Instituto Tecnológico de Orizaba

TECNOLÓGICO NACIONAL DE MÉXICO

CARRERA

ING. INFORMATICA

ASIGNATURA

PROGRAMACION ORIENTADA A OBJETOS

TEMA 4

HERENCIA Y POLIFIRMISMO

ALUMN@:

MAYTE MELLADO HUERTA

KEVIN ALAN ORTIZ FLORES

ANGEL DE JESUS CONTRERAS HERRERA

NO.CONTROL

21010202

21010207

21010180

GRUPO

2a3B

FECHA DE ENTREGA

02/06/2023

INTRODUCCION

El tema principal de este reporte es sobre herencia y polimorfismo en la programación, y se abordan varios subtemas relacionados. A lo largo de este informe, exploraremos en detalle cada uno de estos subtemas, comprendiendo su funcionamiento y aplicando los conceptos aprendidos en ejercicios prácticos.

4.1 Concepto de Herencia y Polimorfismo:

En el desarrollo de software, la herencia y el polimorfismo son conceptos fundamentales que permiten la reutilización de código y la creación de jerarquías de clases. La herencia es un mecanismo mediante el cual una clase puede heredar características y comportamientos de otra clase, conocida como clase base o superclase. El polimorfismo, por otro lado, permite que un objeto pueda comportarse de diferentes formas según el contexto en el que se utilice.

4.2 Definición de una Clase Base:

Una clase base es la clase de la cual se derivan otras clases, también conocidas como clases derivadas o subclases. La clase base define un conjunto de atributos y métodos que pueden ser heredados por las clases derivadas. Estos atributos y métodos proporcionan la base común para todas las clases que heredan de ella.

4.3 Definición de una Clase Derivada:

Una clase derivada, también llamada subclase, es una clase que hereda características y comportamientos de una clase base. La clase derivada puede extender o modificar la funcionalidad heredada de la clase base, agregando nuevos atributos y métodos, o sobrescribiendo los existentes. Esto permite crear una jerarquía de clases en la cual cada clase puede tener características específicas adicionales a las de la clase base.

4.4 Clases Abstractas:

Una clase abstracta es una clase que no puede ser instanciada directamente, sino que se utiliza como base para otras clases. Las clases abstractas proporcionan una estructura común y definiciones de métodos que las clases derivadas deben implementar. Estas clases pueden contener métodos abstractos, los cuales deben ser definidos en las clases derivadas. La idea principal de las clases abstractas es proveer una plantilla o contrato para las clases que las heredan.

4.5 Interfaces:

4.5.1 Definición:

Una interfaz es una colección de métodos abstractos que definen un conjunto de comportamientos. A diferencia de las clases abstractas, las interfaces no pueden tener atributos ni implementaciones de métodos concretos. Las interfaces definen qué métodos deben ser implementados por cualquier clase que las implemente. Esto permite la creación de contratos que las clases deben cumplir, asegurando la consistencia y el polimorfismo.

4.5.2 Implementación:

Para implementar una interfaz en una clase, se utiliza la palabra clave "implements" seguida del nombre de la interfaz. La clase debe proporcionar una implementación de todos los métodos declarados en la interfaz. Esto garantiza que la clase cumpla con el contrato definido por la interfaz y pueda ser tratada como tal.

4.5.3 Variables Polimórficas:

Las variables polimórficas son variables que pueden almacenar referencias a objetos de diferentes clases que sean subclases de una misma clase base o implementen una misma interfaz. Esto permite tratar a los objetos de diferentes clases de manera uniforme, utilizando la interfaz común definida por la clase base o la interfaz implementada. La elección del comportamiento concreto se determina en tiempo de ejecución según el tipo real del objeto almacenado en la variable.

4.6 Reutilización de la Definición de Paquetes/Librerías:

La reutilización de la definición de paquetes o librerías es un enfoque para aprovechar el código existente en diferentes proyectos. Los paquetes o librerías contienen un conjunto de clases y recursos que pueden ser utilizados por otros programas. Al importar y utilizar estos paquetes o librerías en nuevos proyectos, se evita la necesidad de volver a escribir código desde cero, lo cual aumenta la eficiencia y la consistencia en el desarrollo de software.

En resumen, la herencia y el polimorfismo son conceptos esenciales en la programación orientada a objetos que permiten la reutilización de código y la creación de estructuras jerárquicas. Las clases base y derivadas, así como las clases abstractas, proporcionan una forma de organizar y extender la funcionalidad de las clases. Las interfaces definen contratos que las clases deben cumplir, permitiendo el polimorfismo y la utilización de variables polimórficas. Además, la reutilización de la definición de paquetes o librerías es una práctica que promueve la eficiencia y la consistencia en el desarrollo de software.

COMPETENCIA ESPECIPICA:

Comprende y aplica los conceptos de herencia y polimorfismo en programas que utilicen clases base, clases derivadas, clases abstractas e interfaces.

MARCO TEÓRICO:

La programación orientada a objetos (POO) es un paradigma de programación ampliamente utilizado que se basa en el concepto de objetos, los cuales son instancias de clases que encapsulan datos y comportamientos relacionados. Dos conceptos clave en la POO son la herencia y el polimorfismo, los cuales permiten una mayor flexibilidad y reutilización de código en el desarrollo de software.

1. Herencia:

La herencia es un mecanismo fundamental en la POO que permite que una clase, llamada clase derivada o subclase, herede los atributos y métodos de otra clase, conocida como clase base o superclase. La clase derivada puede extender y especializar la funcionalidad de la clase base, añadiendo nuevos atributos y

métodos, o modificando los existentes. Esto facilita la creación de jerarquías de clases, donde las clases derivadas heredan y amplían la funcionalidad de la clase base. La herencia promueve la reutilización de código, ya que las clases derivadas pueden aprovechar y construir sobre el trabajo realizado en la clase base.

2. Polimorfismo:

El polimorfismo es otro concepto clave en la POO que se refiere a la capacidad de un objeto de tomar diferentes formas o comportarse de diferentes maneras según el contexto en el que se utilice. El polimorfismo permite que una variable de un tipo más general, como la clase base o una interfaz, pueda referenciar objetos de diferentes tipos concretos, es decir, objetos de las clases derivadas que heredan de la clase base. Esto permite tratar a los objetos de manera uniforme, independientemente de su tipo específico, lo que conduce a un código más flexible y mantenible. El polimorfismo se basa en la capacidad de las clases derivadas de sobrescribir los métodos heredados de la clase base, proporcionando una implementación específica para cada clase derivada.

3. Clases Abstractas:

Las clases abstractas son aquellas que no pueden ser instanciadas directamente, sino que se utilizan como base para otras clases. Estas clases contienen uno o más métodos abstractos, los cuales son declarados pero no implementados en la clase abstracta. Las clases derivadas de una clase abstracta deben proporcionar una implementación concreta de los métodos abstractos. Las clases abstractas son útiles para definir una estructura común y establecer un contrato que las clases derivadas deben cumplir. Además de los métodos abstractos, una clase abstracta puede contener métodos concretos que ya tienen una implementación definida.

4. Interfaces:

Las interfaces son similares a las clases abstractas en el sentido de que también definen un contrato que las clases deben cumplir, pero a diferencia de las clases abstractas, no pueden contener implementaciones de métodos. Una interfaz define un conjunto de métodos abstractos que deben ser implementados por cualquier

clase que la implemente. Una clase puede implementar múltiples interfaces, lo que proporciona flexibilidad en el diseño y permite que las clases cumplan con diferentes contratos. Las interfaces promueven el polimorfismo al permitir que las clases sean tratadas en función de la interfaz que implementan, en lugar de su tipo específico.

MATERIAL Y EQUIPO:

- ♥ Computadora
- ♥ NetBeans IDE 8.3
- ♥ PDF y presentaciones dadas por el docente

DESARROLLO DE LA PRACTICA

Paso 1: Concepto de Herencia y Polimorfismo

Explica en qué consiste la herencia y el polimorfismo, y su importancia en el desarrollo de software orientado a objetos.

Paso 2: Definición de una Clase Base

Describe qué es una clase base o superclase, y cómo se utiliza como base para derivar otras clases. Explica cómo se heredan atributos y métodos de la clase base a las clases derivadas.

Paso 3: Definición de una Clase Derivada

Explica qué es una clase derivada o subclase, y cómo hereda características y comportamientos de la clase base. Menciona cómo se pueden extender o modificar los atributos y métodos heredados en las clases derivadas.

Paso 4: Clases Abstractas

Describe qué es una clase abstracta y cómo se diferencia de una clase concreta. Explica cómo se utilizan las clases abstractas como plantillas para definir métodos abstractos que deben ser implementados en las clases derivadas.

Paso 5: Interfaces

Explica qué es una interfaz y cómo se utiliza para definir un conjunto de comportamientos. Describe cómo se implementan interfaces en las clases utilizando la palabra clave "implements", y cómo se deben implementar todos los métodos definidos en la interfaz.

