|  |  |  |
| --- | --- | --- |
|  | **Carátula para entrega de prácticas** | |
| Facultad de Ingeniería | | Laboratorio de docencia |

Laboratorios de computación

salas A y B

|  |  |
| --- | --- |
| *Profesor:* | Ing. Guadalupe Lizeth Parrales Romay |
| *Asignatura:* | Programación Orientada a Objetos |
| *Grupo:* | 04 |
| *No de Práctica(s):* | 7 |
| *Integrante(s):* | Dávila Ortega Jesús Eduardo - No. Cuenta: 317199860  Díaz Hernández Marcos Bryan - No. Cuenta: 317027253  Pareja Avila Emiliano - No. Cuenta: 317081345  Vazquez Zavala Oliver Alexis - No. Cuenta: 317202263 |
| *No. de Equipo de cómputo empleado:* |  |
| *No. de Lista o Brigada:* | 12,14, 30, 37 |
| *Semestre:* | 2021-1 |
| *Fecha de entrega:* | 21 de noviembre del 2020 |
| *Observaciones:* |  |
|  |  |

CALIFICACIÓN: \_\_\_\_\_\_\_\_\_

**Objetivo de la práctica.**

Implementar los conceptos de herencia en un lenguaje de programación orientado a

objetos.

**Introducción.**

La herencia es uno de los conceptos fundamentales en el paradigma orientado a objetos, se define como la propiedad que permite que los objetos sean creados o instanciados a partir de otros ya existentes, obteniendo o “heredando” métodos y atributos, similares a los ya existentes de una cierta clase. Es la relación entre una clase general y otra clase más específica, dicho mecanismo permite crear clases derivadas o clases hijas a partir de una o más clases base o clases padre , permitiendo la reutilización de código y el poder compartir automáticamente métodos y datos entre clases, subclases y objetos.

En java, y la gran mayoría de los lenguajes de programación orientados a objetos, la herencia está siempre presente al desarrollar un programa, sin importar la extensión de este, lo anterior debido a que independientemente de declarar una clase como herencia de otra u otras, todas las clases desarrolladas se derivan implícitamente de una superclase llamada Object, la cual es la clase raíz de toda la jerarquía de clases de Java. En consecuencia de lo anterior todas las clases comparten ciertos métodos heredados de la clase Object, tal es el caso del método “equals()” que se utiliza para comparar si dos objetos son iguales, este método devuelve true si los objetos son iguales, o false si no lo son, el método “clone()” el cual crea un nuevo objeto, igual al que se duplica, o el método “toString()” que devuelve una cadena de texto que representa al objeto, por mencionar solo algunos de los métodos heredados de la clase Object.

Para hacer herencia en java se emplea la palabra reservada extends en la declaración de la clase, la sintaxis es la siguiente:

*[modificadores] class NombreClaseDerivada extends NombreClaseBase*

Por ejemplo para indicar que la clase Perro (clase hija o subclase) hereda de la clase Animal (clase padre o superclase), la declaración es la siguiente:

*public class Perro extends Animal {*

*// Código clase Perro*

*}*

Para poder instanciar o crear objetos de las clases derivadas (subclases), esto se realiza de forma idéntica que al instanciar objetos de las clases que se definen sin una relación de herencia explícita, es decir las clases que solo heredan de Object.

Otros conceptos importantes en el desarrollo de programas orientados a objetos y que es utilizado de forma frecuente cuando se presenta una relación de herencia entre una o más clases, es el concepto de sobreescritura y sobrecarga de métodos.

La sobreescritura de métodos es la propiedad que tiene una clase de cambiar o redefinir el comportamiento de los métodos heredados, esto puede ser requerido debido a diversos motivos, entre ellos puede ser que el comportamiento genérico que se hereda del padre no sea suficiente para la clase hija y se requiera de otro mas especializado o que se requiera transformar a un método heredado completamente con un comportamiento distinto, entre otros casos, las reglas para la sobreescritura de métodos son que un método sobrescrito debe tener el mismo nombre, debe tener el mismo tipo y número de parámetros, el tipo de nivel de acceso debe ser igual o más accesible y el valor de retorno debe ser del mismo tipo o un subtipo.

