Programación Orientada a Objetos

Ingeniería en Informática

Herencia y Polimorfismo

Integrantes:

Espinosa Cabada Francisco Eduardo- 22010597

Gutiérrez Montaño Juan Luis- 22010601

Hernández Heredia Kevin- 22010603

Grupo:

2a3B

Fecha de entrega:

31/05/2023

Introducción:

La herencia y el polimorfismo son conceptos fundamentales en la programación orientada a objetos (POO) con Java. Estos conceptos permiten la creación de jerarquías de clases, donde una clase puede heredar atributos y métodos de otra clase base, lo que fomenta la reutilización de código y la organización estructurada del programa.

Comenzaremos por comprender qué es la herencia y cómo se establece una relación de herencia entre clases. Veremos cómo una clase hija puede heredar los atributos y métodos de una clase padre, lo que nos permite aprovechar el código ya existente y agregar nuevas funcionalidades según sea necesario.

Continuaremos examinando el polimorfismo, que se basa en la idea de que un objeto puede tomar varias formas. A través del polimorfismo, podemos tratar a un objeto de una clase derivada como si fuera de su clase base, lo que nos brinda flexibilidad y extensibilidad en el diseño de nuestro programa.

Además de comprender los conceptos teóricos, también abordará la implementación práctica de la herencia y el polimorfismo en Java. Se presentarán ejemplos de código para ilustrar cómo se pueden crear jerarquías de clases, cómo se pueden utilizar los modificadores de acceso para controlar la visibilidad de los miembros heredados y cómo se puede aprovechar el polimorfismo para escribir código más flexible y modular.

Competencia especifica:

La competencia específica en el tema de herencia y polimorfismo en la programación orientada a objetos se centra en comprender y aplicar los conceptos de herencia y polimorfismo en programas que utilicen clases base, clases derivadas, clases abstractas e interfaces.

Para lograr esta competencia, hemos desarrollado la capacidad de comprender los fundamentos de la herencia y cómo se establecen las relaciones entre clases base y clases derivadas. Hemos adquirido conocimientos sobre cómo una clase derivada puede heredar atributos y métodos de una clase base y cómo podemos aprovechar esta característica para reutilizar código y organizar mi programa de manera más estructurada.

Además, he aprendido sobre el uso de clases abstractas en el contexto de la herencia y el polimorfismo. Comprendo cómo una clase abstracta proporciona una base común para las clases derivadas. Hemos practicado la creación y utilización de clases abstractas para establecer contratos y garantizar la coherencia y la interoperabilidad en nuestros programas.

A través de ejercicios y proyectos prácticos, hemos aplicado estos conceptos en el desarrollo de programas. Hemos creado jerarquías de clases, implementado

herencia, utilizado clases abstractas para definir comportamientos comunes y he aplicado el polimorfismo para escribir código más flexible y adaptable.

Al dominar esta competencia, podremos diseñar y desarrollar programas más robustos y escalables utilizando la herencia y el polimorfismo de manera efectiva. Estaremos capacitados para crear jerarquías de clases adecuadas, implementar relaciones de herencia sólidas y utilizar clases abstractas e interfaces para lograr una mayor modularidad y reutilización de código. Además, podremos aprovechar al máximo el polimorfismo para escribir programas flexibles y adaptables a diferentes situaciones y requisitos.

Marco Teórico:

4.1 Concepto de herencia y polimorfismo: La herencia es un concepto fundamental en la programación orientada a objetos que permite crear jerarquías de clases, donde una clase derivada puede heredar atributos y métodos de una clase base. Esto fomenta la reutilización de código y la organización estructurada del programa. El polimorfismo, por otro lado, se basa en la idea de que un objeto puede tomar varias formas. A través del polimorfismo, podemos tratar a un objeto de una clase derivada como si fuera de su clase base, lo que brinda flexibilidad y extensibilidad en el diseño del programa.

4.2 Definición de una clase base: Una clase base, también conocida como clase padre o superclase, es una clase de la cual se derivan otras clases. Define los atributos y métodos comunes que serán heredados por las clases derivadas. La clase base proporciona una base para la implementación de las clases hijas y puede ser utilizada para encapsular comportamientos comunes y establecer una estructura jerárquica en el programa.

4.3 Definición de una clase derivada: Una clase derivada, también conocida como clase hija o subclase, es una clase que hereda atributos y métodos de una clase base. Puede agregar nuevos atributos y métodos específicos, así como sobrescribir o extender los comportamientos heredados de la clase base. Las clases derivadas permiten una especialización y extensión de la funcionalidad de la clase base.

4.4 Clases abstractas: Las clases abstractas son clases que no pueden ser instanciadas directamente, sino que se utilizan como plantillas para derivar subclases concretas. Pueden contener métodos abstractos, que son métodos sin implementación, así como métodos concretos con implementaciones definidas. Las clases abstractas proporcionan una base común para las clases derivadas y pueden contener atributos y métodos comunes que serán heredados.

4.5 Interfaces: Las interfaces son como contratos que especifican un conjunto de métodos que una clase debe implementar. No pueden contener implementaciones de métodos, solo las firmas de los mismos. Las interfaces

permiten la definición de comportamientos comunes que pueden ser implementados por diferentes clases. Las clases pueden implementar múltiples interfaces, lo que brinda flexibilidad en el diseño y promueve la modularidad y el bajo acoplamiento.

4.5.1 Definición de interfaces: Las interfaces se definen mediante la declaración de métodos sin cuerpo y constantes. Establecen un conjunto de métodos que una clase debe implementar y pueden contener especificaciones de firmas de métodos y restricciones adicionales. Las interfaces permiten la abstracción y el diseño orientado a contratos.

4.5.2 Implementación de interfaces: 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 para todos los métodos definidos en la interfaz. Esto asegura que la clase cumpla con el contrato especificado por la interfaz y proporcione los comportamientos requeridos.

4.5.3 Variables polimórficas: Las variables polimórficas son variables que pueden contener objetos de diferentes clases, siempre que esas clases implementen una interfaz común o sean subclases de una misma clase base. Esto permite tratar a los objetos de diferentes clases de manera uniforme, aprovechando el polimorfismo. La elección de qué objeto concreto asignar a una variable polimórfica puede realizarse en tiempo de compilación o en tiempo de ejecución.

4.6 Reutilización de la definición de paquetes/librerías: La reutilización de la definición de paquetes o librerías consiste en aprovechar el código desarrollado previamente y encapsulado en paquetes o librerías para utilizarlo en nuevos proyectos. Los paquetes y las librerías son formas de organizar y agrupar clases relacionadas, lo que facilita su uso posterior. Al utilizar paquetes o librerías existentes, se logra un ahorro de tiempo y esfuerzo en el desarrollo, al aprovechar la funcionalidad ya implementada y probada. Además, la reutilización de paquetes o librerías promueve buenas prácticas de modularidad y facilita el mantenimiento y la actualización del código.

Material y Equipo:

- ✓ Computadora
- ✓ Eclipse IDE for Java Developers – 2023 – 03
- ✓ Java SE Development kit 19.0.2

Desarrollo de la practica:

Desarrollar un programa que modele una cuenta bancaria que tiene los siguientes
atributos, que deben ser de acceso protegido:
• Saldo, de tipo float.

• Número de consignaciones con valor inicial cero, de tipo byte
• Número de retiros con valor inicial cero, de tipo byte.
• Tasa anual (porcentaje), de tipo float.
• Comisión mensual con valor inicial cero, de tipo float.
Recuerde que las consignaciones se realizan cuando una persona deposita dinero en
cuentas de ahorro, cuentas corrientes y cuentas del mercado monetario.
La clase Cuenta tiene un constructor que inicializa los atributos saldo y tasa anual con
valores pasados como parámetros. La clase Cuenta tiene los siguientes métodos:
• Consignar (Depositar) una cantidad de dinero en la cuenta actualizando su saldo.
• Retirar una cantidad de dinero en la cuenta actualizando su saldo. El valor a retirar
no debe superar el saldo.
• Calcular el interés mensual de la cuenta y actualiza el saldo correspondiente.
• Extracto mensual: actualiza el saldo restándole la comisión mensual y calculando
el interés mensual correspondiente (invoca el método anterior).
• Imprimir: muestra en pantalla los valores de los atributos.
La clase Cuenta tiene dos clases hijas:
Cuenta de ahorros: posee un atributo para determinar si la cuenta de ahorros está activa
(tipo boolean). Si el saldo es menor a $10 000, la cuenta está inactiva, en caso contrario se
considera activa.
Los siguientes métodos se redefinen:
✓ Consignar: se puede consignar dinero si la cuenta está activa. Debe invocar al método
heredado.
✓ Retirar: es posible retirar dinero si la cuenta está activa. Debe invocar al método
heredado.
✓ Extracto mensual: si el número de retiros es mayor que 4, por cada retiro adicional, se
cobra $200 como comisión mensual. Al generar el extracto, se determina si la cuenta
está activa o no con el saldo.
✓ Un nuevo método imprimir que muestra en pantalla el saldo de la cuenta, la comisión
mensual y el número de transacciones realizadas (suma de cantidad de consignaciones y retiros).

Cuenta corriente: posee un atributo de sobregiro, el cual se inicializa en cero. Se redefinen los siguientes métodos: Retirar: se retira dinero de la cuenta actualizando su saldo. Se puede retirar dinero superior al saldo. El dinero que se debe queda como sobregiro.

✓ Consignar: invoca al método heredado. Si hay sobregiro, la cantidad consignada reduce el sobregiro.

✓ Extracto mensual: invoca al método heredado.

✓ Un nuevo método imprimir que muestra en pantalla el saldo de la cuenta, la comisión mensual, el número de transacciones realizadas (suma de cantidad de consignaciones y retiros) y el valor de sobregiro.

Cuenta
#float saldo #float tasaAnual #byte numConsignacion #byte numRetiros #float comisionMensual
+Cuenta() +Cuenta(float,float) +float getSaldo() +float getTasaAnual() +byte getNumConsignacion() +byte getNumRetiros() +float getComisionMensual() +void setSaldo(float) +void setTasaAnual(float) +void Consignar(float) +void Retirar(float) +void CalcularInteresMensual() +void ExtractoMensual() +String Imprimir()

CuentaAhorros
-boolean active
+CuenaAhorros(float,float) +void Consignar(float) +void Retirar(float) +void ExtractoMensual() +String Imprimir()

CuentaCorriente
-float sobregiro
+CuentaCorriente (float,float) +void Retirar(float) +void Consignar(float) +void ExtractoMensual() +String Imprimir()

TestCuenta
+static void main(String[]) +static void MenuCuentas(String) +static String cuenta(Cuenta) +static String cuentaAhorro(CuentaAhorros) +static String cuentaCorriente(CuentaCorriente)

Desarrollar un programa que modele una clase Libro para manejar la información asociada
a un libro. La información de interés para un libro es: el título del libro, el autor, editorial, y
el precio. Los métodos de interés son:
Constructores necesarios.
Métodos get y set para cada atributo de un libro.
Metodo que imprima los datos del libro.
Se debe extender la clase Libro definiendo las siguientes clases:
Libros de texto con los atributos: título del libro, el autor, editorial, el precio y con un nuevo
atributo que especifica el nombre del curso (asignatura) al cual está asociado el libro.
Libros de texto del TECNM: subclase de la clase anterior. Esta subclase tiene un atributo
que especifica el nombre del Campus que lo publicó y la fecha de la publicación (deberá de
reutilizar la clase fecha usando en sus programas anteriores)
Libro Novela: con los siguientes atributos (titulo, autor, y precio) y el tipo de novela tipo: (considerar los siguientes tipos: histórica, romántica, policíaca, realista, ciencia ficción o aventuras).
Para cada una de las clases anteriores se debe definir su constructor y redefinir adecuadamente los métodos para visualizar los objetos.
Se solicita:
a) Diagrama de jerarquía de clases.
b) Diagrama de clases.

c) Crear una unidad ejecutable que presente un menú que permita crear objetos según se requieran, deberá de mostrar en una sola ventana emergente los objetos
creados.