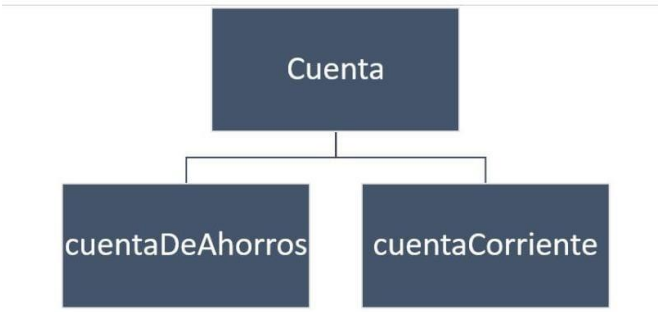
Paso 6: Variables Polimórficas

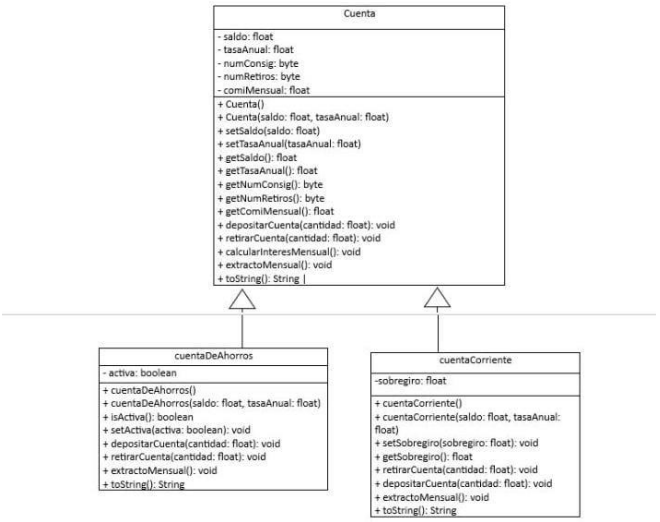
Explica qué son las variables polimórficas y cómo se utilizan para almacenar referencias a objetos de diferentes clases que cumplen con una interfaz común o son subclases de una clase base. Menciona cómo el comportamiento concreto se determina en tiempo de ejecución según el tipo real del objeto almacenado en la variable.

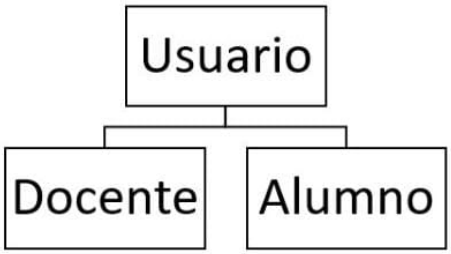
Paso 7: Reutilización de la Definición de Paquetes/Librerías

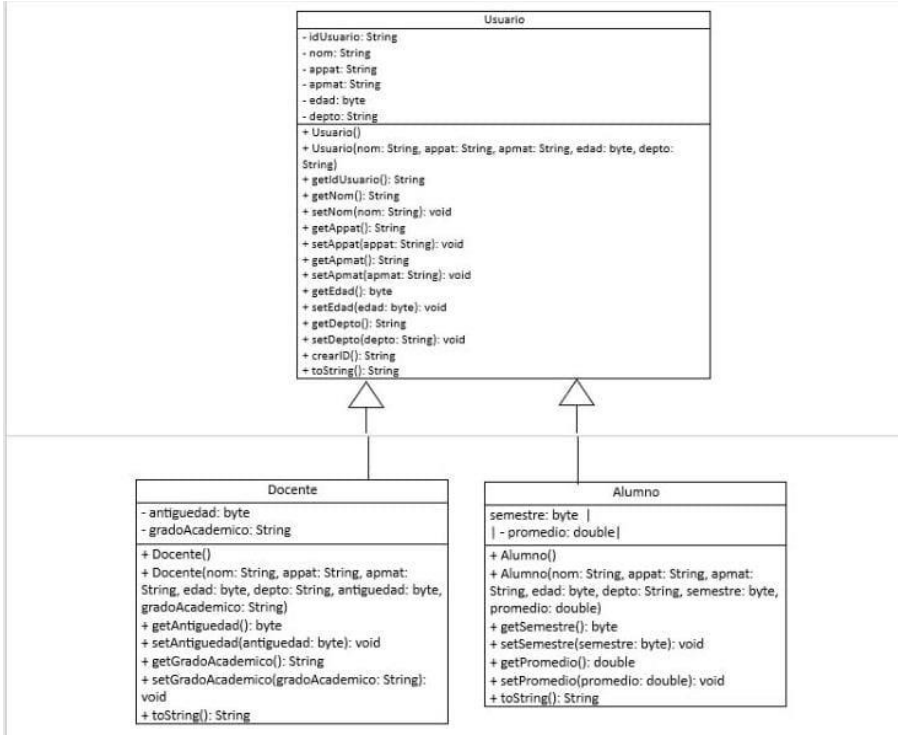
Describe cómo se puede reutilizar la definición de paquetes o librerías en diferentes proyectos para aprovechar el código existente. Explica cómo se importan y utilizan los paquetes o librerías en nuevos proyectos, evitando la necesidad de volver a escribir código desde cero.

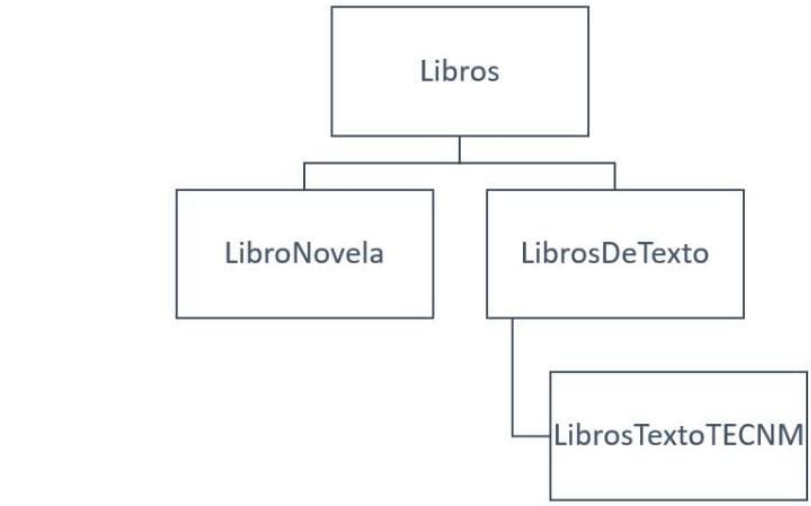
A continuación, se presentarán los diagramas de clase de todos los programas realizados:




Cuenta
- saldo: float
- tasaAnual: float
- numConsig: byte
- numRetiros: byte
- comiMensual: float
+ Cuenta()
+ Cuenta(saldo: float, tasaAnual: float)
+ setSaldo(saldo: float)
+ setTasaAnual(tasaAnual: float)
+ getSaldo(): float
+ getTasaAnual(): float
+ getNumConsig(): byte
+ getNumRetiros(): byte
+ getComiMensual(): float
+ depositarCuenta(cantidad: float): void
+ retirarCuenta(cantidad: float): void
+ calcularInteresMensual(): void
+ extractoMensual(): void
+ toString(): String |
cuentaDeAhorros
- activa: boolean
+ cuentaDeAhorros()
+ cuentaDeAhorros(saldo: float, tasaAnual: float)
+ isActiva(): boolean
+ setActiva(activa: boolean): void
+ depositarCuenta(cantidad: float): void
+ retirarCuenta(cantidad: float): void
+ extractoMensual(): void
+ toString(): String
cuentaCorriente
-sobregiro: float
+ cuentaCorriente()
+ cuentaCorriente(saldo: float, tasaAnual: float)
+ setSobregiro(sobregiro: float): void
+ getSobregiro(): float
+ retirarCuenta(cantidad: float): void
+ depositarCuenta(cantidad: float): void
+ extractoMensual(): void
+ toString(): String


Usuario
Docente
Alumno


Usuario
- idUsuario: String
- nom: String
- appat: String
- apmat: String
- edad: byte
- depto: String
+ Usuario()
+ Usuario(nom: String, appat: String, apmat: String, edad: byte, depto: String)
+ getIdUsuario(): String
+ getNom(): String
+ setNom(nom: String): void
+ getAppat(): String
+ setAppat(appat: String): void
+ getApmat(): String
+ setApmat(apmat: String): void
+ getEdad(): byte
+ setEdad(edad: byte): void
+ getDepto(): String
+ setDepto(depto: String): void
+ crearID(): String
+ toString(): String
Docente
- antiguedad: byte
- gradoAcademico: String
+ Docente()
+ Docente(nom: String, appat: String, apmat: String, edad: byte, depto: String, antiguedad: byte, gradoAcademico: String)
+ getAntiguedad(): byte
+ setAntiguedad(antiguedad: byte): void
+ getGradoAcademico(): String
+ setGradoAcademico(gradoAcademico: String): void
+ toString(): String
Alumno
semestre: byte |
| - promedio: double|
+ Alumno()
+ Alumno(nom: String, appat: String, apmat: String, edad: byte, depto: String, semestre: byte, promedio: double)
+ getSemestre(): byte
+ setSemestre(semestre: byte): void
+ getPromedio(): double
+ setPromedio(promedio: double): void
+ toString(): String



