	Carátula para entrega de prácticas
Facultad de Ingeniería	Laboratorio de docencia

Ingeniería en computación

Laboratorios de computación salas A y B

Profesor: Ing. Rene Adrian Davila Perez

Asignatura: Programación Orientada a Objetos

Grupo: 1

No. de práctica: 7

Integrante: 424121462

Semestre: 2025-1

Fecha de entrega: 4 de octubre de 2024

Observaciones:

CALIFICACIÓN: _____

Índice

Introducción:

La práctica se estructurará en varios ejercicios, cada uno enfocado en resolver problemas específicos. Escribir un programa en Java que implemente una jerarquía de clases para empleados de una compañía. La clase base será Empleado, de la cual derivarán las subclases Manager, Desarrollador y Programador. Cada subclase debe incluir atributos como nombre, dirección, salario y título del trabajo, además de implementar métodos sobrescritos para calcular bonos, generar reportes de desempeño y gestionar proyectos. Es importante que los métodos de generación de reportes y manejo de proyectos se presenten mediante cadenas .

Adicionalmente, se deberán cumplir los siguientes requisitos:

- El ejercicio debe estar encapsulado.
- Se debe definir en un paquete.
- Se debe generar jar y javadoc.
- El reporte debe incluir diagrama de clases indicando la herencia.

El objetivo de estos ejercicios es profundizar en el uso de la herencia y comprender su funcionamiento de manera práctica. Se busca familiarizarse con la técnica de sobrescritura, explorando cómo esta puede modificar métodos definidos en clases superiores. A su vez, se pretende reforzar conocimientos previos sobre la ejecución y compilación de archivos jar y y javadocs.

Se llevará a cabo esta práctica utilizando las mejores herramientas y convenciones posibles, ya que esto no solo mejora la calidad del código, sino que también tiene un impacto en el ámbito laboral. La implementación de comentarios claros y una buena documentación son importantes para facilitar el trabajo en equipo, lo que permite una solución más eficiente de los proyectos. Esta práctica, al ser estructurada como una tarea impuesta por un examinador o una empresa, tiene un enfoque práctico que refleja las demandas del entorno profesional al cual nos enfrentaremos en un futuro.

Marco Teórico:

Hay dos tipos de herencia: Herencia Simple y Herencia Múltiple. La primera indica que se pueden definir nuevas clases solamente a partir de una clase inicial mientras que la segunda indica que se pueden definir nuevas clases a partir de dos o más clases. Java sólo permite herencia simple.

La herencia permite crear una nueva clase basada en una clase existente. El concepto conduce a una estructura jerárquica de árbol, lo cual nos dice que todas las relaciones entre clases deben ajustarse a dicha estructura. En esta estructura jerárquica, cada clase tiene sólo una clase padre.

La clase existente se llama superclase o clase padre y la nueva clase se llama subclase o clase hija. La subclase hereda propiedades y métodos de la superclase, lo que permite la reutilización de código y nos brinda una estructura más organizada y reciclada. [1]

La sobrescritura permite a una clase hija proporcionar una implementación más específica de un método que ya está definido en su clase padre. Mayormente se ve involucrado a la hora de cambiar o ampliar el comportamiento de un método heredado. También se cuenta con reglas para la sobrescritura [2]:

- Mismo nombre y parámetros: El método sobrescrito debe tener el mismo nombre y el mismo método de retorno que el método en la clase padre.
- Acceso: El modificador de acceso del método sobrescrito en la clase hija no puede ser más restrictivo que el de la superclase.
- No se puede sobrescribir métodos estáticos.

Desarrollo:

Para la clase (padre) **Empleado** se tiene:

- nombre: String
- direccion: String
- nombreDeTrabajo: String
- salario: double
- Métodos set y get de cada una de las variables
- Método constructor
- getBono(): retorna un double
- reporteDesempeño(@reporte): retorna una cadena vacía
- manejoDeProyectos(@proyectos): retorna una cadena vacía
-

Para la clase (hija de **Empleado**) **Manager** se tiene:

- Parámetros iguales que los de la clase padre **Empleado**
- Métodos set y get de cada una de las variables
- Método constructor
- getBono(): retorna el salario * 0.25. Método sobreescrito
- reporteDesempeño(@reporte): retorna una cadena respecto al desempeño realizado dado la cadena reporte. Método sobreescrito
- manejoDeProyectos(@proyectos): retorna una cadena que muestra proyectos en los que trabaja el manager dada la cadena proyectos. Método sobreescrito
-