Libro
#String tituloLibro #String autorLibro #String editorialLibro #float precioLibro
+Libro() +Libro(String,String,String,float) +String getTituloLibro() +String getAutorLibro() +String getEditorialLibro() +float getPrecioLibro() +void setTituloLibro(String) +void setAutorLibro(String) +void setEditorialLibro(String) +void setPrecioLibro(float)

LibroNovela
-String tipoNovela
+LibroNovela() +LibroNovela(String,String,String,float,String) +String getTipoNovela() +void setTipoNovela(String) +String toString()

LibrosDeTexto
#String nomCurso
+LibrosDeTexto() +LibrosDeTexto(String,String,String,float,String) +String getNomCurso() +void setNomCurso(String) +String toString()

LibrosDeTextoTECNM
-String campus -Fecha fecha
+ LibrosDeTextoTECNM()

+ LibrosDeTextoTECNM(String,String,String,float,String,String,byte,byte,short) +String getCampus() +Fecha getFecha() +void setCampus(String) +void setFecha(Fecha) +String toString()

TestLibros
+static void main(String[]) +static void MenuCuentas(String)

Practica Empleado

Empleado <abstract>
-String primerNombre -String apellidoPaterno -String numeroSeguroSocial
+Empleado(String, String, String) +String getPrimerNombre() +String getApellidoPaterno() +String getNumeroSeguroSocial() +void setPrimerNombre(String) +void setApellidoPaterno(String) +void setNumeroSeguroSocial(String) +String toString() +double ingresos()

EmpleadoPorComision
-double ventasBrutas -double tarifaComision
+EmpleadoPorComision(String,String,String,double,double) +double getVentasBrutas() +double getTarifaComision() +void setVentasBrutas(double) +void setTarifaComision(double) +double ingresos() +String toString()

EmpleadoAsalariado
-double salarioSemanal

+EmpleadoAsalariado(String,String,String,double) +double getSalarioSemanal() +void setSalarioSemanal(double) +double ingresos() +String toString()

EmpleadoPorHoras
-double sueldo -double horas
+EmpleadoPorHoras(String,String,String,double,double) +double getSueldo() +double getHoras() +void setSueldo(double) +void setHoras(double) +double ingresos() +String toString()

EmpleadoBaseMasComision
-double salarioBase
+EmpleadoBaseMasComision(String,String,String,double,double,double) +double getSalarioBase() +void setSalarioBase(double) +double ingresos() +String toString()

TestEmpleado
+static void main(String[]) +static void menu()

Resultados:

Paquete EntradasSalidas:

Clase Tools:

```java
package EntradasSalidas;

import javax.swing.JOptionPane;

public class Tools {

    public static void imprimeSalida(String msje) {
        JOptionPane.showMessageDialog(null,msje,"Salida de datos",JOptionPane.QUESTION_MESSAGE);
    }

    public static void salidaError(String msje) {
        JOptionPane.showMessageDialog(null, msje,"",JOptionPane.ERROR_MESSAGE);
    }

    public static short leerShort(String msje) {
        return (Short.parseShort(JOptionPane.showInputDialog(null,msje,"Dato de entrada",JOptionPane.INFORMATION_MESSAGE)));
    }

    public static byte leerByte(String msje) {
        return (Byte.parseByte(JOptionPane.showInputDialog(null,msje,"Dato de entrada",JOptionPane.INFORMATION_MESSAGE)));
    }

    public static int leerInt(String msje) {
        return (Integer.parseInt(JOptionPane.showInputDialog(null,msje,"Dato de entrada",JOptionPane.INFORMATION_MESSAGE)));
    }

    public static long leerLong(String msje) {
        return (Long.parseLong(JOptionPane.showInputDialog(null,msje,"Dato de entrada",JOptionPane.INFORMATION_MESSAGE)));
    }

    public static float leerFloat(String msje) {
        return (Float.parseFloat(JOptionPane.showInputDialog(null,msje,"Dato de entrada",JOptionPane.INFORMATION_MESSAGE)));
    }

    public static char leerChar(String msje) {
        return (JOptionPane.showInputDialog(null,msje,"Dato de entrada",JOptionPane.INFORMATION_MESSAGE).charAt(0));
    }

    public static String leerString(String msje) {
        return(JOptionPane.showInputDialog(null,msje,"Dato de entrada",JOptionPane.INFORMATION_MESSAGE));
    }

    public static double leerDouble(String msje) {
        return (Double.parseDouble(JOptionPane.showInputDialog(null,msje,"Dato de entrada",JOptionPane.INFORMATION_MESSAGE)));
    }

    public static int seguirSino() {
        return JOptionPane.showConfirmDialog(null,"Deseas continuar","Capturando datos",JOptionPane.YES_NO_OPTION);
    }

    public static int leerEntero(String msje) {
        return (Integer.parseInt(JOptionPane.showInputDialog(null,msje,"Lectura int: ",JOptionPane.INFORMATION_MESSAGE)));
    }

    public static void imprimePantalla(String msje) {
        JOptionPane.showMessageDialog(null, msje, "Salida",JOptionPane.INFORMATION_MESSAGE);
    }

    //Menu Desplegable
    public static String Desplegable(String menu) {
            String valores[]=menu.split(",");
            String res=(String)JOptionPane.showInputDialog(null,"M E N U","SELECCIONA ALGUNA OPCION: ",
                    JOptionPane.QUESTION_MESSAGE,null,valores,valores[0]);
            return res;
        }

    //Menu de Botones
    public static String Boton(String menu) {
            String valores[]=menu.split(",");
            int n;
            n=JOptionPane.showOptionDialog(null, "M E N U","Seleccione dando clic",JOptionPane.NO_OPTION,
                    JOptionPane.QUESTION_MESSAGE,null,valores,valores[0]);
            return valores[n];
        }

    public static String Desplegable(String cad, String menu) {
        String valores[]=menu.split(",");
        String res=(String)JOptionPane.showInputDialog(null,cad,"M E N U",
                JOptionPane.QUESTION_MESSAGE,null,valores,valores[0]);
        return res;
    }

    public static String Boton(String cad, String menu) {
        String valores[]=menu.split(",");
        int n;
        n=JOptionPane.showOptionDialog(null, cad ,"M E N U", JOptionPane.NO_OPTION,
                JOptionPane.QUESTION_MESSAGE,null,valores,valores[0]);
        return valores[n];
    }
}
```

La clase Tools contiene una serie de métodos estáticos que proporcionan funcionalidades de entrada/salida a través de cuadros de diálogo utilizando la clase JOptionPane de Java Swing.

- imprimeSalida(String msje): Muestra un cuadro de diálogo con el mensaje msje como salida de datos en una ventana emergente.
- salidaError(String msje): Muestra un cuadro de diálogo con el mensaje msje como un mensaje de error en una ventana emergente.
- leerShort(String msje): Muestra un cuadro de diálogo de entrada y lee un valor short proporcionado por el usuario, con el mensaje msje como indicación.
- leerByte(String msje): Muestra un cuadro de diálogo de entrada y lee un valor byte proporcionado por el usuario, con el mensaje msje como indicación.
- leerInt(String msje): Muestra un cuadro de diálogo de entrada y lee un valor int proporcionado por el usuario, con el mensaje msje como indicación.
- leerLong(String msje): Muestra un cuadro de diálogo de entrada y lee un valor long proporcionado por el usuario, con el mensaje msje como indicación.
- leerFloat(String msje): Muestra un cuadro de diálogo de entrada y lee un valor float proporcionado por el usuario, con el mensaje msje como indicación.
- leerChar(String msje): Muestra un cuadro de diálogo de entrada y lee un valor char proporcionado por el usuario, con el mensaje msje como indicación.
- leerString(String msje): Muestra un cuadro de diálogo de entrada y lee una cadena (String) proporcionada por el usuario, con el mensaje msje como indicación.
- leerDouble(String msje): Muestra un cuadro de diálogo de entrada y lee un valor double proporcionado por el usuario, con el mensaje msje como indicación.
- seguirSino(): Muestra un cuadro de diálogo de confirmación con opciones de "Sí" y "No" y devuelve un valor int que indica la opción seleccionada.
- leerEntero(String msje): Muestra un cuadro de diálogo de entrada y lee un valor int proporcionado por el usuario, con el mensaje msje como indicación.
- imprimePantalla(String msje): Muestra un cuadro de diálogo con el mensaje msje como salida de datos en una ventana emergente.
- Desplegable(String menu): Muestra un cuadro de diálogo desplegable con un menú de opciones (definido por la cadena menu) y devuelve la opción seleccionada como String.
- Boton(String menu): Muestra un cuadro de diálogo con botones que representan las opciones (definidas por la cadena menu) y devuelve la opción seleccionada como String.

- Desplegable(String cad, String menu): Muestra un cuadro de diálogo desplegable con un mensaje cad y un menú de opciones (definido por la cadena menu), y devuelve la opción seleccionada como String.
- Boton(String cad, String menu): Muestra un cuadro de diálogo con un mensaje cad y botones que representan las opciones (definidas por la cadena menu), y devuelve la opción seleccionada como String.

Estos métodos facilitan la interacción con el usuario a través de cuadros de diálogo, permitiendo mostrar mensajes, leer valores de diferentes tipos y presentar opciones en forma de menús o botones.

Paquete TDA:

Clase Bateria:

```java
package TDA;

public class Bateria {
    private int miniAmperios;
    private String marca;
    private String tipoConector;

    public Bateria() {}

    public Bateria(int miniAmperios, String marca, String tipoConector) {
        this.miniAmperios=miniAmperios;
        this.marca=marca;
        this.tipoConector=tipoConector;
    }

    public int getMiniAmperios() {
        return miniAmperios;
    }

    public String getMarca() {
        return marca;
    }

    public String getTipoConector() {
        return tipoConector;
    }

    public void setMiniAmperios(int miniAmperios) {
        this.miniAmperios = miniAmperios;
    }

    public void setMarca(String marca) {
        this.marca = marca;
    }

    public void setTipoConector(String tipoConector) {
        this.tipoConector = tipoConector;
    }

    @Override
    public String toString() {
        return "Bateria [miniAmperios=" + miniAmperios + ", marca=" + marca + ", tipoConector=" + tipoConector + "]";
    }

}
```

La clase Bateria representa una batería y tiene tres atributos: miniAmperios, marca y tipoConector.

- Bateria(): Constructor sin argumentos que crea una instancia de la clase Bateria.
- Bateria(int miniAmperios, String marca, String tipoConector): Constructor que recibe los valores de miniAmperios, marca y tipoConector para inicializar los atributos correspondientes de la batería.

- getMiniAmperios(): Método que devuelve el valor de miniAmperios de la batería.
- getMarca(): Método que devuelve el valor de marca de la batería.
- getTipoConector(): Método que devuelve el valor de tipoConector de la batería.
- setMiniAmperios(int miniAmperios): Método que establece el valor de miniAmperios de la batería.
- setMarca(String marca): Método que establece el valor de marca de la batería.
- setTipoConector(String tipoConector): Método que establece el valor de tipoConector de la batería.
- toString(): Método sobrescrito que devuelve una representación en forma de cadena de la batería, mostrando los valores de miniAmperios, marca y tipoConector.

Estos métodos permiten acceder y modificar los atributos de una instancia de la clase Bateria, así como obtener una representación textual de la misma a través del método toString().

Clase Chip:

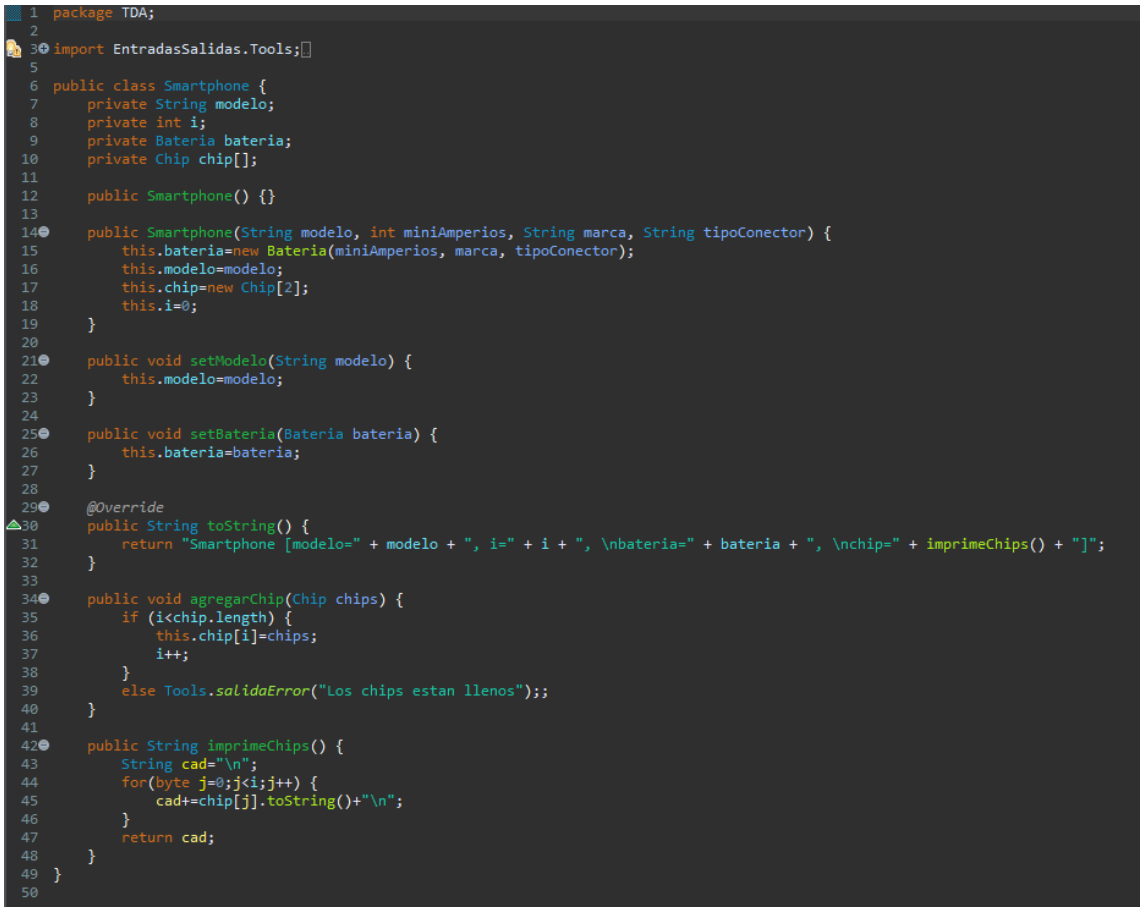
```java
package TDA;

public class Chip {
    private String empresaChip;
    private int numChip;
    private String tipoChip;

    public Chip() {}

    public Chip(String empresaChip, int numChip, String tipoChip) {
        this.empresaChip=empresaChip;
        this.numChip=numChip;
        this.tipoChip=tipoChip;
    }

    public String getEmpresaChip() {
        return empresaChip;
    }

    public int getNumChip() {
        return numChip;
    }

    public String getTipoChip() {
        return tipoChip;
    }

    public void setEmpresaChip(String empresaChip) {
        this.empresaChip = empresaChip;
    }

    public void setNumChip(int numChip) {
        this.numChip = numChip;
    }

    public void setTipoChip(String tipoChip) {
        this.tipoChip = tipoChip;
    }

    @Override
    public String toString() {
        return "Chip [empresaChip=" + empresaChip + ", numChip=" + numChip + ", tipoChip=" + tipoChip + "]";
    }

}
```

La clase Chip representa un chip y tiene tres atributos: empresaChip, numChip y tipoChip.

