Ejemplo práctico: sistema de nómina utilizando polimorfismo.

En este ejemplo se usará la jerarquía ***EmpleadoPorComision-EmpleadoBaseMasComision*** usada anteriormente.
Se usará un *método abstracto* y *polimorfismo* para realizar los cálculos de nómina, con base en una jerarquía de herencia de empleados mejorada que cumpla con los siguientes requerimientos:

> "***Una compañía paga semanalmente a sus empleados, quienes se dividen en cuatro tipos: empleados asalariados que reciben un salario semanal fijo, sin importar el número de horas trabajadas; empleados por horas, que perciben un sueldo por hora y pago por tiempo extra (es decir, 1.5 veces la tarifa de su salario por horas), por todas las horas trabajadas que excedan a 40 horas; empleados por comisión, que perciben un porcentaje de sus ventas, y empleados asalariados por comisión, que obtienen un salario base más un porcentaje de sus ventas. Para este periodo de pago, la compañía ha decidido recompensar a los empleados asalariados por comisión, agregando un 10% a sus salarios base. La compañía desea escribir una aplicación que realice sus cálculos de nómina en forma polimórfica."***

Se usará:

- una clase *abstract* ***Empleado*** para representar el concepto general de un empleado.
- Las subclases que extiendan a ***Empleado***: ***EmpleadoAsalariado***, ***EmpleadoPorComision*** y ***EmpleadoPorHoras***
- La subclase ***EmpleadoBaseMasComision*** que representa el último tipo de empleado.

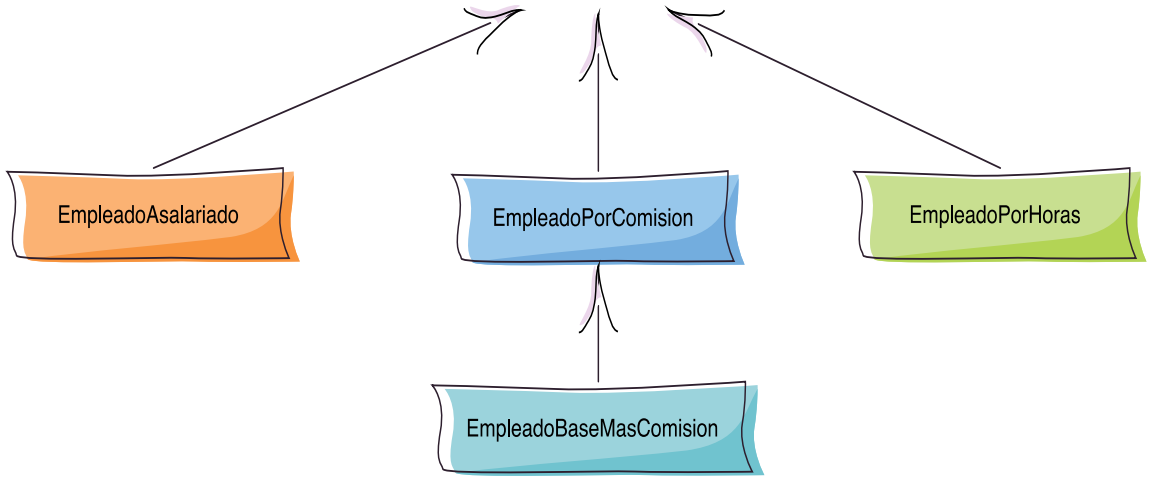

Ilustración 1. Diagrama de clases de UML para la jerarquía Empleado

La superclase abstracta ***Empleado*** declara la "interfaz" para la jerarquía; es decir, el conjunto de métodos que puede invocar un programa en todos los objetos Empleado.
Cada empleado, sin importar la manera en que se calculen sus ingresos, tiene un primer nombre, un apellido paterno y un número de seguro social, por lo que las variables de instancia ***private primerNombre***, ***apellidoPaterno*** y ***numeroSeguroSocial*** aparecen en la superclase abstracta ***Empleado***.