Para la clase (hija de **Empleado**) **Desarrollador** se tiene:

- Parámetros iguales que los de la clase padre **Empleado**
- Métodos set y get de cada una de las variables
- Método constructor
- getBono(): retorna el salario * 0.15. Método sobreescrito
- reporteDesempeño(@reporte): retorna una cadena respecto al desempeño realizado dado la cadena reporte. Método sobreescrito
- manejoDeProyectos(@proyectos): retorna una cadena que muestra proyectos en los que trabaja el manager dada la cadena proyectos. Método sobreescrito
-

Para la clase (hija de **Empleado**) **Programador** se tiene:

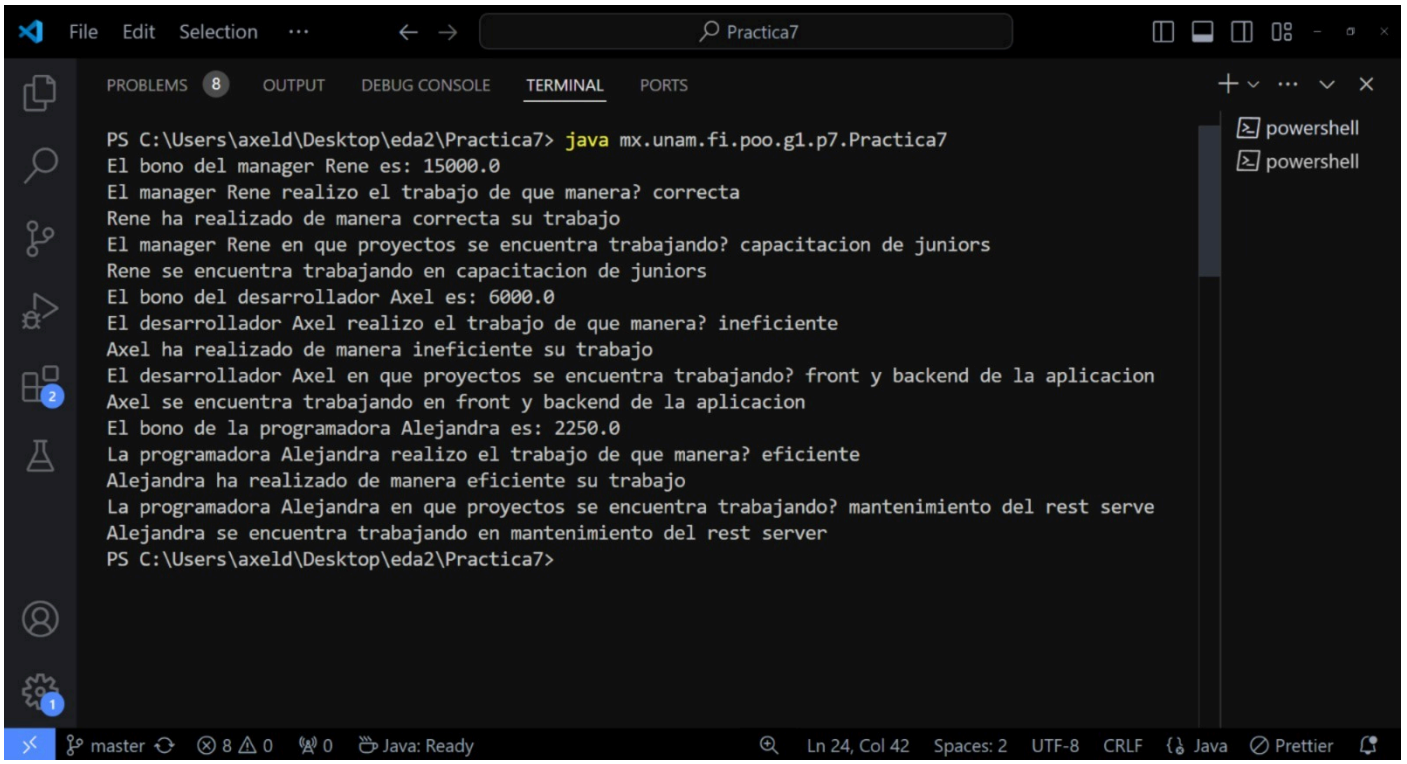
- Parámetros iguales que los de la clase padre **Empleado**
- Métodos set y get de cada una de las variables
- Método constructor
- getBono(): retorna el salario * 0.05. Método sobreescrito
- reporteDesempeño(@reporte): retorna una cadena respecto al desempeño realizado dado la cadena reporte. Método sobreescrito
- manejoDeProyectos(@proyectos): retorna una cadena que muestra proyectos en los que trabaja el manager dada la cadena proyectos. Método sobreescrito