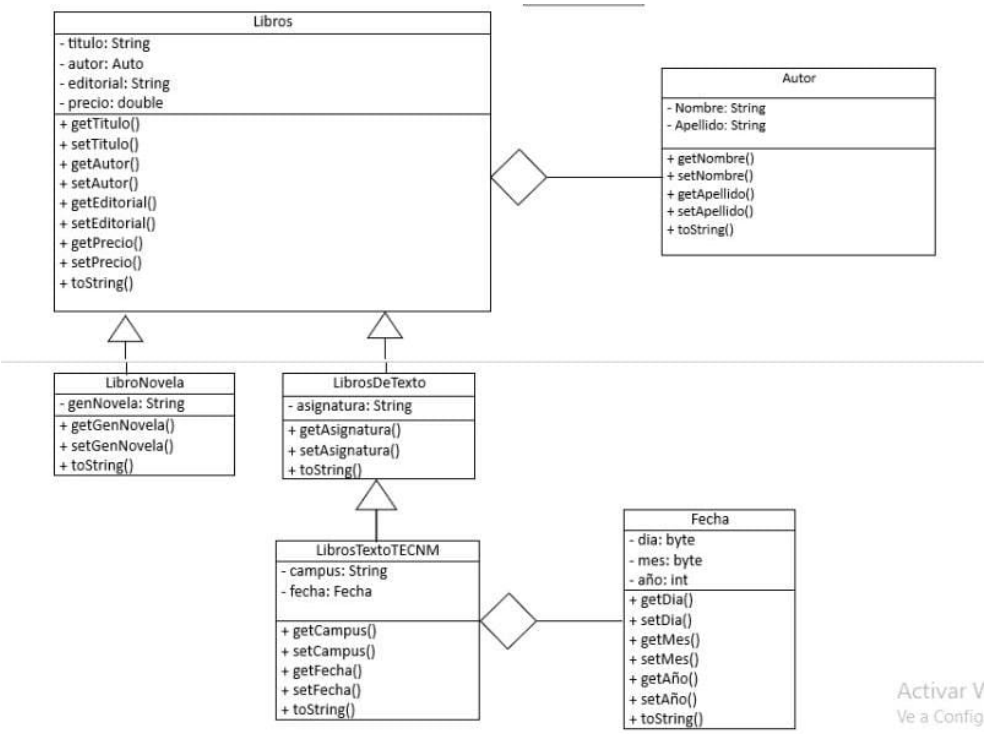



Figura
Cuadrado
Triangulo
Rectangulo

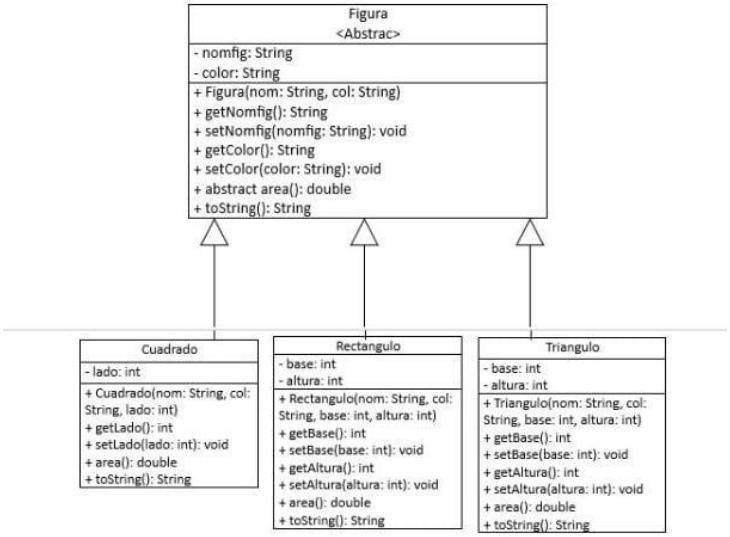
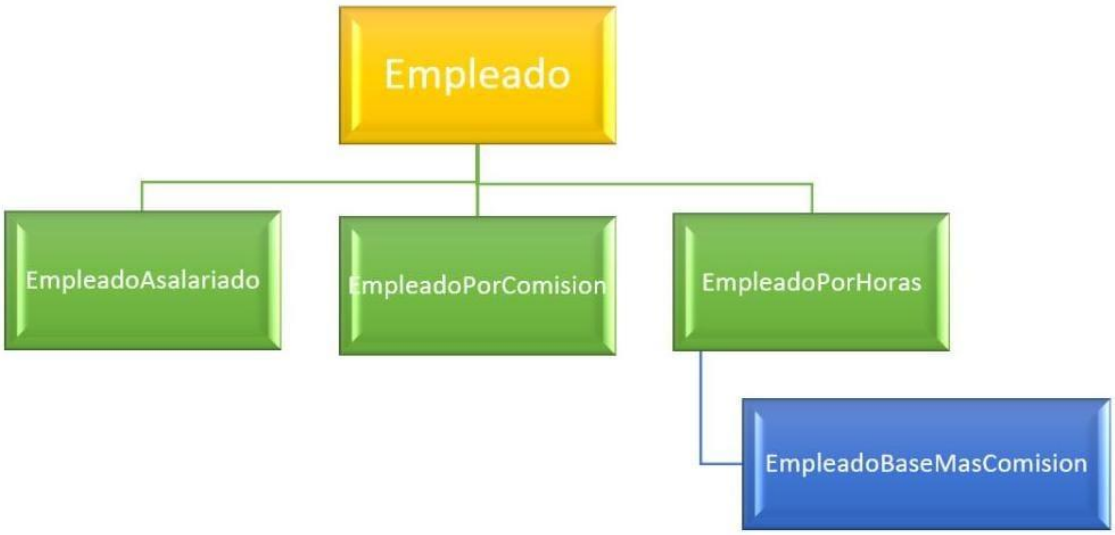
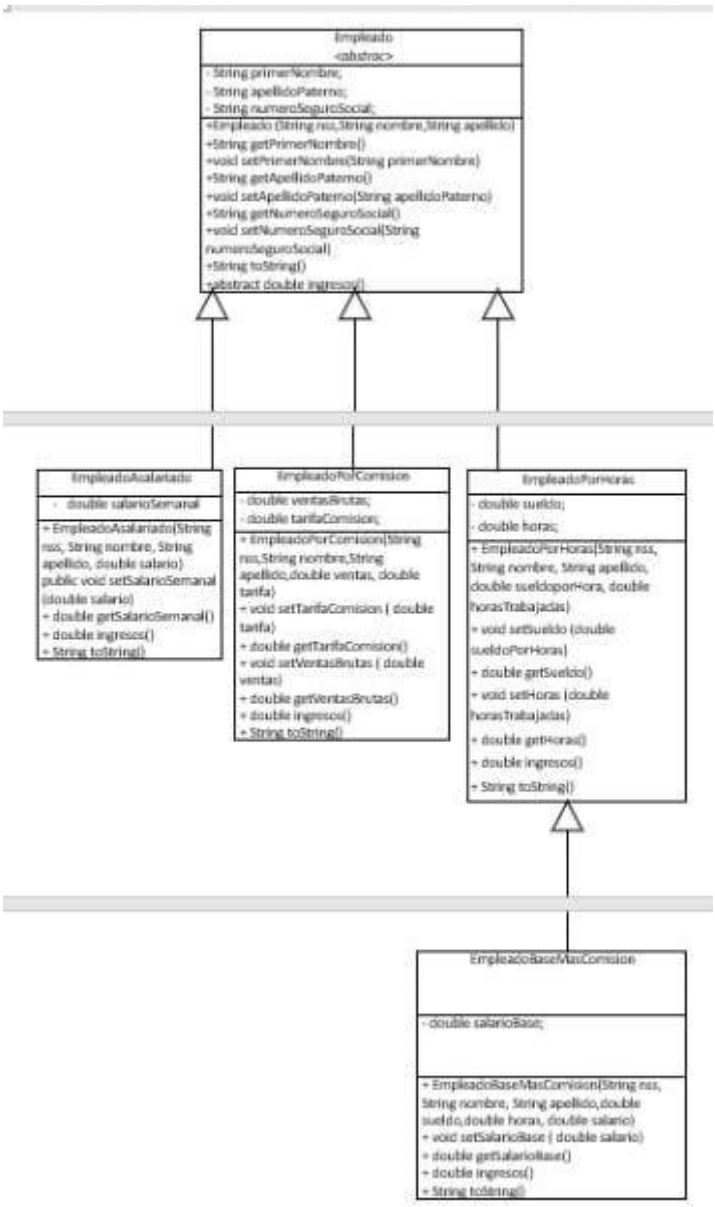


Figura
<Abstrac>
- nomfig: String
- color: String
+ Figura(nom: String, col: String)
+ getNomfig(): String
+ setNomfig(nomfig: String): void
+ getColor(): String
+ setColor(color: String): void
+ abstract area(): double
+ toString(): String
Cuadrado
- lado: int
+ Cuadrado(nom: String, col: String, lado: int)
+ getLado(): int
+ setLado(lado: int): void
+ area(): double
+ toString(): String
Rectangulo
- base: int
- altura: int
+ Rectangulo(nom: String, col: String, base: int, altura: int)
+ getBase(): int
+ setBase(base: int): void
+ getAltura(): int
+ setAltura(altura: int): void
+ area(): double
+ toString(): String
Triangulo
- base: int
- altura: int
+ Triangulo(nom: String, col: String, base: int, altura: int)
+ getBase(): int
+ setBase(base: int): void
+ getAltura(): int
+ setAltura(altura: int): void
+ area(): double
+ toString(): String


