# PROGRAMACIÓN - UD 3

## CASO PRÁCTICO 1

**DESARROLLO DE CLASES EN JAVA**

## Contexto

El jefe de proyecto de tu equipo de programación te ha pedido realizar una serie de clases que tienes que dejar listas en esta semana.

Estas clases son:

1. Crea las siguientes clases con notación UML y, posteriormente, en código Java, con sus correspondientes atributos, y métodos constructores con y sin parámetros y métodos getters y setters:
    a. Clase empleado (en general)
    b. Clase administrativo
    c. Clase contable
    d. Clase informático
2. Para los atributos y métodos indicados, especifique cuáles considera que podrían ser públicos, privados o protegidos y el motivo.
3. Establezca la relación que existe entre las clases de la actividad anterior en notación UML y explica detalladamente.
4. Especifique cómo crearía una instancia para cada una de las clases anteriores. Realiza una clase que contenga un método main y crea una instancia de cada clase, inicializa con los datos correspondientes usando el constructor con parámetros y usando el constructor sin parámetros (utilizando métodos **setter** para asignar valores a los atributos), después muestra los datos que contiene cada objeto por pantalla.

## Cuestiones a resolver

Realiza las clases que se detallan en el apartado anterior.

# PROGRAMACIÓN - UD 3

# Introducción

En este caso práctico, se nos pide crear un conjunto de clases en Java que representan a distintos tipos de empleados dentro de una organización. Estas clases incluyen una clase base `Empleado` y tres subclases especializadas: `Administrativo`, `Contable` e `Informático`.

A lo largo de este análisis, se explorará cómo se deben estructurar las clases, sus atributos, métodos y la relación de herencia entre ellas. Además, se discutirá la visibilidad de los atributos y métodos y cómo afecta la encapsulación y la seguridad del código. El objetivo es crear una jerarquía de clases bien definida que permita modelar correctamente los empleados en diferentes roles dentro de una empresa.

# Desarrollo

## 1. Clases UML y código Java

Para resolver este ejercicio, primero vamos a construir las clases a nivel de diagrama UML y luego las implementaremos en código Java. La relación principal entre estas clases es la **herencia**, donde la clase `Empleado` actúa como clase base y las otras tres clases son subclases que heredan de esta clase base. A continuación, detallamos cada clase.

### Clase Empleado (Base)

La clase `Empleado` es la clase principal que contiene los atributos comunes a todos los empleados de la organización, como **nombre**, **edad**, **salario**, **dirección**, **teléfono**, **número de empleado** y **turno**.

**Atributos**:

- `nombre`: Nombre del empleado.
- `edad`: Edad del empleado.
- `salario`: Salario mensual.
- `direccion`: Dirección de residencia.
- `telefono`: Número de teléfono.
- `numeroEmpleado`: Identificador único del empleado.
- `turno`: Turno de trabajo asignado.

**Métodos**:

- **Constructores**: Uno con parámetros, que permite inicializar todos los atributos, y otro sin parámetros para instanciar el objeto vacío.
- **Métodos Getters y Setters**: Permiten acceder y modificar los valores de los atributos.

# PROGRAMACIÓN - UD 3

- **`mostrarInfo()`**: Método que imprime en consola la información del empleado.
- **`calcularAnualSalario()`**: Método que calcula el salario anual multiplicando el salario mensual por 12.
- **`aumentarSalario()`**: Método para aumentar el salario en un porcentaje dado.

## Subclases: Administrativo, Contable e Informático

Estas subclases heredan los atributos y métodos de `Empleado`, pero agregan atributos adicionales que son específicos para cada tipo de empleado.

### Clase Administrativo

La clase `Administrativo` hereda de `Empleado` y añade el atributo `seccion`, que especifica el área o departamento al que pertenece el administrativo.

**Métodos**:

- Getters y Setters para `seccion`.

### Clase Contable

La clase `Contable` hereda de `Empleado` y añade el atributo `areaFinanciera`, que especifica el área financiera en la que trabaja el contable, como **contabilidad general**, **auditoría**, etc.

**Métodos**:

- Getters y Setters para `areaFinanciera`.

### Clase Informático

La clase `Informático` hereda de `Empleado` y añade el atributo `lenguajeProgramacion`, que especifica el lenguaje de programación que maneja el informático, como **Java**, **Python**, **C++**, etc. Además, esta clase tiene un método adicional **`desarrollarSoftware()`** que simula la acción de desarrollar software en el lenguaje especificado.

