UNIVERSIDAD DE LAS FUERZAS ARMADAS-ESPE SEDE SANTO DOMINGO

DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN - DCCO-SS

CARRERA DE INGENIERÍA EN TECNOLOGÍAS DE LA INFORMACIÓN

PERIODO	:	Abril 2023 – Octubre 2023
ASIGNATURA	:	POO
TEMA	:	Laboratorio 2
NOMBRES	:	Kelvin Quezada
NIVEL-PARALELO	:	Segundo “A”
DOCENTE	:	Ing. Cevallos Farias Javier Moyota
FECHA DE ENTREGA	:	29/06/2023

SANTO DOMINGO - ECUADOR

2023

Contenido

Imágenes

1. Introducción

Mediante artículos científicos nos dieron a conocer que la programación orientada a objetos (POO) es un paradigma de programación que se basa en la creación de objetos, los cuales son instancias de clases. Los objetos tienen características o atributos, y pueden realizar acciones o métodos. A continuación, exploraremos los conceptos clave de la POO:

La herencia permite crear nuevas clases basadas en clases existentes, heredando sus atributos y métodos. Facilita la reutilización de código y la organización jerárquica de las clases. Mientras los atributos y constructores son variables que describen las características de un objeto y se inicializan mediante constructores, que son métodos especiales para asignar valores iniciales.

Las validaciones son técnicas para asegurar que los datos ingresados cumplan reglas o restricciones. Garantizan la integridad y consistencia de los datos y el polimorfismo permite que un objeto pueda presentar múltiples formas o comportamientos. Se logra al referenciar una variable de tipo de clase base o subclase, facilitando la programación genérica y la reutilización de código.

2. Objetivos

2.1.Objetivos Generales

- Desarrollar un ejercicio de sistema de gestión horario en los laboratorios

2.2.Objetivos Especifico

- Mediante este programa se utilizar atributos, métodos, constructores, polimorfismo que cuente con herencia.

3. Desarrollo

3.1.Generalización/Especialización

Según (Vazquez) la relación de especialización/generalización (o de herencia) entre dos clases. Esta relación se considera propia de los lenguajes de POO. Una relación de herencia de la clase B (subclase, o clase hija) con respecto a (superclase o clase base) nos permite decir que la clase B obtiene todos los métodos y atributos de la clase A, y que luego puede añadir algunas características propias.

En el caso anterior se supone que utilizaremos la relación de herencia para decir que la clase B hereda de la clase A. Por medio de este mecanismo, la clase B comparte todas las características (atributos o estado y métodos o comportamiento) de la clase A. Esto no impide que se le pueda añadir a la clase B características adicionales (de nuevo, tanto atributos como métodos), o incluso, se modifique el comportamiento (la definición, no la declaración) de alguno de los métodos heredados.

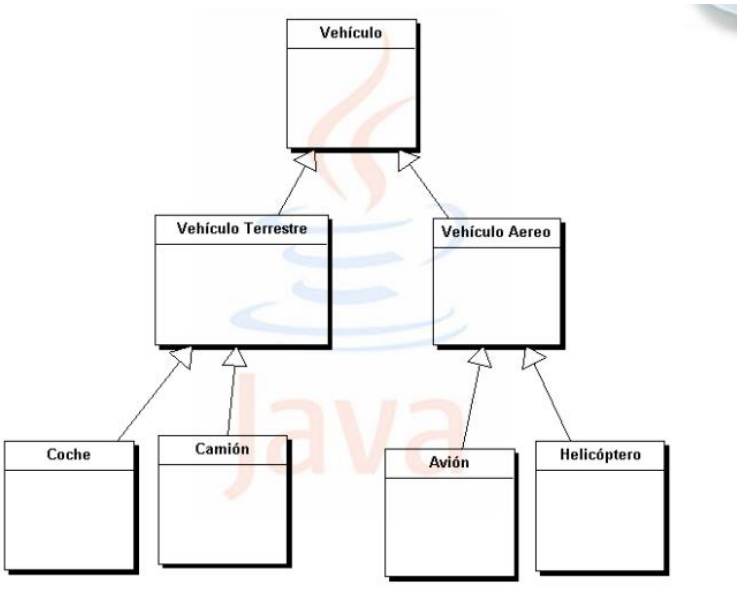

Imagen 1. Generalización/Especialización

La Herencia

Las instancias de una clase incluyen también las de su superclase o superclases. Como resultado, además de los atributos y métodos específicos de la clase, también utilizan los definidos en la superclase.

Esta capacidad se llama herencia, lo que significa que una clase hereda las propiedades y funciones de sus superclases para que sus instancias puedan usarlas. Al colocar el símbolo delante de los atributos y métodos heredados en las subclases, UML brinda la opción de representarlos.

