Informe Ejercicios Biblioteca Herencia Agregación y Composición JAVA

Planear Actividades De Construcción Del Software De Acuerdo Con El Diseño Establecido

Aprendiz:

Santiago Carranza Carrillo

Ficha:

2848530-A

Instructor:

Julio Roberto Galvis Cardozo

SENA

Centro De Diseño Y Metrología

9 De diciembre 2024

Índice

Introducción

En los códigos presentados, se implementa una serie de clases diseñadas para modelar un sistema básico de facturación, que incluye entidades como clientes, productos, comprobantes y fechas. Este sistema está construido siguiendo los principios de la programación orientada a objetos (POO), como la encapsulación, modularidad, reutilización y abstracción.

Estas clases están organizadas dentro del paquete clases y trabajan en conjunto para proporcionar una solución modular y escalable. A través de constructores, métodos *getter* y *setter*,

así como otras funcionalidades específicas, se logra un diseño flexible que puede integrarse en sistemas más amplios.

Agregación y herencia \ Venta

Código app

```
import Clases.Fecha;
import Clases.Producto;
import Clases.Cliente;
import Clases.Factura;

public class App {
    public static void main(String[] args) throws Exception {
        System.out.println("Hello, World!");
    }

    public class Main {
        public static void main(String[] args) {
            Fecha hoy = new Fecha(20, 10, 2011);
            Producto pro1 = new Producto(1, "Cafe", (float) 8.5);
            Producto pro2 = new Producto(2, "Media Luna", 2);
            Cliente cliente = new Cliente(1, "Juana");
            Factura f1 = new Factura('F', 1, hoy, cliente);
            f1.agregarProducto(pro1);
            f1.agregarProducto(pro2);
            f1.mostrar();
        }
    }

}
```

Explicación del código:

1. Importación de Clases

import Clases.Fecha;

import Clases.Producto;

import Clases.Cliente;

import Clases.Factura;

Estas líneas importan las clases necesarias para el funcionamiento del programa. Las clases Fecha, Producto, Cliente y Factura se encuentran dentro del paquete Clases. Esto significa que el proyecto está estructurado en paquetes para organizar mejor el código.

2. Clase Principal: App

```
public class App {
    public static void main(String[] args) throws Exception {
        System.out.println("Hello, World!");
    }
```

Esta clase contiene el punto de entrada principal de la aplicación. La instrucción:

```
System.out.println("Hello, World!");
```

3. Clase Interna: Main

Dentro de la clase App, hay otra clase llamada Main:

```
public class Main {
    public static void main(String[] args) {
        Fecha hoy = new Fecha(20, 10, 2011);
        Producto pro1 = new Producto(1, "Cafe", (float) 8.5);
        Producto pro2 = new Producto(2, "Media Luna", 2);
        Cliente cliente = new Cliente(1, "Juana");
        Factura f1 = new Factura('F', 1, hoy, cliente);
        f1.agregarProducto(pro1);
        f1.agregarProducto(pro2);
        f1.mostrar();
    }
}
```

a. Creación de un objeto Fecha

```
Fecha hoy = new Fecha(20, 10, 2011);
```

b. Creación de objetos Producto

```
Producto pro1 = new Producto(1, "Cafe", (float) 8.5);
Producto pro2 = new Producto(2, "Media Luna", 2);
```

c. Creación de un objeto Cliente

```
Cliente cliente = new Cliente(1, "Juana");
```

d. Creación de un objeto Factura

```
Factura f1 = new Factura('F', 1, hoy, cliente);
```

Se crea un objeto de la clase Factura con los siguientes datos:

Tipo de factura: 'F'.

Número de factura: 1.

Fecha asociada: el objeto hoy (20/10/2011).

Cliente asociado: el objeto cliente.

e. Agregando Productos a la Factura

f1.agregarProducto(pro1);

f1.agregarProducto(pro2);

Se agregan los productos pro1 y pro2 a la factura f1.

f. Mostrando la Factura

f1.mostrar();

Finalmente, se llama al método mostrar de la clase Factura para imprimir en la consola la información de la factura, incluyendo detalles como:

Tipo y número de factura.

Fecha.

Cliente.

Lista de productos y sus precios.

Código clase cliente

```
package clases;

public class Cliente {
    private int codigo;
    private String razonSocial;

    public Cliente() {
    }

    public int getCodigo() {
        return codigo;
    }

    public void setCodigo(int val) {
        this.codigo = val;
    }

    public String getRazonSocial() {
        return razonSocial;
    }
```

```
    public void setRazonSocial(String val) {
        this.razonSocial = val;
    }

    public Cliente(int c, String r) {
        setCodigo(c);
        setRazonSocial(r);
    }

}
```

Explicación del código:

1. Declaración de la Clase

```
package clases;


public class Cliente {
   private int codigo;
   private String razonSocial;
```

package clases: Indica que esta clase pertenece al paquete clases. Esto es útil para organizar el proyecto y evitar conflictos de nombres entre clases.

public class Cliente: Declara la clase Cliente como pública, lo que significa que puede ser utilizada en cualquier lugar del proyecto (si se importa correctamente).

private int codigo y private String razonSocial: Estos son los atributos privados de la clase:

codigo: Representa un identificador único para el cliente (un número entero).

razonSocial: Representa el nombre o razón social del cliente (una cadena de texto).