**Métodos**:

- `desarrollarSoftware()`: Método para simular el proceso de desarrollo de software.

En la programación orientada a objetos (OOP), uno de los principios fundamentales es la **encapsulación**. Este principio establece que los detalles internos de una clase (como los atributos) deben estar ocultos del exterior y solo deben ser accesibles o modificados mediante métodos controlados, llamados **métodos getters** (para obtener los valores) y

**métodos setters** (para modificar los valores). Esto ayuda a proteger los datos de ser manipulados directamente de forma no controlada y asegura que la clase sea más fácil de mantener y evolucionar a lo largo del tiempo.

En este caso, se nos pide especificar qué atributos y métodos deben ser **públicos**, **privados** o **protegidos** en la jerarquía de clases propuesta. A continuación, detallaré la razón de elección de cada tipo de visibilidad para los atributos y métodos de la clase `Empleado` y sus subclases (`Administrativo`, `Contable` e `Informático`).

## 2. Visibilidad de los Atributos

### Atributos privados (`private`):

Los atributos privados deben ser **inaccesibles directamente desde fuera de la clase**. Esto se debe a que los datos sensibles deben ser protegidos para evitar que sean modificados sin el control adecuado. La encapsulación se logra restringiendo el acceso a estos atributos y proporcionándoles **métodos getters** y **setters** para gestionarlos.

En el caso de nuestra clase `Empleado`, los siguientes atributos son adecuados para ser privados:

- **`nombre`**: Es un dato sensible relacionado con la identidad del empleado. Acceder a él debería estar restringido para evitar que se cambie sin autorización.
- **`edad`**: La edad también es un dato personal que no debe ser modificado directamente desde fuera de la clase.
- **`numeroEmpleado`**: Es un identificador único que se asigna al empleado. No debería ser cambiado manualmente una vez asignado, ya que esto comprometería la integridad del sistema de registro de empleados.
- **`turno`**: El turno del empleado es una propiedad que, en algunos contextos, no debería ser modificada directamente por cualquier clase externa, sino a través de un proceso controlado.

### Atributos protegidos (`protected`):

Los atributos protegidos son accesibles dentro de la propia clase y en las subclases que heredan de ella. En este caso, hay ciertos atributos que deben ser accesibles para las subclases, pero **no deben ser modificados directamente desde el exterior de la jerarquía de clases**.

A continuación, se listan los atributos protegidos en la clase `Empleado`:

- **`salario`**: El salario es una propiedad importante del empleado y puede necesitar ser manipulado por las subclases, como en el caso de las subclases `Administrativo`, `Contable` o `Informático` que podrían tener reglas específicas para incrementar o manejar el salario.

- **`direccion`** y **`telefono`**: Si bien estos son datos personales, las subclases podrían necesitarlos para realizar operaciones o modificarlos en función de las necesidades de cada tipo de empleado. No es necesario que sean públicos, pero sí deben estar accesibles dentro de la jerarquía de clases.

### Atributos públicos (`public`):

Los atributos públicos deben ser **accesibles desde cualquier lugar** de la aplicación. Sin embargo, en este caso, **no se recomienda hacer públicos los atributos de la clase `Empleado`**, ya que todos los atributos importantes deberían estar protegidos para mantener la integridad de la clase. En cambio, se debe proporcionar acceso mediante los métodos `getters` y `setters`.

## 3. Visibilidad de los Métodos

### Métodos privados (`private`):

Los métodos privados son utilizados solo dentro de la clase. Por ejemplo, si tienes algún método que realice una operación interna que no deba ser accesible desde fuera de la clase, este debería ser privado.

En el caso de nuestra jerarquía de clases, un ejemplo de método privado que podría ser considerado es **`aumentarSalario()`** si se desea implementar algún tipo de lógica interna para manipular el salario y no se quiere que sea accesible directamente desde las subclases o desde otras clases.

### Métodos protegidos (`protected`):

Los métodos protegidos son aquellos que se deben utilizar en las subclases. Estos métodos deben ser accesibles para las subclases pero no deberían ser utilizados directamente desde fuera de la jerarquía de clases.