3.1.1. Implementación

Según (Solis, 2015) la programación e implementación de clases trata de amoldarse al modo de pensar del hombre y ajustarse a su forma de analizar y gestionar los problemas que tenga que desarrollar. La clase son abstracciones que representa a un conjunto de objetos con un comportamiento e interfaz común. La implementación de una clase comprende dos componentes la declaración y el cuerpo de la clase.

Ejemplo de sintaxis

Definidos los tipos y funciones pasamos a la declaración e implementación de clases con ello podemos identificar la estructura de la programación.

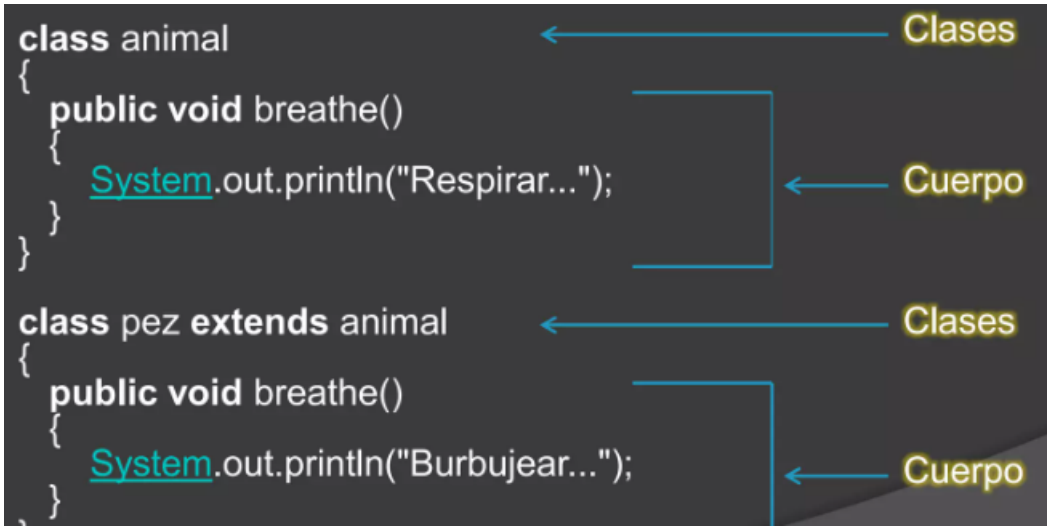

Imagen 2. Implementación

3.2.Gestión de Defectos (testing)

3.2.1. Verificación y Validación

Los procesos de verificación y análisis que aseguran que el software que se está desarrollando está en línea con su especificación y satisface las necesidades de los clientes se denominan verificación y validación. Un proceso de ciclo de vida completo es VandV. Las revisiones de los requisitos vienen primero, luego las revisiones de los diseños y el código, y finalmente las pruebas del producto. En cada etapa del proceso de desarrollo de software, se llevan a cabo actividades de V&V. A pesar de la facilidad con la que pueden confundirse, Boehm (1979) describió sucintamente la diferencia entre verificación y validación de la siguiente manera.

- Verificación: las responsabilidades de Verificación incluyen asegurarse de que el software cumpla con sus especificaciones. Se confirma el cumplimiento del sistema con los requisitos funcionales y no funcionales establecidos.
- La validación es un proceso que se usa más ampliamente: El software necesita ser revisado para ver si cumple con las expectativas del cliente. El software se prueba

para ver si funciona como el usuario espera en lugar de lo especificado, lo que va más allá de determinar si el sistema cumple con sus especificaciones.

Primero, es crucial validar los requisitos del sistema. Es sencillo cometer errores y omisiones durante la fase de análisis de requisitos del sistema y, en tales casos, el software terminado no estará a la altura de las expectativas del cliente. Sin embargo, en realidad, no todos los problemas que presenta una aplicación no se pueden encontrar a través de la validación de requisitos. Cuando el sistema se haya implementado por completo, es posible que se encuentren algunos errores en los requisitos. (Drake, 2009)