Al ser privados, estos atributos no son accesibles directamente desde fuera de la clase. Se accede a ellos mediante los métodos getter y setter.

2. Constructor por Defecto

```
public Cliente() {
}
```

Este es un constructor por defecto vacío. Sirve para crear un objeto Cliente sin inicializar sus atributos. Se utiliza en casos donde se quiere asignar valores más tarde mediante los métodos setter.

3. Métodos Getter y Setter

```
public int getCodigo() {
   return codigo;
```

```
}
public void setCodigo(int val) {
   this.codigo = val;
}
```

getCodigo y setCodigo: Métodos para acceder y modificar el atributo codigo:

getCodigo(): Devuelve el valor del atributo codigo.

setCodigo(int val): Establece el valor del atributo codigo con el valor proporcionado como parámetro (val).

java

Copiar código

```
public String getRazonSocial() {
   return razonSocial;
}

public void setRazonSocial(String val) {
   this.razonSocial = val;
}
```

getRazonSocial y setRazonSocial: Métodos para acceder y modificar el atributo razonSocial:

getRazonSocial(): Devuelve el valor del atributo razonSocial.

setRazonSocial(String val): Establece el valor del atributo razonSocial con el valor proporcionado como parámetro (val).

4. Constructor Sobrecargado

```
public Cliente(int c, String r) {
   setCodigo(c);
   setRazonSocial(r);
}
```

Este es un constructor sobrecargado que permite inicializar un objeto Cliente directamente con valores para sus atributos:

Parámetros:

int c: Código del cliente.

String r: Razón social del cliente.

El constructor utiliza los métodos setter (setCodigo y setRazonSocial) para asignar los valores recibidos a los atributos.

Código clase comprobante

```
package Clases;

public class Comprobante {

private char tipo;
private int numero;
private Fecha fecha;

public Comprobante (){
}

public Fecha getFecha (){
return fecha;
}

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

public int getNumero () {
return numero;
}
public void setNumero (int val) {

this.numero = val;
}
public char getTipo () {
return tipo;
}
public void setTipo (char var) {
this.tipo = var;
}

public Comprobante(char t, int n, Fecha f){
setTipo(t);
setNumero(n);
setFecha(f);
}

}
```

Explicación del código:

1. Declaración de la Clase

```
package Clases;


public class Comprobante {
```

package Clases: Indica que esta clase pertenece al paquete Clases. Esto permite organizar mejor el proyecto.

public class Comprobante: Declara la clase Comprobante como pública, lo que significa que puede ser utilizada desde cualquier otra clase si se importa correctamente.

2. Atributos Privados

```
private char tipo;

private int numero;

private Fecha fecha;
```

private char tipo: Representa el tipo del comprobante, como 'F' (factura) o 'R' (recibo).

private int numero: Representa el número único del comprobante.

private Fecha fecha: Representa la fecha del comprobante. Se usa un objeto de la clase Fecha, que debe estar definida en el mismo proyecto.

Los atributos son privados, lo que significa que solo se pueden acceder y modificar desde dentro de la clase. Esto es parte del concepto de encapsulación.

3. Constructor por Defecto

```
public Comprobante() {

}
```

Este es un constructor vacío, que no recibe parámetros. Se utiliza para crear un objeto Comprobante sin inicializar sus atributos. Los valores de los atributos se pueden asignar más tarde mediante los métodos setter.

4. Métodos Getter y Setter

Estos métodos permiten acceder y modificar los atributos privados de la clase.

Métodos para el atributo fecha

```
public Fecha getFecha() {
   return fecha;
}
```

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

getFecha: Devuelve el objeto Fecha asociado al comprobante.

setFecha(Fecha val): Asigna el objeto Fecha al atributo fecha.

Métodos para el atributo numero

```
public int getNumero() {
   return numero;
}

public void setNumero(int val) {
   this.numero = val;
}
```

getNumero: Devuelve el número del comprobante.

setNumero(int val): Asigna un valor al número del comprobante.

Métodos para el atributo tipo

```
public char getTipo() {
   return tipo;
}

public void setTipo(char var) {
   this.tipo = var;
}
```

getTipo: Devuelve el tipo del comprobante.

setTipo(char var): Asigna un valor al tipo del comprobante.

5. Constructor Sobrecargado

```
public Comprobante(char t, int n, Fecha f) {
   setTipo(t);
   setNumero(n);
   setFecha(f);
```

```
}
```

Este es un constructor sobrecargado que permite inicializar el objeto Comprobante directamente con valores para sus atributos:

Parámetros:

char t: Tipo del comprobante.

int n: Número del comprobante.

Fecha f: Fecha asociada al comprobante (un objeto de la clase Fecha).

Este constructor utiliza los métodos setter para asignar los valores proporcionados a los atributos.

