Servicio Nacional de Aprendizaje – SENA

Centro de Diseño y Metrologia

Informe ejercicios biblioteca herencia agregacion y composicion java

Aprendiz:

Edwin Camilo Lozano Chaparro

No Ficha 2848530-A

Introducción

En el desarrollo de sistemas de gestión, como los relacionados con bibliotecas, préstamos, o inventarios, la organización y el manejo eficiente de las entidades involucradas son fundamentales. En este contexto, el presente informe presenta el desarrollo de varias clases en Java que modelan un sistema de préstamo de libros, donde se gestionan conceptos clave como libros, prestatarios, ciudades, países y editoriales. Estas clases están diseñadas para facilitar el almacenamiento de datos, la relación entre diferentes entidades y la manipulación de objetos mediante la utilización de principios fundamentales de la programación orientada a objetos (POO), como la herencia, los constructores, los getters y setters, y la encapsulación.

El sistema modelado se compone de una serie de clases interrelacionadas, entre las que se incluyen Libro, Prestatario, Pais, Ciudad, y Editorial. Cada clase está estructurada para almacenar información específica, y los objetos de cada una de estas clases interactúan entre sí de manera que se pueda representar un sistema completo de gestión de libros y su préstamo.

Contenido

Agregación y herencia \ Venta

Código app

```java
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();
        }
        }
```

```
}
```

Este programa en **Java** presenta una estructura básica para manejar una aplicación de facturación. Utiliza varias clases relacionadas con un sistema de facturación para crear y mostrar una factura que contiene información sobre productos, el cliente y la fecha de emisión. Aquí hay un desglose detallado:

1. Importación de Clases:

- Estas líneas importan las clases necesarias para trabajar con fechas (Fecha), productos (Producto), clientes (Cliente) y facturas (Factura). Se asume que estas clases están definidas en un paquete llamado Clases.

2. Clase App y su método main:

- Esta es la clase principal que contiene un método main que imprime "Hello, World!". Aunque no es relevante para el resto del programa, parece ser un ejemplo inicial o una prueba.

3. Clase Main y la lógica principal:

Desglose de las operaciones:

1. **Creación de la fecha**:
    - Se instancia un objeto de la clase Fecha que representa el 20 de octubre de 2011.
2. **Creación de productos**:

- Se crean dos objetos Producto, cada uno con un identificador, un nombre y un precio. El precio del primer producto se convierte explícitamente a float.

3. **Creación del cliente**:
   - Se instancia un cliente con un identificador y un nombre.
4. **Creación de la factura**:
   - Se crea una factura f1 con tipo 'F', número 1, la fecha actual (hoy) y el cliente asociado.
5. **Agregar productos a la factura**:
   - Los productos creados (pro1 y pro2) se agregan a la factura mediante el método agregarProducto.
6. **Mostrar la factura**:
   - Se llama al método mostrar de la factura, lo que probablemente imprime los detalles de la factura, como la fecha, los datos del cliente, los productos y el total.

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);
                }
}
```

1. Descripción general de la clase Cliente:

La clase Cliente utiliza el paradigma de **programación orientada a objetos (POO)** y define:

- **Atributos privados**:
    - codigo: Almacena el código único del cliente (probablemente un identificador numérico).
    - razonSocial: Almacena el nombre o razón social del cliente (texto).
- **Métodos get y set**:
    - Se usan para acceder y modificar los valores de los atributos privados, siguiendo el principio de **encapsulamiento**.
- **Constructores**:
    - Un constructor vacío (Cliente()) para instanciar un cliente sin inicializar valores.
    - Un constructor con parámetros (Cliente(int c, String r)) que permite inicializar un cliente directamente con un código y una razón social.

2. Desglose del código:

- Estos atributos solo son accesibles mediante los métodos de la clase (get y set), garantizando el encapsulamiento.
- Este constructor permite crear un objeto de tipo Cliente sin inicializar valores. Es útil para escenarios en los que los atributos serán asignados más tarde.

Métodos get y set:

Los métodos get y set permiten acceder y modificar los valores de los atributos privados.

- **getCodigo() y setCodigo(int val)**:
    - getCodigo() devuelve el valor del atributo codigo.
    - setCodigo(int val) asigna un nuevo valor al atributo codigo.
- **getRazonSocial() y setRazonSocial(String val)**:
    - getRazonSocial() devuelve el valor de razonSocial.
    - setRazonSocial(String val) asigna un nuevo valor al atributo razonSocial.
- Este constructor inicializa un objeto de tipo Cliente asignando valores al código (c) y a la razón social (r).
- Usa los métodos setCodigo y setRazonSocial para asignar valores, lo que asegura consistencia si en el futuro se añaden reglas de validación.

3. Principios de diseño aplicados:

- **Encapsulamiento**:
    - Los atributos son privados y accesibles solo mediante los métodos get y set.
- **Reusabilidad**:
    - El constructor parametrizado permite crear objetos con datos inicializados en una sola línea.
- **Flexibilidad**:
    - El constructor vacío facilita la creación de objetos en escenarios donde los valores serán asignados dinámicamente.

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);
            }
}
```