En la siguiente tabla, se muestra cada una de las cinco clases en la jerarquía, hacia abajo en la columna de la izquierda, y los métodos **ingresos** y **toString** en la fila superior. Para cada clase, el diagrama muestra los resultados deseados de cada método. No se muestran los métodos obtener de la superclase ***Empleado*** porque no se sobreescriben en ninguna de las subclases; éstas heredan y utilizan cada uno de estos métodos "así como están".

	ingresos	toString
Empleado	abstract	*primerNombre apellidoPaterno* numero de seguro social: *NSS*
EmpleadoAsalariado	salarioSemanal	Empleado aslariado: *primerNombre apellidoPaterno* Número de seguro social: *NSS* Salario semanal: *salarioSemanal*

EmpleadoPorHoras	if (horas <= 40)
	sueldo * horas
	else if (horas > 40)
	{
	40 * sueldo +
	(horas – 40) *
	Sueldo * 1.5
	}
		Empleado por horas: *primerNombre apellidoPaterno*
		Número de seguro social: *NSS*
		Sueldo por horas: *sueldo*; horas trabajadas: *horas*

EmpleadoPorComision	tarifaComision * ventasBrutas	Empleado por comisión: *primerNombre apellidoPaterno* Número de seguro social: *NSS* Ventas brutas: *ventasBrutas*; Tarifa de comisión: *tarifaComision*
EmpleadoBaseMasComision	(tarifaComision * ventasBrutas) + salarioBase	Empleado por comisión con salario base: *primerNombre apellidoPaterno* número de seguro social: *NSS* ventas Brutas: *ventasBrutas*; tarida de comisión: *tarifaComision*; salario base: *salarioBase*

Ilustración 2_ Interfaz polimórfica para las clases de la jerarquía de Empleado

Con esta información, deberá implementar la jerarquía de clases de Empleado:

- Superclase abstracta ***Empleado***
- Subclases concretas ***EmpleadoAsalariado***, ***EmpleadoPorComision***, ***EmpleadoPorHoras***, ***EmpleadoBaseMasComision***
- Programa de prueba que crea objetos de todas las clases y procesa esos objetos mediante polimorfismo.

Clase Empleado.

```java
// superclase abstracta Empleado
public abstract class Empleado
{
    private final String primerNombre;
    private final String apellidoPaterno;
    private final String numeroSeguroSocial;

    // constructor
    public Empleado(String primerNombre, String apellidoPaterno, String numeroSeguroSocial)
    {
        this.primerNombre = primerNombre;
        this.apellidoPaterno = apellidoPaterno;
        this.numeroSeguroSocial = numeroSeguroSocial;
    }

    // devuelve el primer nombre
    public String obtenerPrimerNombre()
    {
        return primerNombre;
    }

    // devuelve el apellido paterno
    public String obtenerApellidoPaterno()
    {
        return apellidoPaterno;
    }

    // devuelve el número de seguro social
    public String obtenerNumeroSeguroSocial()
    {
        return numeroSeguroSocial;
    }

    // devuelve representación String de un objeto Empleado
    @Override
    public String toString()
    {
        return String.format("%s %s%nnumero de seguro social: %s",
            obtenerPrimerNombre(), obtenerApellidoPaterno(),
            obtenerNumeroSeguroSocial());
    }

    // método abstracto sobreescrito por las subclases concretas
    public abstract double ingresos(); // aquí no hay implementación
} // fin de la clase abstracta Empleado
```