- Chip(): Constructor sin argumentos que crea una instancia de la clase Chip.
- Chip(String empresaChip, int numChip, String tipoChip): Constructor que recibe los valores de empresaChip, numChip y tipoChip para inicializar los atributos correspondientes del chip.
- getEmpresaChip(): Método que devuelve el valor de empresaChip del chip.
- getNumChip(): Método que devuelve el valor de numChip del chip.
- getTipoChip(): Método que devuelve el valor de tipoChip del chip.
- setEmpresaChip(String empresaChip): Método que establece el valor de empresaChip del chip.
- setNumChip(int numChip): Método que establece el valor de numChip del chip.
- setTipoChip(String tipoChip): Método que establece el valor de tipoChip del chip.
- toString(): Método sobrescrito que devuelve una representación en forma de cadena del chip, mostrando los valores de empresaChip, numChip y tipoChip.

Estos métodos permiten acceder y modificar los atributos de una instancia de la clase Chip, así como obtener una representación textual de la misma a través del método toString().

Clase Smartphone:



```java
package TDA;

import EntradasSalidas.Tools;

public class Smartphone {
    private String modelo;
    private int i;
    private Bateria bateria;
    private Chip chip[];

    public Smartphone() {}

    public Smartphone(String modelo, int miniAmperios, String marca, String tipoConector) {
        this.bateria=new Bateria(miniAmperios, marca, tipoConector);
        this.modelo=modelo;
        this.chip=new Chip[2];
        this.i=0;
    }

    public void setModelo(String modelo) {
        this.modelo=modelo;
    }

    public void setBateria(Bateria bateria) {
        this.bateria=bateria;
    }

    @Override
    public String toString() {
        return "Smartphone [modelo=" + modelo + ", i=" + i + ", \nbateria=" + bateria + ", \nchip=" + imprimeChips() + "]";
    }

    public void agregarChip(Chip chips) {
        if (i<chip.length) {
            this.chip[i]=chips;
            i++;
        }
        else Tools.salidaError("Los chips estan llenos");;
    }

    public String imprimeChips() {
        String cad="\n";
        for(byte j=0;j<i;j++) {
            cad+=chip[j].toString()+"\n";
        }
        return cad;
    }
}

```

La clase Smartphone representa un teléfono inteligente y tiene los siguientes atributos: modelo, i, bateria y chip[].

- Smartphone(): Constructor sin argumentos que crea una instancia de la clase Smartphone.
- Smartphone(String modelo, int miniAmperios, String marca, String tipoConector): Constructor que recibe el modelo del smartphone, así como los valores necesarios para crear una instancia de Bateria, y también inicializa el arreglo chip[].
- setModelo(String modelo): Método que establece el modelo del smartphone.
- setBateria(Bateria bateria): Método que establece la bateria del smartphone.
- toString(): Método sobrescrito que devuelve una representación en forma de cadena del smartphone, mostrando los valores de modelo, i, bateria y los chip agregados.
- agregarChip(Chip chips): Método que agrega un objeto Chip al arreglo chip[] del smartphone. Si el arreglo está lleno, muestra un mensaje de error utilizando el método salidaError de la clase Tools.
- imprimeChips(): Método que recorre el arreglo chip[] y devuelve una representación en forma de cadena de los chip agregados al smartphone.

Estos métodos permiten establecer el modelo y la batería del smartphone, agregar chips al smartphone y obtener una representación textual del smartphone, incluyendo los chips agregados.

Paquete Herencia:

Clase Alumno:

```java
package Herencia;

public class Alumno extends Usuario{

    private byte semestre;
    private float promedio;

    public Alumno() {
        super();
        // TODO Auto-generated constructor stub
    }

    public Alumno(String nombre, String apPaterno, String apMaterno, byte edad, String departamento,byte semestre, float promedio) {
        super(nombre, apPaterno, apMaterno, edad, departamento);
        this.semestre=semestre;
        this.promedio=promedio;
        // TODO Auto-generated constructor stub
    }

    public byte getSemestre() {
        return semestre;
    }

    public float getPromedio() {
        return promedio;
    }

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

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

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

}
```

La clase Alumno es una subclase de Usuario y representa a un alumno. Tiene los siguientes atributos adicionales: semestre y promedio.

- Alumno(): Constructor sin argumentos que crea una instancia de la clase Alumno. Llama al constructor sin argumentos de la clase padre Usuario utilizando super().
- Alumno(String nombre, String apPaterno, String apMaterno, byte edad, String departamento, byte semestre, float promedio): Constructor que recibe los valores necesarios para crear una instancia de Alumno y también llama al constructor de la clase padre Usuario utilizando super().
- getSemestre(): Método que devuelve el valor del semestre del alumno.
- getPromedio(): Método que devuelve el valor del promedio del alumno.
- setSemestre(byte semestre): Método que establece el valor del semestre del alumno.
- setPromedio(float promedio): Método que establece el valor del promedio del alumno.
- toString(): Método sobrescrito que devuelve una representación en forma de cadena del alumno, mostrando los valores de nombre, apPaterno, apMaterno, edad, departamento, semestre y promedio.

Estos métodos permiten acceder y modificar los atributos específicos del alumno, además de heredar los métodos y atributos de la clase padre Usuario.

Clase Cuenta:

```java
package Herencia;

import EntradasSalidas.Tools;

public class Cuenta {
    //Atributos
    protected float saldo;
    protected float tasaAnual;
    protected byte numConsignacion=0;
    protected byte numRetiros=0;
    protected float comisionMensual=0;
    //Constructores
    public Cuenta() {}

    public Cuenta(float saldo, float tasaAnual) {
        this.saldo = saldo;
        this.tasaAnual = tasaAnual;
    }
    //setter and getter

    public float getSaldo() {
        return saldo;
    }

    public float getTasaAnual() {
        return tasaAnual;
    }

    public byte getNumConsignacion() {
        return numConsignacion;
    }

    public byte getNumRetiros() {
        return numRetiros;
    }

    public float getComisionMensual() {
        return comisionMensual;
    }

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

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

    //Metodos
    public void Consignar(float cantidad) {//Depositar dinero en cuenta
        saldo+=cantidad;
        numConsignacion++;
```

```
53      }
54
55      public void Retirar(float cantidad) {
56          if (cantidad<=saldo) {
57              saldo-=cantidad;
58              numRetiros++;
59          }
60          else Tools.salidaError("Lo siento, la cantidad a retirar excede tu saldo");
61      }
62
63      public void CalcularInteresMensual() {
64          float tasaMensual=tasaAnual/12;
65          saldo+=saldo*tasaMensual;
66      }
67
68      public void ExtractoMensual() {
69          saldo-=comisionMensual;
70          CalcularInteresMensual();
71      }
72
73      public String Imprimir() {//Este es el toString
74          return "Cuenta [saldo=" + saldo + ", tasaAnual=" + tasaAnual + ", numConsignacion=" + numConsignacion
75                  + ", numRetiros=" + numRetiros + ", comisionMensual=" + comisionMensual + "]";
76      }
77
78  }
```

La clase Cuenta representa una cuenta bancaria y tiene los siguientes atributos: saldo, tasaAnual, numConsignacion, numRetiros y comisionMensual.

- Cuenta(): Constructor sin argumentos que crea una instancia de la clase Cuenta.
- Cuenta(float saldo, float tasaAnual): Constructor que recibe el saldo inicial y la tasa anual de interés de la cuenta.
- getSaldo(): Método que devuelve el saldo actual de la cuenta.
- getTasaAnual(): Método que devuelve la tasa anual de interés de la cuenta.
- getNumConsignacion(): Método que devuelve el número de consignaciones realizadas en la cuenta.
- getNumRetiros(): Método que devuelve el número de retiros realizados en la cuenta.
- getComisionMensual(): Método que devuelve la comisión mensual de la cuenta.
- setSaldo(float saldo): Método que establece el saldo de la cuenta.
- setTasaAnual(float tasaAnual): Método que establece la tasa anual de interés de la cuenta.
- Consignar(float cantidad): Método que realiza una consignación o depósito en la cuenta, aumentando el saldo y actualizando el número de consignaciones.
- Retirar(float cantidad): Método que realiza un retiro de la cuenta, siempre y cuando haya saldo suficiente. Actualiza el saldo y el número de retiros. Si la cantidad a retirar excede el saldo, muestra un mensaje de error utilizando el método salidaError de la clase Tools.
- CalcularInteresMensual(): Método que calcula el interés mensual de la cuenta utilizando la tasa anual y actualiza el saldo.

- ExtractoMensual(): Método que realiza el cálculo del extracto mensual de la cuenta, restando la comisión mensual al saldo y calculando el interés mensual.
- Imprimir(): Método que devuelve una representación en forma de cadena de la cuenta, mostrando los valores de saldo, tasaAnual, numConsignacion, numRetiros y comisionMensual.

Estos métodos permiten realizar operaciones básicas en una cuenta bancaria, como consignar, retirar, calcular intereses y generar el extracto mensual. El método Imprimir es equivalente a toString, y devuelve una cadena que representa el estado actual de la cuenta.

Clase CuentaAhorros:

```java
package Herencia;

public class CuentaAhorros extends Cuenta{
    //Atributos
    private boolean activa;
    //Constructor
    public CuentaAhorros(float saldo, float tasaAnual) {
        super(saldo, tasaAnual);
        activa=(saldo<10000)?false:true;
    }
    //Metodos sobre escritos
    public void Consignar(float cantidad) {
        if(activa)
            super.Consignar(cantidad);
    }

    public void Retirar(float cantidad) {
        if (activa)
            super.Retirar(cantidad);
        if (saldo<10000)
            activa=false;
    }

    public void ExtractoMensual() {
        if (numRetiros>4)
            comisionMensual+=(numRetiros-4)*200;
        super.ExtractoMensual();
        if (saldo<10000) activa=false;
    }

    public String Imprimir() {
        return super.Imprimir()+"CuentaAhorros [activa=" + activa + "]";
    }



}
```

La clase CuentaAhorros es una subclase de la clase Cuenta y representa una cuenta de ahorros. Además de los atributos heredados de la clase Cuenta, tiene el atributo adicional activa que indica si la cuenta está activa o no.

- CuentaAhorros(float saldo, float tasaAnual): Constructor que recibe el saldo inicial y la tasa anual de interés de la cuenta de ahorros. Llama al constructor de la clase Cuenta y establece el valor del atributo activa según el saldo inicial.
- Consignar(float cantidad): Método que realiza una consignación o depósito en la cuenta de ahorros, solo si la cuenta está activa. Llama al método Consignar de la clase Cuenta si la cuenta está activa.
- Retirar(float cantidad): Método que realiza un retiro de la cuenta de ahorros, solo si la cuenta está activa. Llama al método Retirar de la clase Cuenta si la cuenta está activa. Si el saldo después del retiro es menor a 10000, establece el atributo activa como false.
- ExtractoMensual(): Método que calcula el extracto mensual de la cuenta de ahorros. Si el número de retiros en el mes es mayor a 4, se aplica una comisión adicional por cada retiro excedente. Luego, llama al método ExtractoMensual de la clase Cuenta para realizar los cálculos adicionales. Si el saldo después de los cálculos es menor a 10000, establece el atributo activa como false.
- Imprimir(): Método que devuelve una representación en forma de cadena de la cuenta de ahorros, mostrando los valores de la clase Cuenta y el atributo activa.