1. Descripción general de la clase Comprobante:

La clase Comprobante tiene los siguientes elementos clave:

- **Atributos privados**:
    - tipo: Un carácter (char) que puede representar el tipo de comprobante (por ejemplo, 'F' para factura, 'R' para recibo, etc.).
    - numero: Un número entero (int) que identifica de manera única al comprobante.

- fecha: Un objeto de tipo Fecha, que almacena la fecha asociada al comprobante.

- **Métodos get y set**:
  - Son utilizados para acceder y modificar los valores de los atributos privados.

- **Constructor**:
  - Un constructor vacío (Comprobante()) que permite crear un objeto sin inicializar los valores.
  - Un constructor con parámetros (Comprobante(char t, int n, Fecha f)) que inicializa un comprobante con tipo, número y fecha.

2. Desglose del código:

Atributos privados:

- **tipo**: Se utiliza un char para almacenar el tipo de comprobante (por ejemplo, 'F' o 'R').
- **numero**: Se utiliza un int para almacenar el número único de identificación del comprobante.
- **fecha**: Un objeto de la clase Fecha que se utiliza para almacenar la fecha de emisión del comprobante.

Métodos get y set:

- **Método get y set para tipo**:

- getTipo() devuelve el valor de tipo.
- setTipo(char var) establece el valor de tipo.

- **Método get y set para numero**:
  - getNumero() devuelve el valor de numero.
  - setNumero(int val) establece el valor de numero.

- **Método get y set para fecha**:
  - getFecha() devuelve el objeto Fecha que representa la fecha del comprobante.
  - setFecha(Fecha val) establece la fecha del comprobante.
  - Este constructor permite crear un objeto Comprobante sin especificar valores iniciales para los atributos.

- **Constructor con parámetros**:
  - Este constructor inicializa un objeto Comprobante con el tipo (t), número (n) y fecha (f).
  - Utiliza los métodos setTipo, setNumero y setFecha para asignar los valores a los atributos.

3. Principios de diseño aplicados:

- **Encapsulamiento**: Los atributos de la clase son privados y solo se pueden acceder/modificar a través de métodos get y set, lo que garantiza que el objeto mantenga su integridad y que los datos sean manipulados de manera controlada.

- **Reusabilidad**: La clase está diseñada para ser flexible. Se pueden crear objetos Comprobante tanto con valores predeterminados (usando el constructor vacío) como con valores específicos (usando el constructor con parámetros).

- **Modularidad**: Al depender de la clase Fecha para almacenar la fecha, el código es modular y reutilizable. Si en el futuro se desea cambiar la implementación de la clase Fecha, no es necesario modificar el código de la clase Comprobante.

Código clase factura

```
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());
        }
    }
```

1. Descripción general de la clase Factura:

La clase Factura tiene los siguientes atributos y métodos:

Atributos privados:

- **mProducto**: Un ArrayList<Producto> que almacena los productos que están asociados a la factura.
- **total**: Un float que representa el total calculado de la factura (suma de los precios de los productos).
- **mCliente**: Un objeto de la clase Cliente, que representa al cliente que recibe la factura.

Métodos get y set:

- **getCliente()** y **setCliente()**: Para obtener y establecer el cliente asociado a la factura.
- **getTotal()** y **setTotal()**: Para obtener y establecer el total de la factura.
- **getProducto()** y **setProducto()**: Para obtener y establecer la lista de productos asociados a la factura.

