

|  |  |   |
|--|--|---|
|   | <p align="center"><b>UNIVERSIDAD NACIONAL DE SAN AGUSTIN</b><br/> <b>FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS</b><br/> <b>ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA</b></p> |  |
| <b>Formato:</b> Guía de Práctica de Laboratorio / Talleres / Centros de Simulación |  |   |
| <b>Aprobación:</b> 2022/03/01  | <b>Código:</b> GUIA-PRLE-001   | <b>Página:</b> 1  |

## INFORME DE LABORATORIO

| INFORMACIÓN BÁSICA   |                       |                      |        |                |    |
|--|-----------------------|----------------------|--------|----------------|----|
| ASIGNATURA:  | Tecnología de objetos |                      |        |                |    |
| TÍTULO DE LA PRÁCTICA:   | Tecnología de Objetos |                      |        |                |    |
| NÚMERO DE PRÁCTICA:  | 03                    | AÑO LECTIVO:         | 2025-B | NRO. SEMESTRE: | VI |
| FECHA DE PRESENTACIÓN  | 23/09/2025            | HORA DE PRESENTACIÓN | 23:59  |                |    |
| INTEGRANTE (s):<br>Alexander Valdiviezo Tovar<br>Alvaro Suasaca Pacompia<br>Jean Anco Aymara |                       |                      |        | NOTA:          |    |
| DOCENTE(s): MSc.Edith Cano   |                       |                      |        |                |    |

| SOLUCIÓN Y RESULTADOS  |
|--|
| <p><b>I. SOLUCIÓN DE EJERCICIOS/PROBLEMAS</b></p> <p>Herencia: Clase Persona (atributos: nombre, edad). Clases hijas: Profesor y Estudiante. Composición: Clase Curso está compuesta por un Horario (si el curso deja de existir, el horario también). Agregación: Clase Universidad tiene una lista de Cursos (los cursos pueden existir fuera de la universidad, pero la universidad los “agrupa”).</p> <p>Dependencia: Clase Reporte depende del Estudiante porque genera un reporte temporal de sus datos.</p> <p>Respetar los atributos privados y crear constructores, getters y setters (JavaBeans). Implementar toString() en cada clase. Crear un programa principal (Main). Se crean 2 profesores y 3 estudiantes. Se crean 2 cursos (cada uno con un horario). Se agreguen los cursos a la universidad. Se genera un reporte de un estudiante. Dibujar el diagrama UML mostrando las relaciones:</p> <p><b>Resolución usando C++</b></p> <p>Modelamos un pequeño dominio académico aplicando principios sólidos de POO. Parte de una clase base Persona con atributos privados nombre y edad, y a partir de ella se derivan Profesor y Estudiante (herencia). Esta jerarquía permite reutilizar y especializar comportamiento: ambas subclases heredan getters/setters y redefinen toString() para describirse con su propio matiz (especialidad o matrícula). El uso de un destructor virtual en Persona y de toString() virtual asegura</p> |

|   |   |   |
|---|---|---|
|                                      | <p style="text-align: center;"><b>UNIVERSIDAD NACIONAL DE SAN AGUSTIN</b><br/> <b>FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS</b><br/> <b>ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA</b></p> |  |
| <p style="text-align: center;"><b>Formato:</b> Guía de Práctica de Laboratorio / Talleres / Centros de Simulación</p> |   |   |
| <p><b>Aprobación:</b> 2022/03/01</p>  | <p><b>Código:</b> GUIA-PRLE-001</p>   | <p><b>Página:</b> 2</p>   |

polimorfismo correcto y extensibilidad: si mañana se añaden más tipos de personas, podrán integrarse sin tocar el código cliente.

La composición se ilustra en Curso, que contiene un Horario por valor. Esto expresa una relación “parte-todo” fuerte: el horario no tiene sentido sin su curso y se destruye junto con él. La agregación aparece en Universidad, que mantiene una colección de cursos mediante `std::shared_ptr<Curso>`. Con esto, la universidad “agrupa” cursos que también pueden existir fuera de ella; al emplear punteros inteligentes, se gestiona automáticamente la vida de los objetos y se evita fuga de memoria. Cada clase respeta el encapsulamiento al declarar atributos privados y exponer métodos estilo JavaBeans (`getX`, `setX`), lo que separa la representación interna de la interfaz pública y facilita validaciones futuras sin romper a los consumidores de la clase.