Estos métodos sobrescriben los métodos heredados de la clase Cuenta para añadir lógica adicional relacionada con las cuentas de ahorros, como la activación o desactivación de la cuenta según el saldo y la aplicación de comisiones por retiros excedentes.

Clase CuentaCorriente:

```java
package Herencia;

public class CuentaCorriente extends Cuenta{
    //Atributos
    private float sobregiro=0;
    //Constructor
    public CuentaCorriente(float saldo, float tasaAnual) {
        super(saldo, tasaAnual);
    }
    //Metodos
    public void Retirar(float cantidad) {
        super.Retirar(cantidad);
        if (saldo<cantidad) {
            sobregiro+=cantidad-saldo;
            saldo=0;
        }
    }

    public void Consignar(float cantidad) {
        super.Consignar(cantidad);
        if (sobregiro>0)
            sobregiro-=cantidad;
    }

    public void ExtractoMensual() {
        if (numRetiros>4)
            comisionMensual+=(numRetiros-4)*200;
        super.ExtractoMensual();
    }

    public String Imprimir() {
        return super.Imprimir()+"CuentaCorriente [sobregiro=" + sobregiro + "]";
    }



}
```

La clase CuentaCorriente es una subclase de la clase Cuenta y representa una cuenta corriente. Además de los atributos heredados de la clase Cuenta, tiene el atributo adicional sobregiro que indica el monto acumulado del sobregiro de la cuenta.

- CuentaCorriente(float saldo, float tasaAnual): Constructor que recibe el saldo inicial y la tasa anual de interés de la cuenta corriente. Llama al constructor de la clase Cuenta y no realiza ninguna acción adicional.
- Retirar(float cantidad): Método que realiza un retiro de la cuenta corriente. Llama al método Retirar de la clase Cuenta para realizar el retiro del saldo disponible. Si el saldo no es suficiente, se realiza un sobregiro y se registra el monto sobregirado en el atributo sobregiro.
- Consignar(float cantidad): Método que realiza una consignación o depósito en la cuenta corriente. Llama al método Consignar de la clase Cuenta para realizar la consignación del saldo. Si existe un sobregiro acumulado, se descuenta el monto de la consignación del atributo sobregiro.
- ExtractoMensual(): Método que calcula el extracto mensual de la cuenta corriente. Si el número de retiros en el mes es mayor a 4, se aplica una

comisión adicional por cada retiro excedente. Luego, llama al método ExtractoMensual de la clase Cuenta para realizar los cálculos adicionales.

- Imprimir(): Método que devuelve una representación en forma de cadena de la cuenta corriente, mostrando los valores de la clase Cuenta y el atributo sobregiro.

Estos métodos sobrescriben los métodos heredados de la clase Cuenta para añadir lógica adicional relacionada con las cuentas corrientes, como el registro y descuento de sobregiros.

Clase Docente:

```java
package Herencia;

public class Docente extends Usuario{

    private byte antiguedad;
    private String gradoAcademico;

    public Docente() {
        super();
        // TODO Auto-generated constructor stub
    }

    public Docente(String nombre, String apPaterno, String apMaterno, byte edad, String departamento,byte antiguedad,String gradoAcademico) {
        super(nombre, apPaterno, apMaterno, edad, departamento);
        this.antiguedad=antiguedad;
        this.gradoAcademico=gradoAcademico;
        // TODO Auto-generated constructor stub
    }

    public byte getAntiguedad() {
        return antiguedad;
    }

    public String getGradoAcademico() {
        return gradoAcademico;
    }

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

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

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

}
```

La clase Docente es una subclase de la clase Usuario y representa a un docente. Tiene dos atributos adicionales: antiguedad que indica la cantidad de años de antigüedad del docente y gradoAcademico que representa el grado académico del docente.

- Docente(): Constructor sin argumentos que llama al constructor de la clase Usuario y no realiza ninguna acción adicional.
- Docente(String nombre, String apPaterno, String apMaterno, byte edad, String departamento, byte antiguedad, String gradoAcademico): Constructor que recibe los datos del docente, incluyendo nombre, apellidos, edad, departamento, antigüedad y grado académico. Llama al constructor de la clase Usuario pasando los parámetros correspondientes y asigna los valores de antigüedad y grado académico.
- getAntiguedad(): Método getter que devuelve la antigüedad del docente.
- getGradoAcademico(): Método getter que devuelve el grado académico del docente.

- setAntiguedad(byte antiguedad): Método setter que establece la antigüedad del docente.
- setGradoAcademico(String gradoAcademico): Método setter que establece el grado académico del docente.
- toString(): Método que devuelve una representación en forma de cadena del docente, mostrando los valores de la clase Usuario y los atributos adicionales antiguedad y gradoAcademico.

Estos métodos permiten acceder a la información del docente y actualizarla según sea necesario.

Clase Fecha:

```java
package Herencia;

public class Fecha {
    //atributos
        private byte dia;
        private byte mes;
        private short añio;
        //Constructor vacio
        public Fecha() {}
        //Constructor parametrizado
        public Fecha(byte dia,byte mes,short añio)
        {
            this.dia=dia;
            this.añio=añio;
            this.mes=mes;
        }
        //Encapsular
        public void setDia(byte dia) {
            this.dia=dia;
        }
        public byte getDia() {
            return dia;
        }
        public byte getMes() {
            return mes;
        }
        public void setMes(byte mes) {
            this.mes = mes;
        }
        public short getAñio() {
            return añio;
        }
        public void setAñio(short añio) {
            this.añio = añio;
        }
        @Override
        public String toString() {
            return "Fecha [dia=" + dia + ", mes=" + mes + ", añio=" + añio + "]";
        }
}
```

La clase Fecha representa una fecha con sus atributos dia, mes y año.

- Fecha(): Constructor sin argumentos que no realiza ninguna acción.
- Fecha(byte dia, byte mes, short año): Constructor parametrizado que recibe los valores del día, mes y año y los asigna a los atributos correspondientes.

- setDia(byte dia): Método setter que establece el valor del día.
- getDia(): Método getter que devuelve el valor del día.
- getMes(): Método getter que devuelve el valor del mes.
- setMes(byte mes): Método setter que establece el valor del mes.
- getAño(): Método getter que devuelve el valor del año.
- setAño(short año): Método setter que establece el valor del año.
- toString(): Método que devuelve una representación en forma de cadena de la fecha, mostrando los valores de los atributos dia, mes y año.

Estos métodos permiten acceder a los valores de la fecha y actualizarlos según sea necesario. El método toString() proporciona una representación legible de la fecha en forma de cadena.

Clase Libro:

```java
package Herencia;

public class Libro {

    protected String tituloLibro;
    protected String autorLibro;
    protected String editorialLibro;
    protected float precioLibro;

    public Libro() {}

    public Libro(String tituloLibro, String autorLibro, String editorialLibro, float precioLibro) {
        this.tituloLibro = tituloLibro;
        this.autorLibro = autorLibro;
        this.editorialLibro = editorialLibro;
        this.precioLibro = precioLibro;
    }

    public String getTituloLibro() {
        return tituloLibro;
    }

    public String getAutorLibro() {
        return autorLibro;
    }

    public String getEditorialLibro() {
        return editorialLibro;
    }

    public float getPrecioLibro() {
        return precioLibro;
    }

    public void setTituloLibro(String tituloLibro) {
        this.tituloLibro = tituloLibro;
    }

    public void setAutorLibro(String autorLibro) {
        this.autorLibro = autorLibro;
    }

    public void setEditorialLibro(String editorialLibro) {
        this.editorialLibro = editorialLibro;
    }

    public void setPrecioLibro(float precioLibro) {
        this.precioLibro = precioLibro;
    }

    @Override
    public String toString() {
```

```java
    public String toString() {
        return "Libro [tituloLibro=" + tituloLibro + ", autorLibro=" + autorLibro + ", editorialLibro=" + editorialLibro
                + ", precioLibro=" + precioLibro + "]";
    }


}
```

La clase Libro representa un libro con sus atributos tituloLibro, autorLibro, editorialLibro y precioLibro.

- Libro(): Constructor sin argumentos que no realiza ninguna acción.
- Libro(String tituloLibro, String autorLibro, String editorialLibro, float precioLibro): Constructor parametrizado que recibe los valores del título, autor, editorial y precio del libro y los asigna a los atributos correspondientes.
- getTituloLibro(): Método getter que devuelve el título del libro.
- getAutorLibro(): Método getter que devuelve el autor del libro.
- getEditorialLibro(): Método getter que devuelve la editorial del libro.
- getPrecioLibro(): Método getter que devuelve el precio del libro.
- setTituloLibro(String tituloLibro): Método setter que establece el título del libro.
- setAutorLibro(String autorLibro): Método setter que establece el autor del libro.
- setEditorialLibro(String editorialLibro): Método setter que establece la editorial del libro.
- setPrecioLibro(float precioLibro): Método setter que establece el precio del libro.
- toString(): Método que devuelve una representación en forma de cadena del libro, mostrando los valores de los atributos tituloLibro, autorLibro, editorialLibro y precioLibro.

Estos métodos permiten acceder a los valores del libro y actualizarlos según sea necesario. El método toString() proporciona una representación legible del libro en forma de cadena.

Clase LibroNovela:

```java
package Herencia;

public class LibroNovela extends Libro{

    private String tipoNovela;

    public LibroNovela() {
        super();
    }

    public LibroNovela(String tituloLibro, String autorLibro, String editorialLibro, float precioLibro, String tipoNovela) {
        super(tituloLibro, autorLibro, editorialLibro, precioLibro);
        this.tipoNovela=tipoNovela;
    }

    public String getTipoNovela() {
        return tipoNovela;
    }

    public void setTipoNovela(String tipoNovela) {
        this.tipoNovela = tipoNovela;
    }

    @Override
    public String toString() {
        return super.toString()+"LibroNovela [tipoNovela=" + tipoNovela + "]";
    }

}
```

La clase LibroNovela es una subclase de la clase Libro y representa un libro de novela. Hereda todos los atributos y métodos de la clase Libro y agrega el atributo adicional tipoNovela.

- LibroNovela(): Constructor sin argumentos que llama al constructor sin argumentos de la clase Libro y no realiza ninguna acción adicional.
- LibroNovela(String tituloLibro, String autorLibro, String editorialLibro, float precioLibro, String tipoNovela): Constructor parametrizado que recibe los valores del título, autor, editorial, precio del libro y tipo de novela, y los asigna a los atributos correspondientes heredados de la clase Libro.
- getTipoNovela(): Método getter que devuelve el tipo de novela del libro.
- setTipoNovela(String tipoNovela): Método setter que establece el tipo de novela del libro.
- toString(): Método que devuelve una representación en forma de cadena del libro de novela, mostrando los valores de los atributos heredados de la clase Libro (tituloLibro, autorLibro, editorialLibro, precioLibro) y el atributo adicional tipoNovela.

Estos métodos permiten acceder a los valores específicos del libro de novela y actualizarlos según sea necesario. El método toString() proporciona una representación legible del libro de novela en forma de cadena, incluyendo los atributos heredados de la clase Libro.

Clase LibrosDeTexto:

```java
package Herencia;

public class LibrosDeTexto extends Libro{

    protected String nomCurso;

    public LibrosDeTexto() {
        super();
        // TODO Auto-generated constructor stub
    }

    public LibrosDeTexto(String tituloLibro, String autorLibro, String editorialLibro, float precioLibro, String nomCurso) {
        super(tituloLibro, autorLibro, editorialLibro, precioLibro);
        this.nomCurso=nomCurso;
    }

    public String getNomCurso() {
        return nomCurso;
    }

    public void setNomCurso(String nomCurso) {
        this.nomCurso = nomCurso;
    }

    @Override
    public String toString() {
        return super.toString()+"LibrosDeTexto [nomCurso=" + nomCurso + "]";
    }


}
```

La clase LibrosDeTexto es una subclase de la clase Libro y representa un libro de texto. Hereda todos los atributos y métodos de la clase Libro y agrega el atributo adicional nomCurso.

- LibrosDeTexto(): Constructor sin argumentos que llama al constructor sin argumentos de la clase Libro y no realiza ninguna acción adicional.
- LibrosDeTexto(String tituloLibro, String autorLibro, String editorialLibro, float precioLibro, String nomCurso): Constructor parametrizado que recibe los valores del título, autor, editorial, precio del libro y nombre del curso, y los asigna a los atributos correspondientes heredados de la clase Libro.

- getNomCurso(): Método getter que devuelve el nombre del curso para el libro de texto.
- setNomCurso(String nomCurso): Método setter que establece el nombre del curso para el libro de texto.
- toString(): Método que devuelve una representación en forma de cadena del libro de texto, mostrando los valores de los atributos heredados de la clase Libro (tituloLibro, autorLibro, editorialLibro, precioLibro) y el atributo adicional nomCurso.