Constructores:

- **Constructor vacío**: Permite crear una factura sin inicializar ningún valor.

- **Constructor con parámetros**: Recibe el tipo (t), número (n), fecha (f) y cliente (cli) de la factura, y llama al constructor de la clase base Comprobante para inicializar el tipo, número y fecha. Además, asigna el cliente a la factura.

Métodos adicionales:

- **agregarProducto(Producto p)**: Agrega un producto a la lista mProducto y actualiza el total de la factura sumando el precio del producto.

- **mostrarProductos()**: Muestra en la consola los productos asociados a la factura, mostrando el código, la descripción y el precio de cada uno.

- **mostrar()**: Muestra toda la información de la factura, incluyendo tipo, número, fecha, cliente, productos y total.

2. Desglose del código:

Atributos privados:

- **mProducto**: Es una lista que contiene los productos de la factura, utilizando la clase ArrayList. Cada Producto tiene detalles como el código, la descripción y el precio.

- **total**: Es un valor de tipo float que representa el total de la factura, calculado como la suma de los precios de los productos.

- **mCliente**: Un objeto de la clase Cliente que contiene la información del cliente al que se le emite la factura.

Métodos get y set:

Los métodos get y set permiten acceder y modificar los atributos privados de la clase.

- Los métodos get y set de Cliente permiten acceder y asignar el cliente de la factura.

Constructor con parámetros:

- Este constructor inicializa la factura con un tipo, número y fecha usando el constructor de la clase base Comprobante (que es llamada con super(t, n, f)) y asigna el cliente (cli) usando el método setCliente.

Método agregarProducto:

- Este método agrega un producto a la lista mProducto y actualiza el total de la factura sumando el precio del producto.

Método mostrarProductos:

- Muestra los detalles de cada producto en la factura: código, descripción y precio, utilizando un Iterator para recorrer la lista mProducto.

Método mostrar:

- Este método imprime toda la información de la factura en la consola: tipo, número, fecha, información del cliente, productos asociados y el total de la factura.

3. Principios de diseño aplicados:

- **Herencia**: La clase Factura hereda de Comprobante, lo que permite reutilizar el código de la clase base y extenderlo con funcionalidades específicas de la factura, como la lista de productos y el total.

- **Encapsulamiento**: Los atributos de la clase están encapsulados (es decir, son privados) y solo se pueden modificar o acceder a través de los métodos get y set. Esto asegura el control sobre cómo se accede y se modifica la información de la factura.

- **Colecciones**: Se utiliza un ArrayList para almacenar los productos asociados a la factura. Esto permite manejar dinámicamente la cantidad de productos sin necesidad de manejar índices manualmente.

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);
    }
```

```
}
```

1. Atributos privados:

- **dia**: Un entero que almacena el día de la fecha.
- **mes**: Un entero que almacena el mes de la fecha.
- **año**: Un entero que almacena el año de la fecha.

2. Métodos get y set:

La clase tiene métodos **get** y **set** para cada uno de los atributos. Estos métodos permiten acceder y modificar el valor de los atributos de la clase:

Métodos get y set de los atributos:

- **getDia() y setDia(int val)**: Devuelven y asignan el valor del atributo dia.
- **getMes() y setMes(int val)**: Devuelven y asignan el valor del atributo mes.
- **getAño() y setAño(int val)**: Devuelven y asignan el valor del atributo año.

Estos métodos permiten acceder a los valores de día, mes y año y establecer nuevos valores de manera controlada.

3. Constructor:

La clase también tiene un constructor que permite inicializar los atributos de la fecha en el momento de la creación del objeto Fecha.

- Este constructor recibe tres parámetros: día, mes y año, y establece estos valores mediante los métodos set correspondientes.

4. Constructor vacío:

- El constructor vacío se utiliza para crear una instancia de la clase sin asignar un valor inicial a los atributos dia, mes y año. Posteriormente, los valores pueden ser establecidos usando los métodos set.

5. Uso de la clase Fecha:

- En este caso, se crea un objeto de la clase Fecha con el día 20, mes 10 (octubre) y año 2023.

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);
                }


    }
```

1. Atributos privados:

- **codigo**: Un entero que almacena el código único del producto, utilizado para identificarlo de manera exclusiva.
- **descripcion**: Una cadena de texto (String) que almacena una breve descripción del producto.
- **precio**: Un número decimal (float) que representa el precio del producto.