En `main()` se materializa el escenario: se crean 2 profesores y 3 estudiantes, se definen 2 horarios y a partir de ellos 2 cursos, que luego se agregan a una Universidad. Se imprimen las descripciones (`toString()`) para verificar el estado del sistema y se genera un Reporte de un estudiante, mostrando la dependencia: la clase Reporte no posee ni almacena al estudiante, solo lo “usa” temporalmente para producir una salida. En conjunto, el diseño demuestra con claridad cuatro tipos de relaciones (herencia, composición, agregación y dependencia), aplica buenas prácticas de encapsulamiento y memoria (punteros inteligentes) y deja un esqueleto limpio y extensible para ampliaciones futuras.

```
#include <iostream>
#include <string>
#include <vector>
#include <memory>
#include <sstream>


// ===== Persona (Base) =====
class Persona {
private:
    std::string nombre;
    int edad;

public:
    // Constructores
    Persona() : nombre(""), edad(0) {}
    Persona(const std::string& nombre, int edad) : nombre(nombre), edad(edad) {}

    // Getters / Setters estilo JavaBeans
    std::string getNombre() const { return nombre; }
    void setNombre(const std::string& n) { nombre = n; }

    int getEdad() const { return edad; }
    void setEdad(int e) { edad = e; }

    // toString
    virtual std::string toString() const {
        std::ostringstream oss;
        oss << "Persona{nombre=" << nombre << ", edad=" << edad << "}";
        return oss.str();
    }
};
```

|  |  |   |
|--|--|---|
|                               | <p align="center"><b>UNIVERSIDAD NACIONAL DE SAN AGUSTIN</b><br/> <b>FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS</b><br/> <b>ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA</b></p> |  |
| <p align="center"><b>Formato:</b> Guía de Práctica de Lab<br/> oratorio / Talleres / Centros de Simulación</p> |  |   |
| <p><b>Aprobación:</b> 2022/03/01</p>   | <p align="center"><b>Código:</b> GUIA-PRLE-001</p>   | <p align="right"><b>Página:</b> 3</p>   |

```

    }

    // Destructor virtual para herencia
    virtual ~Persona() = default;
};

// ===== Profesor (Hereda de Persona) =====
class Profesor : public Persona {
private:
    std::string especialidad;

public:
    Profesor() : Persona(), especialidad("") {}
    Profesor(const std::string& nombre, int edad, const std::string& especialidad)
        : Persona(nombre, edad), especialidad(especialidad) {}

    std::string getEspecialidad() const { return especialidad; }
    void setEspecialidad(const std::string& esp) { especialidad = esp; }

    std::string toString() const override {
        std::ostringstream oss;
        oss << "Profesor{nombre=" << getNombre()
            << ", edad=" << getEdad()
            << ", especialidad=" << especialidad << "}";
        return oss.str();
    }
};

// ===== Estudiante (Hereda de Persona) =====
class Estudiante : public Persona {
private:
    std::string matricula; // código o número de matrícula

public:
    Estudiante() : Persona(), matricula("") {}
    Estudiante(const std::string& nombre, int edad, const std::string& matricula)
        : Persona(nombre, edad), matricula(matricula) {}

    std::string getMatricula() const { return matricula; }
    void setMatricula(const std::string& m) { matricula = m; }

    std::string toString() const override {
        std::ostringstream oss;
        oss << "Estudiante{nombre=" << getNombre()
            << ", edad=" << getEdad()
            << ", matricula=" << matricula << "}";
        return oss.str();
    }
};

// ===== Horario (Parte de Curso) =====
class Horario {
private:
    std::string dia;
    std::string horaInicio;

