Este programa implementa un sistema de empleados utilizando los conceptos de interfaces, clases abstractas, herencia y polimorfismo en Java. El sistema es capaz de manejar diferentes tipos de empleados y calcular sus pagos de manera uniforme.

1. Interfaz Pagable

```
public interface Pagable {
   double calcularPago();
}
```

La interfaz Pagable define un contrato que todas las clases de empleados deben seguir. Este contrato incluye el método calcularPago, que debe ser implementado por todas las clases que representen a un empleado. Esta interfaz asegura que todas las clases que la implementen proporcionen una forma de calcular el pago del empleado.

2. Clase Abstracta Empleado

```
public abstract class Empleado implements Pagable {
   private String nombre;
   private static int contador = 0;
   private int idEmpleado;

   public Empleado(String nombre) {
      this.nombre = nombre;
      contador++;
      this.idEmpleado = contador;
   }

   public abstract double calcularPago();

   @Override
   public String toString() {
      return "Empleado [nombre=" + nombre + ", idEmpleado=" + idEmpleado + "]";
   }
```

```
    public void mostrarPago() {
        System.out.println("Pago: $" + calcularPago());
    }
}
```

La clase abstracta Empleado implementa la interfaz Pagable y define un método abstracto calcularPago que debe ser implementado por las subclases. Esta clase también incluye atributos comunes a todos los empleados, como nombre e idEmpleado, y un método toString para proporcionar una representación en cadena del empleado. La clase abstracta no puede ser instanciada directamente y sirve como plantilla para las subclases.

3. Clases Concretas de Empleados

EmpleadoAsalariado

```
public class EmpleadoAsalariado extends Empleado {
    private double salarioAnual;

    public EmpleadoAsalariado(String nombre, double salarioAnual) {
        super(nombre);
        this.salarioAnual = salarioAnual;
    }

    @Override
    public double calcularPago() {
        return salarioAnual / 12;
    }

    @Override
    public String toString() {
        return "EmpleadoAsalariado [salarioAnual=" + salarioAnual + ", calcularPago()=" +
calcularPago() + " " + super.toString() + "]";
    }
```

```
}
```

EmpleadoAsalariado extiende la clase abstracta Empleado e implementa el método calcularPago, que calcula el pago mensual dividiendo el salario anual entre 12.

EmpleadoPorHora

```
public class EmpleadoPorHora extends Empleado {
    private double tarifaPorHora;
    private int horasTrabajadas;

    public EmpleadoPorHora(String nombre, double tarifaPorHora, int horasTrabajadas) {
        super(nombre);
        this.tarifaPorHora = tarifaPorHora;
        this.horasTrabajadas = horasTrabajadas;
    }

    @Override
    public double calcularPago() {
        return tarifaPorHora * horasTrabajadas;
    }

    @Override
    public String toString() {
        return "EmpleadoPorHora [" + super.toString() + " tarifaPorHora=" + tarifaPorHora + ",
horasTrabajadas=" + horasTrabajadas + ", calcularPago()=" + calcularPago() + "]";
    }
}
```

EmpleadoPorHora extiende la clase abstracta Empleado e implementa el método calcularPago, que calcula el pago multiplicando la tarifa por hora por las horas trabajadas.

EmpleadoContratista

```
public class EmpleadoContratista extends Empleado {
   private double pagoContrato;
   private int duracionMeses;

   public EmpleadoContratista(String nombre, double pagoContrato, int duracionMeses) {
      super(nombre);
      this.pagoContrato = pagoContrato;
      this.duracionMeses = duracionMeses;
   }

   @Override
   public double calcularPago() {
      return pagoContrato / duracionMeses;
   }

   @Override
   public String toString() {
      return "EmpleadoContratista [" + super.toString() + "Pago Contrato=" + pagoContrato + ",
duracionMeses=" + duracionMeses + ", calcularPago()=" + calcularPago() + "]";
   }
}
```

EmpleadoContratista extiende la clase abstracta Empleado e implementa el método calcularPago, que calcula el pago mensual dividiendo el pago total del contrato por la duración del contrato en meses.

4. Clase Principal

```
public class Principal {
   public static void main(String[] args) {
```

```
        Empleado empleado1 = new EmpleadoAsalariado("Juan Perez", 120000);
        Empleado empleado2 = new EmpleadoPorHora("Pedro Perez", 150, 30);
        Empleado empleado3 = new EmpleadoContratista("Sergio Gomez", 60000, 3);

        System.out.println(empleado1);
        empleado1.mostrarPago();

        System.out.println(empleado2);
        empleado2.mostrarPago();

        System.out.println(empleado3);
        empleado3.mostrarPago();
    }
}
```

La clase Principal demuestra el uso de las clases concretas de empleados. Crea instancias de EmpleadoAsalariado, EmpleadoPorHora y EmpleadoContratista, y muestra sus detalles y pagos.

Conceptos Aplicados

1. Polimorfismo

El polimorfismo se logra mediante el uso de la interfaz Pagable y la clase abstracta Empleado. Permite tratar a diferentes tipos de empleados de manera uniforme. En el método main, todos los empleados son tratados como instancias de Empleado, pero se comportan de acuerdo con su tipo específico cuando se llama a calcularPago o toString.

Empleado empleado1 = new EmpleadoAsalariado("Juan Perez", 120000);

Empleado empleado2 = new EmpleadoPorHora("Pedro Perez", 150, 30);

Empleado empleado3 = new EmpleadoContratista("Sergio Gomez", 60000, 3);

2. Herencia

La herencia se utiliza para crear clases específicas de empleados que extienden la clase abstracta Empleado. Esto permite que las subclases hereden los atributos y métodos de Empleado, mientras proporcionan implementaciones específicas del método calcularPago.

3. Interfaces

La interfaz Pagable define un contrato que asegura que todas las clases de empleados implementen el método calcularPago. Esto permite que el código trate a cualquier objeto que implemente Pagable de la misma manera.

4. Abstracción

La clase abstracta Empleado proporciona una plantilla para las clases específicas de empleados. Define atributos y métodos comunes y declara un método abstracto calcularPago que debe ser implementado por las subclases. Esto permite que las subclases se centren en los detalles específicos de su implementación, mientras heredan el comportamiento común de la clase abstracta.