Empleado
EmpleadoAsalariado
EmpleadoPorComision
EmpleadoPorHoras
EmpleadoBaseMasComision


Empleado
<abstract>
- String primerNombre;
- String apellidoPaterno;
- String numeroSeguroSocial;
+Empleado (String nss,String nombre,String apellido)
+String getPrimerNombre()
+void setPrimerNombre(String primerNombre)
+String getApellidoPaterno()
+void setApellidoPaterno(String apellidoPaterno)
+String getNumeroSeguroSocial()
+void setNumeroSeguroSocial(String numeroSeguroSocial)
+String toString()
+abstract double ingresos()
EmpleadoAsalariado
- double salarioSemanal
+ EmpleadoAsalariado(String nss, String nombre, String apellido, double salario)
public void setSalarioSemanal (double salario)
+ double getSalarioSemanal()
+ double ingresos()
+ String toString()
EmpleadoPorComision
- double ventasBrutas;
- double tarifaComision;
+ EmpleadoPorComision(String nss,String nombre,String apellido,double ventas, double tarifa)
+ void setTarifaComision (double tarifa)
+ double getTarifaComision()
+ void setVentasBrutas (double ventas)
+ double getVentasBrutas()
+ double ingresos()
+ String toString()
EmpleadoPorHoras
- double sueldo;
- double horas;
+ EmpleadoPorHoras(String nss, String nombre, String apellido, double sueldoporHora, double horasTrabajadas)
+ void setSueldo (double sueldoPorHoras)
+ double getSueldo()
+ void setHoras (double horasTrabajadas)
+ double getHoras()
+ double ingresos()
+ String toString()
EmpleadoBaseMasComision
- double salarioBase;
+ EmpleadoBaseMasComision(String nss, String nombre, String apellido,double sueldo,double horas, double salario)
+ void setSalarioBase (double salario)
+ double getSalarioBase()
+ double ingresos()
+ String toString()

RESULTADOS

```java
package Herencia;

public class Usuario {

    protected String idUsuario;
    protected String nom;
    protected String appat;
    protected String apmat;
    protected byte edad;
    protected String depto;

    public Usuario() {
    }

    public Usuario(String nom, String appat, String apmat, byte edad,
            String depto) {
        this.nom = nom;
        this.appat = appat;
        this.apmat = apmat;
        this.edad = edad;
        this.depto = depto;
        this.idUsuario = crearID();
    }



    public String getIdUsuario() {
        return idUsuario;
    }

    public String getNom() {
        return nom;
    }

    public void setNom(String nom) {
        this.nom = nom;
    }

    public String getAppat() {
        return appat;
    }

    public void setAppat(String appat) {
        this.appat = appat;
    }

```

```java
    public String getApmat() {
        return apmat;
    }

    public void setApmat(String apmat) {
        this.apmat = apmat;
    }

    public byte getEdad() {
        return edad;
    }

    public void setEdad(byte edad) {
        this.edad = edad;
    }

    public String getDepto() {
        return depto;
    }

    public void setDepto(String depto) {
        this.depto = depto;
    }

    public String crearID(){
        String apellidoPaterno = appat.substring(0,3);
        String nombre = nom.substring(0,2);

        if (depto.equals("ISC")) {
            return apellidoPaterno + nombre;
        } else if (depto.equals("INF")) {
            return apellidoPaterno + "_" + nombre;
        }

        return "";
    }


    @Override
    public String toString() {
        return "Usuario{" + "idUsuario=" + idUsuario + ", nom=" + nom
                + ", appat=" + appat + ", apmat=" + apmat + ", edad=" + edad
                + ", depto=" + depto + '}';
    }

}
```

El código define la clase Usuario con atributos y métodos que permiten la creación de objetos de usuario con información personal y la generación de un identificador único.

```java
package Herencia;

public class Docente extends Usuario{
    private byte antiguedad;
    private String gradoAcademico;

    public Docente() {
        super();
    }

    public Docente(String nom, String appat, String apmat, byte edad,
            String depto,byte antiguedad, String gradoAcademico) {
        super(nom, appat, apmat, edad, depto);
        this.antiguedad = antiguedad;
        this.gradoAcademico = gradoAcademico;
    }

    public byte getAntiguedad() {
        return antiguedad;
    }

    public void setAntiguedad(byte antiguedad) {
        this.antiguedad = antiguedad;
    }

    public String getGradoAcademico() {
        return gradoAcademico;
    }

    public void setGradoAcademico(String gradoAcademico) {
        this.gradoAcademico = gradoAcademico;
    }

    @Override
    public String toString() {
        return super.toString()+"Docente{" + "antiguedad=" + antiguedad
                + ", gradoAcademico=" + gradoAcademico + '}';
    }

}
```

La clase Docente extiende la clase Usuario y agrega atributos y métodos específicos para representar a un docente con su antigüedad y grado académico. Al heredar de la clase Usuario, la clase Docente puede aprovechar la funcionalidad y los atributos definidos en la clase Usuario`, promoviendo la reutilización de código y siguiendo el principio de herencia en la programación orientada a objetos.

```java
package Herencia;

public class Alumno extends Usuario{

    private byte semestre;
    private double promedio;

    public Alumno() {
        super();
    }

    public Alumno(String nom, String appat, String apmat, byte edad,
            String depto,byte semestre, double promedio) {
        super(nom, appat, apmat, edad, depto);
        this.semestre = semestre;
        this.promedio = promedio;
    }

    public byte getSemestre() {
        return semestre;
    }

    public void setSemestre(byte semestre) {
        this.semestre = semestre;
    }

    public double getPromedio() {
        return promedio;
    }

    public void setPromedio(double promedio) {
        this.promedio = promedio;
    }

    @Override
    public String toString() {
        return super.toString()+"Alumno{" + "semestre=" + semestre
                + ", promedio=" + promedio + '}';
    }


}
```

La clase Alumno extiende la clase Usuario y agrega atributos y métodos específicos para representar a un alumno con su semestre y promedio. Al heredar de la clase Usuario, la clase Alumno puede aprovechar la funcionalidad y los atributos definidos en la clase Usuario, promoviendo la reutilización de código y siguiendo el principio de herencia en la programación orientada a objetos.

```java
package Test;

import EntradaSalida.Tools;
import Herencia.Docente;
import Herencia.Alumno;

public class testUsuario {
    public static void main (String [] args) {

        Docente docente = new Docente("Maria Jacinta", "Martinez", "Castillo",
                (byte) 30, "INF", (byte) 10, "Maestría");
        Alumno alumno = new Alumno("Mayte", "Mellado", "Huerta",
                (byte) 20, "INF", (byte) 4, 9.0);

        Tools.imprimeSalida(docente.toString());
        Tools.imprimeSalida(alumno.toString());
    }
}
```

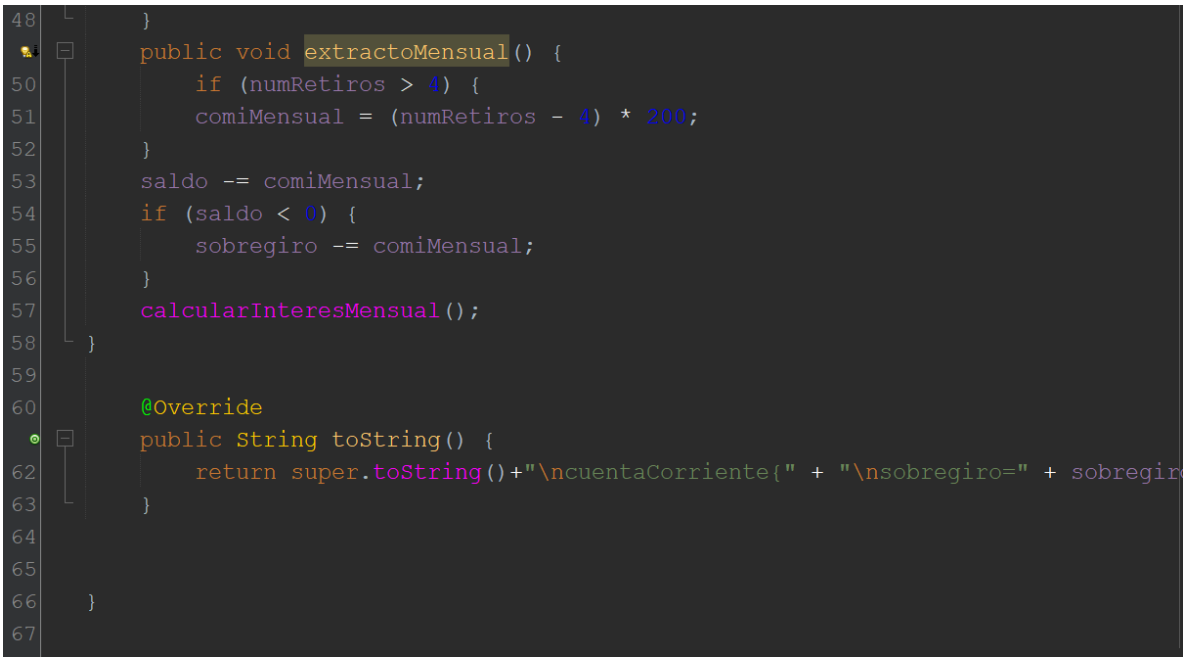
El código proporcionado es una clase de prueba llamada testUsuario, que se encuentra en el paquete Test. Esta clase demuestra el uso de las clases Docente y Alumno del paquete Herencia.

```java
package Herencia;

import EntradaSalida.Tools;

public class Cuenta {
    protected float saldo;
    protected float tasaAnual;
    protected byte numConsig=0;
    protected byte numRetiros=0;
    protected float comiMensual=0;

    public Cuenta() {}

    public Cuenta(float saldo, float tasaAnual) {

        this.saldo = saldo;
        this.tasaAnual = tasaAnual;

    }

    public void setSaldo(float saldo) {
        this.saldo = saldo;
    }

    public void setTasaAnual(float tasaAnual) {
        this.tasaAnual = tasaAnual;
    }

    public float getSaldo() {
        return saldo;
    }

    public float getTasaAnual() {
        return tasaAnual;
    }

    public byte getNumConsig() {
        return numConsig;
    }

    public byte getNumRetiros() {
        return numRetiros;
    }

    public float getComiMensual() {
        return comiMensual;
    }
```

```
48
    public void depositarCuenta (float cantidad){
50
        saldo+=cantidad;
51      numConsig++;
52
53  }
54  public void retirarCuenta(float cantidad){
56      if(cantidad<=saldo){
57          saldo-=cantidad;
58          numRetiros++;
59      }
60      else
61          Tools.salidaError("Lo siento, la cantidad a retirar excede tu saldo"
62
63  }
64
65
66  public void calcularInteresMensual(){
67  float tasaMensual = tasaAnual / 12;
68  saldo += saldo * tasaMensual;
69
70  }
    public void extractoMensual() {
72      saldo -= comiMensual;
73      calcularInteresMensual();
74
75  }
76
77  @Override
    public String toString() {
79      return "Cuenta{" + "saldo=" + saldo + "\nTasa Anual: " + tasaAnual
80              +"\nNúmero de Consignaciones: " + numConsig
81              +"\nNúmero de Retiros: " + numRetiros
82              +"\nComisión Mensual: " + comiMensual;
83
84  }
85 }
```