Código clase factura

```java
package Clases;

import java.util.ArrayList;
import java.util.Iterator;

public class Factura extends Comprobante {

    private ArrayList<Producto> mProducto;
    private float total;
    private Cliente mCliente;

    public Factura() {
    }

    public Cliente getCliente() {
        return mCliente;

    }

    public void setCliente(Cliente val) {
        this.mCliente = val;
    }

    public float getTotal() {
        return total;
    }

    public void setTotal(float val) {
        this.total = val;
    }

    public ArrayList<Producto> getProducto() {
```

```
        return mProducto;
    }

    public void setProducto(ArrayList<Producto> val) {
        this.mProducto = val;
    }

    public Factura(char t, int n, Fecha f, Cliente cli) {
        super(t, n, f);
        setCliente(cli);
    }

    public void agregarProducto(Producto p) {
        mProducto.add(p);
        setTotal(getTotal() + p.getPrecio());
    }

public void mostrarProductos() {
Iterator<Producto> iter = mProducto.iterator();
while (iter.hasNext()) {
Producto p = iter.next();
System.out.printf("Codigo: %d Descripcion: %s Precio: %5.2f

\n",

p.getCodigo(), p.getDescripcion(), p.getPrecio());

}

}

    public void mostrar() {
        System.out.printf("Tipo: %c Número: %d Fecha: %d/%d/%d\n",
                getTipo(), getNumero(),
                getFecha().getDia(), getFecha().getMes(),

                getFecha().getAño());

        System.out.printf("Cliente: \n");
        System.out.printf("Codigo: %d Razon Social: %s \n",
                mCliente.getCodigo(), mCliente.getRazonSocial());
        System.out.printf("Productos: \n");

        mostrarProductos();
        System.out.printf("Total: %6.2f \n", getTotal());
```

```
    }
}
```

Explicación del código:

1. Declaración de la Clase

```
package Clases;


import java.util.ArrayList;

import java.util.Iterator;
```

public class Factura extends Comprobante {

package Clases: La clase pertenece al paquete Clases.

import java.util.ArrayList;: Importa la clase ArrayList, que se usa para manejar listas dinámicas.

import java.util.Iterator;: Importa la interfaz Iterator, que se usa para recorrer los elementos de una lista.

public class Factura extends Comprobante: Indica que Factura es una subclase de Comprobante. Hereda sus atributos y métodos, y añade funcionalidades específicas para las facturas.

2. Atributos Privados

```
private ArrayList<Producto> mProducto;

private float total;

private Cliente mCliente;
```

ArrayList<Producto> mProducto: Representa una lista dinámica de productos asociados a la factura.

float total: Almacena el total de la factura, calculado sumando los precios de los productos.

Cliente mCliente: Representa al cliente asociado a la factura.

Estos atributos son privados, por lo que solo se pueden acceder y modificar mediante los métodos definidos en la clase.

3. Constructores

a. Constructor por Defecto

```
public Factura() {

}
```

Es un constructor vacío, que no inicializa los atributos. Se puede usar si los valores se asignarán más tarde mediante los métodos setter.

b. Constructor Sobrecargado

```
public Factura(char t, int n, Fecha f, Cliente cli) {
```

```
    super(t, n, f);
    setCliente(cli);
}
```

Este constructor permite inicializar los parámetros:

char t: Tipo de comprobante (por ejemplo, 'F' para factura).

int n: Número de la factura.

Fecha f: Fecha de la factura.

Cliente cli: Cliente asociado a la factura.

super(t, n, f): Llama al constructor de la clase base Comprobante para inicializar el tipo, número y fecha.

setCliente(cli): Asigna el cliente al atributo mCliente.

4. Métodos Getter y Setter

Los métodos getter y setter proporcionan acceso y modificación a los atributos privados:

Para el Cliente:

```
public Cliente getCliente() {
    return mCliente;
}
public void setCliente(Cliente val) {
    this.mCliente = val;
}
```

Para el Total:

```
public float getTotal() {
    return total;
}
public void setTotal(float val) {
    this.total = val;
}
```

Para la Lista de Productos:

```
public ArrayList<Producto> getProducto() {
    return mProducto;
}
```

```
public void setProducto(ArrayList<Producto> val) {
   this.mProducto = val;
}
```

5. Métodos Específicos

a. Agregar Producto

```
public void agregarProducto(Producto p) {
   mProducto.add(p);
   setTotal(getTotal() + p.getPrecio());
}
```

mProducto.add(p): Añade un producto a la lista mProducto.

setTotal(getTotal() + p.getPrecio()): Actualiza el total de la factura sumando el precio del producto agregado.

b. Mostrar Productos

```
public void mostrarProductos() {
   Iterator<Producto> iter = mProducto.iterator();
   while (iter.hasNext()) {
      Producto p = iter.next();
      System.out.printf("Codigo: %d Descripcion: %s Precio: %5.2f\n",
         p.getCodigo(), p.getDescripcion(), p.getPrecio());
   }
}
```

Utiliza un Iterator para recorrer la lista de productos.

