  }
};
```

Indique si el código anterior es válido. En caso afirmativo, escriba un ejemplo de uso para el método **getSalary()**. En caso contrario, justifique.

R:
El código **no es válido**. El método `getSalary()` está declarado como `const` (2pts), por lo que **no puede modificar el atributo** `sueldo`. La instrucción `sueldo += 1000` viola esta restricción, generando un **error de compilación** (3pts).

B. (5pts) A continuación se presentan tres fragmentos de código relacionados con la implementación de una clase **Producto** en C++.
Para cada fragmento, indique si este debería ubicarse en un archivo **.h** (archivo de cabecera) o en un archivo **.cpp** (archivo de implementación), y justifique brevemente su decisión.

a)

```
class Producto {
public:
  Producto(string nombre, double precio);
  double getPrecio() {return precio;}
  static int n_productos;
private:
  string nombre;
  double precio;
};
```

b)

```
Producto::Producto(string nombre, double precio) {
  this->nombre = nombre;
  this->precio = precio;
}
```

c)

```
int Producto::n_productos = 0;
```

R: Un archivo `.h` se usa para declarar clases y métodos, mientras que un archivo `.cpp` contiene implementaciones.

a) Archivo `.h` (2pts)
Declara la clase, su constructor, método y atributo estático. Va en el archivo de cabecera.

b) Archivo `.cpp` (2pts) Implementa el constructor. Debe ir en el archivo `.cpp`.

c) Archivo `.cpp` (1pt) Define el atributo estático. Se ubica en el `.cpp` para evitar múltiples definiciones.

C. (5pts) Complete las definiciones e implementaciones para que el código del **main** compile y se pueda ejecutar sin errores. Por cada fragmento de código adicional que agregue, indique el número de línea correspondiente y el contenido exacto que debería insertarse después de dicha línea.

```
1.  #include <vector>
2.  #include <iostream>
3.  class Manzana {
4.  public:
5.      Manzana(float peso): kg(peso) {}
6.
7.  private:
8.      static float precio;  // precio por Kg.
9.      float kg;             // peso en Kg.
10. };
11.
12. float Manzana::precio = 900;
13.
14. int main (){
15.     std::vector<Manzana *> p;
16.     p.push_back(new Manzana(2.0));
17.     float total = 0;
18.     for (int i = 0; i < p.size(); i++)
19.         total = total + *p[i];
20.     std::cout << "Valor a pagar: " << total << std::endl;  // debería
imprimir 1800 como total.
21.     return 0;
22. }
```

R:

```
// Luego de línea 5
friend float operator +(float i, const Manzana &m);   //  (3pts.)

// Luego de línea 12:

float operator + (float i, const Manzana &p){
  return i+p.kg*p.precio;                          // (2pts.)
}
```

D. (5pts) ¿Qué imprime el siguiente código?

```
#include <iostream>
using namespace std;

class A {
public:
    virtual void saludoVirtual() { cout << "Virtual A" << endl; }
    void saludo() { cout << "Saludo A" << endl; }
};

class B : public A {
public:
    void saludoVirtual() { cout << "Virtual B" << endl; }
    void saludo() { cout << "Saludo B" << endl; }
};

int main() {
    B b;
    A &a = b;
    A *pa = &b;
    b.saludoVirtual();
    a.saludoVirtual();
    a.saludo();
    pa->saludoVirtual();
    pa->saludo();
    return 0;
}
```

R: La salida es (1 pts cada una):

```
Virtual B
Virtual B
Saludo A
Virtual B
Saludo A
```

E. (5pts) A continuación se muestra un archivo típico **main.cpp** perteneciente a un proyecto Qt con interfaz gráfica:

```cpp
// main.cpp
#include "mainwindow.h"
#include <QApplication>

int main(int argc, char *argv[]) {
    QApplication a(argc, argv);
    MainWindow w;
    w.show();

    return a.exec();
}
```

Explique claramente el propósito de las siguientes instrucciones:

1. **#include "mainwindow.h"**
2. **#include <QApplication>**
3. **QApplication a(argc, argv);**
4. **w.show();**
5. **return a.exec();**

R:

1. `#include "mainwindow.h"`
   Incluye la definición de la clase `MainWindow`, necesaria para instanciar y usar la ventana principal del programa. (1 pto)

2. `#include <QApplication>`
   Permite utilizar la clase `QApplication`, necesaria para inicializar una aplicación Qt. (1 pto)

3. `QApplication a(argc, argv);`
   Inicializa el sistema de aplicación Qt, gestionando argumentos por consola (argc y argv) y preparando el entorno gráfico. (1 pto)

4. `w.show();`
   Muestra la ventana principal en pantalla. Sin esta instrucción, la GUI no es visible. (1 pto)

5. `return a.exec();`
   Inicia el bucle de eventos de Qt, permitiendo que la interfaz reaccione a interacciones del usuario. (1 pto)

F. (5pts) En el contexto de excepciones en C++, responda:

1. ¿Cuáles son las ventajas de utilizar una clase personalizada como excepción en lugar de lanzar directamente un **std::string**?
2. ¿Qué diferencia existe al capturar una excepción por valor en lugar de capturarla por referencia?

R:

1. (3 ptos) Ventajas:

- Permite encapsular más información relevante (mensaje, código, contexto). (1 pto)
- Se puede extender con jerarquía de excepciones y polimorfismo. (1 pto)
- Mejora la claridad, mantenibilidad y reutilización del código. (1 pto)

2. (2 ptos) Problemas:

- Crea una copia innecesaria, lo cual puede ser costoso. (1 pto)
- Puede causar pérdida de información si la clase maneja memoria dinámica (problemas de copia superficial). (1 pto)