Por ejemplo, un método como **`calcularAnualSalario()`** podría ser protegido porque no tiene sentido que otras clases externas modifiquen el cálculo del salario anual. Sin embargo, las subclases (como `Contable` o `Informático`) podrían necesitar este método para realizar sus propias operaciones o cálculos.

### Métodos públicos (`public`):

Los métodos públicos son aquellos que deben ser accesibles desde cualquier lugar del código. Estos métodos proporcionan la interfaz pública de la clase.

- **`getters` y `setters`**: Los métodos que permiten acceder o modificar los atributos privados de la clase son públicos, como **`getNombre()`**, **`setNombre(nombre)`**, **`getEdad()`**, etc. De esta manera, cualquier parte del código que necesite manipular

# PROGRAMACIÓN - UD 3

los datos de un empleado podrá hacerlo a través de estos métodos sin acceder directamente a los atributos.

- **`mostrarInfo()`**: Este método debe ser público para que cualquier clase externa pueda acceder a él y obtener la información del empleado. Además, al ser un método genérico que muestra datos comunes a todos los empleados (como nombre, edad, salario), no tiene sentido restringirlo.

## 4. La Relación Entre las Clases y el Uso de Herencia

La **herencia** es un concepto clave en la programación orientada a objetos y permite que las subclases heredan comportamientos y atributos de la clase base. En este caso, la relación entre la clase `Empleado` y las subclases `Administrativo`, `Contable` e `Informático` es una relación de herencia directa, ya que estas subclases comparten un conjunto de atributos y métodos comunes con la clase base `Empleado`, pero también agregan atributos y métodos específicos.

La clase `Empleado` actúa como una **clase base** o **superclase** que tiene los atributos comunes a todos los empleados, como el **nombre**, **edad**, **salario**, **dirección**, **teléfono**, etc. Las subclases, como `Administrativo`, `Contable` e `Informático`, **heredan estos atributos y métodos** de la clase `Empleado`, pero cada una tiene atributos adicionales que son específicos para su tipo de empleado.

**Ventajas de esta relación de herencia:**

- **Reutilización de código**: Las subclases no tienen que volver a definir los atributos y métodos comunes; simplemente los heredan de la clase base.
- **Modularidad y mantenimiento**: Si se realiza un cambio en la clase base `Empleado`, las subclases automáticamente heredan esos cambios, lo que facilita el mantenimiento y la extensión de la aplicación.
- **Polimorfismo**: Aunque las subclases tienen métodos propios, pueden sobrescribir o extender los métodos de la clase base según sea necesario.