Por cada producto, imprime su código, descripción y precio.

c. Mostrar Factura Completa

```
public void mostrar() {
   System.out.printf("Tipo: %c Número: %d Fecha: %d/%d/%d\n",
         getTipo(), getNumero(),
         getFecha().getDia(), getFecha().getMes(),
         getFecha().getAño());

   System.out.printf("Cliente: \n");
```

```
    System.out.printf("Codigo: %d Razon Social: %s \n",
            mCliente.getCodigo(), mCliente.getRazonSocial());
    System.out.printf("Productos: \n");

    mostrarProductos();
    System.out.printf("Total: %6.2f \n", getTotal());
}
```

Este método muestra los detalles completos de la factura

Tipo, número y fecha (usando los métodos heredados de Comprobante).

Datos del cliente (usando los métodos de la clase Cliente).

Lista de productos (llama a mostrarProductos).

Total, de la factura.

Código clase fecha

```
package Clases;

public class Fecha {

    private int dia;
    private int mes;
    private int año;

    public Fecha() {

    }

    public int getAño() {
        return año;

    }

    public void setAño(int val) {
        this.año = val;
    }

    public int getDia() {
        return dia;
    }

    public void setDia(int val) {
```

```
        this.dia = val;
    }

    public int getMes() {
        return mes;
    }

    public void setMes(int val) {
        this.mes = val;

    }

    public Fecha(int d, int m, int a) {
        setDia(d);
        setMes(m);
        setAño(a);
    }

}
```

Explicación del código:

1. Declaración de la Clase

package Clases;

public class Fecha {

package Clases: Indica que esta clase pertenece al paquete Clases. Esto organiza las clases dentro del proyecto.

public class Fecha: Define la clase Fecha como pública, lo que significa que puede ser utilizada desde otras clases si se importa correctamente.

2. Atributos Privados

private int dia;

private int mes;

private int año;

int dia: Representa el día de la fecha.

int mes: Representa el mes de la fecha.

int año: Representa el año de la fecha.

Los atributos son privados, lo que significa que solo pueden ser accedidos o modificados mediante los métodos getter y setter. Esto es parte del concepto de encapsulación.

3. Constructores

a. Constructor por Defecto

```
public Fecha() {

}
```

Este es un constructor vacío que no inicializa los atributos. Se utiliza cuando deseas crear un objeto Fecha y asignar los valores más adelante mediante los métodos setter.

b. Constructor Sobrecargado

```
public Fecha(int d, int m, int a) {
   setDia(d);
   setMes(m);
   setAño(a);
}
```

Este es un constructor sobrecargado, que permite inicializar los atributos al momento de crear el objeto.

Parámetros:

int d: Día.

int m: Mes.

int a: Año.

Los valores proporcionados se asignan a los atributos mediante los métodos setter.

4. Métodos Getter y Setter

Estos métodos permiten acceder y modificar los atributos privados.

Para el Atributo año:

```
public int getAño() {
   return año;
}
public void setAño(int val) {
   this.año = val;
}
```

getAño: Devuelve el valor del atributo año.

setAño(int val): Asigna un valor al atributo año.

Para el Atributo dia:

```
public int getDia() {
```

```
    return dia;
}
public void setDia(int val) {
    this.dia = val;
}
```

getDia: Devuelve el valor del atributo dia.

setDia(int val): Asigna un valor al atributo dia.

Para el Atributo mes:

```
public int getMes() {
    return mes;
}
public void setMes(int val) {
    this.mes = val;
}
```

getMes: Devuelve el valor del atributo mes.

setMes(int val): Asigna un valor al atributo mes.

Código clase producto

```
package Clases;

public class Producto {

    private int codigo;
    private String descripcion;
    private float precio;

    public Producto() {

    }

    public int getCodigo() {

        return codigo;
    }

    public void setCodigo(int val) {
        this.codigo = val;
    }
```

```
    public String getDescripcion() {
        return descripcion;
    }

    public void setDescripcion(String val) {
        this.descripcion = val;
    }

    public float getPrecio() {
        return precio;
    }

    public void setPrecio(float val) {
        this.precio = val;
    }

    public Producto(int c, String d, float p) {
        setCodigo(c);
        setDescripcion(d);
        setPrecio(p);
    }

}
```

Explicación del código:

1. Declaración de la Clase

package Clases;

public class Producto {

package Clases: Indica que la clase pertenece al paquete Clases.

public class Producto: Define la clase Producto como pública, lo que significa que puede ser utilizada desde otras clases si se importa correctamente.

2. Atributos Privados

private int codigo;

private String descripcion;

private float precio;

int codigo: Representa un código numérico único para identificar el producto.

String descripcion: Almacena una descripción del producto, como su nombre o detalles relevantes.

float precio: Representa el precio del producto.

Los atributos son privados para protegerlos de accesos directos desde otras clases. Solo se pueden acceder o modificar a través de los métodos de la clase (encapsulación).

3. Constructores

a. Constructor por Defecto

```
public Producto() {

}
```

Este es un constructor vacío que no inicializa los atributos. Se usa cuando los valores se asignarán más adelante utilizando los métodos setter.

b. Constructor Sobrecargado

```
public Producto(int c, String d, float p) {
   setCodigo(c);
   setDescripcion(d);
   setPrecio(p);
}
```

Este constructor permite inicializar los atributos al momento de crear el objeto

Parámetros:

int c: Código del producto.

String d: Descripción del producto.

float p: Precio del producto.