2. Métodos get y set:

La clase tiene métodos **get** y **set** para cada uno de los atributos. Estos métodos permiten acceder y modificar los valores de los atributos de la clase de forma controlada:

Métodos get y set de los atributos:

- **getCodigo() y setCodigo(int val)**: Devuelven y asignan el valor del atributo codigo.
- **getDescripcion() y setDescripcion(String val)**: Devuelven y asignan el valor del atributo descripcion.
- **getPrecio() y setPrecio(float val)**: Devuelven y asignan el valor del atributo precio.

Estos métodos proporcionan acceso y permiten la modificación segura de los atributos.

3. Constructor:

La clase también tiene un constructor que permite inicializar los atributos de un producto al momento de crear el objeto Producto.

- Este constructor recibe tres parámetros: código (c), descripción (d) y precio (p).

A través de los métodos set, se asignan estos valores a los atributos correspondientes.

4. Constructor vacío:

- El constructor vacío es útil para crear un objeto Producto sin asignar valores iniciales. Posteriormente, los valores pueden ser establecidos usando los métodos set.

5. Uso de la clase Producto:

- En este caso, se crea un objeto Producto con el código 101, la descripción "Café" y el precio 8.5 utilizando el constructor con parámetros.

Composición \Biblioteca

Código CargarLibro

```
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());
        }
    }
```

1. Clase principal CargarLibro

Esta clase tiene dos funciones principales:

1. Cargar los datos de un libro ingresados por el usuario.
2. Imprimir en la consola los detalles del libro.

Métodos principales

a) Método main

1. **Objetivo**:
    - Ejecutar el flujo principal del programa.
    - Crear una instancia de CargarLibro.
    - Llamar al método cargarLibro para obtener los datos ingresados por el usuario.
    - Llamar a imprimirLibro para mostrar los detalles del libro.

b) Método cargarLibro

1. **Propósito**:
    - Recopilar datos del libro ingresados por el usuario mediante la consola.
2. **Uso de Scanner**:
    - Se utiliza para leer los datos introducidos por el usuario.
    - Se limpia el buffer con scanner.nextLine() después de leer un entero para evitar problemas con la lectura de cadenas.
3. **Datos solicitados**:
    - **Código del libro** (int): Un identificador único.
    - **Nombre del libro** (String): El título del libro.
    - **Edición del libro** (int): Número de la edición.
    - **Año de publicación** (int): Año en que fue publicado.
4. **Retorno**:
    - Crea y retorna un objeto de tipo Libro utilizando un constructor.

c) Método imprimirLibro

1. **Propósito**:
    - Mostrar los detalles del libro recibido como argumento.
2. **Uso de métodos get**:
    - Se acceden a los atributos del libro mediante los métodos get definidos en la clase Libro.

2. Clase auxiliar Libro

Se asume que la clase Libro contiene atributos y métodos que permiten encapsular los datos de un libro.

Atributos sugeridos para la clase Libro:

- **int codigo**: Código único del libro.
- **String nombre**: Nombre del libro.
- **int edicion**: Número de edición.
- **int añoPublicacion**: Año de publicación.
- **int stock**: (opcional) Cantidad disponible del libro.

Métodos sugeridos en Libro:

- **Métodos get** para cada atributo, como getCodigo(), getNombre(), etc.

Código ciudad

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

    public Ciudad() {

    }
```

```java
        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 de la clase Ciudad

La clase Ciudad representa una entidad que contiene información sobre una ciudad, su nombre y el país al que pertenece. Extiende de una clase base llamada General, lo que permite reutilizar

funcionalidad común a múltiples entidades (aunque no se proporciona el código de General, se deduce que es una clase generalizada).