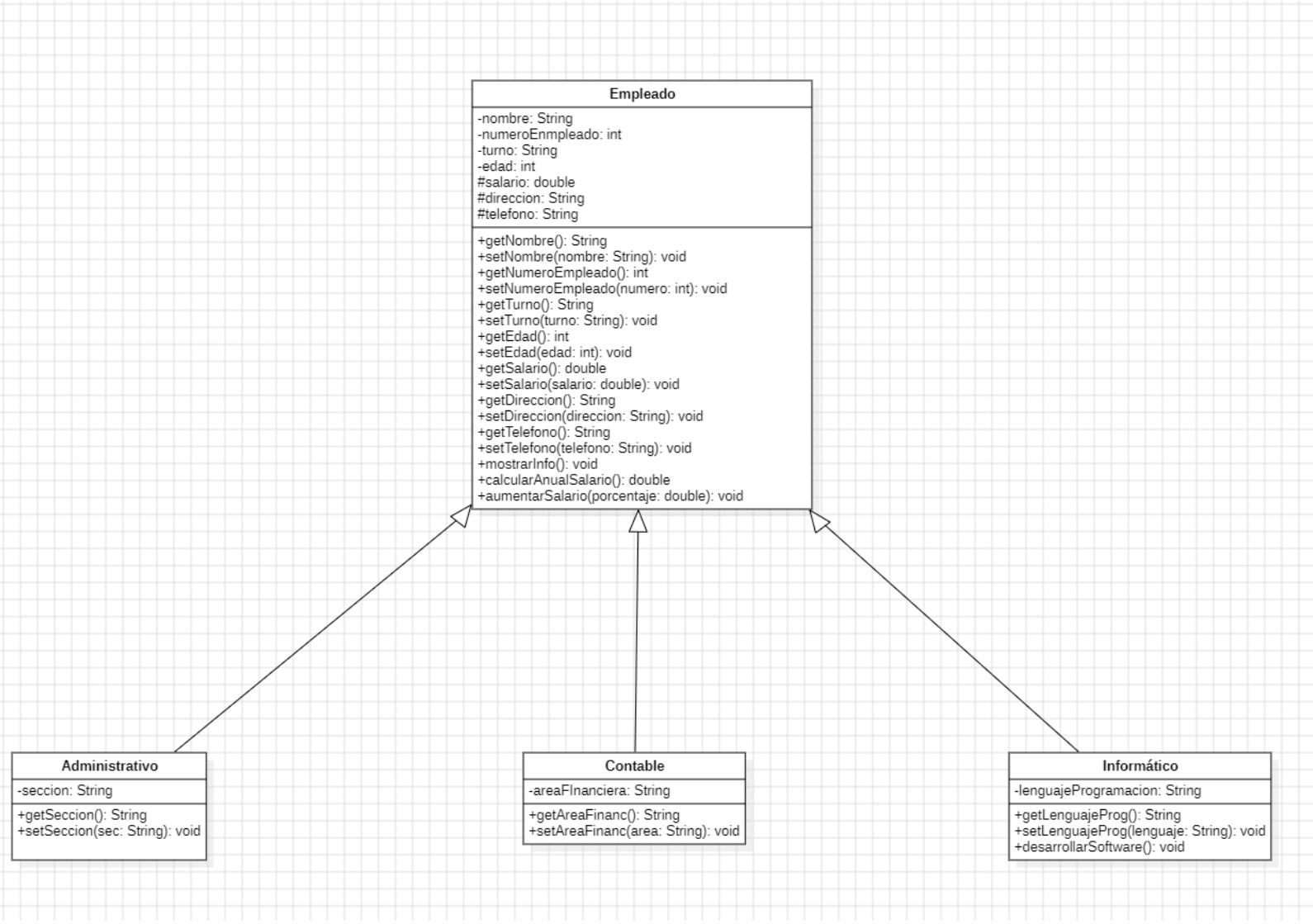
## 5. Conclusión sobre la Visibilidad

La elección de la visibilidad de los atributos y métodos es crucial para garantizar que las clases sean seguras, reutilizables y fáciles de mantener. En este caso, los **atributos privados** aseguran que los datos sensibles (como el salario, nombre, etc.) no puedan ser modificados directamente desde fuera de la clase, mientras que los **métodos públicos** proporcionan una interfaz controlada para acceder y modificar estos atributos. Los **atributos y métodos protegidos** permiten que las subclases accedan y modifiquen los datos de manera controlada, asegurando que no se violen las reglas de la encapsulación.

En resumen, el uso adecuado de los modificadores de acceso **privado**, **protegido** y **público** en las clases es fundamental para implementar una arquitectura robusta y segura en la programación orientada a objetos.