Usa los métodos setter para asignar valores a los atributos, lo que asegura consistencia en caso de que exista lógica adicional en esos métodos.

4. Métodos Getter y Setter

Estos métodos permiten acceder y modificar los atributos privados.

Para el Atributo codigo:

```
public int getCodigo() {
   return codigo;
}
public void setCodigo(int val) {
   this.codigo = val;
}
```

getCodigo: Devuelve el valor del atributo codigo.

setCodigo(int val): Asigna un valor al atributo codigo.

Para el Atributo descripcion:

```
public String getDescripcion() {
   return descripcion;
}
public void setDescripcion(String val) {
   this.descripcion = val;
}
```

getDescripcion: Devuelve el valor del atributo descripcion.

setDescripcion(String val): Asigna un valor al atributo descripcion.

Para el Atributo precio:

```
public float getPrecio() {
   return precio;
}
public void setPrecio(float val) {
   this.precio = val;
}
```

getPrecio: Devuelve el valor del atributo precio.

setPrecio(float val): Asigna un valor al atributo precio.

Conclusión (Agregación y herencia)

La implementación presentada demuestra una solución clara y bien estructurada para manejar entidades fundamentales en un sistema de facturación. Cada clase está diseñada para ser reutilizable y extensible, lo que facilita su integración con otros módulos o sistemas. Este sistema constituye una base sólida para construir aplicaciones más complejas, como sistemas de inventarios, gestión de clientes o incluso plataformas de comercio electrónico. Con algunas mejoras adicionales, como validaciones más robustas, manejo de excepciones y una interfaz gráfica o de usuario, podría convertirse en una solución completa y funcional.

Composición y Biblioteca

Introducción

Los códigos proporcionados forman parte de un sistema orientado a objetos en Java diseñado para manejar entidades relacionadas con libros, personas y ubicaciones. Estas clases están organizadas para representar un modelo de datos que podría usarse, por ejemplo, en una aplicación de gestión de bibliotecas, préstamos o editoriales. A través de la herencia y la encapsulación, estas clases garantizan la reutilización de código y un diseño claro y modular. En conjunto, estos códigos demuestran un uso efectivo de conceptos como herencia, encapsulación, y composición, lo que resulta en una estructura bien diseñada para un sistema de gestión.

Código cargar libro

```java
import java.util.Scanner;

public class CargarLibro {

    public static void main(String[] args) {
        CargarLibro cargador = new CargarLibro();

        Libro libro = cargador.cargarLibro();
        cargador.imprimirLibro(libro);
    }

    public Libro cargarLibro() {
        Scanner scanner = new Scanner(System.in);

        System.out.println("Ingrese el código del libro:");
        int codigo = scanner.nextInt();
        scanner.nextLine(); // Limpiar el buffer

        System.out.println("Ingrese el nombre del libro:");
        String nombre = scanner.nextLine();

        System.out.println("Ingrese la edición del libro:");
        int edicion = scanner.nextInt();

        System.out.println("Ingrese el año de publicación del libro:");
        int anioPublicacion = scanner.nextInt();
```

```
        return new Libro(codigo, nombre, edicion, añoPublicacion);
    }

    public void imprimirLibro(Libro libro) {
        System.out.println("Detalles del libro:");
        System.out.println("Código: " + libro.getCodigo());

        System.out.println("Nombre: " + libro.getNombre());
        System.out.println("Edición: " + libro.getEdicion());
        System.out.println("Año de publicación: " +

                libro.getAñoPublicacion());

        System.out.println("Stock: " + libro.getStock());
    }
}
```

Explicación del código:

1. Definición de la clase CargarLibro

```
public class CargarLibro {
```

Esta es la clase principal llamada CargarLibro, que contiene los métodos para cargar la información del libro y luego imprimirla.

2. Método main

```
public static void main(String[] args) {
   CargarLibro cargador = new CargarLibro();
   Libro libro = cargador.cargarLibro();
   cargador.imprimirLibro(libro);
}
```

El método main es el punto de entrada del programa. Aquí se crea una instancia de la clase CargarLibro, llamada cargador, que luego se utiliza para llamar a dos métodos:

cargarLibro(): Se utiliza para cargar los datos del libro desde el usuario.

imprimirLibro(libro): Se utiliza para imprimir la información del libro en la consola.

3. Método cargarLibro

```
public Libro cargarLibro() {
   Scanner scanner = new Scanner(System.in);
   System.out.println("Ingrese el código del libro:");
   int codigo = scanner.nextInt();
```

```
    scanner.nextLine(); // Limpiar el buffer

    System.out.println("Ingrese el nombre del libro:");
    String nombre = scanner.nextLine();

    System.out.println("Ingrese la edición del libro:");
    int edicion = scanner.nextInt();

    System.out.println("Ingrese el año de publicación del libro:");
    int anioPublicacion = scanner.nextInt();

    return new Libro(codigo, nombre, edicion, anioPublicacion);
}
```

Este método solicita al usuario que ingrese varios datos sobre el libro

Código: Un número que representa un identificador único del libro.

Nombre: El título del libro.

Edición: El número de la edición del libro.