Clase EmpleadoPorComision.

```java
// La clase EmpleadoPorComision extiende a Empleado

public class EmpleadoPorComision extends Empleado {

    private double ventasBrutas; // ventas totales por semana
    private double tarifaComision; // porcentaje de comision

    public EmpleadoPorComision(String primerNombre, String apellidoPaterno,
    String numeroSeguroSocial, double ventasBrutas, double tarifaComision )
    {
        super(primerNombre, apellidoPaterno, numeroSeguroSocial);

        if (ventasBrutas < 0.0)
            throw new IllegalArgumentException("Las ventas brutas deben ser >= 0.0");

        if (tarifaComision <= 0.0 || tarifaComision >= 1.0)
            throw new IllegalArgumentException("La tarifa de comisión debe ser > 0.0 y < 1.0");


        this.ventasBrutas = ventasBrutas;
        this.tarifaComision = tarifaComision;
    }


    public void establecerVentasBrutas(double ventasBrutas){
        if (ventasBrutas < 0.0)
            throw new IllegalArgumentException("Las ventas brutas deben ser >= 0.0");
        this.ventasBrutas = ventasBrutas;
    }
    public void establecerTarifaComision(double tarifaComision){
        if (tarifaComision <= 0.0 || tarifaComision >= 1.0)
            throw new IllegalArgumentException("La tarifa de comisión debe ser > 0.0 y < 1.0");
        this.tarifaComision = tarifaComision;
    }

    public double obtenerVentasBrutas()
    {
        return ventasBrutas;
    }
    public double obtenerTarifaComision() {
        return tarifaComision;
    }

    // calcula los ingresos; sobre escribe el método abstracto ingresos en Empleado
    @Override
    public double ingresos()
    {
        return obtenerTarifaComision() * obtenerVentasBrutas();
    }

    // Devuelve representación String de un objeto EmpleadoPorComision
    @Override
    public String toString()
    {
        return String.format("%s: %s%n%s: $%,.2f; %s: %.2f",
            "Empleado por comisión", super.toString(),
            "ventas brutas", obtenerVentasBrutas(),
            "tarifa por comisión", obtenerTarifaComision());
    }
}// fin de la clase EmpleadoPorComision

```

Clase EmpleadoBaseMasComision.

```java
public class EmpleadoBaseMasComision extends EmpleadoPorComision
{
    private double salarioBase;

    public EmpleadoBaseMasComision(String primerNombre, String apellidoPaterno,
        String numeroSeguroSocial, double ventasBrutas, double tarifaComision,
        double salarioBase)
    {
        // llamada al constructor de la superclase EmpleadoPorComision
        super(primerNombre, apellidoPaterno, numeroSeguroSocial, ventasBrutas, tarifaComision);

        if (salarioBase < 0.0)
            throw new IllegalArgumentException ("El salario base debe ser >= 0.0");
        this.salarioBase = salarioBase;
    }

    public void establecerSalarioBase(double salarioBase)
    {
        if (salarioBase < 0.0)
            throw new IllegalArgumentException ("El salario base debe ser >= 0.0");
        this.salarioBase = salarioBase;
    }

    public double obtenerSalarioBase()
    {
        return salarioBase;
    }

    @Override
    public double ingresos(){
        return obtenerSalarioBase() + super.ingresos();
    }

    @Override
    public String toString()
    {
        return String.format("%s %s; %s: $%,.2f",
            "Con salario base", super.toString(),
            "salario base", obtenerSalarioBase());
    }
}
```

Clase EmpleadoAsalariado.

```java
// la clase concreta EmpleadoAsalariado extiende a la clase abstracta Empleado

public class EmpleadoAsalariado extends Empleado
{
    private double salarioSemanal;

    // constructor
    public EmpleadoAsalariado(String primerNombre, String apellidoPaterno,
        String numeroSeguroSocial, double salarioSemanal)
    {
        super(primerNombre, apellidoPaterno, numeroSeguroSocial);

        if(salarioSemanal < 0.0)
            throw new IllegalArgumentException(
                "El salario semanal debe ser >= 0.0");

        this.salarioSemanal = salarioSemanal;
    }

    // establece el salario
    public void establecerSalarioSemanal(double salarioSemanal)
    {
        if(salarioSemanal < 0.0)
            throw new IllegalArgumentException(
                "El salario semanal debe ser >= 0.0");

        this.salarioSemanal = salarioSemanal;

    }

    // devuelve el salario
    public double obtenerSalarioSemanal()
    {
        return salarioSemanal;
    }

    // calcula los ingresos: sobrescribe el método abstracto ingresos en Empleado
    @Override
    public double ingresos()
    {
        return obtenerSalarioSemanal();
    }

    // devuelve representación String de un objeto EmpleadoAsalariado
    @Override
    public String toString()
    {
        return String.format("Empleado asalariado: %s%n%s: $%,.2f",
            super.toString(), "salario semanal", obtenerSalarioSemanal());
    }
} // fin de la clase EmpleadoAsalariado
```