Para el archivo ejecutable **Practica7** se tiene:

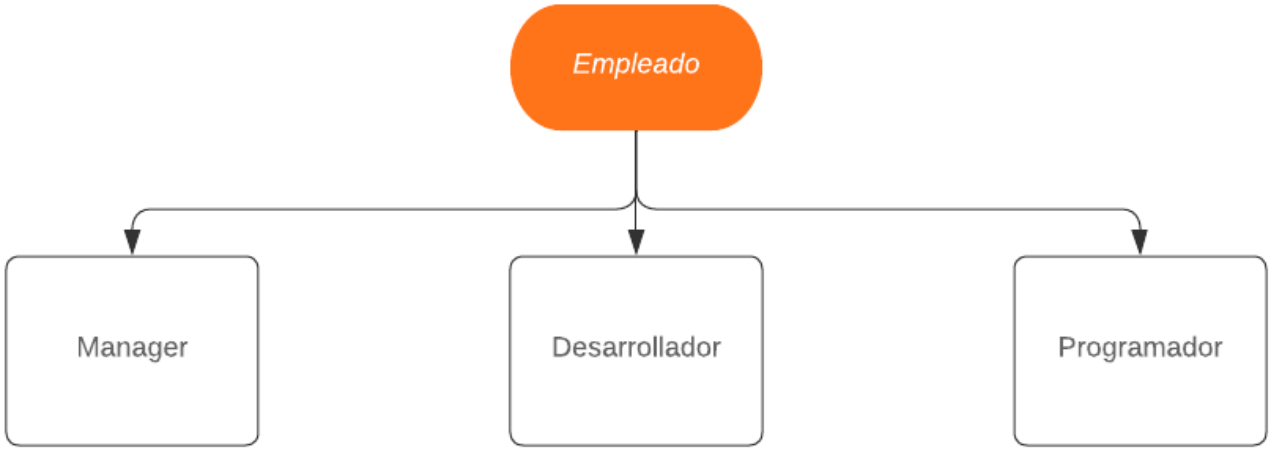
- Método main(@args)

Se le solicita al usuario que ingrese alguna palabra referente a cómo se realizó el trabajo (bien, eficiente, mal, etc) para así generar el reporte, también se le solicitará al usuario que de uno o varios proyectos para la función correspondiente. Para probar que se ejecute de la manera esperada se ingresaron las siguientes respuestas: “correcta” y “capacitación de juniors” para el caso del Manager, “ineficiente” y “front y backend de la aplicación” para el caso del Desarrollador, también se ingresó “eficiente” y “mantenimiento del rest server” para el caso de Programador. La respuesta por parte del programa fue la esperada pero podría presentar ciertos problemas por la manera en la que están redactadas las cadenas mostradas, por lo que se podría hacer de manera más específica o más elaborada para así tener mejores funciones.

Resultados:



En la imagen se muestra la salida en pantalla de lo ingresado, a su vez también se muestran algunos de los valores definidos en el main para cada una de las clases.



En la imagen se muestra el diagrama de clases indicando la herencia.

Conclusiones:

Conocer de la herencia y sobrescritura en Java nos ayuda a realizar una práctica eficiente. Estos conceptos proporcionan una fuente sólida para entender cómo estructurar el código de manera eficiente y organizada, facilitando la reutilización y la extensibilidad. Sin una buena comprensión teórica, sería difícil aplicar correctamente estas herramientas en proyectos reales, lo que puede llevar a un código desorganizado y menos mantenible. Por lo tanto, dominar la teoría no solo optimiza el desarrollo, sino que también mejora la calidad y la claridad del software.

Referencias:

[1] "III - Herencia," UNAM. [En línea]. Disponible:

http://profesores.fi-b.unam.mx/carlos/java/java_basico3_4.html. [Accedido: Oct. 4, 2024].

[2] ChatGPT. [En línea]. Disponible: https://chatgpt.com. [Accedido: Sep. 29, 2024]. Pregunta:

“Platícame más acerca de la sobrescritura en Java”