Estos métodos permiten acceder a los valores específicos del libro de texto y actualizarlos según sea necesario. El método toString() proporciona una representación legible del libro de texto en forma de cadena, incluyendo los atributos heredados de la clase Libro.

Clase LibrosDeTextoTECNM:

```
package Herencia;

public class LibrosDeTextoTECNM extends LibrosDeTexto{

    private String campus;
    private Fecha fecha;

    public LibrosDeTextoTECNM() {
        super();
        // TODO Auto-generated constructor stub
    }
    public LibrosDeTextoTECNM(String tituloLibro, String autorLibro, String editorialLibro, float precioLibro, String nomCurso, String campus, byte dia,byte mes,short año) {
        super(tituloLibro, autorLibro, editorialLibro, precioLibro, nomCurso);
        this.campus=campus;
        this.fecha=new Fecha(dia,mes,año);
        // TODO Auto-generated constructor stub
    }
    public String getCampus() {
        return campus;
    }

    public Fecha getFecha() {
        return fecha;
    }

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

    public void setFecha(Fecha fecha) {
        this.fecha = fecha;
    }
    @Override
    public String toString() {
        return super.toString()+"LibrosDeTextoTECNM [campus=" + campus + ", fecha=" + fecha + "]";
    }

}
```

La clase LibrosDeTextoTECNM es una subclase de la clase LibrosDeTexto y representa un libro de texto específico para el Tecnológico Nacional de México (TECNM). Además de los atributos heredados de la clase LibrosDeTexto, agrega dos atributos adicionales: campus y fecha.

- LibrosDeTextoTECNM(): Constructor sin argumentos que llama al constructor sin argumentos de la clase LibrosDeTexto y no realiza ninguna acción adicional.
- LibrosDeTextoTECNM(String tituloLibro, String autorLibro, String editorialLibro, float precioLibro, String nomCurso, String campus, byte dia, byte mes, short año): Constructor parametrizado que recibe los valores del título, autor, editorial, precio del libro, nombre del curso, campus, día, mes y año. Llama al constructor parametrizado de la clase LibrosDeTexto para establecer los valores de los atributos heredados y crea un objeto Fecha utilizando los parámetros dia, mes y año.

- getCampus(): Método getter que devuelve el campus asociado al libro de texto.
- getFecha(): Método getter que devuelve el objeto Fecha que representa la fecha asociada al libro de texto.
- setCampus(String campus): Método setter que establece el campus asociado al libro de texto.
- setFecha(Fecha fecha): Método setter que establece el objeto Fecha que representa la fecha asociada al libro de texto.
- toString(): Método que devuelve una representación en forma de cadena del libro de texto, mostrando los valores de los atributos heredados de la clase LibrosDeTexto y los atributos adicionales campus y fecha.

Estos métodos permiten acceder a los valores específicos del libro de texto del TECNM y actualizarlos según sea necesario. El método toString() proporciona una representación legible del libro de texto en forma de cadena, incluyendo los atributos heredados de la clase LibrosDeTexto, así como los atributos adicionales campus y fecha.

Clase Usuario:

```java
package Herencia;

public class Usuario {

    protected String idUsuario;
    protected String nombre;
    protected String apPaterno;
    protected String apMaterno;
    protected byte edad;
    protected String departamento;

    public Usuario() {}

    public Usuario(String nombre, String apPaterno, String apMaterno, byte edad, String departamento) {
        this.nombre = nombre;
        this.apPaterno = apPaterno;
        this.apMaterno = apMaterno;
        this.edad = edad;
        this.departamento = departamento;
        this.idUsuario=GeneraId();
    }

    public String getIdUsuario() {
        return idUsuario;
    }

    public String getNombre() {
        return nombre;
    }

    public String getApPaterno() {
        return apPaterno;
    }

    public String getApMaterno() {
        return apMaterno;
    }

    public byte getEdad() {
        return edad;
    }

    public String getDepartamento() {
        return departamento;
    }

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

    public void setApPaterno(String apPaterno) {
        this.apPaterno = apPaterno;
```

```
    }

    public void setApMaterno(String apMaterno) {
        this.apMaterno = apMaterno;
    }

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

    public void setDepartamento(String departamento) {
        this.departamento = departamento;
    }

    @Override
    public String toString() {
        return "Usuario [idUsuario=" + idUsuario + ", nombre=" + nombre + ", apPaterno=" + apPaterno + ", apMaterno="
                + apMaterno + ", edad=" + edad + ", departamento=" + departamento + "]";
    }

    public String GeneraId() {
        String id="";
        switch(departamento) {
        case "ISC":
            id+=letrasApellido()+letrasNombre();break;
        case "INF":
            id+=letrasApellido()+"_"+letrasNombre();break;
        }

        id=id.toUpperCase();
        return id;
    }

    public String letrasApellido() {
        return apPaterno.substring(0, 3);
    }

    public String letrasNombre() {
        return nombre.substring(0, 2);
    }

}
```

La clase Usuario representa un usuario con atributos como el ID del usuario, nombre, apellidos, edad y departamento.

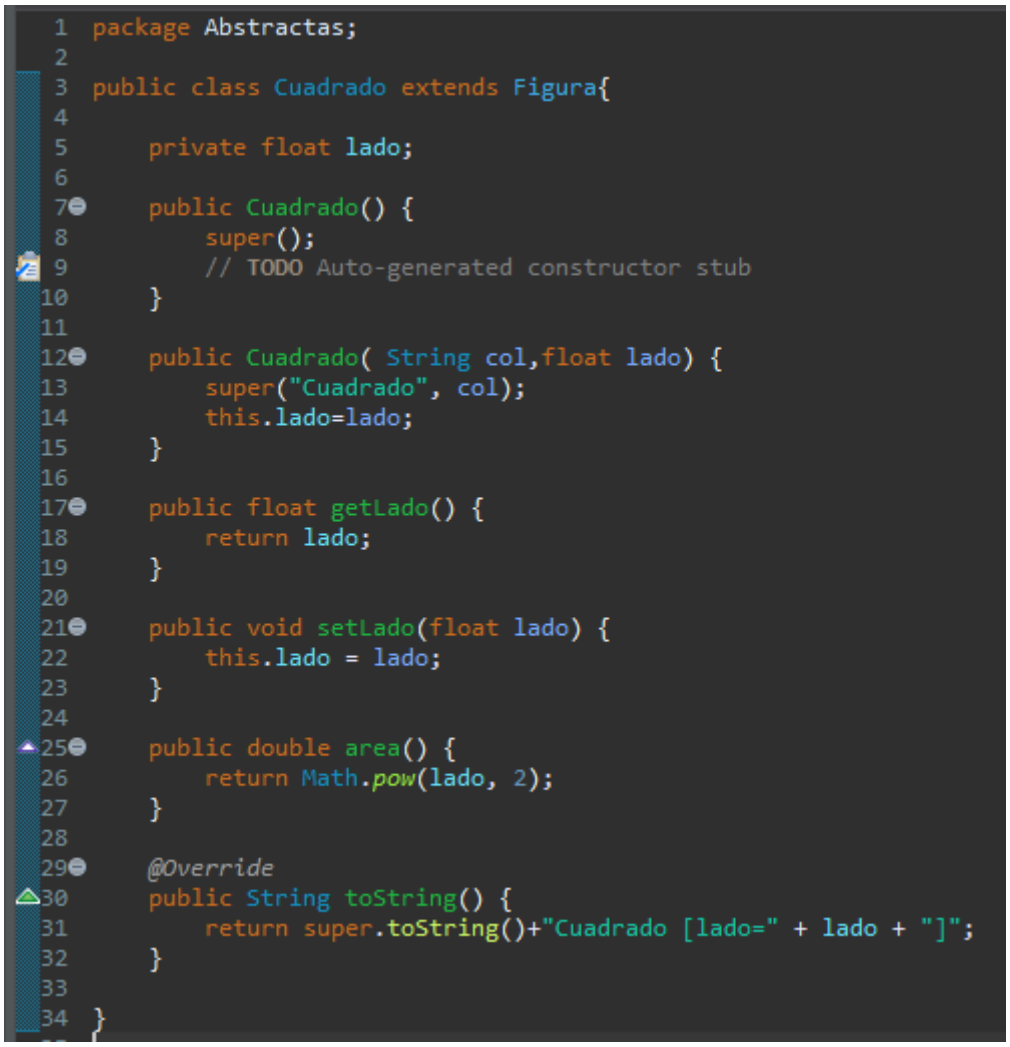
- Usuario(): Constructor sin argumentos que no realiza ninguna acción adicional.
- Usuario(String nombre, String apPaterno, String apMaterno, byte edad, String departamento): Constructor parametrizado que recibe los valores del nombre, apellidos, edad y departamento del usuario. Establece los valores de los atributos correspondientes y también llama al método GeneraId() para generar el ID del usuario.
- getIdUsuario(): Método getter que devuelve el ID del usuario.
- getNombre(): Método getter que devuelve el nombre del usuario.
- getApPaterno(): Método getter que devuelve el apellido paterno del usuario.
- getApMaterno(): Método getter que devuelve el apellido materno del usuario.
- getEdad(): Método getter que devuelve la edad del usuario.
- getDepartamento(): Método getter que devuelve el departamento del usuario.
- setNombre(String nombre): Método setter que establece el nombre del usuario.
- setApPaterno(String apPaterno): Método setter que establece el apellido paterno del usuario.

- setApMaterno(String apMaterno): Método setter que establece el apellido materno del usuario.
- setEdad(byte edad): Método setter que establece la edad del usuario.
- setDepartamento(String departamento): Método setter que establece el departamento del usuario.
- toString(): Método que devuelve una representación en forma de cadena del usuario, mostrando los valores de los atributos del usuario.
- GeneraId(): Método que genera el ID del usuario basado en el departamento y los nombres y apellidos. Dependiendo del departamento, se generará un ID en un formato específico.
- letrasApellido(): Método que devuelve las primeras tres letras del apellido paterno del usuario.
- letrasNombre(): Método que devuelve las primeras dos letras del nombre del usuario.

Estos métodos permiten acceder a los valores del usuario y actualizarlos según sea necesario. Además, el método GeneraId() se utiliza para generar el ID del usuario en función de su departamento y nombres y apellidos. El método toString() proporciona una representación legible del usuario en forma de cadena, mostrando todos los atributos del usuario.

Paquete Abstractas:

Clase Cuadrado:



```java
package Abstractas;

public class Cuadrado extends Figura{

    private float lado;

    public Cuadrado() {
        super();
        // TODO Auto-generated constructor stub
    }

    public Cuadrado( String col,float lado) {
        super("Cuadrado", col);
        this.lado=lado;
    }

    public float getLado() {
        return lado;
    }

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

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

    @Override
    public String toString() {
        return super.toString()+"Cuadrado [lado=" + lado + "]";
    }

}
```

La clase Cuadrado es una subclase de la clase Figura y representa un cuadrado.

- Cuadrado(): Constructor sin argumentos que no realiza ninguna acción adicional.
- Cuadrado(String col, float lado): Constructor parametrizado que recibe el color y el lado del cuadrado. Llama al constructor de la clase Figura para establecer el tipo y el color de la figura, y establece el valor del lado.
- getLado(): Método getter que devuelve el valor del lado del cuadrado.
- setLado(float lado): Método setter que establece el valor del lado del cuadrado.
- area(): Método que calcula y devuelve el área del cuadrado. El área se calcula elevando al cuadrado el valor del lado.
- toString(): Método que devuelve una representación en forma de cadena del cuadrado, mostrando el tipo, el color y el valor del lado.

Además de los métodos descritos, la clase también hereda los atributos y métodos de la clase Figura, que incluyen el tipo y el color de la figura.

Estos métodos y atributos permiten acceder a las propiedades del cuadrado, como el lado y el área, y obtener una representación legible del cuadrado en forma de cadena.

Clase Empleado:

```java
package Abstractas;

public abstract class Empleado {

    private String primerNombre;
    private String apellidoPaterno;
    private String numeroSeguroSocial;

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

    public String getPrimerNombre() {
        return primerNombre;
    }

    public String getApellidoPaterno() {
        return apellidoPaterno;
    }

    public String getNumeroSeguroSocial() {
        return numeroSeguroSocial;
    }

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

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

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

    @Override
    public String toString() {
        return "Empleado [primerNombre=" + primerNombre + ", apellidoPaterno=" + apellidoPaterno
                + ", numeroSeguroSocial=" + numeroSeguroSocial + "]";
    }

    public abstract double ingresos();

}
```

La clase Empleado es una clase abstracta que sirve como base para representar a un empleado.

- Empleado(String nss, String nombre, String apellido): Constructor parametrizado que recibe el número de seguro social, el primer nombre y el apellido del empleado. Establece los valores de los atributos correspondientes.
- getPrimerNombre(): Método getter que devuelve el primer nombre del empleado.
- getApellidoPaterno(): Método getter que devuelve el apellido paterno del empleado.