Componentes principales

1. **Atributos**:
    - String nombre: Almacena el nombre de la ciudad.
    - Pais pais: Es una referencia a un objeto de la clase Pais, que indica el país al que pertenece la ciudad.
2. **Constructores**:
    - **Constructor vacío (Ciudad())**:
        - Proporciona una forma de crear un objeto de Ciudad sin inicializar sus atributos.
    - **Constructor parametrizado (Ciudad(int id, String nombre, Pais pais))**:
        - Permite inicializar los atributos id (probablemente heredado de General), nombre y pais al momento de crear el objeto.
3. **Métodos get y set**:
    - **getNombre y setNombre**:
        - Acceden o modifican el atributo nombre.
    - **getPais y setPais**:
        - Acceden o modifican el atributo pais, que es un objeto de la clase Pais.

Código explicado

1. Clase Ciudad

- **extends General**:
    - Indica que Ciudad hereda de la clase General, probablemente reutilizando propiedades y métodos como id o fechaCreacion.

2. Constructor vacío

- Útil para crear objetos sin inicializar sus atributos inmediatamente.

3. Constructor parametrizado

- Inicializa:
    - id (heredado de General).
    - nombre, con el nombre de la ciudad.
    - pais, con un objeto que representa el país asociado.

4. Métodos get y set

- **getNombre**: Retorna el valor actual del atributo nombre.
- **setNombre**: Permite establecer un nuevo valor para nombre.
- **getPais**: Retorna el objeto Pais asociado.
- **setPais**: Asocia la ciudad con un nuevo objeto Pais.

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 de la clase Editorial

La clase Editorial modela una entidad que representa una editorial, asociándola con su nombre y la ciudad donde opera. Al igual que la clase Ciudad, extiende la clase General, lo que permite heredar propiedades comunes (como id) y compartir métodos estándar entre diferentes entidades.

Componentes principales

1. **Atributos**:
    - String nombre: Almacena el nombre de la editorial.
    - Ciudad ciudad: Referencia a un objeto de la clase Ciudad, que representa la ciudad donde está ubicada la editorial.
2. **Constructores**:
    - **Constructor vacío (Editorial())**:
        - Permite crear un objeto Editorial sin inicializar sus atributos.
    - **Constructor parametrizado (Editorial(int id, String nombre, Ciudad ciudad))**:
        - Inicializa los atributos id (heredado de General), nombre, y la ciudad asociada (ciudad).
3. **Métodos get y set**:

- **getNombre y setNombre**:
  - Acceden o modifican el nombre de la editorial.
- **getCiudad y setCiudad**:
  - Acceden o modifican el objeto Ciudad asociado con la editorial.

Código explicado

1. Clase Editorial

- **Atributos**:
  - nombre: El nombre de la editorial.
  - ciudad: Relación directa con la clase Ciudad.

2. Constructor vacío

- Permite instanciar la clase sin valores iniciales.

3. Constructor parametrizado

- Inicializa:
  - id (heredado de General).
  - nombre, con el nombre de la editorial.
  - ciudad, con una referencia a un objeto Ciudad.

4. Métodos get y set

- **getNombre**: Devuelve el nombre de la editorial.
- **setNombre**: Permite modificar el nombre.

- **getCiudad**: Retorna el objeto Ciudad asociado.
- **setCiudad**: Establece una nueva relación entre la editorial y una ciudad.

Código general

```
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 de la clase General

La clase General es una clase base o abstracta que contiene atributos comunes para varias entidades, como un identificador (id) y un nombre. Esta clase puede ser extendida por otras clases, como Ciudad, Editorial, o cualquier otra entidad que requiera estos atributos. Al proporcionar una estructura básica para estas clases, General facilita la reutilización de código y ayuda a organizar la jerarquía de las clases en el sistema.

Componentes principales

1. **Atributos**:
    - int id: Este atributo almacena un identificador único para cada instancia de General (y sus clases hijas).
    - String nombre: Este atributo almacena el nombre asociado a la entidad representada por la clase.
2. **Constructores**:
    - **Constructor vacío (General())**:
        - Permite crear un objeto de la clase sin inicializar sus atributos.

- **Constructor parametrizado (General(int id, String nombre))**:
  - Inicializa los atributos id y nombre con los valores proporcionados al crear una instancia de la clase.

3. **Métodos get y set**:
   - **getId y setId**:
     - Permiten acceder y modificar el valor del atributo id.
   - **getNombre y setNombre**:
     - Permiten acceder y modificar el valor del atributo nombre.

Código explicado

1. Clase General

- **Atributos**:
  - id: El identificador único de cada instancia de General.
  - nombre: El nombre asociado a la entidad representada.

2. Constructor vacío

- Constructor que crea un objeto de la clase General sin inicializar los atributos.

3. Constructor parametrizado

- Inicializa los atributos id y nombre con los valores proporcionados cuando se crea el objeto.

4. Métodos get y set

- **getId**: Devuelve el valor del id.
- **setId**: Establece un nuevo valor para el id.
- **getNombre**: Devuelve el valor del nombre.
- **setNombre**: Establece un nuevo valor para el nombre.

Código libro