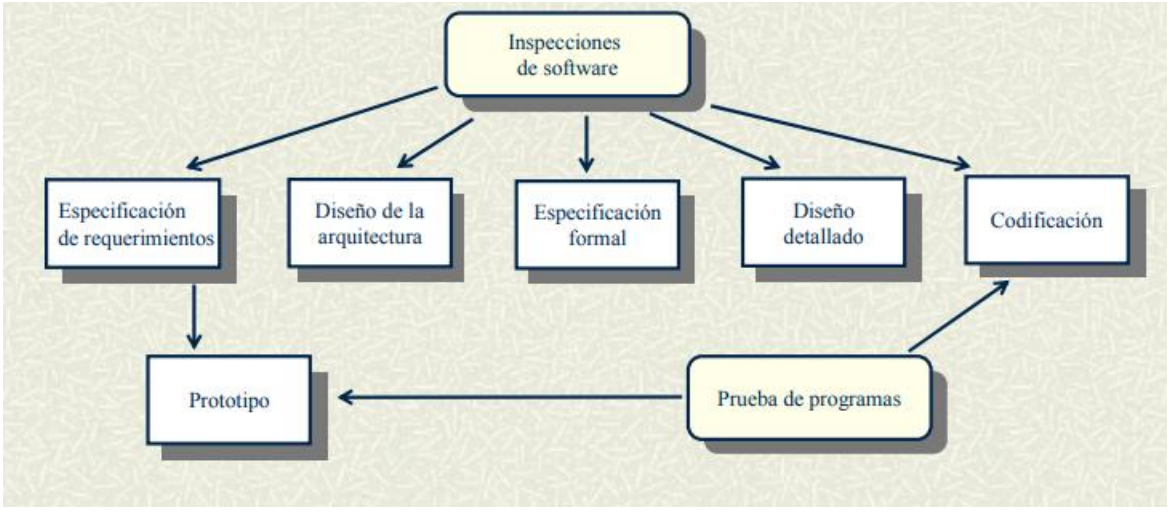

Imagen 3. Verificación y Validación

3.2.2. Pruebas vs Depuración

Según (tech, 2022) caso de prueba ejecución de un método bajo unas condiciones particulares (valores de los parámetros + estado de la clase), el resultado permite determinar si, para este caso particular, la clase se ha comportado acorde a su especificación. En general, el número de posibles casos de prueba es muy alto (muchas veces infinito).

Objetivo de la prueba de clases

- Encontrar el mayor número de defectos posible.

- Utilizar un número “pequeño” de casos de prueba.
- Nunca olvidar que las pruebas permiten probar la existencia de defectos, no la ausencia de estos.

Eclipse se utiliza para la depuración.

En muchas situaciones, es un desafío comprender lo que está sucediendo en un programa. Depurar el código del programa es la solución. Incluso desde una máquina remota, la máquina virtual Java se puede utilizar para depurar código. Eclipse simplifica la depuración. La depuración implica ir paso a paso a través de un programa y verificar los valores de sus variables. A continuación, puede ver con precisión lo que sucede en el programa cuando se ejecuta una determinada línea de código. El primer paso para depurar un programa Java en Eclipse es insertar un punto de interrupción, también conocido como "punto de interrupción", en la línea de código que desea revisar o vigilar.

A continuación, el programa debe iniciarse con la opción de depuración seleccionada. Mientras no se alcance el punto de interrupción, este programa continuará ejecutándose normalmente. Cuando Eclipse quiere cambiar a una perspectiva de depuración diferente, pausará la ejecución y solicitará permiso. Toca "Sí" después de seleccionar "Recordar mi decisión". Luego, la línea se resaltará en la perspectiva de depuración, que luego mostrará detalles sobre el estado de ejecución.

3.2.3. Pruebas de unidad

Según (KeepCoding, 2023) los procesos de verificación de la funcionalidad del software de un programa o aplicación en particular se denominan pruebas o pruebas de código. Para ello, hace uso de herramientas como las pruebas unitarias de software, que se encargan de comprobar

que una parte del código fuente funciona correctamente. Este mecanismo se caracteriza por ofrecer una garantía de calidad del sistema.

Por lo tanto, las pruebas unitarias de software podrían ser muy útiles para garantizar que su código se ejecute, por lo que es importante conocer todos los detalles sobre él, como sus características clave, utilidades y propiedades.