```

```
std::string horaFin;

public:
    Horario() : dia(""), horaInicio(""), horaFin("") {}
    Horario(const std::string& dia, const std::string& inicio, const std::string&
fin)
        : dia(dia), horaInicio(inicio), horaFin(fin) {}

    std::string getDia() const { return dia; }
    void setDia(const std::string& d) { dia = d; }

    std::string getHoraInicio() const { return horaInicio; }
    void setHoraInicio(const std::string& hi) { horaInicio = hi; }

    std::string getHoraFin() const { return horaFin; }
    void setHoraFin(const std::string& hf) { horaFin = hf; }

    std::string toString() const {
        std::ostringstream oss;
        oss << "Horario{dia=" << dia
            << ", inicio=" << horaInicio
            << ", fin=" << horaFin << "}";
        return oss.str();
    }
};

// ===== Curso (Compuesto por Horario) =====
class Curso {
private:
    std::string nombre;
    Horario horario; // Composición: el horario pertenece al curso

public:
    Curso() : nombre(""), horario() {}
    Curso(const std::string& nombre, const Horario& horario)
        : nombre(nombre), horario(horario) {}

    std::string getNombre() const { return nombre; }
    void setNombre(const std::string& n) { nombre = n; }

    Horario getHorario() const { return horario; }
    void setHorario(const Horario& h) { horario = h; }

    std::string toString() const {
        std::ostringstream oss;
        oss << "Curso{nombre=" << nombre
            << ", " << horario.toString() << "}";
        return oss.str();
    }
};

// ===== Universidad (Agrega Cursos existentes) =====
class Universidad {
private:
    std::string nombre;
```

```
// Agregación: guarda referencias compartidas a cursos que pueden existir fuera
std::vector<std::shared_ptr<Curso>> cursos;

public:
    Universidad() : nombre("") {}
    explicit Universidad(const std::string& nombre) : nombre(nombre) {}

    std::string getNombre() const { return nombre; }
    void setNombre(const std::string& n) { nombre = n; }

    void agregarCurso(const std::shared_ptr<Curso>& curso) {
        cursos.push_back(curso);
    }


    const std::vector<std::shared_ptr<Curso>>& getCursos() const { return cursos; }

    std::string toString() const {
        std::ostringstream oss;
        oss << "Universidad{nombre=" << nombre << ", cursos=[";
        for (size_t i = 0; i < cursos.size(); ++i) {
            oss << cursos[i]->getNombre();
            if (i + 1 < cursos.size()) oss << ", ";
        }
        oss << "]}";
        return oss.str();
    }
};

// ===== Reporte (Depende de Estudiante) =====
class Reporte {
public:
    // Dependencia: usa Estudiante temporalmente para generar un informe
    std::string generar(const Estudiante& est) const {
        std::ostringstream oss;
        oss << "=== REPORTE DE ESTUDIANTE ===\n";
        oss << "Nombre: " << est.getNombre() << "\n";
        oss << "Edad: " << est.getEdad() << "\n";
        oss << "Matrícula: " << est.getMatricula() << "\n";
        oss << "Resumen: " << est.toString() << "\n";
        oss << "=====\n";
        return oss.str();
    }
};

// ===== Programa Principal =====
int main() {
    // 2 Profesores
    Profesor p1("Ana Torres", 45, "Programación");
    Profesor p2("Luis Gómez", 50, "Matemática");

    // 3 Estudiantes
    Estudiante e1("Carla Pérez", 20, "2025-001");
    Estudiante e2("Jorge Ramírez", 22, "2025-002");
    Estudiante e3("María López", 19, "2025-003");
```

