Programación Orientada a Objetos. Ejemplo práctico

Introducción

Cuando nos enfrentamos a la tarea de **diseñar un sistema informático** tenemos que hacer la **simulación de una serie de modelo conceptuales y físicos**.

Para poder llevar a cabo esta tarea es necesario reducir la complejidad a algo comprensible por el usuario. Las personas **gestionamos la complejidad a través de la abstracción**. Por ejemplo, no pensamos en nuestro coche como una lista de partes insondables, sino que lo vemos como un objeto bien definido con un comportamiento específico. Ignoramos lo que representa el motor y cómo funciona, así como los detalles de funcionamiento de la transmisión o sistema de frenos. Sin embargo, si tenemos la noción de cómo funciona un coche en su conjunto. Tenemos una capacidad poderosa para **gestionar esa abstracción de una forma jerárquica** que nos permite, a nuestra voluntad, dividir en partes sistemas complejos. Desde el exterior, el coche es el coche; pero luego, en su interior, podemos distinguir el volante, los frenos, la radio, los cinturones de seguridad, los sillones, etc.

Los **programas tradicionales se pueden descomponer en los objetos con que se trata el proceso**. Una secuencia de pasos de un proceso **se puede convertir en un trasiego de mensajes entre objetos autónomos**, cada uno con su propio comportamiento. Tratamos estos objetos como e**ntidades del mundo real que responden a mensajes que es dicen que hagan algo**. Como ya se ha expuesto, esta es la esencia de la **Programación Orientada a Objetos**.

Para comprender todo lo aprendido de la POO: sus elementos (clases y objetos) junto a sus principios (encapsulación, herencia y polimorfismo) se desarrollará un ejemplo práctico: un Sistema de Gestión de Nóminas de Empleados.

Para ello, se distinguen diferentes clases las cuales se deberán modelar.

Descripción del Sistema de Gestión de Nóminas de Empleados

Se trata de crear un sistema informático que permita calcular las nóminas de los distintos tipos de empleados, teniendo en cuenta los siguientes requisitos.

Todos los empleados vienen definidos por los siguientes datos: con los datos nombre, apellidos, identificador de empleado, sexo (‘M’/’H’/’?’), teléfono y email

Luego, dependiendo del tipo de empleado pueden tener datos concretos. Dentro de los empleados, se pueden distinguir empleados:

- empleado jefe: con los datos nombre, apellidos, identificador de empleado, sexo (‘M’/’H’/’?’), teléfono, email y salario semanal
- empleado con comisión: con los datos nombre, apellidos, identificador de empleado, sexo (‘M’/’H’/’?’), teléfono, email, un salario base por semana, una comisión por artículo vendido y la cantidad total de artículos vendidos por semana.
- empleado por hora: con los datos nombre, apellidos, identificador de empleado, sexo (‘M’/’H’/’?’), teléfono, email, un salario por hora y el número de horas trabajadas a la semana.
- Empleado por piezas: con los datos nombre, apellidos, identificador de empleado, sexo (‘M’/’H’/’?’), teléfono, email, un salario por pieza producida y la cantidad semana de piezas hechas.

El sistema informático podrá crear objetos de diferentes tipos de empleados, mostrar toda su información, acceder/modificar el valor de sus propiedades y calcular el sueldo en función del empleado que sea.

Analizando el problema

Para abordar el problema, aplicaremos la Programación Orientada a Objetos en las que iremos identificando las clases necesarias y estableciendo relaciones entre ellas.

Para ello, deberemos comenzar a definir una clase con todos los atributos y comportamientos comunes a todos los empleados. De esta forma, crearemos una primera clase base de la cual pueden ir heredando otras subclases en las que almacenaremos los comportamientos y características propias.

¿Qué atributos son comunes para todos los tipos de empleados?

Revisando los requisitos todos los empleados tienen los siguientes datos: nombre, apellidos, identificador de empleado, sexo ('M'/'H'/'?'), teléfono y email. Por tanto, podemos crear una clase Empleado que contenga todos estos atributos.

Todos los empleados tienen un salario, pero no es igual para todos ya que dependiendo del tipo de empleado que sea el salario será diferente.

Así pues, podemos distinguir para los tipos de empleados los siguientes atributos propios para cada uno:

- empleado jefe: salario semanal
- empleado con comisión: un salario base por semana, una comisión por artículo vendido y la cantidad total de artículos vendidos por semana.
- empleado por hora: un salario por hora y el número de horas trabajadas a la semana.
- Empleado por piezas: un salario por pieza producida y la cantidad semana de piezas hechas.

Por tanto, nos crearemos cuatro clases más Jefe, EmpleadoComision, EmpleadoHora y EmpleadoPieza incluyendo en cada una de ellas sus propios atributos.

De tal manera, que estas cuatro clases serán subclases de la clase Empleado; heredando así todos sus atributos y comportamientos.

¿Qué comportamientos son comunes a todas las clases?