Año de publicación: El año en que el libro fue publicado.

Después de solicitar cada dato, el método crea y devuelve un objeto de la clase Libro con los datos ingresados.

4. Método imprimirLibro

```
public void imprimirLibro(Libro libro) {
    System.out.println("Detalles del libro:");
    System.out.println("Código: " + libro.getCodigo());
    System.out.println("Nombre: " + libro.getNombre());
    System.out.println("Edición: " + libro.getEdicion());
    System.out.println("Año de publicación: " + libro.getAñoPublicacion());
    System.out.println("Stock: " + libro.getStock());
}
```

Este método recibe un objeto de tipo Libro como argumento y luego imprime los detalles de ese libro:

Código: Llamando al método getCodigo() del objeto Libro.

Nombre: Llamando a getNombre().

Edición: Llamando a getEdicion().

Año de publicación: Llamando a getAñoPublicacion().

Stock: Llamando a getStock() (aunque este valor no se ingresa en el código, se asume que la clase Libro tiene un atributo stock con un método getStock()).

Código ciudad

```
public class Ciudad extends General {
    private String nombre;
    private Pais pais;

    public Ciudad() {

    }

    public Ciudad(int id, String nombre, Pais pais) {
        this.nombre = nombre;
        this.pais = pais;
    }

    public String getNombre() {
        return nombre;
    }

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

    public Pais getPais() {
        return pais;
    }

    public void setPais(Pais pais) {
        this.pais = pais;
    }
}
```

Explicación del código:

1. Definición de la clase Ciudad

public class Ciudad extends General {

La clase Ciudad extiende a la clase General, lo que significa que hereda los atributos y métodos de General, pero también puede tener sus propios atributos y métodos.

2. Atributos de la clase Ciudad

```
private String nombre;

private Pais pais;
```

La clase Ciudad tiene dos atributos:

nombre: Una cadena de texto (String) que representará el nombre de la ciudad.

pais: Un objeto de tipo Pais que representa el país al que pertenece la ciudad. Aquí, Pais debe ser otra clase definida en el programa.

3. Constructores de la clase Ciudad

```
public Ciudad() {

}
```

Este es el constructor vacío de la clase Ciudad. No hace nada en particular, solo permite crear un objeto Ciudad sin especificar parámetros.

```
public Ciudad(int id, String nombre, Pais pais) {
   this.nombre = nombre;
   this.pais = pais;
}
```

Este es otro constructor que toma tres parámetros:

id: Un identificador que podría usarse para identificar de manera única una ciudad. Aunque no se usa en el código proporcionado, probablemente está relacionado con la clase General (ya que esta clase es la que probablemente tiene el atributo id, dado que la clase Ciudad lo hereda).

nombre: El nombre de la ciudad.

pais: El objeto Pais al que pertenece la ciudad.

Dentro de este constructor, los valores de nombre y pais se asignan a los atributos correspondientes de la clase Ciudad.

4. Métodos Getter y Setter

Getter para nombre:

```
public String getNombre() {
   return nombre;
}
```

Este método devuelve el valor del atributo nombre de la ciudad.

Setter para nombre:

```
public void setNombre(String nombre) {
```

```
    this.nombre = nombre;
}
```

Este método establece el valor del atributo nombre de la ciudad.

Getter para pais:

```
public Pais getPais() {
    return pais;
}
```

Este método devuelve el objeto Pais asociado con la ciudad.

Setter para pais:

```
public void setPais(Pais pais) {
    this.pais = pais;
}
```

Este método establece el valor del atributo pais de la ciudad, permitiendo modificar a qué país pertenece la ciudad.

Código editorial

```java
public class Editorial extends General {
    private String nombre;
    private Ciudad ciudad;

    public Editorial() {

    }

    public Editorial(int id, String nombre, Ciudad ciudad) {
        this.nombre = nombre;
        this.ciudad = ciudad;
    }

    public String getNombre() {
        return nombre;
    }

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

    public Ciudad getCiudad() {
```

```
        return ciudad;
    }

    public void setCiudad(Ciudad ciudad) {

        this.ciudad = ciudad;
    }
}
```

Explicación del código:

1. Definición de la clase Editorial

public class Editorial extends General {

2. Atributos de la clase Editorial

private String nombre;

private Ciudad ciudad;

La clase Editorial define dos atributos:

nombre: Es un texto (String) que almacena el nombre de la editorial.

ciudad: Es un objeto de tipo Ciudad que representa la ciudad en la que se encuentra la editorial.

Estos atributos son privados, lo que significa que solo pueden ser accedidos o modificados desde dentro de la clase Editorial mediante métodos getter y setter.

3. Constructores

La clase define dos constructores:

public Editorial() {

}

Este es un constructor por defecto que no hace nada y permite crear un objeto de tipo Editorial sin inicializar sus atributos.

public Editorial(int id, String nombre, Ciudad ciudad) {

this.nombre = nombre;

this.ciudad = ciudad;

}

Este constructor permite inicializar los atributos de la clase al momento de crear un objeto.

nombre: Se inicializa con el valor pasado como parámetro.

ciudad: Se inicializa con el objeto Ciudad que se pasa como parámetro.