|   |   |   |
|---|---|---|
|    | <p style="text-align: center;"><b>UNIVERSIDAD NACIONAL DE SAN AGUSTIN</b><br/> <b>FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS</b><br/> <b>ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA</b></p> |  |
| <p style="text-align: center;"><b>Formato:</b> Guía de Práctica de Lab<br/> oratorio / Talleres / Centros de Simulación</p> |   |   |
| <b>Aprobación: 2022/03/01</b>   | <b>Código: GUIA-PRLE-001</b>  | <b>Página: 6</b>  |

```
// 2 Cursos (cada uno con su Horario) - Composición
Horario h1("Lunes", "08:00", "10:00");
Horario h2("Miércoles", "10:00", "12:00");

auto c1 = std::make_shared<Curso>("Estructuras de Datos", h1);
auto c2 = std::make_shared<Curso>("Cálculo II", h2);

// Universidad (Agregación de cursos)
Universidad uni("Universidad Nacional de Arequipa");
uni.agregarCurso(c1);
uni.agregarCurso(c2);

// Mostrar entidades
std::cout << p1.toString() << "\n";
std::cout << p2.toString() << "\n\n";

std::cout << e1.toString() << "\n";
std::cout << e2.toString() << "\n";
std::cout << e3.toString() << "\n\n";

std::cout << c1->toString() << "\n";
std::cout << c2->toString() << "\n\n";

std::cout << uni.toString() << "\n\n";

// Dependencia: generar un reporte temporal de un estudiante
Reporte rep;
std::cout << rep.generar(e2) << std::endl;

return 0;
}
```

## Salida del programa en C++


```
Profesor{nombre=Ana Torres, edad=45, especialidad=Programación}
Profesor{nombre=Luis Gómez, edad=50, especialidad=Matemática}

Estudiante{nombre=Carla Pérez, edad=20, matricula=2025-001}
Estudiante{nombre=Jorge Ramírez, edad=22, matricula=2025-002}
Estudiante{nombre=María López, edad=19, matricula=2025-003}

Curso{nombre=Estructuras de Datos, Horario{dia=Lunes, inicio=08:00, fin=10:00}}
Curso{nombre=Cálculo II, Horario{dia=Miércoles, inicio=10:00, fin=12:00}}

Universidad{nombre=Universidad Nacional de Arequipa, cursos=[Estructuras de Datos, Cálculo II]}

=== REPORTE DE ESTUDIANTE ===
Nombre: Jorge Ramírez
Edad: 22
Matrícula: 2025-002
Resumen: Estudiante{nombre=Jorge Ramírez, edad=22, matricula=2025-002}
=====
```

|   |  |   |
|---|--|---|
|                                      | <p style="text-align: center;">UNIVERSIDAD NACIONAL DE SAN AGUSTIN<br/>FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS<br/>ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA</p> |  |
| <p style="text-align: center;"><b>Formato:</b> Guía de Práctica de Laboratorio / Talleres / Centros de Simulación</p> |  |   |
| <p>Aprobación: 2022/03/01</p>   | <p>Código: GUIA-PRLE-001</p>   | <p>Página: 7</p>  |

## Resolución usando Java

Para la clase persona tenemos los atributos de nombre y edad, sus getters y setters correspondientes y el metodo toString para poder sobrescribirlo en futuras clases hijas de ser necesario.

```
class Persona {
    private String nombre;
    private int edad;

    public Persona(String nombre, int edad) {
        this.nombre = nombre;
        this.edad = edad;
    }

    public String getNombre() { return nombre; }
    public void setNombre(String nombre) { this.nombre = nombre; }

    public int getEdad() { return edad; }
    public void setEdad(int edad) { this.edad = edad; }

    @Override
    public String toString() {
        return "Nombre: " + nombre + ", Edad: " + edad;
    }
}
```

Para las clases hijas de la clase persona creamos 2, la clase estudiante y la clase profesor, los atributos de estas clases hijas son los mismo que los de la padre, pero en el metodo toString se añade la dof

```
class Profesor extends Persona {
    public Profesor(String nombre, int edad) {
        super(nombre, edad);
    }


    @Override
    public String toString() {
        return "Profesor -> " + super.toString();
    }
}

class Estudiante extends Persona {
    public Estudiante(String nombre, int edad) {
        super(nombre, edad);
    }

    @Override
    public String toString() {
        return "Estudiante -> " + super.toString();
    }
}
```

Como clase auxiliar tenemos la clase horario con los atributos de día y hora, le ponemos los getters y setter correspondientes y la clase to string de misma manera.