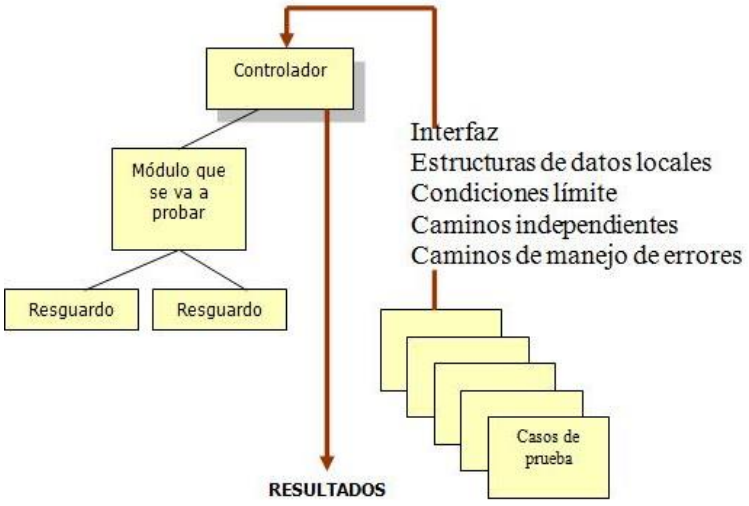

Imagen 4. Pruebas de unidad

3.3.Polimorfismo

Según (Fernandez, s.f.) al crear objetos con comportamientos compartidos, el polimorfismo nos permite procesar objetos de diversas formas. Implica tener la capacidad de mostrar la misma interfaz de usuario para varios formularios o tipos de datos subyacentes. Los objetos pueden reemplazar comportamientos primarios comunes con comportamientos secundarios particulares mediante el uso de la herencia. La sobrecarga de métodos y la anulación de métodos son dos formas en que el polimorfismo permite que el mismo método lleve a cabo varios comportamientos.

Las clases con tipos compatibles se pueden usar en cualquier parte de nuestro código gracias al polimorfismo. La compatibilidad de tipos en Java se refiere a cómo una clase extiende

a otra o cómo una clase implementa una interfaz. Informalmente: Podemos enlazar referencias de sus hijas a una referencia de tipo padre. Cualquier instancia de una clase que implemente la interfaz se puede conectar a una referencia de tipo de interfaz.

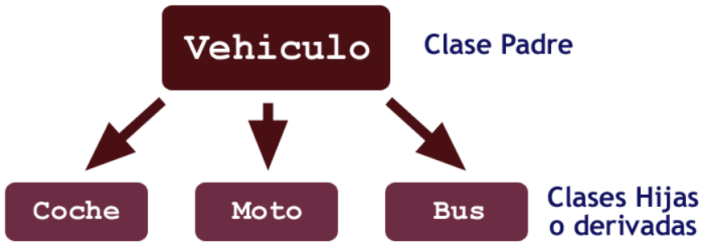

Imagen 5. Polimorfismo

3.3.1. Sobrecarga de métodos

Como ya sabemos, los métodos se pueden identificar por sus firmas, que están compuestas por el nombre del método, así como por la cantidad y el tipo de sus argumentos. Debe tener en cuenta que el tipo de devolución de un método no es un componente de su firma y, por lo tanto, es inútil para la diferenciación de métodos. Suponga que estamos definiendo un punto en tres dimensiones y escriba la siguiente implementación.

Tenemos el constructor sobrecargado cuatro veces, es demasiado. Los constructores con uno y dos parámetros son ambiguos:

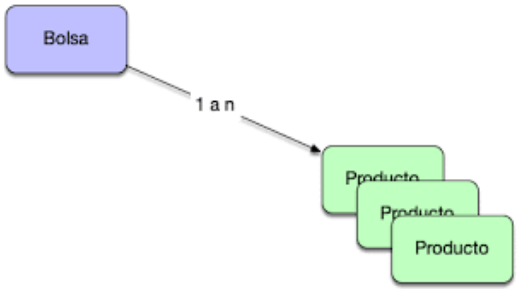

Imagen 6. Sobrecarga de métodos

3.3.2. Sobre escritura de métodos

A menos que la subclase anule los métodos, una subclase hereda todos los métodos accesibles de su superclase. Se dice que un método definido por una subclase que comparte el mismo nombre, número y tipo de argumentos que un método definido por una superclase está anulado por la subclase. La funcionalidad de un método que se heredó de una clase padre puede ser añadida o cambiada por las subclases con mayor frecuencia mediante anulaciones de métodos.

```
Ejemplo

class ClaseA
{
   void miMetodo(int var1, int var2)
   { ... }

   String miOtroMetodo( )
   { ... }
}

class ClaseB extends ClaseA
{
   /* Estos métodos sobreescriben a los métodos
   de la clase padre */

   void miMetodo (int var1 ,int var2)
   { ... }

   String miOtroMetodo( )
   { ... }
}
```

