Universidad Andrés Bello

Facultad de Ingeniería

Paradigmas de la Programación

Trabajo 1
La taxonomía de Bloom

Autores:
Raphaël Maufroy
José Salazar Cabello

Profesor: Juan Calderón Maureira

Fecha: Abril 2025

Año Académico: 2025, Semestre 1

1. Introducción

El presente trabajo tiene como objetivo desarrollar un sistema en C++ que permita a un usuario crear y gestionar pruebas escritas utilizando la Taxonomía de Bloom como referencia principal.

Dentro del desarrollo del trabajo se buscó aplicar conceptos fundamentales de programación orientada a objetos, como lo son la herencia, el polimorfismo y el manejo de memoria dinámica. Además, se consideró diseñar una solución modular y estructurada que permita un fácil mantenimiento y comprensión del código.

La funcionalidad central del sistema consiste en gestionar pruebas compuestas por distintos tipos de preguntas, tales como preguntas de Verdadero o Falso y preguntas de Selección Múltiple. Cada una de estas preguntas se encuentra asociada a un nivel taxonómico, lo que permite categorizar y evaluar las habilidades que se desean medir en el evaluado.

2. Descripción de la solución

La solución se estructura en cuatro clases principales:

1. La clase madre **Pregunta**, de la cual heredan tanto atributos como métodos las clases **Verdadero_Falso** y **Seleccion_Mult**. Se define en `preguntas.h` y `preguntas.cpp`
2. La clase **Prueba**, encargada de almacenar punteros de objetos de clase Pregunta. Se define en `prueba.h` y `prueba.cpp`
3. La clase **Menu**, que se encarga de la lógica general del programa. Se define en `utils.h` y `utils.cpp`

2.1. Clase Pregunta

La clase base `Pregunta` define la estructura común para todos los tipos de preguntas:

```
class Pregunta {
  private:
    int n_pregunta;
    string enunciado;
    string niv_tax;
    float tiempo_est;
    Pregunta *siguiente;
  public:
    Pregunta();
    virtual ~Pregunta();
    // M\'etodos getters y setters
    virtual void set_correct_resp()=0;
};
```

Los atributos principales incluyen:

- `n_pregunta`: Identificador único de la pregunta
- `enunciado`: Texto de la pregunta
- `niv_tax`: Nivel taxonómico según Bloom

- `tiempo_est`: Tiempo estimado para responder

2.2. Clases Derivadas

2.2.1. Clase Seleccion_Mult

Esta clase implementa las preguntas de selección múltiple:

```
class Seleccion_Mult : public Pregunta {
  private:
    string correct_resp;
    vector<string> dists;
  public:
    // Constructores y m\'etodos
};
```

2.2.2. Clase Verdadero_Falso

Esta clase maneja las preguntas de verdadero o falso:

```
class Verdadero_Falso : public Pregunta {
  private:
    bool correct_resp;
    string justificacion;
  public:
    // Constructores y m\'etodos
};
```

2.3. Clase Prueba

La clase `Prueba` gestiona un conjunto de preguntas:

```
class Prueba {
  private:
    int tot_preguntas;
    float tot_tiempo;
    vector<Pregunta *> preguntas;
  public:
    // M\'etodos de gesti\'on de preguntas
};
```

2.4. Clase Menu

La clase `Menu` implementa la interfaz de usuario:

```
class Menu {
  private:
    int opcion;
    vector<Prueba *> pruebas;
  public:
    void mostrar_menu();
    void ejecutar_opcion_1();
    // ... m\'as m\'etodos
};
```

3. Funcionalidades Implementadas

El sistema permite:

1. Crear nuevas pruebas especificando cantidad de preguntas y tiempo total
2. Insertar preguntas de dos tipos diferentes
3. Eliminar preguntas específicas
4. Modificar preguntas existentes
5. Mostrar todas las preguntas de una prueba
6. Eliminar pruebas completas

4. Conclusión

El desarrollo de este trabajo permitió implementar de manera práctica los principales conceptos de la programación orientada a objetos en C++. Se logró crear un sistema modular y estructurado que gestiona eficientemente pruebas y preguntas siguiendo la taxonomía de Bloom.

El uso de herencia y polimorfismo resultó fundamental para manejar los diferentes tipos de preguntas de manera uniforme, mientras que el uso de memoria dinámica permitió una gestión flexible de las pruebas y sus componentes.

La solución desarrollada demuestra la aplicabilidad de los conceptos de POO en problemas reales, resultando en un código mantenible, extensible y bien estructurado.