4. Métodos Getter y Setter

Getter y Setter para nombre

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

getNombre: Devuelve el valor del atributo nombre.

setNombre: Permite modificar el valor del atributo nombre.

Getter y Setter para ciudad

```
public Ciudad getCiudad() {
   return ciudad;
}
public void setCiudad(Ciudad ciudad) {
   this.ciudad = ciudad;
}
```

getCiudad: Devuelve el objeto Ciudad asociado a la editorial.

setCiudad: Permite establecer o cambiar la ciudad en la que se encuentra la editorial.

Código general

```java
public class General {
    private int id;
    private String nombre;

    public General() {
    }

    public General(int id, String nombre) {
        this.id = id;
        this.nombre = nombre;
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
```

```
    }

    public String getNombre() {
        return nombre;

    }

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

}
```

Explicación del código:

1. Definición de la clase General

public class General {

2. Atributos

private int id;

private String nombre;

La clase tiene dos atributos privados:

id: Un número entero que representa un identificador único para cada objeto de esta clase.

nombre: Una cadena de texto (String) que contiene el nombre asociado al objeto.

3. Constructores

Constructor vacío

public General() {

}

Este constructor vacío permite crear un objeto General sin inicializar los valores de sus atributos.

Constructor con parámetros

public General(int id, String nombre) {

this.id = id;

this.nombre = nombre;

}

Este constructor permite inicializar los atributos id y nombre al momento de crear un objeto General.

id: Se asigna al atributo id de la instancia utilizando this.id.

nombre: Se asigna al atributo nombre de la instancia utilizando this.nombre.

4. Métodos Getter y Setter

Getter y Setter para id

```
public int getId() {
   return id;
}
public void setId(int id) {
   this.id = id;
}
```

getId: Devuelve el valor actual del atributo id.

setId: Permite establecer o cambiar el valor del atributo id.

Getter para nombre

```
public String getNombre() {
   return nombre;
}
```

Este método devuelve el valor actual del atributo nombre.

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

Código libro

```java
public class Libro {
    private int edicion;
    private int añoPublicacion;

    public Libro() {

    }

    // contructor
    public Libro(int codigo, int edicion, int añoPublicacion) {
        this.edicion = edicion;
        this.añoPublicacion = añoPublicacion;
    }

    public int getStock() {
        return 10;
    }
```

```
    public int getEdicion() {
        return edicion;
    }

    public void setEdicion(int edicion) {
        this.edicion = edicion;
    }

    public int getAñoPublicacion() {
        return añoPublicacion;
    }

    public void setAñopublicacion(int añoPublicacion) {
        this.añoPublicacion = añoPublicacion;
    }
}
```

Explicación del código:

1. Definición de la clase Libro

public class Libro {

2. Atributos

private int edicion;

private int añoPublicacion;

edicion: Es un entero que representa la edición del libro (por ejemplo, primera, segunda, tercera edición, etc.).

añoPublicacion: Es un entero que representa el año en que se publicó el libro.

3. Constructores

public Libro() {

}

Es un constructor por defecto que no hace nada. Permite crear un objeto Libro sin inicializar sus atributos.

public Libro(int codigo, int edicion, int añoPublicacion) {

this.edicion = edicion;

this.añoPublicacion = añoPublicacion;

}

Este constructor permite crear un objeto Libro e inicializar directamente los atributos edicion y añoPublicacion.

El constructor asigna:

edicion: Al atributo correspondiente utilizando this.edicion.

añoPublicacion: Al atributo correspondiente utilizando this.añoPublicacion.

4. Métodos Getter y Setter

Getter para edicion

```
public int getEdicion() {
   return edicion;
}
```

Este método devuelve el valor actual del atributo edicion.

Setter para edicion

```
public void setEdicion(int edicion) {
   this.edicion = edicion;
}
```

Este método permite modificar el valor del atributo edicion.

Getter para añoPublicacion

```
public int getAñoPublicacion() {
   return añoPublicacion;
}
```

Este método devuelve el valor actual del atributo añoPublicacion.

Setter para añoPublicacion

```
public void setAñopublicacion(int añoPublicacion) {
   this.añoPublicacion = añoPublicacion;
}
```

Este método permite modificar el valor del atributo añoPublicacion.

Método getStock

```
public int getStock() {
   return 10;
}
```

Este método devuelve un valor fijo, 10.

Código país

```java
public class Pais extends General {
    private String nombre;

    public Pais() {
    }

    public Pais(int id, String nombre) {
        this.nombre = nombre;
    }

    public String getNombre() {
        return nombre;
    }

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

Explicación del código:

1. Definición de la clase Pais

public class Pais extends General {

Pais: Es una clase pública que representa un país con un atributo adicional, nombre.

extends General: Indica que Pais hereda de la clase General. Esto significa que Pais tiene acceso a los atributos y métodos definidos en General (como id y sus métodos getter y setter).

2. Atributo

private String nombre;

Este atributo privado almacena el nombre del país (por ejemplo, "Argentina", "Brasil", etc.).

3. Constructores

Constructor vacío

public Pais() {

}

Este constructor vacío permite crear un objeto Pais sin inicializar sus atributos.

Constructor con parámetros

public Pais(int id, String nombre) {

this.nombre = nombre;

}

Este constructor permite crear un objeto Pais e inicializar el atributo nombre directamente.

