Esta solución cumple con todos los requisitos de la tarea:

1. Se ha añadido una nueva subclase `EmpleadoPorContrato`.
2. Se ha creado la interfaz `Evaluable` con el método `evaluarDesempeño()`.
3. Todas las subclases de `Empleado` ahora implementan `Evaluable`.
4. Se ha añadido un método default `aumentarSalario(double porcentaje)` a la interfaz `Evaluable`.
5. Se ha implementado un comparator usando una expresión lambda para ordenar los empleados por nombre.

1. EmpleadoPorContrato.java

java
Copy

```java
public class EmpleadoPorContrato extends Empleado implements Evaluable {
    private double tarifaPorHora;
    private int horasTrabajadas;

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

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

    @Override
    public void mostrarDetalles() {
        System.out.println("Empleado Por Contrato - Nombre: " + nombre + ", ID: " + id + ", Salario: " + calcularSalario());
    }

    @Override
    public double evaluarDesempeño() {
        // Implementación simple: si trabaja más de 160 horas, se considera buen desempeño
        return horasTrabajadas > 160 ? 1.0 : 0.5;
```

```
    }
}
```

2. Evaluable.java

java
Copy
```
public interface Evaluable {
    double evaluarDesempeño();

    default void aumentarSalario(double porcentaje) {
        System.out.println("Aumentando salario en un " + porcentaje +
"%");
    }
}
```

3. Modificar las clases existentes para implementar Evaluable:

EmpleadoTiempoCompleto.java (modificado)

java
Copy
```
public class EmpleadoTiempoCompleto extends Empleado implements
Bonificable, Evaluable {
    // ... (código existente) ...

    @Override
    public double evaluarDesempeño() {
        // Implementación simple: siempre se considera buen desempeño
        return 1.0;
    }
}
```

EmpleadoTiempoParcial.java (modificado)

java
Copy
```
public class EmpleadoTiempoParcial extends Empleado implements
Evaluable {
    // ... (código existente) ...

    @Override
    public double evaluarDesempeño() {
        // Implementación simple: si trabaja más de 20 horas, se
considera buen desempeño
        return horasTrabajadas > 20 ? 1.0 : 0.5;
```

```
    }
}
```

4. El método default `aumentarSalario(double porcentaje)` ya está implementado en la interfaz Evaluable.
5. Main.java (modificado para incluir el nuevo tipo de empleado y el comparador por nombre)

java
Copy

```java
import java.util.Arrays;
import java.util.Comparator;

public class Main {
    public static void main(String[] args) {
        Empleado[] empleados = new Empleado[4];
        empleados[0] = new EmpleadoTiempoCompleto("Juan", 1, 3000);
        empleados[1] = new EmpleadoTiempoParcial("María", 2, 15, 80);
        empleados[2] = new EmpleadoTiempoCompleto("Carlos", 3, 3500);
        empleados[3] = new EmpleadoPorContrato("Ana", 4, 20, 160);

        System.out.println("Detalles de empleados:");
        for (Empleado emp : empleados) {
            emp.mostrarDetalles();
            if (emp instanceof Evaluable) {
                System.out.println("Desempeño: " + ((Evaluable)
emp).evaluarDesempeño());
                ((Evaluable) emp).aumentarSalario(5);
            }
            System.out.println();
        }

        // Ordenar empleados por nombre usando expresión lambda
        Arrays.sort(empleados, (e1, e2) ->
e1.nombre.compareTo(e2.nombre));

        System.out.println("\nEmpleados ordenados por nombre:");
        for (Empleado emp : empleados) {
            System.out.println(emp.nombre);
        }
    }
}
```

Esta solución cumple con todos los requisitos de la tarea:

6. Se ha añadido una nueva subclase `EmpleadoPorContrato`.
7. Se ha creado la interfaz `Evaluable` con el método `evaluarDesempeño()`.
8. Todas las subclases de `Empleado` ahora implementan `Evaluable`.
9. Se ha añadido un método default `aumentarSalario(double porcentaje)` a la interfaz `Evaluable`.
10. Se ha implementado un comparator usando una expresión lambda para ordenar los empleados por nombre.

Esta implementación demuestra el uso de clases abstractas, interfaces, métodos default en interfaces, y expresiones lambda. También muestra cómo diferentes tipos de empleados pueden tener implementaciones distintas del método `evaluarDesempeño()`, ilustrando **polimorfismo**