|   |  |   |
|---|--|---|
|          | <p>UNIVERSIDAD NACIONAL DE SAN AGUSTIN<br/>FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS<br/>ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA</p> |  |
| <p><b>Formato:</b> Guía de Práctica de Laboratorio / Talleres / Centros de Simulación</p> |  |   |
| <p>Aprobación: 2022/03/01</p>   | <p>Código: GUIA-PRLE-001</p>   | <p>Página: 9</p>  |

```
class Horario {
    private String dia;
    private String hora;

    public Horario(String dia, String hora) {
        this.dia = dia;
        this.hora = hora;
    }


    public String getDia() { return dia; }
    public void setDia(String dia) { this.dia = dia; }

    public String getHora() { return hora; }
    public void setHora(String hora) { this.hora = hora; }

    @Override
    public String toString() {
        return "Horario: " + dia + " a las " + hora;
    }
}
```

Como clase que usa a horario tenemos la clase curso que utiliza un string como atributo nombre y un objeto horario que servirá para definir los parámetros de día y hora.

También complementamos la clase con sus getters y setters junto con la clase toString que llama a la clase toString de la clase horario, esto hace más sencillo la lectura de código y evita escribir el mismo código en cada clase si no es necesario.

|   |  |   |
|---|--|---|
|          | <p>UNIVERSIDAD NACIONAL DE SAN AGUSTIN<br/>FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS<br/>ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA</p> |  |
| <p><b>Formato:</b> Guía de Práctica de Laboratorio / Talleres / Centros de Simulación</p> |  |   |
| <p>Aprobación: 2022/03/01</p>   | <p>Código: GUIA-PRLE-001</p>   | <p>Página: 10</p>   |

```
class Curso {
    private String nombre;
    private Horario horario;

    public Curso(String nombre, Horario horario) {
        this.nombre = nombre;
        this.horario = horario;
    }

    public String getNombre() { return nombre; }
    public void setNombre(String nombre) { this.nombre = nombre; }

    public Horario getHorario() { return horario; }
    public void setHorario(Horario horario) { this.horario = horario; }

    @Override
    public String toString() {
        return "Curso: " + nombre + " | " + horario.toString();
    }
}
```

Para manejar todas las clases anteriores tenemos la clase universidad que guarda una lista de cursos para poder asignarlos a un estudiante en el futuro, de la misma manera que en los metodos anteriores se usa el metodo toString para sobrescribir los metodos anteriores y poder complementarlos con la nueva informacion necesaria.

```
class Universidad {
    private String nombre;
    private List<Curso> cursos;

    public Universidad(String nombre) {
        this.nombre = nombre;
        this.cursos = new ArrayList<>();
    }

    public void agregarCurso(Curso curso) {
        cursos.add(curso);
    }

    public List<Curso> getCursos() {
        return cursos;
    }

    @Override
    public String toString() {
        StringBuilder sb = new StringBuilder("Universidad: " + nombre + "\nCursos:\n");
        for (Curso c : cursos) {
            sb.append("- ").append(c.toString()).append("\n");
        }
        return sb.toString();
    }
}
```

Por ultimo tenemos la clase reporte estudiante que devuelve los atributos nombre y edad de un objeto estudiante.

```
class Reporte {
    public String generar(Estudiente est) {
        return "Reporte del estudiante\n-----\n" +
            "Nombre: " + est.getNombre() + "\n" +
            "Edad: " + est.getEdad();
    }
}
```

Para la clase main creamos varios objetos estudiante, profesor y cursos, se crea un objeto universidad y agregamos los cursos creados, por ultimo tenemos la impresion de los atributos por el metodo to string y el como todos los metodos to String se complementaron por herencia o polimorfismo al agregarlos a clases mas grandes.

```
public class Main {  
    public static void main(String[] args) {  
  
        Profesor prof1 = new Profesor("Juan Pérez", 45);  
        Profesor prof2 = new Profesor("María López", 39);  
  
        Estudiante est1 = new Estudiante("Ana Torres", 20);  
        Estudiante est2 = new Estudiante("Luis Gómez", 22);  
        Estudiante est3 = new Estudiante("Carla Ríos", 19);  
  
        Curso curso1 = new Curso("Programación", new Horario("Lunes", "10:00 AM"));  
        Curso curso2 = new Curso("Matemáticas", new Horario("Martes", "2:00 PM"));  
  
        Universidad uni = new Universidad("Universidad Nacional");  
        uni.agregarCurso(curso1);  
        uni.agregarCurso(curso2);  
  
        System.out.println(prof1);  
        System.out.println(prof2);  
        System.out.println(est1);  
        System.out.println(est2);  
        System.out.println(est3);  
        System.out.println();  
        System.out.println(uni);  
  
        Reporte reporte = new Reporte();  
        System.out.println(reporte.generar(est2));  
    }  
}
```