Clase EmpleadoPorHoras.

```java
// la clase EmpleadoPorHoras extiende a Empleado

public class EmpleadoPorHoras extends Empleado{
    private double sueldo; // sueldo por hora
    private double horas; // horas trabajadas por semana

    // constructor
    public EmpleadoPorHoras(String primerNombre, String apellidoPaterno,
        String numeroSeguroSocial, double sueldo, double horas)
    {
        super(primerNombre, apellidoPaterno, numeroSeguroSocial);

        if(sueldo < 0.0) // valida sueldo
            throw new IllegalArgumentException(
                "El sueldo por horas debe ser >= 0.0");


        if((horas < 0.0) || (horas > 168.0)) // valida horas
            throw new IllegalArgumentException(
                "Las horas trabajadas deben ser >= 0.0 y <= 168.0");

        this.sueldo = sueldo;
        this.horas = horas;
    }

    // establece el sueldo
    public void establecerSueldo(double sueldo)
    {
        if(sueldo < 0.0) // valida sueldo
            throw new IllegalArgumentException(
                "El sueldo por horas debe ser >= 0.0");

        this.sueldo = sueldo;
    }

    // devuelve el sueldo
    public double obtenerSueldo()
    {
        return sueldo;
    }

    // estable las horas trabajadas
    public void establecerHoras(double horas)
    {
        if((horas < 0.0) || (horas > 168.0)) // valida horas
            throw new IllegalArgumentException(
                "Las horas trabajadas deben ser >= 0.0 y <= 168.0");
```

```java

        this.horas = horas;
    }

    // devuelve las horas trabajadas
    public double obtenerHoras()
    {
        return horas;
    }

    // calcula los ingresos; sobrescribe el método abstracto ingresos en Empleado
    @Override
    public double ingresos()
    {
        if (obtenerHoras() <= 40) // no hay tiempo extra
            return obtenerSueldo() * obtenerHoras();
        else
            return 40 * obtenerSueldo() + (obtenerHoras() - 40) * obtenerSueldo() * 1.5;
    }

    // devuelve representación String de un objeto EmpleadoPorHoras
    @Override
    public String toString()
    {
        return String.format("Empleado por horas: %s%n%s: $%,.2f; %s: %,.2f",
            super.toString(), "sueldo por hora", obtenerSueldo(),
            "horas trabajadas", obtenerHoras());
    }


} // fin de la clase EmpleadoPorHoras