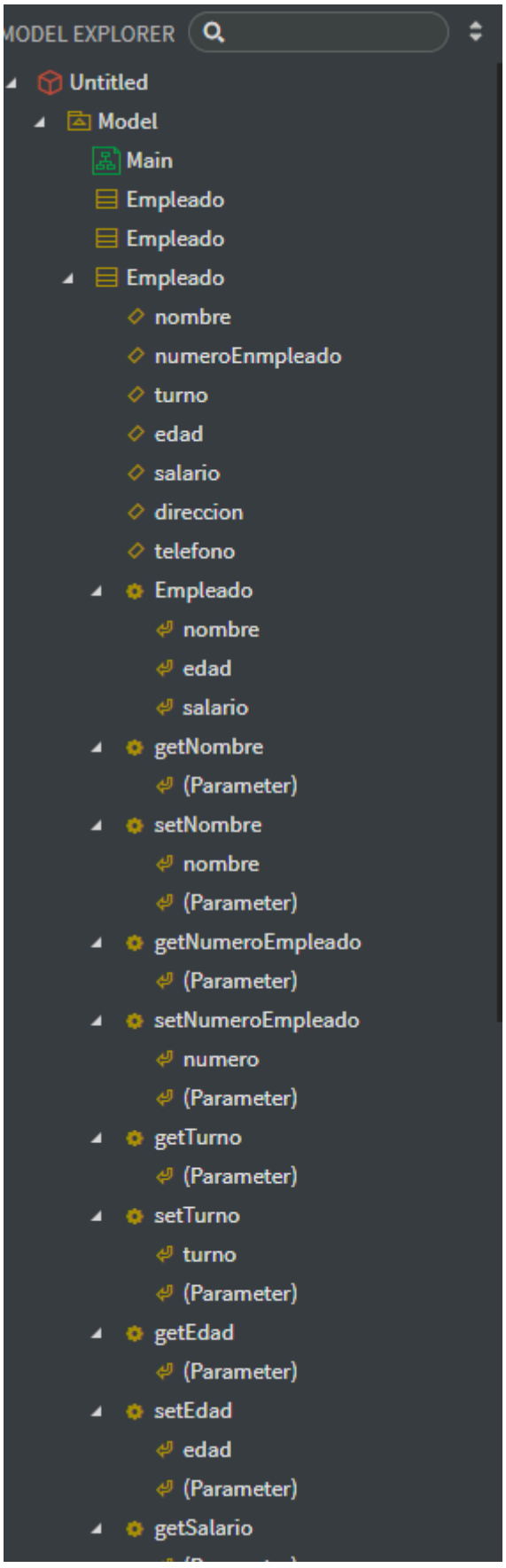
# PROGRAMACIÓN - UD 3

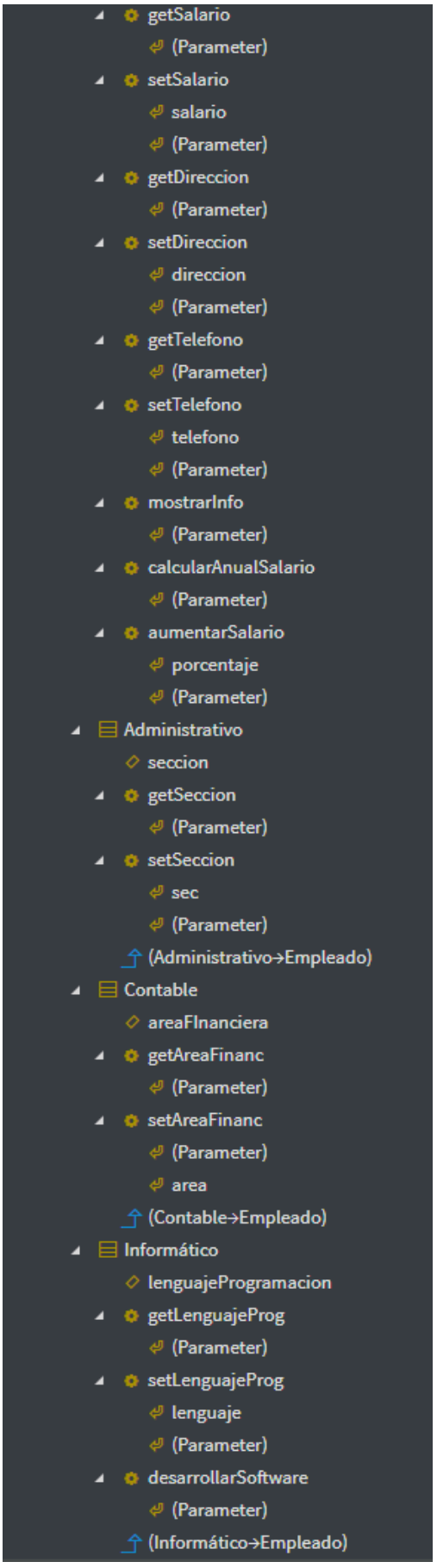
## 6. Diagrama UML



# PROGRAMACIÓN - UD 3

## 7. EXPLORADOR DEL DIAGRAMA UML



# PROGRAMACIÓN - UD 3

## 8. Código

### 8.1 Empleado.java

```
package ud3caso1app;

public class Empleado {
    private String nombre;
    private int edad;
    protected double salario;
    protected String direccion;
    protected String telefono;
    private int numeroEmpleado;
    private String turno;

    public Empleado() {
    }

    public Empleado(String nombre, int edad, double salario, String direccion, String telefono,
int numeroEmpleado, String turno) {
        this.nombre = nombre;
        this.edad = edad;
        this.salario = salario;
        this.direccion = direccion;
        this.telefono = telefono;
        this.numeroEmpleado = numeroEmpleado;
        this.turno = turno;
    }

    public String getNombre() {
        return nombre;
    }

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

    public int getEdad() {
        return edad;
    }

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

    public double getSalario() {
```