La clase Cuenta es una representación de una cuenta bancaria y proporciona métodos para realizar operaciones como depósitos, retiros, cálculo de intereses y generación de extractos mensuales. Los atributos de la cuenta se pueden establecer y obtener mediante métodos específicos.

```java
package Herencia;

import EntradaSalida.Tools;

public class cuentaCorriente extends Cuenta{
    private float sobregiro;

    public cuentaCorriente() {
        super();
    }

    public cuentaCorriente(float saldo, float tasaAnual) {
        super(saldo, tasaAnual);

    }
    public void setSobregiro(float sobregiro) {
        this.sobregiro = sobregiro;
    }

    public float getSobregiro() {
        return sobregiro;
    }
    public void retirarCuenta(float cantidad) {
        if (cantidad <= saldo + sobregiro) {
            if (cantidad <= saldo) {
                saldo -= cantidad;
            } else {
                sobregiro -= (cantidad - saldo);
                saldo = 0;
            }
            numRetiros++;
        } else {
            Tools.imprimeSalida("Lo siento, la cantidad a retirar excede el sald
        }
    }
    public void depositarCuenta(float cantidad) {
        if (sobregiro > 0) {
            if (cantidad <= sobregiro) {
                sobregiro -= cantidad;
            } else {
                saldo += (cantidad - sobregiro);
                sobregiro = 0;
            }
        } else {
            saldo += cantidad;
        }
        numConsig++;
```



```
48      }
        public void extractoMensual() {
50          if (numRetiros > 4) {
51          comiMensual = (numRetiros - 4) * 200;
52      }
53      saldo -= comiMensual;
54      if (saldo < 0) {
55          sobregiro -= comiMensual;
56      }
57      calcularInteresMensual();
58  }
59
60      @Override
        public String toString() {
62          return super.toString()+"\ncuentaCorriente{" + "\nsobregiro=" + sobregir
63      }
64
65
66  }
67
```

La clase cuentaCorriente extiende la funcionalidad de la clase Cuenta para adaptarla a una cuenta corriente bancaria, agregando el concepto de sobregiro. Proporciona métodos personalizados para manejar retiros, depósitos y generación de extractos mensuales, teniendo en cuenta tanto el saldo como el sobregiro disponible.

```java
package Herencia;

public class cuentaDeAhorros extends Cuenta{
    private boolean activa;

    public cuentaDeAhorros() {
    super();
    }
    public cuentaDeAhorros(float saldo, float tasaAnual) {
        super(saldo, tasaAnual);
        activa=(saldo>=1000)?false:true;
    }

    public boolean isActiva() {
        return activa;
    }

    public void setActiva(boolean activa) {
        this.activa = activa;
    }

    public void depositarCuenta(float cantidad){
        if(activa){
            super.depositarCuenta(cantidad);
        }
    }
    public void retirarCuenta(float cantidad){
        if(activa){
            super.retirarCuenta(cantidad);
        }

    }
    public void extractoMensual(){

        calcularInteresMensual(); // Primero calcular el interés mensual
        saldo -= comiMensual; // Restar la comisión mensual al saldo
        if (numRetiros > 4) {
        comiMensual += (numRetiros - 4) * 200;
        }
        if (saldo < 10000) {
        activa = false;
    }
}


    @Override
    public String toString() {
        return super.toString()+"\ncuentaDeAhorros{" + "\nactiva=" + activa + '}
    }

```

La clase cuentaDeAhorros extiende la funcionalidad de la clase Cuenta para adaptarla a una cuenta de ahorros bancaria. Agrega la noción de cuenta

activa/inactiva y proporciona métodos personalizados para manejar depósitos, retiros y generación de extractos mensuales, aplicando restricciones específicas para una cuenta de ahorros activa.

```java
package Test;

import EntradaSalida.Tools;
import Herencia.cuentaCorriente;
import Herencia.cuentaDeAhorros;

public class TestCuenta {
    public static void main(String [] args){

        menu("Cuenta Corriente,Cuenta de Ahorro,Salir");

    }
    public static void menu(String menu)
    {
        String sel="";
        do {
            sel=Tools.boton(menu);
            switch(sel){
                case "Cuenta Corriente":
                String cc = "Depositar,Retirar,Extracto Mensual,Imprimir,Salir";
                menuCC(cc);
                break;
                case "Cuenta de Ahorro":
                String ca = "Depositar,Retirar,Extracto Mensual,Imprimir,Salir";
                menuCA(ca);
                break;
                case "Salir":;  break;
            }
        }while(!sel.equalsIgnoreCase("Salir"));
    }
    public static void menuCA(String menu)
    {
        cuentaDeAhorros ca = new cuentaDeAhorros(15000, (float)2.5);
        String sel="";
        do {
            sel=Tools.boton(menu);
            switch(sel){
                case "Depositar":
                    Tools.imprimePantalla(ca.toString());
                    ca.depositarCuenta(Tools.leerFloat("Ingrese la cantidad a de
                break;
                case "Retirar":
                    Tools.imprimePantalla(ca.toString());
                    ca.retirarCuenta(Tools.leerFloat("Ingrese la cantidad que de
                break;
                case "Extracto Mensual":
                    ca.extractoMensual();
                    Tools.imprimePantalla(ca.toString());
                break;
                case "Imprimir":
```

```
51                      Tools.imprimePantalla(ca.toString());
52                  break;
53                  case "Salir":;  break;
54              }
55          }while(!sel.equalsIgnoreCase("Salir"));
56      }
57      public static void menuCC(String menu)
58      {
59          cuentaCorriente cc = new cuentaCorriente(15000, (float)2.5);
            String sel="";
61          do {
62              sel=Tools.boton(menu);
63              switch(sel){
64                  case "Depositar":
65                      Tools.imprimePantalla(cc.toString());
66                      cc.depositarCuenta(Tools.leerFloat("Ingrese la cantidad a de
67                  break;
68                  case "Retirar":
69                      Tools.imprimePantalla(cc.toString());
70                      cc.retirarCuenta(Tools.leerFloat("Ingrese la cantidad que de
71                  break;
72                  case "Extracto Mensual":
73                      cc.extractoMensual();
74                      Tools.imprimePantalla(cc.toString());
75                      break;
76                  case "Imprimir":
77                      Tools.imprimePantalla(cc.toString());
78                  break;
79                  case "Salir":;  break;
80              }
81          }while(!sel.equalsIgnoreCase("Salir"));
82      }
83
84
85  }
86
```

La clase TestCuenta es una clase de prueba que permite al usuario interactuar con las cuentas corrientes y de ahorros, realizando operaciones como depositar, retirar y ver el extracto mensual.

```java
package Herencia;

public class Libros {
    protected String titulo;
    protected Autor autor;
    protected String editorial;
    protected double precio;

    public Libros() {

    }

    public Libros(String titulo, Autor autor, String editorial, double precio) {
        this.titulo = titulo;
        this.autor = autor;
        this.editorial = editorial;
        this.precio = precio;
    }

    public String getTitulo() {
        return titulo;
    }

    public void setTitulo(String titulo) {
        this.titulo = titulo;
    }

    public Autor getAutor() {
        return autor;
    }

    public void setAutor(Autor autor) {
        this.autor = autor;
    }

    public String getEditorial() {
        return editorial;
    }

    public void setEditorial(String editorial) {
        this.editorial = editorial;
    }

    public double getPrecio() {
        return precio;
    }

    public void setPrecio(double precio) {
        this.precio = precio;
    }
```

```
51
52          @Override
53          public String toString() {
54              return "Libros: " + "\ntitulo=" + titulo + "\nautor=" + autor
55                      + "\neditorial=" + editorial + "\nprecio=" + precio ;
56          }
57
58
59      }
60
```