```
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 de la clase Libro

La clase Libro representa un libro en un sistema. Esta clase contiene atributos relacionados con el libro, como su edición y año de publicación, así como los métodos necesarios para acceder y modificar estos atributos. Además, la clase incluye un método getStock que simula la cantidad de ejemplares del libro en stock, aunque no está relacionado directamente con los atributos del libro en el código proporcionado.

Componentes principales de la clase Libro

1. **Atributos**:
    - int edicion: Representa la edición del libro (por ejemplo, primera, segunda, etc.).
    - int añoPublicacion: Representa el año en que el libro fue publicado.
2. **Constructores**:

- **Constructor vacío (Libro())**:
  - Permite crear un objeto de la clase Libro sin inicializar los atributos.
- **Constructor parametrizado (Libro(int edicion, int añoPublicacion))**:
  - Inicializa los atributos edicion y añoPublicacion con los valores proporcionados.

3. **Métodos**:
   - **getStock()**:
     - Este método devuelve un valor constante de 10, lo que indica que, en este contexto, el libro tiene 10 ejemplares en stock. (En un sistema real, este valor podría estar relacionado con una base de datos o una variable dinámica).
   - **getEdicion() y setEdicion(int edicion)**:
     - Permiten obtener y establecer el valor de la edición del libro.
   - **getAñoPublicacion() y setAñoPublicacion(int añoPublicacion)**:
     - Permiten obtener y establecer el año de publicación del libro.

Código Explicado

1. Atributos

- **edicion**: Almacena la edición del libro.
- **añoPublicacion**: Almacena el año en que el libro fue publicado.

2. Constructor vacío

- Constructor que permite crear una instancia de Libro sin inicializar sus atributos.

Esto puede ser útil si se desea configurar los valores de los atributos más tarde.

3. Constructor parametrizado

- Este constructor inicializa los atributos edicion y añoPublicacion con los valores

proporcionados cuando se crea una nueva instancia de Libro.

4. Métodos get y set

- **getEdicion() y setEdicion(int edicion)**:
  - Obtienen y configuran el valor de la edición del libro.
- **getAñoPublicacion() y setAñoPublicacion(int añoPublicacion)**:
  - Obtienen y configuran el valor del año de publicación del libro.
- **getStock()**:
  - Devuelve un valor fijo de 10, indicando que siempre habrá 10

ejemplares disponibles del libro. En un sistema más complejo, este método podría consultar una base de datos o variable para obtener el valor real del stock.