---

run:

Profesor -> Nombre: Juan Pérez, Edad: 45  
Profesor -> Nombre: María López, Edad: 39  
Estudiante -> Nombre: Ana Torres, Edad: 20  
Estudiante -> Nombre: Luis Gómez, Edad: 22  
Estudiante -> Nombre: Carla Ríos, Edad: 19

Universidad: Universidad Nacional  
Cursos:

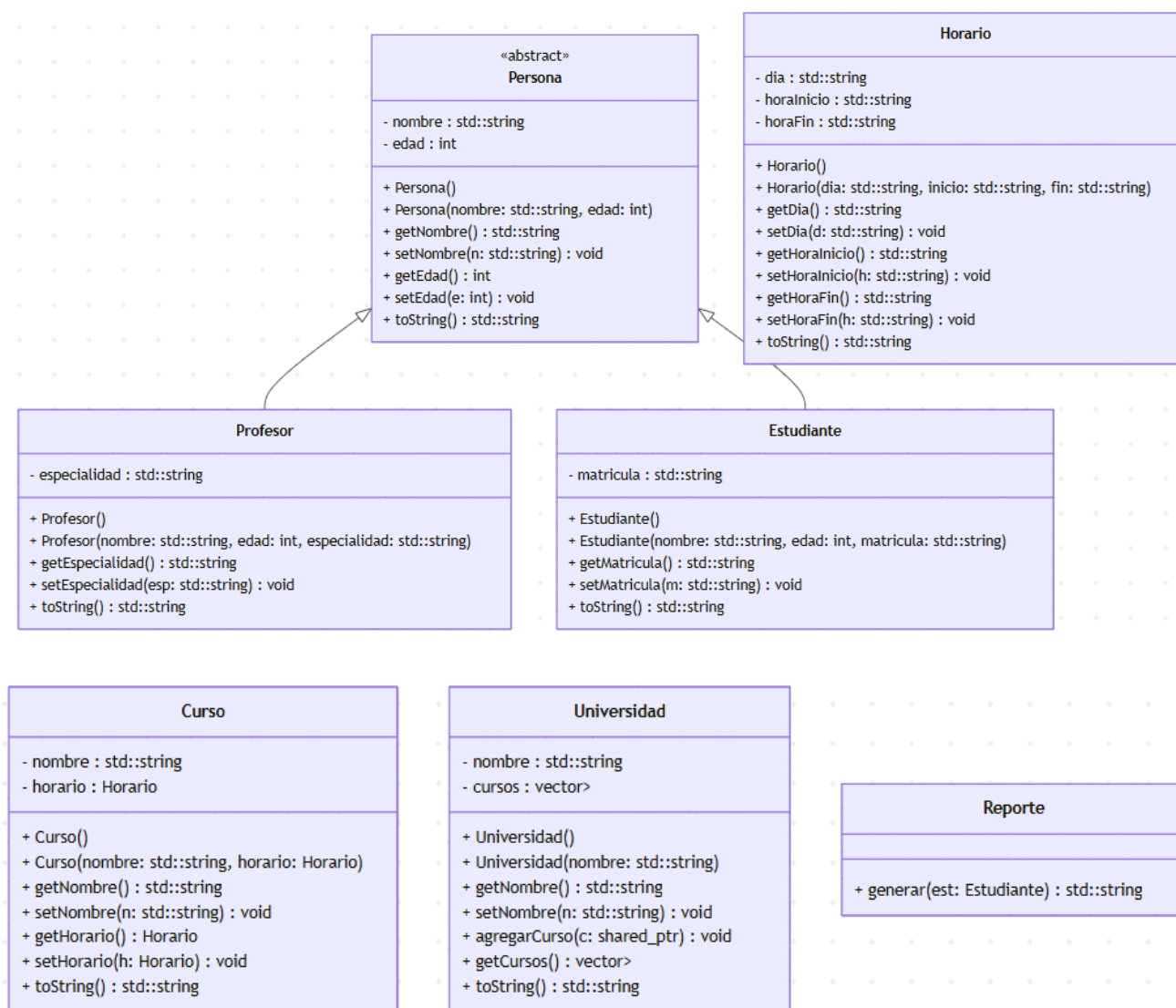
- Curso: Programación | Horario: Lunes a las 10:00 AM
- Curso: Matemáticas | Horario: Martes a las 2:00 PM

