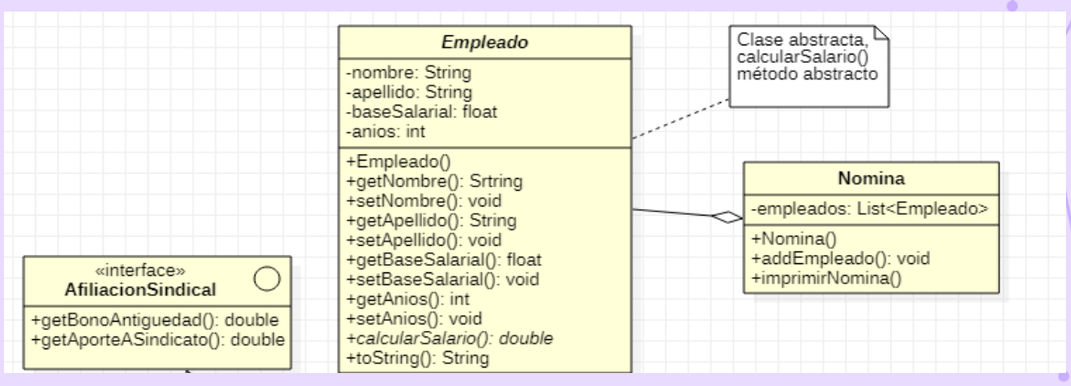
Ejemplo práctico clases abstractas e interfaces

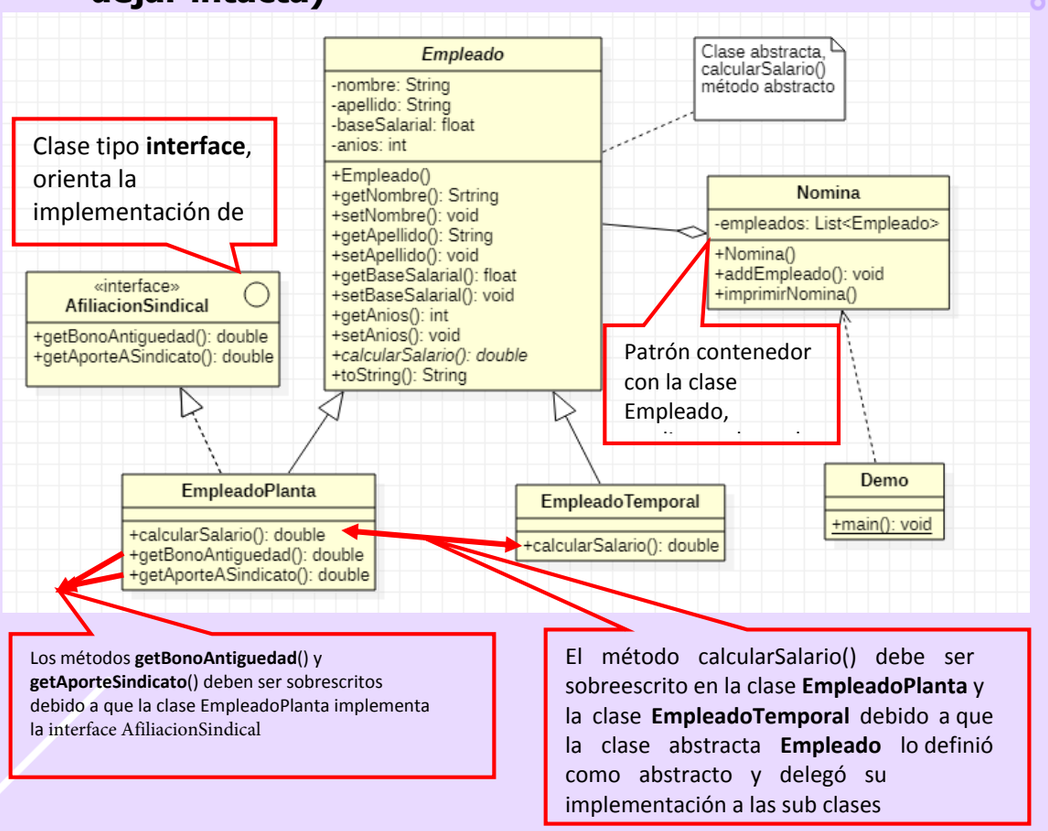
En la empresa soluciones clic derecho, se requiere un software que permita calcular el salario neto a pagar para sus empleados, el líder del equipo de desarrollo, dispuso 3 clases base para el desarrollo según lo acordado con el profesional encargado del área de pagos:

- La clase abstracta Empleado
- La Interfaz AfiliacionSindical
- La clase Nomina



Y dio las siguientes consideraciones:

- Se debe implementar la clase **EmpleadoPlanta**
- Se debe implementar la clase **EmpleadoTemporal**
- La clase Nomina (**se debe dejar intacta)**
- La clase Empleado (**se debe dejar intacta) Clase abstracta**
- Existe una clase Demo con un método *main* para probar que las funcionalidades estén correctas con un método que imprime el resultado de los cálculos realizados y la información de cada objeto. (**se debe dejar intacta)**



Basado en la anterior información:

a. Implemente en Java la clase **EmpleadoTemporal**

- Aplique herencia de la clase Empleado

```
public class EmpleadoTemporal extends Empleado{

}
```

- Incluya el constructor de la clase y realice encadenamiento

```
public EmpleadoTemporal(String nombre, String apellido, float baseSalarial, int anios) {
    super(nombre,apellido,baseSalarial,anios);
}
```