```
public Pais(int id, String nombre) {
   super(id, nombre);
   this.nombre = nombre;
}
```

4. Métodos Getter y Setter

Getter para nombre

```
public String getNombre() {
   return nombre;
}
```

Devuelve el valor del atributo nombre.

Setter para nombre

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

Permite modificar el valor del atributo nombre.

Código prestatario

```
public class Prestatario extends General {
    private String nombre;
    private String apellido;
    private String direccion;
    private Ciudad ciudad;
    private String telefono;

    public Prestatario() {

    }

    public Prestatario(int id, String nombre, String apellido, String

    direccion, Ciudad ciudad, String telefono) {

    this.nombre = nombre;
    this.apellido = apellido;

    this.direccion = direccion;
    this.ciudad = ciudad;
```

```
    this.telefono = telefono;
    }

    // Getters y setters
    public String getNombre() {
    return nombre;
    }

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

    public String getApellido() {
    return apellido;
    }

    public void setApellido(String apellido) {
    this.apellido = apellido;
    }

    public String getDireccion() {
    return direccion;
    }

    public void setDireccion(String direccion) {
    this.direccion = direccion;
    }

    public Ciudad getCiudad() {
    return ciudad;
    }

    public void setCiudad(Ciudad ciudad) {
    this.ciudad = ciudad;
    }

    public String getTelefono() {
    return telefono;
    }

    public void setTelefono(String telefono) {
    this.telefono = telefono;
    }
    }
```

Explicación del código:

1. Definición de la clase Prestatario

```
public class Prestatario extends General {
```

La clase Prestatario es pública, por lo que puede ser utilizada en cualquier parte del programa.

2. Atributos

```
private String nombre;

private String apellido;

private String direccion;

private Ciudad ciudad;

private String telefono;
```

Todos los atributos son privados (private), lo que significa que no pueden ser accedidos directamente desde fuera de la clase. Para interactuar con ellos, se usan métodos getter y setter.

3. Constructores

Constructor vacío

```
public Prestatario() {

}
```

Este es un constructor vacío que no inicializa ningún atributo. Permite crear un objeto de tipo Prestatario y asignar valores a sus atributos más adelante.

Constructor con parámetros

```
public Prestatario(int id, String nombre, String apellido, String direccion, Ciudad ciudad, String telefono) {
    this.nombre = nombre;
    this.apellido = apellido;
    this.direccion = direccion;
    this.ciudad = ciudad;
    this.telefono = telefono;
}
```

Este constructor permite crear un objeto de tipo Prestatario e inicializar sus atributos al momento de la creación.

4. Métodos Getter y Setter

Los métodos getter y setter permiten acceder y modificar los atributos privados de la clase.

Getter y Setter para nombre

```
public String getNombre() {
```

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

getNombre: Devuelve el valor actual del atributo nombre.

setNombre: Permite modificar el valor de nombre.

Getter y Setter para apellido

```
public String getApellido() {
    return apellido;
}
public void setApellido(String apellido) {
    this.apellido = apellido;
}
```

getApellido: Devuelve el valor actual del atributo apellido.

setApellido: Permite modificar el valor de apellido.

Getter y Setter para direccion

```
public String getDireccion() {
    return direccion;
}
public void setDireccion(String direccion) {
    this.direccion = direccion;
}
```

getDireccion: Devuelve el valor actual del atributo direccion.

setDireccion: Permite modificar el valor de direccion.

Getter y Setter para ciudad

```
public Ciudad getCiudad() {
    return ciudad;
}
public void setCiudad(Ciudad ciudad) {
```

```
    this.ciudad = ciudad;
}
```

getCiudad: Devuelve el objeto Ciudad asociado al prestatario.

setCiudad: Permite asociar un nuevo objeto Ciudad al prestatario.

Getter y Setter para telefono

```
public String getTelefono() {
    return telefono;
}
public void setTelefono(String telefono) {
    this.telefono = telefono;
}
```

getTelefono: Devuelve el número de teléfono del prestatario.

setTelefono: Permite modificar el número de teléfono del prestatario.

Conclusión (Composición y Biblioteca)

La implementación de estas clases proporciona una base sólida para desarrollar un sistema funcional y escalable, como una biblioteca digital o un sistema de préstamos. La combinación de relaciones jerárquicas entre clases y el uso de constructores y métodos getter/setter permite manejar datos de manera eficiente y coherente.

Cada clase desempeña un rol claro dentro del sistema:

General actúa como la clase base, ofreciendo un diseño genérico para las clases derivadas.

Pais, Ciudad, y Editorial modelan información geográfica y organizacional.

Libro y Prestatario son las entidades centrales que interactúan en un posible flujo de préstamo de libros.

Con este diseño modular y orientado a objetos, el sistema no solo es fácil de entender y mantener, sino que también se puede extender fácilmente para añadir nuevas funcionalidades o integrarse con otros sistemas. Esto destaca la importancia de seguir principios sólidos de diseño al desarrollar software.