```

Clase PruebaSistemaNomina.

```java
// Programa de prueba para la jerarquia de Empleado

public class PruebaSistemaNomina
{
    public static void main(String[] args)
    {
        // crea objetos de las subclases
        EmpleadoAsalariado empleadoAsalariado = new EmpleadoAsalariado("John", "Smith", "111-11-1111", 800.00);
        EmpleadoPorHoras empleadoPorHoras = new EmpleadoPorHoras("Karen", "Price", "222-22-2222", 16.75, 40);
        EmpleadoPorComision empleadoPorComision = new EmpleadoPorComision("Sue", "Jones", "333-33-3333", 10000, .06);
        EmpleadoBaseMasComision empleadoBaseMasComision = new EmpleadoBaseMasComision("Bob", "Lewis", "444-44-4444",
            5000, .04, 300);

        System.out.println("Empleados procesados por separado: ");
        System.out.printf("%n%s%n%s: $%,.2f%n%n", empleadoAsalariado, "ingresos", empleadoAsalariado.ingresos());
        System.out.printf("%s%n%s: $%,.2f%n%n", empleadoPorHoras, "ingresos", empleadoPorHoras.ingresos());
        System.out.printf("%s%n%s: $%,.2f%n%n", empleadoPorComision, "ingresos", empleadoPorComision.ingresos());
        System.out.printf("%s%n%s: $%,.2f%n%n", empleadoBaseMasComision, "ingresos", empleadoBaseMasComision.ingresos());

        // crea un arreglo Empleado de cuatro elementos
        Empleado[] empleados = new Empleado[4];

        // inicializa el arreglo con objetos Empleado
        empleados[0] = empleadoAsalariado;
        empleados[1] = empleadoPorHoras;
        empleados[2] = empleadoPorComision;
        empleados[3] = empleadoBaseMasComision;

        System.out.println("Empleados procesados en forma polimorfica:%n%n");

        // procesa en forma genérica a cada empleado en el arreglo de empleados
        for (Empleado empleadoActual : empleados)
        {
            System.out.println(empleadoActual); // invoca a toString

            // determina si el elemento es un EmpleadoBaseMasComision
            if(empleadoActual instanceof EmpleadoBaseMasComision)
            {
                // conversión descendente de la referencia de Empleado
                // a una referencia de EmpleadoBaseMasComision
                EmpleadoBaseMasComision empleado = (EmpleadoBaseMasComision) empleadoActual;

                empleado.establecerSalarioBase(1.10 * empleado.obtenerSalarioBase());

                System.out.printf("El nuevo salario base con 10%% de aumento es: $%,.2f%n",
                    empleado.obtenerSalarioBase());
            } // fin del if

            System.out.printf("ingresos $%,.2f%n%n", empleadoActual.ingresos());
        } // fin de for

        // obtiene el nombre del tipo de cada objeto en el arreglo de empleados
        for(int j = 0; j < empleados.length; j++)
            System.out.printf("El empleado %d es un %s%n", j, empleados[j].getClass().getName());
    } // fin de main
} // fin de la clase PruebaSistemaNomina
```

Clase PruebaEmpleadoPorComision.

```java

public class PruebaEmpleadoPorComision
{
    public static void main(String[] args)
    {
        EmpleadoPorComision empleado = new EmpleadoPorComision("Antonio", "Dominguez",
            "2222-2222", 10000, 0.6);
        System.out.println("Información del empleado obtenida por los metodos establecer: ");
        System.out.printf("%n%s %s%n", "El primer nombre es", empleado.obtenerPrimerNombre());
        System.out.printf("%s %s%n", "El apellido paterno es", empleado.obtenerApellidoPaterno());
        System.out.printf("%s %s%n", "El numero de seguro social es",
            empleado.obtenerNumeroSeguroSocial());
        System.out.printf("%s %.2f%n", "Las ventas brutas son", empleado.obtenerVentasBrutas());
        System.out.printf("%s %.2f%n", "La tarifa de comision es",
        empleado.obtenerTarifaComision());

        empleado.establecerVentasBrutas(500);
        empleado.establecerTarifaComision(.1);

        System.out.printf("%n%s:%n%n%s%n",
            "Informacion actualizada del empleado, obtenida mediante toString",
            empleado.toString());
    }
}

```

Conclusiones:

- Aunque un objeto de una subclase también ***es un*** objeto de una superclase, en realidad ambos son distintos.
- Los objetos de una subclase pueden tratarse como si fueran objetos de la superclase. Sin embargo, como la subclase tiene miembros adicionales que sólo pertenecen a ella, no se permite asignar una referencia de la superclase a una variable de la subclase sin una *conversión explícita*, ya que dicha asignación dejaría a los miembros de la subclase indefinidos para el objeto de la superclase.
- Hemos visto tres maneras apropiadas de asignar referencias de una superclase y de una subclase a las variables de sus tipos:
    - Asignar una referencia de la superclase a una variable de ésta es un proceso simple y directo.
    - Asignar una referencia de la subclase a una variable de éstaes un proceso simple y directo.
    - Asignar una referencia de la subclase a una variable de la superclase es seguro, ya que el objeto de la subclase es un objeto de su superclase. No obstante, la variable de la superclase puede usarse para referirse sólo a los miembros de la superclase. Si este código hace referencia a los miembros que pertenezcan sólo a la subclase, a través de la variable de la superclase, el compilador reporta errores.