Imagen 7. Sobre escritura de métodos

3.4.Anexos

Este ejercicio se dará a conocer el

3.4.1. Diagrama UML

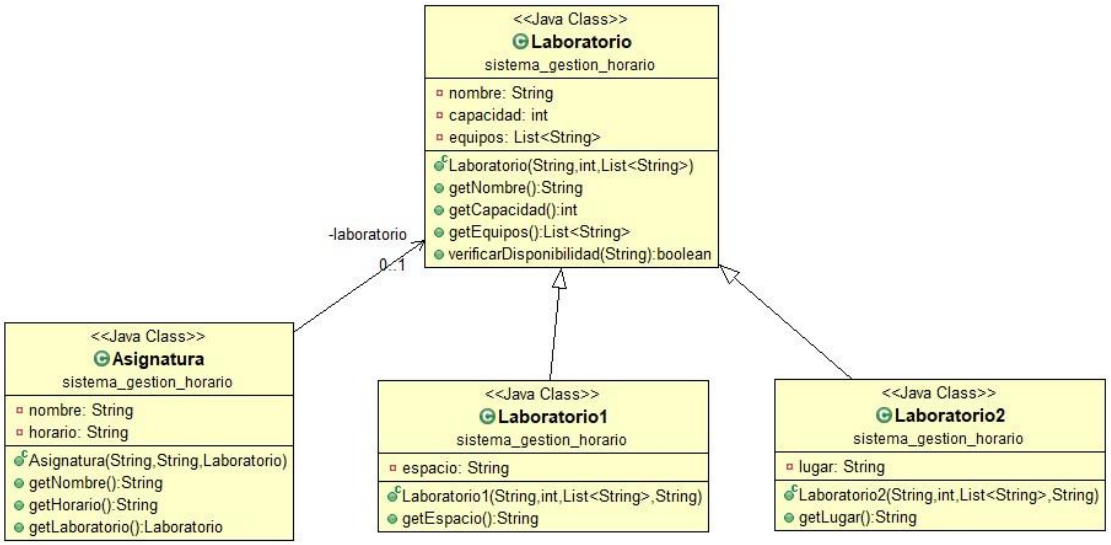

Imagen 8. Diagrama UML

3.4.2. Diagrama de Uso

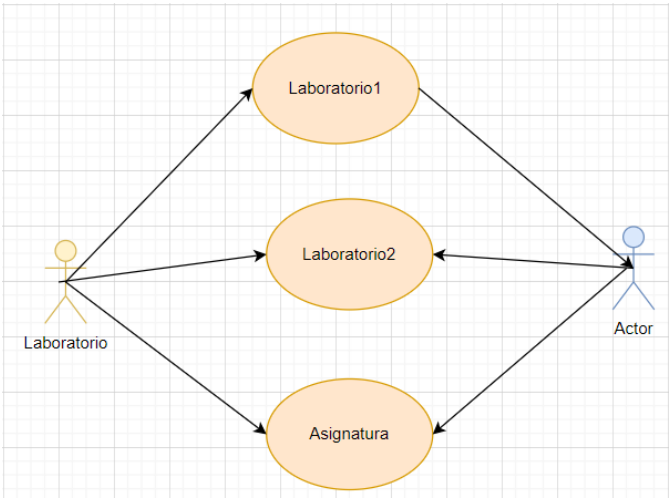

Imagen 9 Diagrama De Uso

3.4.3. Anexos de Código


```java
package sistema_gestion_horario;
import java.util.List;
public class Laboratorio {
    //Atributo
    private String nombre;
    private int capacidad;
    private List<String> equipos;
    //constructores
    public Laboratorio(String nombre, int capacidad,
            List<String> equipos) {
        this.nombre = nombre;
        this.capacidad = capacidad;
        this.equipos = equipos;
    }
        //metodos get  para mostrar datos
    public String getNombre() {
        return nombre;
    }
        //metodos get  para mostrar datos
    public int getCapacidad() {
        return capacidad;
    }
    public List<String> getEquipos() {
        return equipos;
    }
    public boolean verificarDisponibilidad(String horario) {

        return true;
    }
}
```

Imagen 10. Clase Laboratorio

```java
package sistema_gestion_horario;
public class Asignatura {
    //Atributos
    private String nombre;
    private String horario;
    private Laboratorio laboratorio;
//constructores
    public Asignatura(String nombre, String horario,
            Laboratorio laboratorio) {
        this.nombre = nombre;
        this.horario = horario;
        this.laboratorio = laboratorio;
    }
        //metodos get  para mostrar datos
    public String getNombre() {
        return nombre;
    }
        //metodos get  para mostrar datos
    public String getHorario() {
        return horario;
    }
        //metodos get  para mostrar datos
    public Laboratorio getLaboratorio() {
        return laboratorio;
    }
}
```

Imagen 11. Asignatura

```java
package sistema_gestion_horario;

import java.util.List;
//utilizamos el extends para hacer herencia del Laboratorio
//que hereda los atributos de la clase
public class Laboratorio1 extends Laboratorio {
    private String espacio;
//constructor
    public Laboratorio1(String nombre, int capacidad,
            List<String> equipos, String espacio) {
        super(nombre, capacidad, equipos);
        this.espacio = espacio;
    }
    //metodos get  para mostrar datos
    public String getEspacio() {
        return espacio;
    }
}
```