La clase Libros es una representación de un libro con atributos como título, autor, editorial y precio. Permite acceder y modificar estos atributos, y proporciona una representación en forma de cadena del libro.

```
1   package Herencia;
2
3   public class LibrosDeTexto extends Libros{
4       private String asignatura;
5
6       public LibrosDeTexto() {
7           super();
8       }
9
10      public LibrosDeTexto(String titulo, Autor autor, String editorial,
11              double precio,String asignatura) {
12          super(titulo, autor, editorial, precio);
13          this.asignatura = asignatura;
14      }
15
16      public String getAsignatura() {
17          return asignatura;
18      }
19
20      public void setAsignatura(String asignatura) {
21          this.asignatura = asignatura;
22      }
23
24      @Override
25      public String toString() {
26          return super.toString()+"\nLibrosDeTexto: "
27                  + "\nasignatura=" + asignatura ;
28      }
29
30
31  }
```

La clase LibrosDeTexto es una extensión de la clase Libros que representa libros de texto. Además de los atributos heredados, tiene un atributo adicional llamado asignatura que representa la asignatura del libro de texto.

```java
package Herencia;

public class LibroNovela extends Libros{
    private String genNovela;

    public LibroNovela() {
        super();
    }

    public LibroNovela(String titulo, Autor autor, String editorial,
            double precio,String genNovela) {
        super(titulo, autor, editorial, precio);
        this.genNovela = genNovela;
    }

    public String getGenNovela() {
        return genNovela;
    }

    public void setGenNovela(String genNovela) {
        this.genNovela = genNovela;
    }

    @Override
    public String toString() {
        return super.toString()+"\nLibroNovela: " + "\ngenNovela=" + genNovela ;
    }


}
```

La clase LibroNovela es una extensión de la clase Libros que representa libros de novelas. Además de los atributos heredados, tiene un atributo adicional llamado genNovela que representa el género de la novela.

```java
package Herencia;

public class LibrosTextoTECNM extends LibrosDeTexto{
    private String campus;
    private Fecha fecha;

    public LibrosTextoTECNM() {
        super();
    }

    public LibrosTextoTECNM(String titulo, Autor autor, String editorial,
            double precio, String asignatura,String campus, Fecha fecha) {
        super(titulo, autor, editorial, precio, asignatura);
        this.campus = campus;
        this.fecha = fecha;
    }

    public String getCampus() {
        return campus;
    }

    public void setCampus(String campus) {
        this.campus = campus;
    }

    public Fecha getFecha() {
        return fecha;
    }

    public void setFecha(Fecha fecha) {
        this.fecha = fecha;
    }

    @Override
    public String toString() {
        return super.toString()+"\nLibrosTextoTECNM: " + "\ncampus=" + campus
                + "\nfecha=" + fecha ;
    }
}
```

La clase LibrosTextoTECNM es una extensión de la clase LibrosDeTexto que representa libros de texto utilizados en el Tecnológico Nacional de México. Además de los atributos heredados, tiene dos atributos adicionales: campus y fecha.

```java
package Herencia;

public class Autor {
    private String Nombre;
        private String Apellido;

        public Autor () {}
        public Autor(String Nombre,String Apellido) {

                this.Nombre=Nombre;
                this.Apellido=Apellido;
        }
        public String getNombre() {
                return Nombre;
        }
        public void setNombre(String nombre) {
                Nombre = nombre;
        }
        public String getApellido() {
                return Apellido;
        }
        public void setApellido(String apellido) {
                Apellido = apellido;
        }

        public String toString(){
                return " "+Nombre+" "+Apellido;
        }
}
```

La clase Autor es una clase simple que representa a un autor de libros y proporciona métodos para acceder y modificar su nombre y apellido.

```java
package Herencia;

public class Fecha {

	private byte dia, mes;
	private int año;
	public Fecha() {}
	public Fecha(byte dia, byte mes, int año) {
		this.dia=dia;
		this.mes=mes;
		this.año=año;
	}
	public byte getDia() {
		return dia; }
	public void setDia(byte dia) {
		this.dia = dia; }
	public byte getMes() {
		return mes; }
	public void setMes(byte mes) {
		this.mes = mes; }
	public int getAño() {
		return año; }
	public void setAño(int año) {
		this.año = año; }
	public String toString () {
		return "DIA:"+dia+"del mes:"+mes+"del año:"+año;}
}

```

La clase Fecha es una clase simple que representa una fecha y proporciona métodos para acceder y modificar el día, el mes y el año, así como para obtener una representación legible de la fecha en forma de cadena.

```java
package Test;

import EntradaSalida.Tools;
import Herencia.Autor;
import Herencia.Fecha;
import Herencia.LibroNovela;
import Herencia.LibrosDeTexto;
import Herencia.LibrosTextoTECNM;

public class TestLibros {
    public static void main(String[] args) {
        menu();
    }

    public static void menu() {
        String sel = "";
        do {
            sel = Tools.boton("Crear Libro de Texto,Crear Libro de Novela,Salir"

            switch (sel) {
                case "Crear Libro de Texto":
                    menuLibrosDeTexto();
                    break;
                case "Crear Libro de Novela":
                    crearLibroDeNovela();
                    break;
                case "Salir":
                    break;

            }
        } while (!sel.equalsIgnoreCase("Salir"));
    }
    public static void menuLibrosDeTexto() {
        String sel;
        sel = Tools.boton("Libro de Texto,Libro de Texto TECNM");

        switch (sel) {
            case "Libro de Texto":
                crearLibroDeTexto();
                break;
            case "Libro de Texto TECNM":
                crearLibroDeTextoTECNM();
                break;

        }
    }
```

```
47
48      public static void crearLibroDeTexto() {
49
50          String titulo = Tools.leerString("Ingrese el título del libro de texto")
51          String nombreAutor = Tools.leerString("Ingrese el nombre del autor");
52          String apellidoAutor = Tools.leerString("Ingrese el apellido del autor")
53          Autor autor = new Autor(nombreAutor, apellidoAutor);
54          String editorial = Tools.leerString("Ingrese la editorial");
55          double precio = Tools.leerDouble("Ingrese el precio");
56          String asignatura = Tools.leerString("Ingrese la asignatura");
57
58
59          LibrosDeTexto libroTexto = new LibrosDeTexto(titulo, autor, editorial, p
60          Tools.imprimePantalla(libroTexto.toString());
61      }
62
63      public static void crearLibroDeNovela() {
64
65          String titulo = Tools.leerString("Ingrese el título del libro de novela"
66          String nombreAutor = Tools.leerString("Ingrese el nombre del autor");
67          String apellidoAutor = Tools.leerString("Ingrese el apellido del autor")
68          Autor autor = new Autor(nombreAutor, apellidoAutor);
69          String editorial = Tools.leerString("Ingrese la editorial");
70          double precio = Tools.leerDouble("Ingrese el precio");
71          String genero = Tools.leerString("Ingrese el género de la novela");
72
73          LibroNovela libroNovela = new LibroNovela(titulo, autor, editorial, prec
74          Tools.imprimePantalla(libroNovela.toString());
75      }
76      public static void crearLibroDeTextoTECNM() {
77
78          String titulo = Tools.leerString("Ingrese el título del libro de texto")
79          String nombreAutor = Tools.leerString("Ingrese el nombre del autor");
80          String apellidoAutor = Tools.leerString("Ingrese el apellido del autor")
81          Autor autor = new Autor(nombreAutor, apellidoAutor);
82          String editorial = Tools.leerString("Ingrese la editorial");
83          double precio = Tools.leerDouble("Ingrese el precio");
84          String asignatura = Tools.leerString("Ingrese la asignatura");
85          String campus = Tools.leerString("Ingrese el campus");
86          byte dia = Tools.leerByte("Ingrese el día de la fecha");
87          byte mes = Tools.leerByte("Ingrese el mes de la fecha");
88          int año = Tools.leerInt("Ingrese el año de la fecha");
89          Fecha fecha = new Fecha(dia, mes, año);
90
91          LibrosTextoTECNM libroTextoTECNM = new LibrosTextoTECNM(titulo, autor, e
92          Tools.imprimePantalla(libroTextoTECNM.toString());
93      }
94  }
```

La clase TestLibros permite al usuario crear y mostrar libros de texto y libros de novela, interactuando con las clases del paquete Herencia.

```java
package Abstractas;

public abstract class Figura {
    private String nomfig;
    private String color;

    public Figura(String nom, String col) {
        nomfig = nom;
        color = col;
    }

    public String getNomfig() {
        return nomfig;
    }

    public void setNomfig(String nomfig) {
        this.nomfig = nomfig;
    }

    public String getColor() {
        return color;
    }

    public void setColor(String color) {
        this.color = color;
    }
    public abstract double area();

    @Override
    public String toString() {
        return "Figura{" + "nomfig=" + nomfig + ", color=" + color + '}';
    }

}