# PROGRAMACIÓN - UD 3

```
        return salario;
    }

    public void setSalario(double salario) {
        this.salario = salario;
    }

    public String getDireccion() {
        return direccion;
    }

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

    public String getTelefono() {
        return telefono;
    }

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

    public int getNumeroEmpleado() {
        return numeroEmpleado;
    }

    public void setNumeroEmpleado(int numeroEmpleado) {
        this.numeroEmpleado = numeroEmpleado;
    }

    public String getTurno() {
        return turno;
    }

    public void setTurno(String turno) {
        this.turno = turno;
    }

    public void mostrarInfo() {
        System.out.println("Empleado: " + nombre + ", Edad: " + edad + ", Salario: " + salario +
                ", Dirección: " + direccion + ", Teléfono: " + telefono +
                ", Número Empleado: " + numeroEmpleado + ", Turno: " + turno);
    }
}
```

# PROGRAMACIÓN - UD 3

## 8.2 Administrativo.java

```
package ud3caso1app;

public class Administrativo extends Empleado {
    private String seccion;

    public Administrativo() {
    }

    public Administrativo(String nombre, int edad, double salario, String direccion, String
telefono, int numeroEmpleado, String turno, String seccion) {
        super(nombre, edad, salario, direccion, telefono, numeroEmpleado, turno);
        this.seccion = seccion;
    }

    public String getSeccion() {
        return seccion;
    }

    public void setSeccion(String seccion) {
        this.seccion = seccion;
    }
}
```

## 8.3 Contable.java

```
package ud3caso1app;

public class Contable extends Empleado {
    private String areaFinanciera;

    public Contable() {
    }

    public Contable(String nombre, int edad, double salario, String direccion, String telefono,
int numeroEmpleado, String turno, String areaFinanciera) {
        super(nombre, edad, salario, direccion, telefono, numeroEmpleado, turno);
        this.areaFinanciera = areaFinanciera;
    }

    public String getAreaFinanciera() {
        return areaFinanciera;
    }

    public void setAreaFinanciera(String areaFinanciera) {
```

# PROGRAMACIÓN - UD 3

```
        this.areaFinanciera = areaFinanciera;
    }
}
```

### 8.4 Informatico.java

```
package ud3caso1app;

public class Informatico extends Empleado {
    private String lenguajeProgramacion;

    public Informatico() {
    }

    public Informatico(String nombre, int edad, double salario, String direccion, String
telefono, int numeroEmpleado, String turno, String lenguajeProgramacion) {
        super(nombre, edad, salario, direccion, telefono, numeroEmpleado, turno);
        this.lenguajeProgramacion = lenguajeProgramacion;
    }

    public String getLenguajeProgramacion() {
        return lenguajeProgramacion;
    }

    public void setLenguajeProgramacion(String lenguajeProgramacion) {
        this.lenguajeProgramacion = lenguajeProgramacion;
    }

    public void desarrollarSoftware() {
        System.out.println(getNombre() + " está desarrollando software en " +
lenguajeProgramacion);
    }
}
```

### 8.5 Principal.java

```
package ud3caso1app;

public class Principal {
    public static void main(String[] args) {
        // Crear instancia de Empleado
        Empleado empleado = new Empleado("Juan", 30, 2000.0, "Calle Falsa 123",
"123456789", 1, "Mañana");
        empleado.mostrarInfo();

        // Crear instancia de Administrativo
```

# PROGRAMACIÓN - UD 3

```
        Administrativo admin = new Administrativo("Laura", 28, 1800.0, "Calle Real 456",
"987654321", 2, "Tarde", "Recursos Humanos");
        admin.mostrarInfo();

        // Crear instancia de Contable
        Contable contable = new Contable("Carlos", 35, 2200.0, "Avenida Central 789",
"654321987", 3, "Noche", "Impuestos");
        contable.mostrarInfo();

        // Crear instancia de Informático
        Informatico informatico = new Informatico("Ana", 25, 2500.0, "Calle Nueva 321",
"456789123", 4, "Mañana", "Java");
        informatico.mostrarInfo();
        informatico.desarrollarSoftware();
    }
}
```

## 9. Pruebas