- Sobreescriba el método **calcularSalario,** teniendo en cuenta que la fórmula para el cálculo del salario está dada por la siguiente fórmula: $salario = 2.5 * baseSalarial - baseSalarial * 0.286$

```
@Override
public  double calcularSalario(){
    return 2.5*getBaseSalarial()-getBaseSalarial()*0.286;
}
```

b. Implemente en Java la clase **EmpleadoPlanta**

- Aplique herencia de la clase Empleado

```
public class EmpleadoPlanta extends Empleado
```

- Incluya el constructor de la clase y realice encadenamiento

```
public EmpleadoPlanta(String nombre, String apellido, float baseSalarial, int anios) {
    super(nombre,apellido,baseSalarial,anios);
}
```

- Implemente la interfaz AfiliacionSindical

```
public class EmpleadoPlanta extends Empleado implements AfiliacionSindical{
```

- Sobreescriba el método getBonoAntiguedad, teniendo en cuenta que el valor del bono se da así:
  - Si el empleado lleva más de 10 años en la empresa, se calcula su valor como el 10% de la base salarial
  - Si el empleado lleva 10 años o menos en la empresa, se calcula su valor como el 5% de la base salarial

```
@Override
public double getBonoAntiguedad(){

    if(getAnios()>10){
        return getBaseSalarial()*0.1;
    }else{
        return getBaseSalarial()*0.05;
    }

}
```

- Sobreescriba el método getAporteASindicato, teniendo en cuenta que el valor del aporte se da así:
  - Si el empleado lleva más de 10 años en la empresa, se calcula su

valor como el 3% de la base salarial
- Si el empleado lleva 10 años o menos en la empresa, se calcula su valor como el 1.5% de la base salarial

```
@Override
public double getAporteASindicato(){
    if(getAnios()>10){
        return getBaseSalarial()*0.03;
    }else{
        return getBaseSalarial()*0.015;
    }
}
```

- Sobreescriba el método **calcularSalario,** teniendo en cuenta que la fórmula para el cálculo del salario está dada por la siguiente fórmula: $2.5 * baseSalarial + bonoAntiguedad - aporteASindicato$

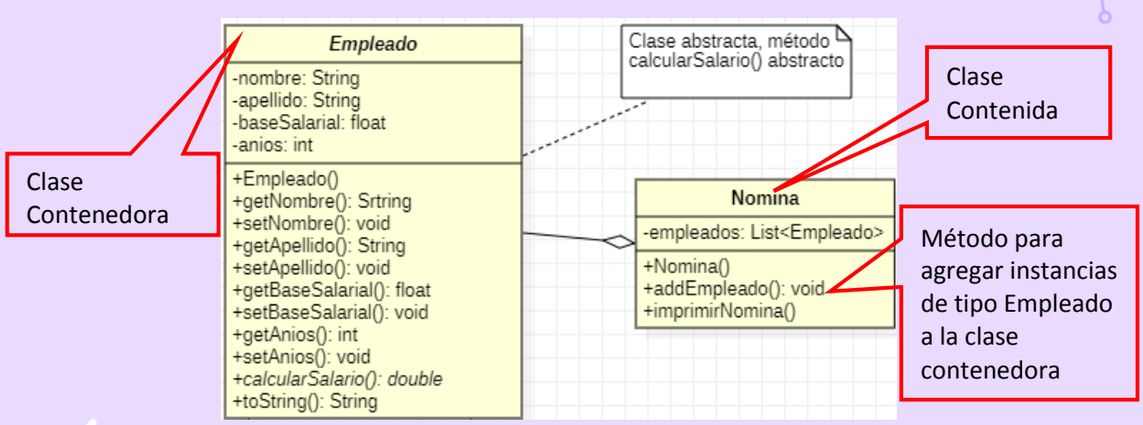
```
@Override
public  double calcularSalario(){
    return 2.5*getBaseSalarial()+getBonoAntiguedad()-getAporteASindicato();
}
```

Notas:

- Tenga en cuenta las consideraciones dadas por el líder del equipo de desarrollo y no modifique las clases que pidió que se dejen intactas
- Analice detenidamente las clases que se dieron como base
- Descargue el proyecto base del siguiente enlace para realizar el ejercicio por su cuenta: https://drive.google.com/drive/folders/1ijKRLYMzCrDmtW7mTNh32vy1xOVXY06S?usp=sharing

Patrón contenedor

En este mismo ejercicio, podemos ver la aplicación de un patrón contenedor mediante el uso de la Interface **List** y su implementación **ArrayList** ***(ver código fuente)***, note que la clase contenedora es abstracta, así que puede referenciar instancias de los subtipos debido a que al ser definida como abstracta, no permite la creación de objetos de tipo Empleado, pero nos sirve como variable polimórfica (***recordar el tema polimorfismo***)



Código fuente de la clase contenedora

```java
import java.util.ArrayList;// implementacion de la coleccion List
import java.util.List;//Interfaz de la coleccion list


public class Nomina {

    private final List<Empleado> empleados;

    public Nomina() {
        empleados = new ArrayList<>();
    }

    public void addEmpleado(Empleado e) {
        empleados.add(e);
    }

    public void imprimirNomina(){
        for (Empleado empleado : empleados) {
            System.out.println(empleado);
        }
    }
}
```

Enlaces adicionales

- https://youtu.be/q8ZmVQ8B-tE
- https://youtube.com/playlist?list=PLuLE9s9J8IVDDI350EpPnPJRmjLeuteEV