Todos los empleados deberán poder los siguientes comportamientos:

- Constructor de cada clase. Con los atributos propios de cada clase e invocando al constructor de su super clase para poder asignar los atributos heredados
- Acceder a sus atributos. Para ello, se crearán los métodos getters y setters para cada uno de los atributos propios de la clase. Pudiendo llamar a los heredados de su superclase.
- Mostrar la información total de los empleados. Nos referimos a todos los valores de todos los atributos. Para ello, deberemos mostrar los atributos propios y llamar a mostrar toda la información de la super clase.

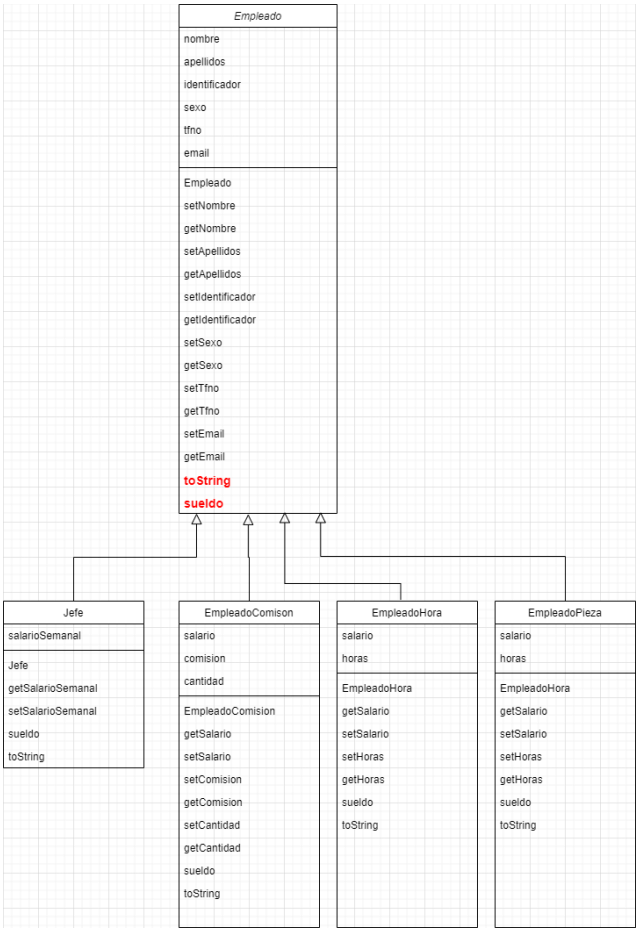
- Mostrar el sueldo que gana cada empleado. Este método tendrá implementaciones diferentes en función del tipo de empleado que lo esté calculando. Recordamos los los sueldos serían:
    - Para los empleados jefe sería su sueldo semanal
    - Para los empleados con comisión su sueldo sería su salario base más la comisión por la cantidad de productos vendidos
    - Para los empleados por hora su sueldo será su salario por el número de horas.
    - Para los empleados por pieza su sueldo será la cantidad de piezas producidas por el salario por pieza

Una vez recopilado todos los comportamientos observamos que todos estarán en la clase Empleado, sin embargo para algunos de ellos no podremos implementarlos. Esto sucederá para los comportamientos de calcular el sueldo y mostrar la información, ya que en la clase Empleado desconocemos de qué tipo de empleado se trata. Estos métodos serán abstractos y por tanto, la clase Empleado será abstracta.

Se recuerda que al definir la clase como abstracta no podremos crear ninguna instancia de la misma, teniendo que elegir entre alguna de las subclases.

Diagrama de clases con UML

Por tanto, se puede comenzar a modelar las clases necesarias tal y como se muestra en la siguiente imagen. Se recuerda que se utiliza UML



Según el diagrama la superclase o clase madre es la clase Empleado. Las clases Jefe, EmpleadoComision, EmpleadoHora y EmpleadoPieza son subclases o clases hijas. De tal manera, que las subclases heredan las propiedades y métodos de la clase hija. Para ello, se está aplicando la **Herencia**.

Los métodos señalados en color rojo serán métodos abstractos los cuales no podremos implementar en la clase Empleado. Se implementarán en las clases hijas las cuales. Recuerda que esto es una manera de aplicar el **Polimorfismo**.

Así mismo, aunque e el diagrama de clases solo aparece un método constructor para cada clase se podrán implementar varios con diferentes parámetros. De tal manera que mediante la sobrecarga de métodos también estaremos aplicando el Polimorfismo.

Codificación de las clases

Comenzamos creando una carpeta en la que iremos creando todas las clases. Podemos incluir estas clases dentro del paquete *sistemanominasv1*

Comenzamos creando el primero fichero Empleado.java que será la clase abstracta, dentro de la cual definimos los atributos y con la opción del entorno de desarrollo crearemos rápidamente sus métodos constructor, getters y setters.

Codificación del programa principal

Una vez que hemos creado las clases, crearemos otra clase que contenga la función main donde se crearán los diferentes objetos empleados y se probarán los métodos creados.