Código país

```
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 de la clase Pais

La clase Pais es una representación de un país, heredando de la clase General y añadiendo un atributo nombre que almacena el nombre del país. Además, proporciona métodos para acceder y modificar este atributo, así como un constructor que permite la inicialización tanto del id heredado de la clase General como del nombre del país.

Componentes principales de la clase Pais

1. **Atributos**:
    - **String nombre**: Este atributo almacena el nombre del país.

2. **Constructores**:

   - **Constructor vacío (Pais())**: Permite crear un objeto Pais sin inicializar el nombre ni el id. Este constructor puede ser útil si los valores del país se establecen después de la creación del objeto.

   - **Constructor parametrizado (Pais(int id, String nombre))**: Inicializa el objeto Pais con los valores proporcionados, configurando tanto el id heredado de la clase General como el nombre del país.

3. **Métodos**:

   - **getNombre() y setNombre(String nombre)**: Permiten obtener y establecer el nombre del país.

Código Explicado

1. Atributo nombre

- Este atributo almacena el nombre del país, el cual es una cadena de texto.

2. Constructor vacío

- Constructor que permite crear un objeto de la clase Pais sin inicializar sus atributos. Es útil cuando los valores de los atributos se asignan después de la creación del objeto.

3. Constructor parametrizado

- Este constructor inicializa el atributo nombre con el valor proporcionado y también permite establecer el id utilizando el constructor de la clase General (heredada por Pais).

4. Métodos get y set

- **Método getNombre()**:
- Obtiene el valor del atributo nombre del país.
- **Método setNombre(String nombre)**:
- Establece el valor del atributo nombre con el valor proporcionado.

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 de la clase Prestatario

La clase Prestatario extiende de la clase General y se utiliza para representar a una persona que recibe un préstamo o crédito, conteniendo atributos que describen su identidad, contacto y ubicación.

Estos atributos incluyen nombre, apellido, direccion, ciudad (que está asociada a un objeto de la clase Ciudad), y telefono.

Componentes principales de la clase Prestatario

1. **Atributos**:
    - **nombre**: Almacena el nombre del prestatario.
    - **apellido**: Almacena el apellido del prestatario.
    - **direccion**: Almacena la dirección de residencia del prestatario.
    - **ciudad**: Es un objeto de la clase Ciudad, que representa la ciudad de residencia del prestatario.
    - **telefono**: Almacena el número de teléfono del prestatario.
2. **Constructores**:
    - **Constructor vacío (Prestatario())**: Permite crear un objeto Prestatario sin inicializar sus atributos. Este constructor puede ser útil si los valores de los atributos se asignan después de la creación del objeto.
    - **Constructor parametrizado (Prestatario(int id, String nombre, String apellido, String direccion, Ciudad ciudad, String telefono))**: Inicializa todos los atributos de la clase, incluido el id heredado de la clase General.
3. **Métodos**:
    - **Getters y setters**: Permiten obtener y establecer los valores de los atributos privados de la clase, como el nombre, apellido, direccion, ciudad, y telefono.

Código Explicado

1. Atributos

- **nombre**, **apellido**, **direccion**, y **telefono** son atributos de tipo String que representan información personal del prestatario.
- **ciudad** es un objeto de tipo Ciudad, lo que permite asociar una ciudad al prestatario.

2. Constructor vacío

- Constructor sin parámetros que permite crear un objeto Prestatario sin inicializar los atributos. En este caso, los valores de los atributos pueden asignarse posteriormente a través de los métodos set.

3. Constructor parametrizado

- Este constructor inicializa los atributos nombre, apellido, direccion, ciudad y telefono con los valores proporcionados, permitiendo una inicialización más directa y conveniente del objeto Prestatario.

4. Métodos get y set

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

- **Método getNombre() y setNombre(String nombre)**: Obtienen y establecen el valor de nombre.
- **Método getApellido() y setApellido(String apellido)**: Obtienen y establecen el valor de apellido.

- **Método getDireccion() y setDireccion(String direccion)**: Obtienen y establecen el valor de direccion.

- **Método getCiudad() y setCiudad(Ciudad ciudad)**: Obtienen y establecen el valor del objeto Ciudad asociado al prestatario.

- **Método getTelefono() y setTelefono(String telefono)**: Obtienen y establecen el valor de telefono.

Conclusión

El diseño orientado a objetos de las clases presentadas permite una adecuada representación de un sistema de gestión de libros y prestatarios. Mediante el uso de clases como Libro, Prestatario, Pais, Ciudad, y Editorial, se puede almacenar y gestionar la información esencial relacionada con el préstamo de libros, facilitando la relación entre los diferentes actores (libros, prestatarios, editoriales, etc.) dentro de un sistema. Además, el uso de principios como la herencia y la encapsulación en el diseño de las clases garantiza la flexibilidad, la reutilización del código y la facilidad de mantenimiento del sistema.

Este enfoque modular permite que el sistema se amplíe de manera sencilla, añadiendo nuevas funcionalidades o entidades si fuera necesario, sin afectar la integridad del diseño original. En resumen, este conjunto de clases representa un ejemplo sólido de cómo se puede utilizar la programación orientada a objetos para modelar un sistema de gestión eficiente y escalable.