- getNumeroSeguroSocial(): Método getter que devuelve el número de seguro social del empleado.
- setPrimerNombre(String primerNombre): Método setter que establece el primer nombre del empleado.
- setApellidoPaterno(String apellidoPaterno): Método setter que establece el apellido paterno del empleado.
- setNumeroSeguroSocial(String numeroSeguroSocial): Método setter que establece el número de seguro social del empleado.
- toString(): Método que devuelve una representación en forma de cadena del empleado, mostrando el primer nombre, el apellido paterno y el número de seguro social.
- ingresos(): Método abstracto que debe ser implementado por las subclases de Empleado. Representa los ingresos del empleado y debe ser implementado de manera específica en cada subclase.

La clase Empleado también proporciona una base común para las subclases que extienden la funcionalidad y definen cómo se calculan los ingresos para diferentes tipos de empleados.

Cabe destacar que al ser una clase abstracta, no se pueden crear instancias directas de la clase Empleado, sino que se debe usar como base para crear subclases concretas que implementen el método ingresos() de acuerdo a sus características específicas.

Clase EmpleadoAsalariado:

```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);
        // TODO Auto-generated constructor stub
    }

    public double getSalarioSemanal() {
        return salarioSemanal;
    }

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

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

    @Override
    public String toString() {
        return "Empleado Asalariado" + super.toString() +" salarioSemanal=" + getSalarioSemanal();
    }



}

```

La clase EmpleadoAsalariado es una subclase de Empleado que representa a un empleado asalariado.

- EmpleadoAsalariado(String nss, String nombre, String apellido, double salario): Constructor parametrizado que recibe el número de seguro social, el primer nombre, el apellido y el salario semanal del empleado. Llama al constructor de la superclase Empleado y establece el salario semanal utilizando el método setSalarioSemanal().
- getSalarioSemanal(): Método getter que devuelve el salario semanal del empleado.
- setSalarioSemanal(double salario): Método setter que establece el salario semanal del empleado. Si el salario es negativo, se establece como 0.
- ingresos(): Implementación del método abstracto ingresos() de la clase Empleado. Retorna el salario semanal del empleado.
- toString(): Método que devuelve una representación en forma de cadena del empleado asalariado, mostrando el tipo de empleado, el primer nombre, el apellido paterno y el salario semanal.

La clase EmpleadoAsalariado hereda los atributos y métodos de la clase Empleado y agrega el atributo específico salarioSemanal y su funcionalidad.

Clase EmpleadoBaseMasComision:

```java
package Abstractas;

public class EmpleadoBaseMasComision extends EmpleadoPorHoras{

    private double salarioBase;

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

    public double getSalarioBase() {
        return salarioBase;
    }

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

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

    public String toString(){
        return "Empleado con salario base" + super.toString() + "Salario base: " + getSalarioBase();
    }

}
```

La clase EmpleadoBaseMasComision es una subclase de EmpleadoPorHoras y representa a un empleado que recibe un salario base más comisiones.

- EmpleadoBaseMasComision(String nss, String nombre, String apellido, double sueldoPorHoras, double horasTrabajadas, double salario): Constructor parametrizado que recibe el número de seguro social, el primer nombre, el apellido, el sueldo por horas, las horas trabajadas y el salario base del empleado. Llama al constructor de la superclase EmpleadoPorHoras y establece el salario base utilizando el método setSalarioBase().

- getSalarioBase(): Método getter que devuelve el salario base del empleado.
- setSalarioBase(double salario): Método setter que establece el salario base del empleado. Si el salario es negativo, se establece como 0.
- ingresos(): Implementación del método ingresos() de la superclase EmpleadoPorHoras. Retorna el salario base del empleado más los ingresos calculados por las horas trabajadas.
- toString(): Método que devuelve una representación en forma de cadena del empleado con salario base, mostrando el tipo de empleado, el primer nombre, el apellido paterno, el sueldo por horas, las horas trabajadas y el salario base.

La clase EmpleadoBaseMasComision hereda los atributos y métodos de la clase EmpleadoPorHoras y agrega el atributo específico salarioBase y su funcionalidad.

Clase EmpleadoPorComision:

```java
package Abstractas;

public class EmpleadoPorComision extends Empleado{

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

    public double getVentasBrutas() {
        return ventasBrutas;
    }

    public double getTarifaComision() {
        return tarifaComision;
    }

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

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

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

    public String toString(){
        return "Empleado por comision" + super.toString() + "Ventasbrutas: " + getVentasBrutas() + "Tarifa de comision: " + getTarifaComision();
    }

}
```

La clase EmpleadoPorComision es una subclase de Empleado y representa a un empleado que recibe un salario basado en comisiones por las ventas realizadas.

- EmpleadoPorComision(String nss, String nombre, String apellido, double ventas, double tarifa): Constructor parametrizado que recibe el número de seguro social, el primer nombre, el apellido, las ventas brutas y la tarifa de comisión del empleado. Llama al constructor de la superclase Empleado y establece las ventas brutas y la tarifa de comisión utilizando los métodos setVentasBrutas() y setTarifaComision().
- getVentasBrutas(): Método getter que devuelve el monto total de las ventas brutas del empleado.

- getTarifaComision(): Método getter que devuelve la tarifa de comisión del empleado.
- setVentasBrutas(double ventas): Método setter que establece el monto total de las ventas brutas del empleado. Si las ventas son negativas, se establece como 0.
- setTarifaComision(double tarifa): Método setter que establece la tarifa de comisión del empleado. Si la tarifa es menor que 0 o mayor que 1, se establece como 0.
- ingresos(): Implementación del método ingresos() de la superclase Empleado. Retorna el producto de la tarifa de comisión y las ventas brutas del empleado.
- toString(): Método que devuelve una representación en forma de cadena del empleado por comisión, mostrando el tipo de empleado, el primer nombre, el apellido paterno, las ventas brutas y la tarifa de comisión.

La clase EmpleadoPorComision hereda los atributos y métodos de la clase Empleado y agrega los atributos específicos ventasBrutas y tarifaComision, así como su funcionalidad correspondiente.

Clase EmpleadoPorHoras:

```java
package Abstractas;

public class EmpleadoPorHoras extends Empleado{

    private double sueldo;
    private double horas;

    public EmpleadoPorHoras(String nss, String nombre, String apellido, double sueldoPorHoras, double horasTrabajadas ) {
        super(nss, nombre, apellido);
        setSueldo(sueldoPorHoras);
        setHoras(horasTrabajadas);
    }

    public double getSueldo() {
        return sueldo;
    }

    public double getHoras() {
        return horas;
    }

    public void setSueldo( double sueldoPorHoras ){
        sueldo = ( sueldoPorHoras < 0.0 ) ? 0.0 : sueldoPorHoras;
    }

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

    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" + super.toString() + "Sueldo por hora:" + getSueldo() + " Horas trabajadas: " + getHoras();
        }

}
```

La clase EmpleadoPorHoras es una subclase de Empleado y representa a un empleado que recibe un salario basado en horas trabajadas.

- EmpleadoPorHoras(String nss, String nombre, String apellido, double sueldoPorHoras, double horasTrabajadas): Constructor parametrizado que recibe el número de seguro social, el primer nombre, el apellido, el

sueldo por hora y las horas trabajadas del empleado. Llama al constructor de la superclase Empleado y establece el sueldo por hora y las horas trabajadas utilizando los métodos setSueldo() y setHoras().

- getSueldo(): Método getter que devuelve el sueldo por hora del empleado.
- getHoras(): Método getter que devuelve las horas trabajadas del empleado.
- setSueldo(double sueldoPorHoras): Método setter que establece el sueldo por hora del empleado. Si el sueldo es negativo, se establece como 0.
- setHoras(double horasTrabajadas): Método setter que establece las horas trabajadas del empleado. Si las horas son menores que 0 o mayores que 168 (que representa una semana completa de trabajo), se establecen como 0.
- ingresos(): Implementación del método ingresos() de la superclase Empleado. Calcula los ingresos del empleado basados en el sueldo por hora y las horas trabajadas. Si las horas trabajadas son menores o iguales a 40, se multiplica el sueldo por hora por las horas trabajadas. Si las horas trabajadas son mayores a 40, se calcula el sueldo por hora para las primeras 40 horas y se le agrega el 50% adicional para las horas extras.
- toString(): Método que devuelve una representación en forma de cadena del empleado por horas, mostrando el tipo de empleado, el primer nombre, el apellido paterno, el sueldo por hora y las horas trabajadas.

La clase EmpleadoPorHoras hereda los atributos y métodos de la clase Empleado y agrega los atributos específicos sueldo y horas, así como su funcionalidad correspondiente.

Clase Figura:

```java
package Abstractas;

public abstract class Figura {

    private String nomFig;
    private String color;

    public Figura() {}

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

    public String getNomFig() {
        return nomFig;
    }

    public String getColor() {
        return color;
    }

    public void setNomFig(String nomFig) {
        this.nomFig = nomFig;
    }

    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 proporciona una base para representar diferentes figuras geométricas.

- Figura(): Constructor por defecto que no recibe ningún parámetro.
- Figura(String nom, String col): Constructor parametrizado que recibe el nombre de la figura y el color. Establece estos valores en los atributos nomFig y color.
- getNomFig(): Método getter que devuelve el nombre de la figura.
- getColor(): Método getter que devuelve el color de la figura.
- setNomFig(String nomFig): Método setter que establece el nombre de la figura.
- setColor(String color): Método setter que establece el color de la figura.
- area(): Método abstracto que debe ser implementado por las subclases. Representa el cálculo del área de la figura.
- toString(): Método que devuelve una representación en forma de cadena de la figura, mostrando el nombre de la figura y el color.

La clase Figura es una clase abstracta, lo que significa que no se pueden crear instancias directamente de esta clase, sino que se utilizan como base para crear subclases que implementan los detalles específicos de cada figura geométrica. Las subclases deben implementar el método area() de acuerdo con la fórmula de cálculo de área correspondiente a cada figura específica.

Clase Triangulo:

```java
package Abstractas;

public class Triangulo extends Figura{

    private float base,altura;

    public Triangulo() {
        super();
    }

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

    public float getBase() {
        return base;
    }

    public float getAltura() {
        return altura;
    }

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

    public void setAltura(float altura) {
        this.altura = altura;
    }

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

    @Override
    public String toString() {
        return super.toString()+"Triangulo [base=" + base + ", altura=" + altura + "]";
    }

}
```

La clase Triangulo es una subclase de la clase abstracta Figura y representa un triángulo geométrico. Aquí está una descripción de los métodos y atributos en esta clase:

- Triangulo(): Constructor por defecto que no recibe ningún parámetro. Llama al constructor de la clase padre (Figura) sin argumentos.
- Triangulo(String col, float base, float altura): Constructor parametrizado que recibe el color, la base y la altura del triángulo. Llama al constructor de la clase padre (Figura) con el nombre "Triangulo" y el color especificado, y establece los valores de base y altura en los atributos correspondientes.

- getBase(): Método getter que devuelve el valor de la base del triángulo.
- getAltura(): Método getter que devuelve el valor de la altura del triángulo.
- setBase(float base): Método setter que establece el valor de la base del triángulo.
- setAltura(float altura): Método setter que establece el valor de la altura del triángulo.
- area(): Método que calcula y devuelve el área del triángulo utilizando la fórmula (base * altura) / 2.
- toString(): Método que devuelve una representación en forma de cadena del triángulo, mostrando el nombre de la figura, el color, la base y la altura.

La clase Triangulo hereda el método area() de la clase Figura y lo implementa específicamente para calcular el área de un triángulo. También hereda el método toString() de la clase Figura y lo modifica para incluir la información adicional del triángulo.

Paquete Test:

Clase DocenteAlumnos:

```
1  package Test;
2
3  import EntradasSalidas.Tools;
5
6  public class DocenteAlumnos {
7
8      public static void main(String[] args) {
9          Docente doc=new Docente(Tools.leerString("Nombre"),Tools.leerString("Apellido paterno"),Tools.leerString("Apellido Materno"),
10                 Tools.leerByte("Edad"),Tools.Desplegable("ISC,INF"),Tools.leerByte("antiguedad"),Tools.leerString("Grado adademico"));
11         Alumno alu=new Alumno(Tools.leerString("Nombre"),Tools.leerString("Apellido paterno"),Tools.leerString("Apellido Materno"),
12                 Tools.leerByte("Edad"),Tools.Desplegable("ISC,INF"),Tools.leerByte("Semestre"),Tools.leerFloat("Promedio"));
13         Tools.imprimePantalla("Docente: \n"+doc.toString()+"\n\nAlumno:\n"+alu.toString());
14     }
15
16 }
```

La clase llamada DocenteAlumnos contiene un método main para probar las clases Docente y Alumno.

Se crea un objeto doc de la clase Docente utilizando valores ingresados por el usuario. Los valores solicitados son: nombre, apellido paterno, apellido materno, edad, departamento (ISC o INF), antigüedad y grado académico. Se utiliza la clase Tools para leer los valores del usuario.

Se crea un objeto alu de la clase Alumno utilizando valores ingresados por el usuario. Los valores solicitados son: nombre, apellido paterno, apellido materno, edad, departamento (ISC o INF), semestre y promedio. También se utiliza la clase Tools para leer los valores del usuario.

Se utiliza el método toString() de los objetos doc y alu para obtener una representación en forma de cadena de los mismos.

Se utiliza el método imprimePantalla() de la clase Tools para mostrar en pantalla la información del docente y el alumno.

Clase PruebaSmartphone:

```java
package Test;

import EntradasSalidas.Tools;

public class PruebaSmartphone {

    public static void main(String[] args) {
        Smartphone celPedro=new Smartphone("JPRO",3400,"SAMSUNG","microUSB");
        Chip telcel=new Chip("telcel", 45678978,"Micro");
        Chip unefon=new Chip("unefon", 4136896,"Micro");
        Chip rad=new Chip("unefon", 45678978,"Micro");

        celPedro.agregarChip(telcel);
        celPedro.agregarChip(unefon);
        celPedro.agregarChip(rad);
        Tools.imprimePantalla(""+celPedro.toString());
    }

}
```

La clase llamada PruebaSmartphone contiene un método main para probar la clase Smartphone y la clase Chip.

Se crea un objeto celPedro de la clase Smartphone con los siguientes valores: modelo "JPRO", capacidad de batería 3400, marca "SAMSUNG" y tipo de conector "microUSB".

Se crean tres objetos de la clase Chip con diferentes valores: telcel, unefon y rad. Cada objeto Chip se crea con un nombre de compañía, un número de teléfono y un tipo de chip.

Se utilizan los métodos agregarChip() del objeto celPedro para agregar los objetos telcel, unefon y rad como chips asociados al smartphone.

Se utiliza el método toString() del objeto celPedro para obtener una representación en forma de cadena del mismo.

Se utiliza el método imprimePantalla() de la clase Tools para mostrar en pantalla la información del smartphone.

Clase TestCuenta:

```java
package Test;

import EntradasSalidas.Tools;

public class TestCuenta {

    public static void main(String[] args) {
        MenuCuentas("Cuenta normal,Cuenta Ahorros,Cuenta Corriente,Imprimir,Salir");
    }

    public static void MenuCuentas(String menu) {
        String sel="", cn="",ca="",cc="";
        do {
            sel=Tools.Boton(menu);
            switch (sel) {
            case "Cuenta normal":
                Cuenta cue=new Cuenta(Tools.leerFloat("Saldo: "),Tools.leerFloat("Tasa anual"));
                cn+=cuenta(cue)+"\n";
                break;
            case "Cuenta Ahorros":
                CuentaAhorros cueAh=new CuentaAhorros(Tools.leerFloat("Saldo: "),Tools.leerFloat("Tasa anual"));
                ca+=cuentaAhorro(cueAh)+"\n";
                break;
            case "Cuenta Corriente":
                CuentaCorriente cueCor=new CuentaCorriente(Tools.leerFloat("Saldo: "),Tools.leerFloat("Tasa anual"));
                cc+=cuentaCorriente(cueCor)+"\n";
                break;
            case "Imprimir":
                Tools.imprimePantalla("Cuentas capturadas: \n\n Cuenta normales: \n"+cn+"\n Cuentas Ahorro: \n"+ca+"\n Cuentas Corrientes\n"+cc);
                break;
            case "Salir":
            }
        }while(!sel.equalsIgnoreCase("Salir"));
    }

    public static String cuenta(Cuenta cuenta) {
        String sel="";
        do {
            sel=Tools.Desplegable("Consignar,Retirar,Calcular Interes mensual,Extracto Mensual,Imprimir,Salir");
            switch(sel) {
            case "Consignar":
                cuenta.Consignar(Tools.leerFloat("Cantidad a depositar: "));
                Tools.imprimePantalla("Cantidad de saldo actual: "+cuenta.getSaldo());
                break;
            case "Retirar":
                cuenta.Retirar(Tools.leerFloat("Cantidad a retirar: "));
                break;
            case "Calcular Interes mensual: ":
                cuenta.CalcularInteresMensual();
                break;
            case "Extracto Mensual":
                cuenta.ExtractoMensual();
```

```
54                  break;
55              case "Imprimir":
56                  Tools.imprimePantalla(cuenta.Imprimir());
57                  break;
58              case "Salir":
59                  break;
60              }
61          }while(!sel.equalsIgnoreCase("Salir"));
62          return cuenta.Imprimir();
63      }
64
65      public static String cuentaAhorro(CuentaAhorros cuenta) {
66          String sel="";
67          do {
68              sel=Tools.Desplegable("Consignar,Retirar,Extracto Mensual,Imprimir,Salir");
69              switch(sel) {
70              case "Consignar":
71                  cuenta.Consignar(Tools.leerFloat("Cantidad a depositar: "));
72                  Tools.imprimePantalla("Cantidad de saldo actual: "+cuenta.getSaldo());
73                  break;
74              case "Retirar":
75                  cuenta.Retirar(Tools.leerFloat("Cantidad a retirar: "));
76                  break;
77              case "Extracto Mensual":
78                  cuenta.ExtractoMensual();
79                  break;
80              case "Imprimir":
81                  Tools.imprimePantalla(cuenta.Imprimir());
82                  break;
83              case "Salir":
84                  break;
85              }
86          }while(!sel.equalsIgnoreCase("Salir"));
87          return cuenta.Imprimir();
88      }
89
90      public static String cuentaCorriente(CuentaCorriente cuenta) {
91          String sel="";
92          do {
93              sel=Tools.Desplegable("Consignar,Retirar,Extracto Mensual,Imprimir,Salir");
94              switch(sel) {
95              case "Consignar":
96                  cuenta.Consignar(Tools.leerFloat("Cantidad a depositar: "));
97                  Tools.imprimePantalla("Cantidad de saldo actual: "+cuenta.getSaldo());
98                  break;
99              case "Retirar":
100                 cuenta.Retirar(Tools.leerFloat("Cantidad a retirar: "));
101                 break;
102             case "Extracto Mensual":
103                 cuenta.ExtractoMensual();
104                 break;
105             case "Imprimir":
```

```
105             case "Imprimir":
106                 Tools.imprimePantalla(cuenta.Imprimir());
107                 break;
108             case "Salir":
109                 break;
110             }
111         }while(!sel.equalsIgnoreCase("Salir"));
112         return cuenta.Imprimir();
113     }
114 }
```

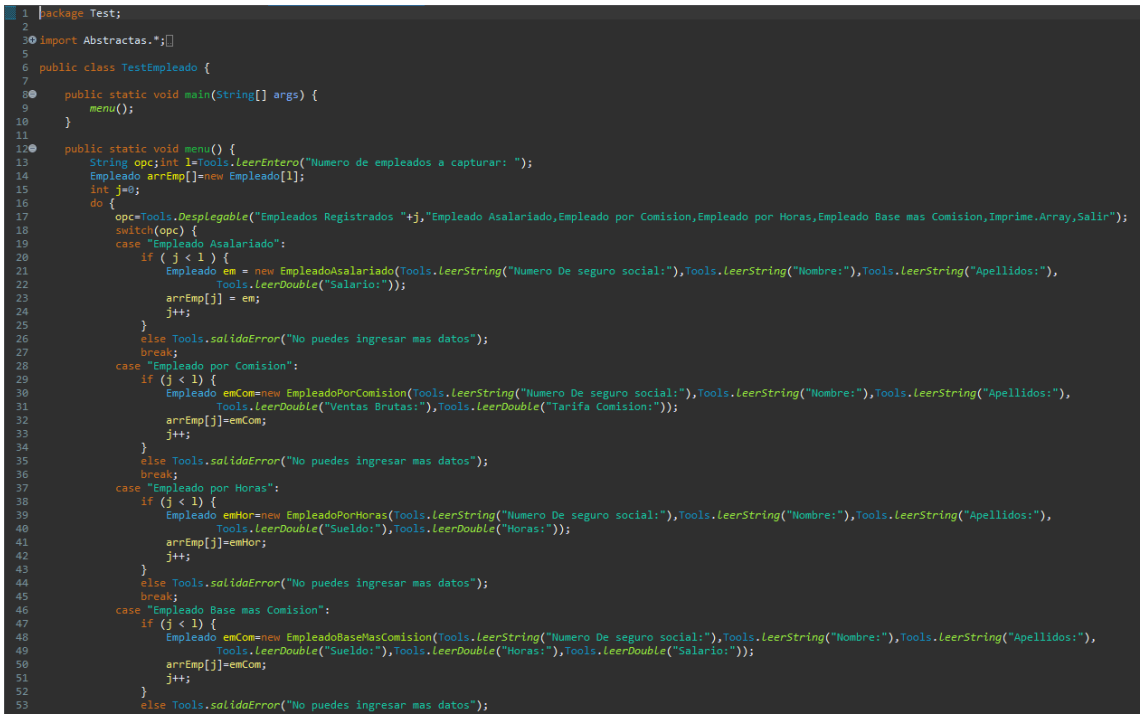
La clase llamada TestCuenta contiene un método main y otros métodos auxiliares para probar y manejar cuentas bancarias.

- En el método main, se llama al método MenuCuentas() pasando como argumento un menú de opciones representado como una cadena. El

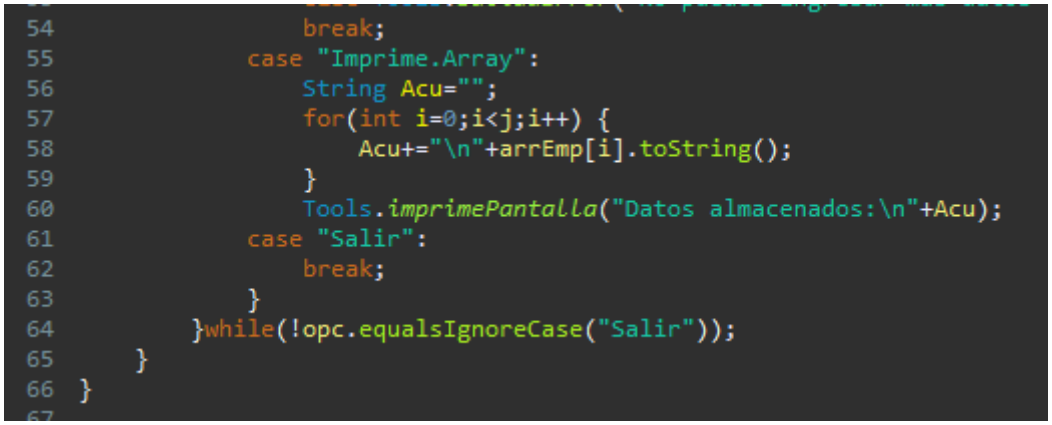
menú incluye las opciones "Cuenta normal", "Cuenta Ahorros", "Cuenta Corriente", "Imprimir" y "Salir".

- El método MenuCuentas() toma una cadena menu como argumento y realiza un bucle do-while que se ejecuta hasta que la opción seleccionada sea "Salir". Dentro del bucle, se muestra el menú utilizando el método Boton() de la clase Tools y se utiliza una declaración switch para realizar diferentes acciones según la opción seleccionada.
- Si se selecciona "Cuenta normal", se crea un objeto de la clase Cuenta mediante la entrada del saldo y la tasa anual utilizando el método leerFloat() de la clase Tools. Luego se llama al método cuenta() pasando la cuenta creada y se guarda la cadena resultante en la variable cn.
- Si se selecciona "Cuenta Ahorros", se realiza un proceso similar al paso anterior, pero utilizando la clase CuentaAhorros en lugar de Cuenta. La cadena resultante se guarda en la variable ca.
- Si se selecciona "Cuenta Corriente", se realiza un proceso similar al paso anterior, pero utilizando la clase CuentaCorriente en lugar de Cuenta. La cadena resultante se guarda en la variable cc.
- Si se selecciona "Imprimir", se utiliza el método imprimePantalla() de la clase Tools para mostrar en pantalla la información de las cuentas capturadas.
- El bucle continúa hasta que se seleccione la opción "Salir".
- Los métodos cuenta(), cuentaAhorro() y cuentaCorriente() se utilizan para manejar las acciones correspondientes a cada tipo de cuenta. Estos métodos muestran un submenú utilizando el método Desplegable() de la clase Tools y realizan acciones según la opción seleccionada.
- Después de realizar la acción correspondiente, se muestra información adicional utilizando los métodos de la clase Cuenta correspondientes.
- El bucle en cada método continúa hasta que se seleccione la opción "Salir".
- Al final de cada método, se devuelve la información de la cuenta utilizando el método Imprimir() de la clase Cuenta.

Clase TestEmpleado:



```
package Test;

import Abstractas.*;

public class TestEmpleado {

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

    public static void menu() {
        String opc;int l=Tools.leerEntero("Numero de empleados a capturar: ");
        Empleado arrEmp[]=new Empleado[l];
        int j=0;
        do {
            opc=Tools.Desplegable("Empleados Registrados "+j,"Empleado Asalariado,Empleado por Comision,Empleado por Horas,Empleado Base mas Comision,Imprime.Array,Salir");
            switch(opc) {
            case "Empleado Asalariado":
                if ( j < l ) {
                    Empleado em = new EmpleadoAsalariado(Tools.leerString("Numero De seguro social:"),Tools.leerString("Nombre:"),Tools.leerString("Apellidos:"),
                            Tools.leerDouble("Salario:"));
                    arrEmp[j] = em;
                    j++;
                }
                else Tools.salidaError("No puedes ingresar mas datos");
                break;
            case "Empleado por Comision":
                if (j < l) {
                    Empleado emCom=new EmpleadoPorComision(Tools.leerString("Numero De seguro social:"),Tools.leerString("Nombre:"),Tools.leerString("Apellidos:"),
                            Tools.leerDouble("Ventas Brutas:"),Tools.leerDouble("Tarifa Comision:"));
                    arrEmp[j]=emCom;
                    j++;
                }
                else Tools.salidaError("No puedes ingresar mas datos");
                break;
            case "Empleado por Horas":
                if (j < l) {
                    Empleado emHor=new EmpleadoPorHoras(Tools.leerString("Numero De seguro social:"),Tools.leerString("Nombre:"),Tools.leerString("Apellidos:"),
                            Tools.leerDouble("Sueldo:"),Tools.leerDouble("Horas:"));
                    arrEmp[j]=emHor;
                    j++;
                }
                else Tools.salidaError("No puedes ingresar mas datos");
                break;
            case "Empleado Base mas Comision":
                if (j < l) {
                    Empleado emCom=new EmpleadoBaseMasComision(Tools.leerString("Numero De seguro social:"),Tools.leerString("Nombre:"),Tools.leerString("Apellidos:"),
                            Tools.leerDouble("Sueldo:"),Tools.leerDouble("Horas:"),Tools.leerDouble("Salario:"));
                    arrEmp[j]=emCom;
                    j++;
                }
                else Tools.salidaError("No puedes ingresar mas datos");
```



```
                break;
            case "Imprime.Array":
                String Acu="";
                for(int i=0;i<j;i++) {
                    Acu+="\n"+arrEmp[i].toString();
                }
                Tools.imprimePantalla("Datos almacenados:\n"+Acu);
            case "Salir":
                break;
            }
        }while(!opc.equalsIgnoreCase("Salir"));
    }
}
```

La clase llamada TestEmpleado contiene un método main y otros métodos auxiliares para probar y gestionar empleados.

- En el método main, se llama al método menu() que muestra un menú interactivo.
- El método menu() realiza un bucle do-while que se ejecuta hasta que la opción seleccionada sea "Salir". Dentro del bucle, se muestra un menú utilizando el método Desplegable() de la clase Tools y se utiliza una declaración switch para realizar diferentes acciones según la opción seleccionada.
- Si se selecciona "Empleado Asalariado", se crea un objeto de la clase EmpleadoAsalariado mediante la entrada del número de seguro social, nombre, apellidos y salario utilizando los métodos leerString() y leerDouble() de la clase Tools. El objeto se guarda en el array arrEmp[] en la posición j y se incrementa j en 1.

- Si se selecciona "Empleado por Comision", se realiza un proceso similar al paso anterior, pero utilizando la clase EmpleadoPorComision en lugar de EmpleadoAsalariado. Se ingresan los datos adicionales de ventas brutas y tarifa de comisión.
- Si se selecciona "Empleado por Horas", se realiza un proceso similar al paso anterior, pero utilizando la clase EmpleadoPorHoras en lugar de EmpleadoAsalariado. Se ingresan los datos adicionales de sueldo y horas trabajadas.
- Si se selecciona "Empleado Base mas Comision", se realiza un proceso similar al paso anterior, pero utilizando la clase EmpleadoBaseMasComision en lugar de EmpleadoAsalariado. Se ingresan los datos adicionales de sueldo, horas trabajadas y salario base.
- Si se selecciona "Imprime.Array", se recorre el array arrEmp[] y se muestra en pantalla la información de todos los empleados registrados utilizando el método toString() de la clase Empleado.
- Si se selecciona "Salir", se sale del bucle y finaliza el programa.

Clase TestFigura:

```java
package Test;

import Abstractas.*;

public class TestFigura {

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

    public static void menu() {
        String opc;
        Figura arrFig[]=new Figura[10];
        int j=0;
        do {
            opc=Tools.Desplegable("Figuras Registradas"+j,"Triangulo,Cuadrado,Imprime.Array,Salir");
            switch(opc) {
            case "Triangulo":
                if(j<10) {
                    Figura Trian=new Triangulo(Tools.leerString("Color"),Tools.leerFloat("Base"),Tools.leerFloat("Altura"));
                    arrFig[j]=Trian;
                    j++;
                }
                else {Tools.salidaError("No puedes ingresar mas datos");}
                break;
            case "Cuadrado":
                if(j<10) {
                    Figura Rectan=new Cuadrado(Tools.leerString("Color"),Tools.leerFloat("Lado"));
                    arrFig[j]=Rectan;
                    j++;
                }
                else {Tools.salidaError("No puedes ingresar mas datos");}
                break;
            case "Imprime.Array":
                String Acu="";
                for(int i=0;i<j;i++) {
                    Acu+="\n"+arrFig[i].toString();
                }
                Tools.imprimePantalla("Datos almacenados:\n"+Acu);
            case "Salir":
                break;
            }
        }while(!opc.equalsIgnoreCase("Salir"));
    }
}
```

La clase llamada TestFigura contiene un método main y otros métodos auxiliares para probar y gestionar figuras.

- En el método main, se llama al método menu() que muestra un menú interactivo.

- El método menu() realiza un bucle do-while que se ejecuta hasta que la opción seleccionada sea "Salir". Dentro del bucle, se muestra un menú utilizando el método Desplegable() de la clase Tools y se utiliza una declaración switch para realizar diferentes acciones según la opción seleccionada.
- Si se selecciona "Triangulo", se crea un objeto de la clase Triangulo mediante la entrada de color, base y altura utilizando los métodos leerString() y leerFloat() de la clase Tools. El objeto se guarda en el array arrFig[] en la posición j y se incrementa j en 1.
- Si se selecciona "Cuadrado", se realiza un proceso similar al paso anterior, pero utilizando la clase Cuadrado en lugar de Triangulo. Se ingresa el dato adicional del lado del cuadrado.
- Si se selecciona "Imprime.Array", se recorre el array arrFig[] y se muestra en pantalla la información de todas las figuras registradas utilizando el método toString() de cada objeto figura.
- Si se selecciona "Salir", se sale del bucle y finaliza el programa.

Clase TestLibros:

```java
package Test;

import EntradasSalidas.Tools;

public class TestLibros {
    public static void main(String[] args) {
        MenuCuentas("Libro,Libro de texto,Libro de texto TECNM,Libro de novela,Imprimir,Salir");
    }

    public static void MenuCuentas(String menu) {
        String sel="",li="",lite="",litetec="",lino="";
        do {
            sel=Tools.Desplegable("Tipos de libros",menu);
            switch (sel) {
            case "Libro":
                Libro lib=new Libro(Tools.leerString("Nombre del libro"),Tools.leerString("Nombre del autor"),Tools.leerString("Editorial"),
                        Tools.leerFloat("Precio del libro"));
                li+=lib.toString()+"\n";
                break;
            case "Libro de texto":
                LibrosDeTexto libTex=new LibrosDeTexto(Tools.leerString("Nombre del libro"),Tools.leerString("Nombre del autor"),
                        Tools.leerString("Editorial"),Tools.leerFloat("Precio del libro"),Tools.leerString("Nombre del curso asociado al libro"));
                lite+=libTex.toString()+"\n";
                break;
            case "Libro de texto TECNM":
                LibrosDeTextoTECNM libTexTec=new LibrosDeTextoTECNM(Tools.leerString("Nombre del libro"),Tools.leerString("Nombre del autor"),
                        Tools.leerString("Editorial"),Tools.leerFloat("Precio del libro"),Tools.leerString("Nombre del curso asociado al libro"),
                        Tools.leerString("Campus que lo publico"),Tools.leerByte("Dia de publicación"),Tools.leerByte("Mes de publicación"),
                        Tools.leerShort("Año de publicación"));
                litetec+=libTexTec.toString()+"\n";
                break;
            case "Libro de novela":
                LibroNovela libNov=new LibroNovela(Tools.leerString("Nombre del libro"),Tools.leerString("Nombre del autor"),
                        Tools.leerString("Editorial"),Tools.leerFloat("Precio del libro"),
                        Tools.Desplegable("Tipo de novela","Histórica,Romántica,Policíaca,Realista,Ciencia Ficción,Aventuras"));
                lino+=libNov.toString()+"\n";
                break;
            case "Imprimir":
                Tools.imprimePantalla("Libros capturados\n\nLibros normales:\n"+li+"\n\nLibros de Texto:\n"+lite+"\n\nLibros de texto del TECNM:\n"+
                        litetec+"\n\nLibros de Novelas:\n"+lino);
                break;
            case "Salir":
            }
        }while(!sel.equalsIgnoreCase("Salir"));
    }
}
```

El código proporcionado muestra una clase llamada TestLibros que contiene un método main y otros métodos auxiliares para probar y gestionar libros. Aquí está una descripción de lo que hace el programa:

- En el método main, se llama al método MenuCuentas() y se le pasa como argumento una cadena de texto que representa las opciones del menú.
- El método MenuCuentas() recibe una cadena de texto que representa las opciones del menú y realiza un bucle do-while que se ejecuta hasta que

la opción seleccionada sea "Salir". Dentro del bucle, se muestra un menú utilizando el método Desplegable() de la clase Tools y se utiliza una declaración switch para realizar diferentes acciones según la opción seleccionada.

- Si se selecciona "Libro", se creará un objeto de la clase Libro mediante la entrada del nombre del libro, nombre del autor, editorial y precio del libro utilizando los métodos leerString() y leerFloat() de la clase Tools. El objeto se guarda en la cadena li.
- Si se selecciona "Libro de texto", se realiza un proceso similar al paso anterior, pero utilizando la clase LibrosDeTexto en lugar de Libro. Se ingresa un dato adicional para el nombre del curso asociado al libro.
- Si se selecciona "Libro de texto TECNM", se realiza un proceso similar al paso anterior, pero utilizando la clase LibrosDeTextoTECNM en lugar de LibrosDeTexto. Se ingresan datos adicionales como el campus que lo publicó, la fecha de publicación y el año de publicación.
- Si se selecciona "Libro de novela", se realiza un proceso similar al paso anterior, pero utilizando la clase LibroNovela en lugar de LibrosDeTextoTECNM. Se ingresa un dato adicional para el tipo de novela seleccionado.
- Si se selecciona "Imprimir", se muestra en pantalla la información de los libros capturados, separada por categorías.
- Si se selecciona "Salir", se sale del bucle y finaliza el programa.

Conclusión:

En la programación orientada a objetos, la herencia, el polimorfismo, las clases abstractas y la reutilización de la definición de paquetes/librerías son conceptos esenciales que permiten crear código modular, flexible y reutilizable.

La herencia y el polimorfismo son pilares fundamentales de la programación orientada a objetos. La herencia nos permite crear jerarquías de clases, donde las clases derivadas heredan atributos y métodos de la clase base, lo que fomenta la reutilización de código y facilita la organización del programa. El polimorfismo nos permite tratar objetos de diferentes clases de manera uniforme, lo que promueve la flexibilidad y la escritura de código genérico.

Las clases abstractas proporcionan una forma de definir comportamientos comunes y establecer una estructura para las clases derivadas. Permiten la creación de métodos abstractos, que deben ser implementados por las clases derivadas, lo que garantiza que cumplan con un contrato establecido.

Por último, la reutilización de la definición de paquetes/librerías nos permite aprovechar el código existente y utilizarlo en nuestros proyectos. Esto ahorra tiempo y esfuerzo, ya que no es necesario volver a escribir código que ya ha sido implementado y probado.

Bibliografía:

Guia de reporte de practicas. (2023). Padlet. https://padlet.com/mtzcastillo2023/tema-3-metodos-7ykd9bx1lgv1xqwr/wish/2538860880

ING. INFORMATICA ENCUADRE POO. (2023). Padlet. https://padlet.com/mtzcastillo2023/ing-informatica-encuadre-poo-2nnwhmi63h3nizu7

KevinDCCsHeOs. (2023, June 2). *KevinDCCsHeOs/Tema4*. GitHub. https://github.com/KevinDCCsHeOs/Tema4

Programación ATS. (2023). [YouTube Video]. In *YouTube*. https://www.youtube.com/@ProgramacionATS