Imagen 12. Clase Laboratorio1

```java
package sistema_gestion_horario;
import java.util.List;
//utilizamos el extends para hacer herencia del Laboratorio
//que hereda los atributos de la clase
public class Laboratorio2 extends Laboratorio {
    //Atributo
    private String lugar;
    //constructor
    public Laboratorio2(String nombre, int capacidad,
            List<String> equipos, String lugar) {
        super(nombre, capacidad, equipos);
        this.lugar = lugar;
    }
    //metodos get  para mostrar datos
    public String getLugar() {
        return lugar;
    }
}
```

Imagen 13. Clase Laboratorio2

```java
public class Principal {
    public static void main(String[] args) {
        //ingresamos el sacanner para el ingreso de consola
        Scanner leer = new Scanner(source:System.in);
        //creamos el Liss para mostrar las lista de los arreglos ArrayList
        List<Laboratorio> laboratorios = new ArrayList<>();
        System.out.print(s: "Ingrese la cantidad de laboratorios a agregar: ");
        int numLaboratorios = leer.nextInt();
        //for (inicializamos, dondicion, aumento o decremento)
        for (int i = 0; i < numLaboratorios; i++) {
            System.out.println("Laboratorio #" + (i + 1));
            System.out.print(s: "Ingrese el nombre del laboratorio: ");
            String nombre = leer.next();
            System.out.print(s: "Ingrese la capacidad del laboratorio: ");
            int capacidad = leer.nextInt();
            System.out.print(s: "Ingrese la cantidad de equipos del laboratorio:");
            int numEquipos = leer.nextInt();
            //llamaremos el List para la lsita del arreglo de los equipos
            List<String> equipos = new ArrayList<>();
            //for (inicializamos, dondicion, aumento o decremento)
            for (int j = 0; j < numEquipos; j++) {
                System.out.print("Ingrese el nombre del equipo #" + (j + 1) + ": ");
                String equipo = leer.next();
                //nos permite añadir un elemto al final del ArrayList
                equipos.add(e: equipo);
            }
            System.out.println(x: "Seleccione el tipo de laboratorio:");
            System.out.println(x: "1. Laboratorio1");
            System.out.println(x: "2. Laboratorio2");
            int tipo = leer.nextInt();
            //llamamos el nombre de la clase Principal Laboratorio y el objeto
            //laboratorio
            Laboratorio laboratorio;
            //creamos el if para comparar el tipo de numero que sea igual 1
            if (tipo == 1) {
                System.out.print(s: "Ingrese el espacio del Laboratorio1: ");
                String espacio = leer.next();
                //llamamos al objeto laboratorio y llamamos al la clase hija
                //Laboratorio1 con sus atributos de la clase principal Laboratorio
                // y la clase secundaria Laboratorio1
                laboratorio = new Laboratorio1(nombre, capacidad, equipos, espacio);
            } else {
                System.out.print(s: "Ingrese el lugar del Laboratorio2: ");
                String lugar = leer.next();
                //llamamos al objeto laboratorio y llamamos al la clase hija
                //Laboratorio1 con sus atributos de la clase principal Laboratorio
                // y la clase secundaria Laboratorio2
                laboratorio = new Laboratorio2(nombre, capacidad, equipos, lugar);
            }
            //nos permite añadir un elemto al final del ArrayList
            laboratorios.add(e: laboratorio);
        }
```

Imagen 14. Principal

```java
        System.out.print(s: "Ingrese la cantidad de asignaturas a agregar: ");
        int numAsignaturas = leer.nextInt();
        //Llamamos el arreglo List de Asignatura del arreglo ArrayList
        List<Asignatura> asignaturas = new ArrayList<>();
        //for (inicializamos, dondicion, aumento o decremento)
        for (int i = 0; i < numAsignaturas; i++) {
            System.out.println("Asignatura #" + (i + 1));
            System.out.print(s: "Ingrese el nombre de la asignatura: ");
            String nombre = leer.next();
            System.out.print(s: "Ingrese el horario de la asignatura: ");
            String horario = leer.next();

            System.out.println(x: "Seleccione el laboratorio asignado para la asignatura:")
            //for (inicializamos, condicion, el size nos devuelve un entero con
            //el tamaño de la lista, aumento o decremento)
            for (int j = 0; j < laboratorios.size(); j++) {
                //nos dara el incremento del laboratorio y llamamos al metodo getNombre
                System.out.println((j + 1) + ". " + laboratorios.get(index: j).getNombre());
            }

            int labIndex = leer.nextInt() - 1;
            //llamaremos a la clase principal laboratorio y creamos el objeto
            //y llamaos el metodo
            Laboratorio laboratorio = laboratorios.get(index: labIndex);
            //llamaremos a la clase principal Asignatura y creamos el objeto y llamaos
            //los atributos de la clase asignatura
            Asignatura asignatura = new Asignatura(nombre, horario, laboratorio);
            //nos permite añadir un elemto al final del ArrayList
            asignaturas.add(e: asignatura);
        }
        asignaturas.add(e: asignatura);
    }
    //imprimimos todos los metodos de la clase y instanciamos.
    System.out.println(x: "Información de los laboratorios:");
    for (Laboratorio laboratorio : laboratorios) {
        System.out.println("Nombre: " + laboratorio.getNombre());
        System.out.println("Capacidad: " + laboratorio.getCapacidad());
        System.out.println("Equipos: " + laboratorio.getEquipos());
        if (laboratorio instanceof Laboratorio1) {
            System.out.println("Espacio: " + ((Laboratorio1) laboratorio).getEspacio());
        } else if (laboratorio instanceof Laboratorio2) {
            System.out.println("Lugar: " + ((Laboratorio2) laboratorio).getLugar());
        }
        System.out.println();
        }

        leer.close();
    }
}
```