Reporte del estudiante

-----  
Nombre: Luis Gómez  
Edad: 22

**BUILD SUCCESSFUL (total time: 0 seconds)**

## Diagrama UML



El diagrama UML muestra cómo se relacionan las clases dentro del sistema. La clase Persona actúa como base y se especializa en Profesor y Estudiante a través de herencia, lo que permite reutilizar atributos y comportamientos comunes como nombre y edad. Por otro lado, la clase Curso está ligada fuertemente a Horario mediante una relación de composición (rombo sólido), indicando que el horario no puede existir sin el curso. De esta forma se representan relaciones de vida compartida entre objetos.

|   |   |   |
|---|---|---|
|                                      | <p style="text-align: center;"><b>UNIVERSIDAD NACIONAL DE SAN AGUSTIN</b><br/> <b>FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS</b><br/> <b>ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA</b></p> |  |
| <p style="text-align: center;"><b>Formato:</b> Guía de Práctica de Laboratorio / Talleres / Centros de Simulación</p> |   |   |
| <p><b>Aprobación:</b> 2022/03/01</p>  | <p><b>Código:</b> GUIA-PRLE-001</p>   | <p><b>Página:</b> 14</p>  |

En un nivel más flexible, la clase Universidad se conecta con Curso mediante agregación (rombo vacío), lo que refleja que la universidad organiza y mantiene cursos que también podrían existir fuera de ella. Finalmente, la clase Reporte tiene una dependencia con Estudiante (flecha discontinua), ya que genera reportes temporales basados en sus datos sin mantener una referencia permanente.

### **Diferencias entre implementación Java y C++**

Como se puede observar en el código, la implementación entre dos lenguajes orientados objetos modernos como Java y C++ es la sintaxis y el cómo estos manejan el uso de memoria, en Java contamos con el garbage collector que libera memoria automáticamente, sin embargo en C++ tenemos otras bondades usadas en el código como el uso de punteros.

Para el uso de librerías, cada lenguaje tiene sus propias librerías, pero la mayoría tienen el mismo final de resolver y dar facilidad a un problema para poder implementar programas más complejos, "import java.util.\*;" en caso de Java y "#include <iostream> #include <string>" en caso de C++.

Mientras en Java para la herencia podemos usar el atributo extends en C++ se usa :public caso\_base para poder heredar, existen varias más diferencias en sintaxis usadas como el polimorfismo en Java con override y el uso de virtual en C++, pero los principales cambios es que para poder correr el programa Java se necesita de el JVM y un compilador, mientras para poder ejecutar el código C++ se pudo lograr sin nada de esto al ser un lenguaje más cerca al hardware, esto nos da mejor control pero al mismo tiempo puede hacer que el programa en C++ sea más complicado de analizar y entender por lo complicado de la sintaxis.

## **II. CUESTIONARIO**

### **1. ¿Qué diferencias puede resaltar en las implementaciones de los dos lenguajes de programación en POO?**

Escogimos lo que es lenguaje C++ y Java para poder implementar y lo comparamos con Python y vemos que la principal diferencia es la sintaxis pues en C++ y Java por ejemplo en Java se tiene que declarar el tipo de dato de la variable en Python no pues lo toma como un objeto, entonces diremos que Python es más sencillo y conciso en sus sintaxis mientras que C++ y Java es más estricto y los errores más pequeños puede que no se ejecute el código.

Otra diferencia que encontramos es en la indentación o sangría pues en el lenguaje de Python si no está indentado de manera correcta puede haber errores como por ejemplo en la creación de objetos o métodos si está mal indentado no lo considera dentro del método o objeto creado.