```

La clase Figura es una clase abstracta que define propiedades y métodos básicos para representar una figura geométrica. Las clases concretas que hereden de esta clase deben implementar el método area() para calcular el área específica de la figura.

```java
package Abstractas;

public class Triangulo extends Figura{
    private int base,altura;

    public Triangulo(String nom, String col,int base, int altura) {
        super(nom, col);
        this.base = base;
        this.altura = altura;
    }

    public int getBase() {
        return base;
    }

    public void setBase(int base) {
        this.base = base;
    }

    public int getAltura() {
        return altura;
    }

    public void setAltura(int altura) {
        this.altura = altura;
    }
    public double area(){
        return(this.base*this.altura)/2;
    }

    @Override
    public String toString() {
        return super.toString()+"\nTriangulo{" + "\nbase=" + base
                + "\naltura=" + altura + "\narea="+area()+'}';
    }

}
```

La clase Triangulo es una clase concreta que hereda de la clase abstracta Figura y representa un triángulo. Proporciona funcionalidades para establecer la base y la altura, calcular el área y mostrar la información del triángulo en una representación de cadena.

```java
package Abstractas;

public class Rectangulo extends Figura{
    private int base;
    private int altura;

    public Rectangulo(String nom, String col,int base, int altura) {
        super(nom, col);
        this.base = base;
        this.altura = altura;
    }

    public double getBase() {
        return base;
    }

    public void setBase(int base) {
        this.base = base;
    }

    public double getAltura() {
        return altura;
    }

    public void setAltura(int altura) {
        this.altura = altura;
    }
    public double area(){
        return base*altura;
    }

    @Override
    public String toString() {
        return super.toString()+"\nRectangulo{" + "\nbase=" + base
                + "\naltura=" + altura + "\narea="+area()+'}';
    }

}
```

La clase Rectangulo es una clase concreta que hereda de la clase abstracta Figura y representa un rectángulo. Proporciona funcionalidades para establecer la base y la altura, calcular el área y mostrar la información del rectángulo en una representación de cadena.

```java
package Abstractas;

public class Cuadrado extends Figura{

    private int lado;

    public Cuadrado(String nom, String col,int lado) {
        super(nom, col);
        this.lado = lado;
    }

    public double getLado() {
        return lado;
    }

    public void setLado(int lado) {
        this.lado = lado;
    }

    public double area(){
        return Math.pow(lado, 2);
    }

    @Override
    public String toString() {
        return super.toString()+"\nCuadrado{" + "\nlado=" + lado
                +"\narea="+area()+ '}';
    }

}
```

La clase Cuadrado es una clase concreta que hereda de la clase abstracta Figura y representa un cuadrado. Proporciona funcionalidades para establecer el lado, calcular el área y mostrar la información del cuadrado en una representación de cadena.

```java
package Test;

import Abstractas.Cuadrado;
import Abstractas.Figura;
import Abstractas.Rectangulo;
import Abstractas.Triangulo;
import EntradaSalida.Tools;

public class TestFiguras {
    public static void main(String[] args) {
        menu();
    }

    public static void menu() {
        String sel = "";
        Figura arrFig[]=new Figura[10];
        int j=0;
        do {
            sel = Tools.boton("Triangulo,Cuadrado,Rectangulo,Imprimir,Salir");

            switch (sel) {
                case "Triangulo":
                    if(j<10){
                        Figura Triangulo=new Triangulo("Triangulo","Rojo",
                                Tools.leerInt("Ingrese la base"),
                                Tools.leerInt("Ingrese la altura"));
                        arrFig[j]=Triangulo;
                        j++;
                    }else{
                        Tools.imprimePantalla("No puedes ingresar mas datos");
                    }

                    break;
                case "Cuadrado":
                   if(j<10){
                        Figura Cuadrado=new Cuadrado("Cuadrado","Azul",
                                Tools.leerInt("Ingrese un lado"));
                        arrFig[j]=Cuadrado;
                        j++;
                    }else{
                        Tools.imprimePantalla("No puedes ingresar mas datos");
                    }
                    break;
                case "Rectangulo":
                    if(j<10){
                        Figura Rectangulo=new Rectangulo("Rectangulo","Negro",
                                Tools.leerInt("Ingrese un la base"),
                                Tools.leerInt("Ingrese la altura"));
                        arrFig[j]=Rectangulo;
                        j++;
```

```
51                      }else{
52                          Tools.imprimePantalla("No puedes ingresar mas datos");
53                      }
54                      break;
55                  case "Imprimir":
56                      String Acu=" ";
57                      for(int i=0;i<j;i++){
58                          Acu+="\n"+arrFig[i];
59
60                      }
61                      Tools.imprimePantalla(Acu);
62                      break;
63                  case "Salir":
64                      break;
65
66              }
67          } while (!sel.equalsIgnoreCase("Salir"));
68      }
69  }
70
```

La clase TestFiguras permite al usuario crear y almacenar figuras geométricas en un arreglo, y luego imprimir la información de las figuras ingresadas. El programa muestra un menú interactivo y utiliza las clases Triangulo, Cuadrado, Rectangulo y Figura del paquete Abstractas para representar y operar con las figuras geométricas.

```java
package Abstractas;

public abstract class Empleado {
    protected String primerNombre;
    protected String apellidoPaterno;
    protected String numeroSeguroSocial;

    public Empleado(String nombre, String apellido,String nss){
        numeroSeguroSocial = nss;
        primerNombre = nombre;
        apellidoPaterno = apellido;
        }

    public String getPrimerNombre() {
        return primerNombre;
    }

    public void setPrimerNombre(String primerNombre) {
        this.primerNombre = primerNombre;
    }

    public String getApellidoPaterno() {
        return apellidoPaterno;
    }

    public void setApellidoPaterno(String apellidoPaterno) {
        this.apellidoPaterno = apellidoPaterno;
    }

    public String getNumeroSeguroSocial() {
        return numeroSeguroSocial;
    }

    public void setNumeroSeguroSocial(String numeroSeguroSocial) {
        this.numeroSeguroSocial = numeroSeguroSocial;
    }

    public String toString() {
        return "Empleado [Nombre=" + primerNombre
                + ", apellidoPaterno=" +apellidoPaterno+ ", numeroSeguroSocial="
                + numeroSeguroSocial + "]";
    }
    public abstract double ingresos();
}
```

La clase Empleado define las características comunes de un empleado y proporciona un método abstracto ingresos() que se implementará en las subclases para calcular los ingresos específicos de cada tipo de empleado.

```java
package Abstractas;

public class EmpleadoAsalariado extends Empleado {

    private double salarioSemanal;

    public EmpleadoAsalariado(String nss, String nombre, String apellido,
            double salario ){
        super(nss, nombre, apellido);
        setSalarioSemanal( salario );
    }

    public void setSalarioSemanal( double salario ){
        salarioSemanal = (salario < 0.0) ? 0.0 :salario;
    }

    public double getSalarioSemanal(){
        return salarioSemanal;
    }

    public double ingresos(){
        return getSalarioSemanal();
    }

    public String toString(){
        return "EMPLEADO ASALARIADO\n"+super.toString()
                +"\nSalario semanal: "+ingresos();
    }
}

```

La clase EmpleadoAsalariado es una subclase de Empleado que representa a un empleado asalariado con un salario fijo semanal. Implementa el método abstracto ingresos() para calcular los ingresos y proporciona métodos para acceder y modificar el salario semanal.

```java
package Abstractas;

public class EmpleadoBaseMasComision extends EmpleadoPorHoras {

        private double salarioBase;

    public EmpleadoBaseMasComision( String nombre,String apellido, String nss,
            double ventas,double tarifa, double salario ){
        super( nombre, apellido, nss, ventas, tarifa );
        setSalarioBase( salario );
    }

    public void setSalarioBase( double salario ){
        salarioBase = ( salario < 0.0 ) ? 0.0 : salario;
    }

    public double getSalarioBase(){
        return salarioBase;
    }

    public double ingresos() {
        return getSalarioBase() + super.ingresos();
    }

    public String toString(){
        return "CON SALARIO BASE\n"+super.toString()
                +"\nSalario base:"+getSalarioBase()+"\nIngresos: "+ingresos();
    }
}
```

La clase EmpleadoBaseMasComision es una subclase de EmpleadoPorHoras que representa a un empleado con un salario base más una comisión basada en las ventas. Implementa el método ingresos() para calcular los ingresos totales y proporciona métodos para acceder y modificar el salario base.

```java
package Abstractas;

public class EmpleadoPorComision extends Empleado {

    private double ventasBrutas;
    private double tarifaComision;

    public EmpleadoPorComision(String nombre, String apellido, String nss,
            double ventas, double tarifa) {
        super(nombre, apellido, nss);
        setVentasBrutas(ventas);
        setTarifaComision(tarifa);
    }

    public void setTarifaComision(double tarifa) {
        tarifaComision = (tarifa > 0.0 && tarifa < 1.0) ? tarifa : 0.0;
    }

    public double getTarifaComision() {
        return tarifaComision;
    }

    public void setVentasBrutas(double ventas) {
        ventasBrutas = (ventas < 0.0) ? 0.0 : ventas;
    }

    public double getVentasBrutas() {
        return ventasBrutas;
    }

    public double ingresos() {
        return getTarifaComision() * getVentasBrutas();
    }

    public String toString() {
        return "EMPLEADO POR COMISION\n" + super.toString()
                + "\nVentas brutas: " + getVentasBrutas()
                + "\nTarifa de comision: " + getTarifaComision()
                + "\nIngresos: " + ingresos();
    }
}
```