Imagen 15. Principal2

3.4.4. Código del programa

Clase Laboratorio	Clase Asignatura

import java.util.List;
public class Laboratorio {
//Atributo
private String nombre;
private int capacidad;
private List<String> equipos;
//constructores
public Laboratorio(String nombre, int capacidad, List<String> equipos) {
this.nombre = nombre;
	package sistema_gestion_horario;
	public class Asignatura {
	//Atributos
	private String nombre;
	private String horario;
	private Laboratorio laboratorio;
	//constructores
	public Asignatura(String nombre, String horario, Laboratorio laboratorio) {
	this.nombre = nombre;

this.capacidad = capacidad;
this.equipos = equipos;
}
//metodos get para mostrar datos
public String getNombre() {
return nombre;
}
//metodos get para mostrar datos
public int getCapacidad() {
return capacidad;
}
//metodos get para mostrar datos
public List<String> getEquipos() {
return equipos;
}
public boolean verificarDisponibilidad(String horario) {
 	
return true;
}
}
	this.horario = horario;
	this.laboratorio = laboratorio;
	}
	//metodos get para mostrar datos
	public String getNombre() {
	return nombre;
	}
	//metodos get para mostrar datos
	public String getHorario() {
	return horario;
	}
	//metodos get para mostrar datos
	public Laboratorio getLaboratorio() {
	return laboratorio;
	}
	}

Clase Laboratorio1	**Clase Laboratorio2**

package sistema_gestion_horario;
 	
import java.util.List;
//utilizamos el extends para hacer herencia del Laboratorio que hereda los atributos de la clase
public class Laboratorio1 extends Laboratorio {
private String espacio;
//constructor
public Laboratorio1(String nombre, int capacidad, List<String> equipos, String espacio) {
super(nombre, capacidad, equipos);
this.espacio = espacio;
}
//metodos get para mostrar datos
public String getEspacio() {
return espacio;
}
}
	package sistema_gestion_horario;
	import java.util.List;
	//utilizamos el extends para hacer herencia del Laboratorio que hereda los atributos de la clase
	public class Laboratorio2 extends Laboratorio {
	//Atributo
	private String lugar;
	//constructor
	public Laboratorio2(String nombre, int capacidad, List<String> equipos, String lugar) {
	super(nombre, capacidad, equipos);
	this.lugar = lugar;
	}
	//metodos get para mostrar datos
	public String getLugar() {
	return lugar;
	}
	}