También podemos hacer la diferencia de en qué tipos de áreas se desarrolla más por ejemplo Python está más enfocado en lo que es inteligencia artificial y ciencia de datos (machine

|   |   |   |
|---|---|---|
|                                      | <p style="text-align: center;"><b>UNIVERSIDAD NACIONAL DE SAN AGUSTIN</b><br/> <b>FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS</b><br/> <b>ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA</b></p> |  |
| <p style="text-align: center;"><b>Formato:</b> Guía de Práctica de Laboratorio / Talleres / Centros de Simulación</p> |   |   |
| <p><b>Aprobación:</b> 2022/03/01</p>  | <p><b>Código:</b> GUIA-PRLE-001</p>   | <p><b>Página:</b> 15</p>  |

learning), aplicaciones web, por otro lado tenemos a java que se utiliza más para aplicaciones móviles, servidores web y sistemas integrados.

## 2. ¿En su implementación dónde se puede evidenciar la protección de datos o la seguridad utilizando la técnica de la programación orientada a objetos?

En lo que es c++ y java tenemos lo que llamamos encapsulamiento donde tenemos la opción de poner el estado interno de un objeto de manera privada con private además de protected si se usa lo que es herencia además de los métodos de acceso a contenido privado lo que son los getters y setters que nos ayudaran con el acceso a estos objetos protegidos. Lo tenemos en el código más que nada en lo que son los atributos y sus setters y getters para su acceso.

### III. CONCLUSIONES

- Primero, el diagrama UML sirvió como contrato de diseño: al definir herencia (Persona→Profesor/Estudiante), composición (Curso–Horario), agregación (Universidad–Cursos) y dependencia (Reporte→Estudiante), permitió implementar rápidamente una solución en C++ (y sería igual de directa en Java) con una estructura clara y extensible. Ese modelado previo reduce ambigüedades, facilita pruebas y promueve principios SOLID (especialmente responsabilidad única y sustitución), logrando código mantenible y fácil de evolucionar.
- Segundo, aunque C++ y Java comparten POO clásica (clases, tipos estáticos, visibilidad, sobrescritura), difieren en detalles operativos: C++ expone gestión de memoria explícita y múltiples paradigmas, mientras Java aporta recolección de basura y un ecosistema uniformado; Python, por su parte, ofrece tipado dinámico y sintaxis concisa que acelera prototipos, pero delega a pruebas y disciplina aspectos que en C++/Java se validan en compilación. También cambia la sensibilidad a la sintaxis: Python usa indentación significativa; C++/Java dependen de llaves y tipos declarados.
- Tercero, la seguridad y protección de datos se evidencian mediante encapsulamiento: atributos privados y acceso controlado por getters/setters, más protected para herencia, limitan exposición del estado y habilitan validaciones, invariantes y reglas de negocio en un solo punto. En C++ puede reforzarse con const/referencias, RAII y punteros inteligentes; en Java, con inmutabilidad, clases de registro y validación en setters. En conjunto, el diseño orientado a objetos no solo organiza el dominio: también reduce superficie de ataque y errores, permitiendo aplicar “privacy by design” desde el modelo hasta la implementación.

|   |  |   |
|---|--|---|
|          | <p>UNIVERSIDAD NACIONAL DE SAN AGUSTIN<br/>FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS<br/>ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA</p> |  |
| <p><b>Formato:</b> Guía de Práctica de Laboratorio / Talleres / Centros de Simulación</p> |  |   |
| <p>Aprobación: 2022/03/01</p>   | <p>Código: GUIA-PRLE-001</p>   | <p>Página: 16</p>   |

| REFERENCIAS Y BIBLIOGRAFÍA  |
|---|
| <p><a href="https://www.ionos.com/es-us/digitalguide/paginas-web/desarrollo-web/python-vs-java/">https://www.ionos.com/es-us/digitalguide/paginas-web/desarrollo-web/python-vs-java/</a><br/> <a href="https://www.geeksforgeeks.org/blogs/cpp-vs-java/">https://www.geeksforgeeks.org/blogs/cpp-vs-java/</a></p> |