La clase EmpleadoPorComision es una subclase de Empleado que representa a un empleado cuyo salario se basa en una tarifa de comisión y las ventas brutas realizadas. Implementa el método ingresos() para calcular los ingresos totales y proporciona métodos para acceder y modificar las ventas brutas y la tarifa de comisión.

```java
package Abstractas;

public class EmpleadoPorHoras extends Empleado {

    private double sueldo;
    private double horas;

    public EmpleadoPorHoras( String nombre, String apellido, String nss,
            double sueldoPorHoras, double horasTrabajadas ){
        super( nombre, apellido, nss );
        setSueldo( sueldoPorHoras );
        setHoras( horasTrabajadas );
    }
    public void setSueldo( double sueldoPorHoras ){
        sueldo = ( sueldoPorHoras < 0.0 ) ? 0.0 : sueldoPorHoras;
    }

    public double getSueldo(){
        return sueldo;
    }

    public void setHoras( double horasTrabajadas ){
        horas = ( ( horasTrabajadas >= 0.0 ) && ( horasTrabajadas <= 168.0 ) )
                ?horasTrabajadas : 0.0;
    }

    public double getHoras(){
        return horas;
    }

    public double ingresos(){
        if ( getHoras() <= 40 )
    return getSueldo() * getHoras();
        else
    return 40 * getSueldo() + ( getHoras() - 40 ) * getSueldo() * 1.5;
    }

    public String toString(){
        return "EMPLEADO POR HORAS\n"+super.toString()+"\nSueldo por hora:"
                +getSueldo()+ "\nHoras trabajadas: "+getHoras()
                +"\nIngresos: "+ingresos();
    }
}

```

La clase EmpleadoPorHoras es una subclase de Empleado que representa a un empleado cuyo salario se basa en la cantidad de horas trabajadas y el sueldo por hora. Implementa el método ingresos() para calcular los ingresos totales y proporciona métodos para acceder y modificar el sueldo por hora y las horas trabajadas.

```java
package Test;

import EntradaSalida.Tools;
import Abstractas.Empleado;
import Abstractas.EmpleadoAsalariado;
import Abstractas.EmpleadoPorComision;
import Abstractas.EmpleadoPorHoras;
import Abstractas.EmpleadoBaseMasComision;

public class TestEmpleado {
    private static Empleado arrEmpleado[] = new Empleado[10];
    private static EmpleadoBaseMasComision arrEmpleadoBaseMasComision[] =
            new EmpleadoBaseMasComision[10];
    private static int j = 0;

    public static void main(String[] args) {
        menu();
    }

    public static void menu() {
        String sel = "";
        do {
            sel = Tools.boton("Empleado Asalariado,Empleado Por Comision,"
                    + "Empleado Por Horas,Imprimir,Salir");

            switch (sel) {
                case "Empleado Asalariado":
                    if (j < 10) {
                        EmpleadoAsalariado EmpleAsal = new EmpleadoAsalariado(
                                Tools.leerString("Ingresar nombre"),
                                Tools.leerString("Ingresa apellido"),
                                Tools.leerString("Ingrese numero de seguro socia
                                Tools.leerDouble("Ingrese el salario"));
                        arrEmpleado[j] = EmpleAsal;
                        j++;
                    } else {
                        Tools.imprimePantalla("No puedes ingresar mas datos");
                    }
                    break;
                case "Empleado Por Comision":
                    if (j < 10) {
                        EmpleadoPorComision EmplePorComision =
                                new EmpleadoPorComision(Tools.leerString("Ingres
                                        Tools.leerString("Ingrese apellido"),
                                        Tools.leerString("Ingrese numero de segu
                                        Tools.leerDouble("Ingrese el numero de v
                                        Tools.leerDouble("Ingrese la tarifa"));
                        arrEmpleado[j] = EmplePorComision;
                        j++;
```

```
                        String op = Tools.boton2("Sí,No");
                        if (op.equals("Sí")) {
                            menuEmpleadoBaseMasComision();
                        }
                    } else {
                        Tools.imprimePantalla("No puedes ingresar más datos");
                    }
                    break;
                case "Empleado Por Horas":
                    if (j < 10) {
                        EmpleadoPorHoras EmplePorHrs = new EmpleadoPorHoras(
                                Tools.leerString("Ingrese nombre"),
                                Tools.leerString("Ingrese apellido"),
                                Tools.leerString("Ingrese numero de seguro socia
                                Tools.leerDouble("Ingrase sueldo por horas"),
                                Tools.leerDouble("Ingrese las horas trabajadas")
                        arrEmpleado[j] = EmplePorHrs;
                        j++;
                    } else {
                        Tools.imprimePantalla("No puedes ingresar mas datos");
                    }
                    break;
                case "Imprimir":
                    String Acu=" ";
                    for(int i=0;i<j;i++){
                        Acu+="\n"+arrEmpleado[i];

                    }
                    Tools.imprimePantalla(Acu);
                    break;
                case "Salir":
                    break;
            }
        } while (!sel.equalsIgnoreCase("Salir"));
    }

    public static void menuEmpleadoBaseMasComision() {
        String sel = "";
        do {
            sel = Tools.boton("Empleado Base Mas Comision,Imprimir,Regresar");

            switch (sel) {
                case "Empleado Base Mas Comision":
                    if (j < 10) {
                        EmpleadoPorComision EmplePorComision = new EmpleadoPorCo
                                Tools.leerString("Ingrese nombre"),
                                Tools.leerString("Ingrese apellido"),
                                Tools.leerString("Ingrese numero de seguro socia
                                Tools.leerDouble("Ingrese el numero de ventas"),
                                Tools.leerDouble("Ingrese la tarifa"));
```

```
100                         double salario = Tools.leerDouble("Ingresar salario");
101                         EmpleadoBaseMasComision EmpleBasMasComi = new EmpleadoBa
102                                 EmplePorComision.getPrimerNombre(),
103                                 EmplePorComision.getApellidoPaterno(),
104                                 EmplePorComision.getNumeroSeguroSocial(),
105                                 EmplePorComision.getVentasBrutas(),
106                                 EmplePorComision.getTarifaComision(), salario);
107                         arrEmpleadoBaseMasComision[j] = EmpleBasMasComi;
108                         j++;
109                     } else {
110                         Tools.imprimePantalla("No puedes ingresar mas datos");
111                     }
112                     break;
113                 case "Imprimir":
114                     String Acu2 = "";
115                        for (int i = 0; i < j; i++) {
116                             Acu2 += "\n" + arrEmpleadoBaseMasComision[i];
117                     }
118                     Tools.imprimePantalla(Acu2);
119                     break;
120                 case "Regresar":
121                     break;
122             }
123         } while (!sel.equalsIgnoreCase("Regresar"));
124     }
```

La clase TestEmpleado proporciona un programa interactivo para probar y gestionar diferentes tipos de empleados. Los empleados ingresados se almacenan en arreglos y se puede imprimir la información de los empleados en cualquier momento.

CONCLUSIONES

En conclusión, la herencia y el polimorfismo son conceptos fundamentales en la programación orientada a objetos que permiten la reutilización de código, la creación de jerarquías de clases y la flexibilidad en el diseño de software. La herencia proporciona la capacidad de extender y especializar la funcionalidad de una clase base en las clases derivadas, lo que facilita la reutilización de código y la organización de las clases en una estructura jerárquica. Por otro lado, el polimorfismo permite tratar a los objetos de manera uniforme a través de variables polimórficas, lo que aumenta la flexibilidad y el modularidad del código.

Las clases abstractas y las interfaces desempeñan un papel importante en la implementación de herencia y polimorfismo. Las clases abstractas proporcionan

una estructura común y establecen un contrato que las clases derivadas deben cumplir, mientras que las interfaces definen un conjunto de métodos que deben ser implementados por cualquier clase que las implemente. Estas herramientas fomentan la consistencia y el modularidad en el desarrollo de software, al tiempo que permiten la adaptabilidad y el intercambio de componentes.

En resumen, la herencia y el polimorfismo son conceptos poderosos que permiten construir programas más robustos y flexibles. Al utilizar herencia y polimorfismo de manera adecuada, los desarrolladores pueden mejorar la eficiencia, reutilizar el código y crear diseños de software más mantenibles y escalables. Comprender y aplicar estos conceptos es fundamental para aprovechar al máximo el potencial de la programación orientada a objetos.

BIBLIOGRAFÍA

Stroustrup, B. (2013). C++ Programming Language: The Definitive Reference (C++ Reference Series). Addison-Wesley Professional.

Schildt, H. (2017). Java: The Complete Reference. McGraw-Hill Education.

Matthes, E. (2019). Python Crash Course: A Hands-On, Project-Based Introduction to Programming. No Starch Press.

Wiener, R., & Pinson, L. J. (2017). Fundamentals of OOP and Data Structures in Java. Springer.

Mueller, J. P. (2017). C# 7.0 All-in-One For Dummies. For Dummies.