Princiala	
int labIndex = leer.nextInt() - 1; //llamaremos a la clase principal laboratorio y creamos el objeto y llamaos el metodo	System.out.print("Ingrese la cantidad de asignaturas a agregar: "); int numAsignaturas = leer.nextInt();

```
        Laboratorio laboratorio = laboratorios.get(labIndex);
        //llamaremos a la clase principal Asignatura y creamos el objeto y llamaos los atributos de la clase asignatura
        Asignatura asignatura = new Asignatura(nombre, horario, laboratorio);
        //nos permite añadir un elemto al final del ArrayList
        asignaturas.add(asignatura);
      }
      //imprimimos todos los metodos de la clase y instanciamos.
      System.out.println("Información de los laboratorios:");
      for (Laboratorio laboratorio : laboratorios) {
        System.out.println("Nombre: " + laboratorio.getNombre());
        System.out.println("Capacidad: " + laboratorio.getCapacidad());
        System.out.println("Equipos: " + laboratorio.getEquipos());
        if (laboratorio instanceof Laboratorio1) {
          System.out.println("Espacio: " + ((Laboratorio1) laboratorio).getEspacio());
        } else if (laboratorio instanceof Laboratorio2) {
          System.out.println("Lugar: " + ((Laboratorio2) laboratorio).getLugar());
        }
        System.out.println();
      }

      leer.close();
    }
}
```

```
      //Llamamos el arreglo List de Asignatura del arreglo ArrayList
      List<Asignatura> asignaturas = new ArrayList<>();
      //for (inicializamos, dondicion, aumento o decremento)
      for (int i = 0; i < numAsignaturas; i++) {
        System.out.println("Asignatura #" + (i + 1));
        System.out.print("Ingrese el nombre de la asignatura: ");
        String nombre = leer.next();
        System.out.print("Ingrese el horario de la asignatura: ");
        String horario = leer.next();

        System.out.println("Seleccione el laboratorio asignado para la asignatura:");
        //for (inicializamos, condicion, el size nos devuelve un entero con el tamaño de la lista, aumento o decremento)
        for (int j = 0; j < laboratorios.size(); j++) {
          //nos dara el incremento del laboratorio y llamamos al metodo getNombre
          System.out.println((j + 1) + ". " + laboratorios.get(j).getNombre());
        }

        int labIndex = leer.nextInt() - 1;
        //llamaremos a la clase principal laboratorio y creamos el objeto y llamaos el metodo
        Laboratorio laboratorio = laboratorios.get(labIndex);
        //llamaremos a la clase principal Asignatura y creamos el objeto y llamaos los atributos de la clase asignatura
        Asignatura asignatura = new Asignatura(nombre, horario, laboratorio);
        //nos permite añadir un elemto al final del ArrayList
        asignaturas.add(asignatura);
      }
      //imprimimos todos los metodos de la clase y instanciamos.
      System.out.println("Información de los laboratorios:");
      for (Laboratorio laboratorio : laboratorios) {
        System.out.println("Nombre: " + laboratorio.getNombre());
```

```
            System.out.println("Capacidad: " +
laboratorio.getCapacidad());
            System.out.println("Equipos: " +
laboratorio.getEquipos());
            if (laboratorio instanceof Laboratorio1) {
                System.out.println("Espacio: " + ((Laboratorio1)
laboratorio).getEspacio());
            } else if (laboratorio instanceof Laboratorio2) {
                System.out.println("Lugar: " + ((Laboratorio2)
laboratorio).getLugar());
            }
            System.out.println();
        }

        leer.close();
    }
}
```

4. Conclusiones

- Mediante en un artículo científico nos dieron a conocer la utilización de herencia de la manera adecuada y evitar una jerarquía demasiado compleja.
- Utilizar atributos constructores ya que se los constructores adecuados para establecer valores iniciales y mantener un estado coherente del objeto.
- Las validaciones son para implementar validaciones en atributos y métodos para garantizar la integridad y consistencia de los datos

5. Recomendaciones

- Mediante este ejercicio de sistema de horarios para laboratorio tuvimos algunos problemas ya que utilizamos tipos de arreglos en nuestro ejercicio para ingresar datos de laboratorios, materia tuvimos que ver videos para resolver dicho ejercicio.

- Recomiendo realizar mas ejercicios que ayuden a la compresión de herencia polimorfismo, sobrecarga, la utilización de arreglos ya que es muy importante para aplicar a nuestro ejercicio.

6. Bibliografía

Drake, J. (2009). *Ingeneria de Programacion*. Obtenido de Ingeneria de Programacion: https://www.ctr.unican.es/asignaturas/Ingenieria_Software_4_F/Doc/M7_09_VerificacionValidacion-2011.pdf

Fernandez, O. B. (s.f.). *Programación Avanzada*. Obtenido de Polimorfismo y sobrecarga: http://www3.uji.es/~belfern/Docencia/Presentaciones/ProgramacionAvanzada/Tema1/conceptosPolimorfismoSobrecarga.html#18

KeepCoding. (3 de Mayo de 2023). *KEEPCODING*. Obtenido de KEEPCODING: https://keepcoding.io/blog/que-son-las-pruebas-unitarias-de-software/

Solis, F. (22 de Abril de 2015). *slideshare.* Obtenido de slideshare: https://es.slideshare.net/efsolis/5-47311297

tech. (29 de Julio de 2022). Obtenido de tech: https://www.techtitute.com/ec/informatica/cursos-software/blog/depuracion-prueba-java

Vazquez, J. (s.f.). *Programación Orientada a Objetos.* Obtenido de Programación Orientada a Objetos: http://ri.uaemex.mx/bitstream/handle/20.500.11799/34080/secme-16400.pdf?sequence=1

Link de video
https://www.youtube.com/watch?v=UdLsYp31z-0&t=16s
https://www.youtube.com/watch?v=_BZtjmc-2y0
https://www.youtube.com/watch?v=-FIUWVQnZVE
https://www.youtube.com/watch?v=92WfJeeYdX8