Por otro lado la sobrecarga de métodos se refiere a la creación de varios métodos con el mismo nombre pero con diferente lista de tipos de parámetros(diferencia principal con la sobreescritura), en el caso de java se utiliza el número y tipo de parámetros para seleccionar de un conjunto de métodos sobrecargados cual ejecutar, ya que java diferencia los métodos sobrecargados con base en el número y tipo de parámetros o argumentos que tiene el método y no por el tipo que devuelve, como podría llegar a creerse, esta propiedad también aplica en los llamados métodos constructores, ya que es posible tener una sobrecarga de métodos constructores en una clase, asi como de otros métodos no constructores.

**Análisis de los ejercicios.**

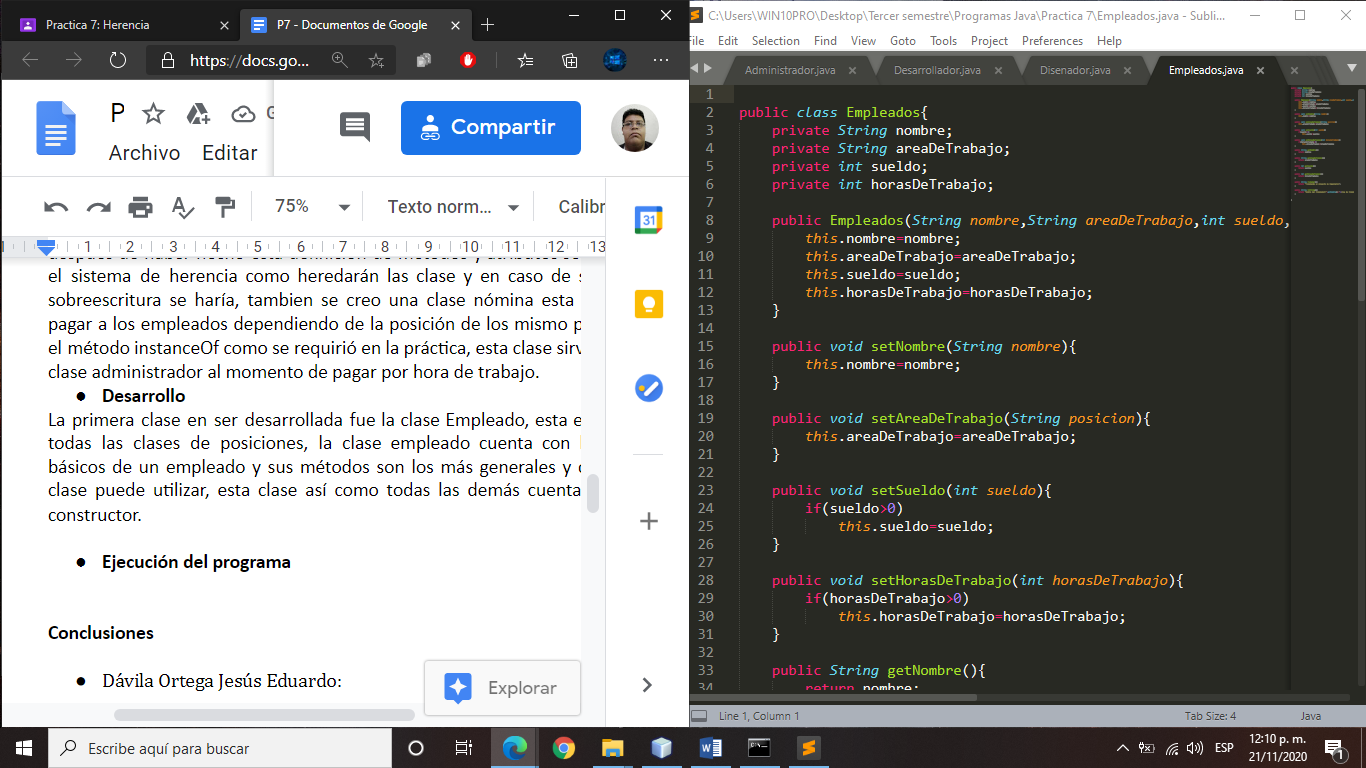
**Ejercicio 1**

* **Análisis previo**

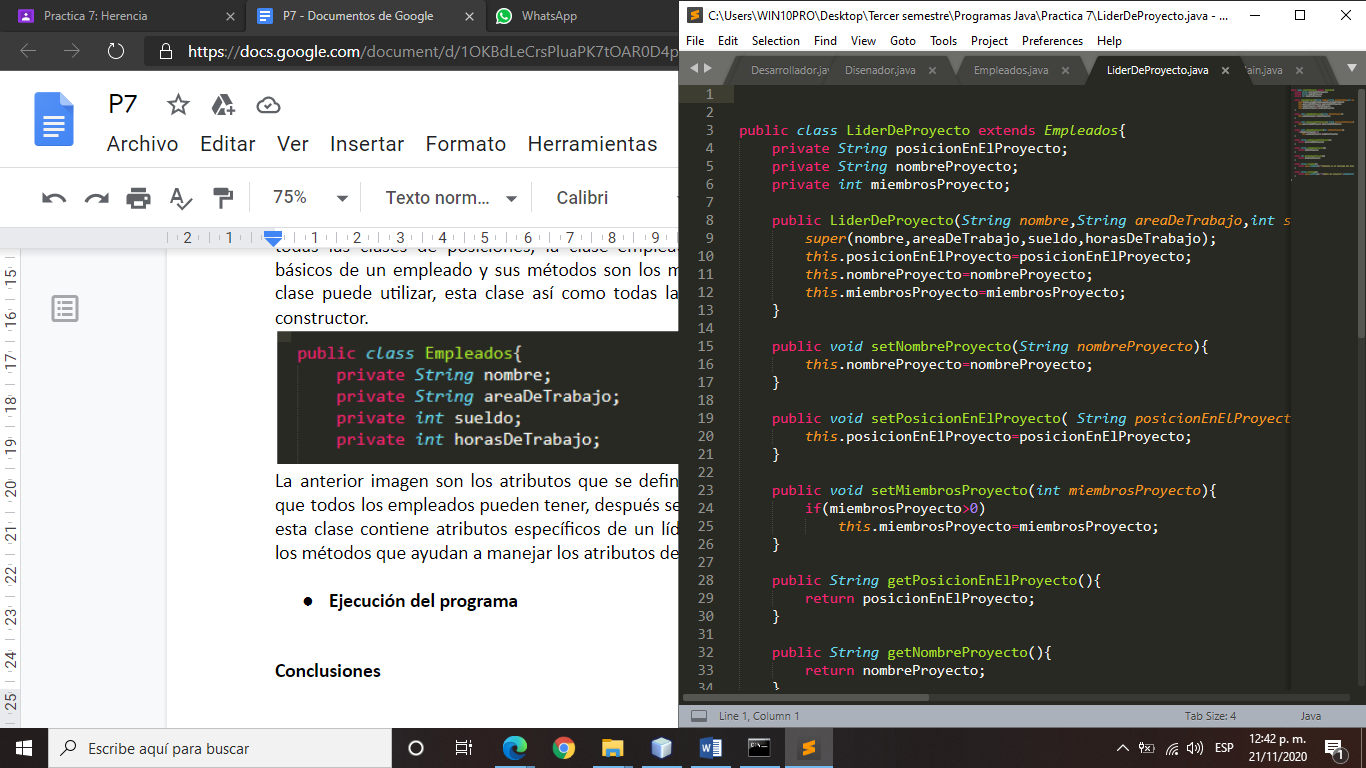
Al momento de analizar cómo se iba a realizar la implementación del problema propuesto, lo primero que hicimos fue decidir cuáles serían las características que tendría cada posición, después de definir los atributos únicos y aquellos que pueden ser generales , se empezaron a definir los métodos de cada tipo de trabajador y en caso de que todos tuvieran ese mismo método se podría considerar meter ese método como el método de empleado así al momento de heredar los hijos tendrían ese método definido lo mismo pasaría con los atributos de cada tipo. Posteriormente después de haber hecho esta definición de métodos y atributos se empezó a diseñar el sistema de herencia como heredarán las clases y en caso de ser necesaria una sobreescritura se haría, también se creó una clase nómina esta tiene como fin el pagar a los empleados dependiendo de la posición de los mismo para esto se utilizó el método instanceOf como se requirió en la práctica, esta clase sirve para apoyar a la clase administrador al momento de pagar por hora de trabajo.

* **Desarrollo**

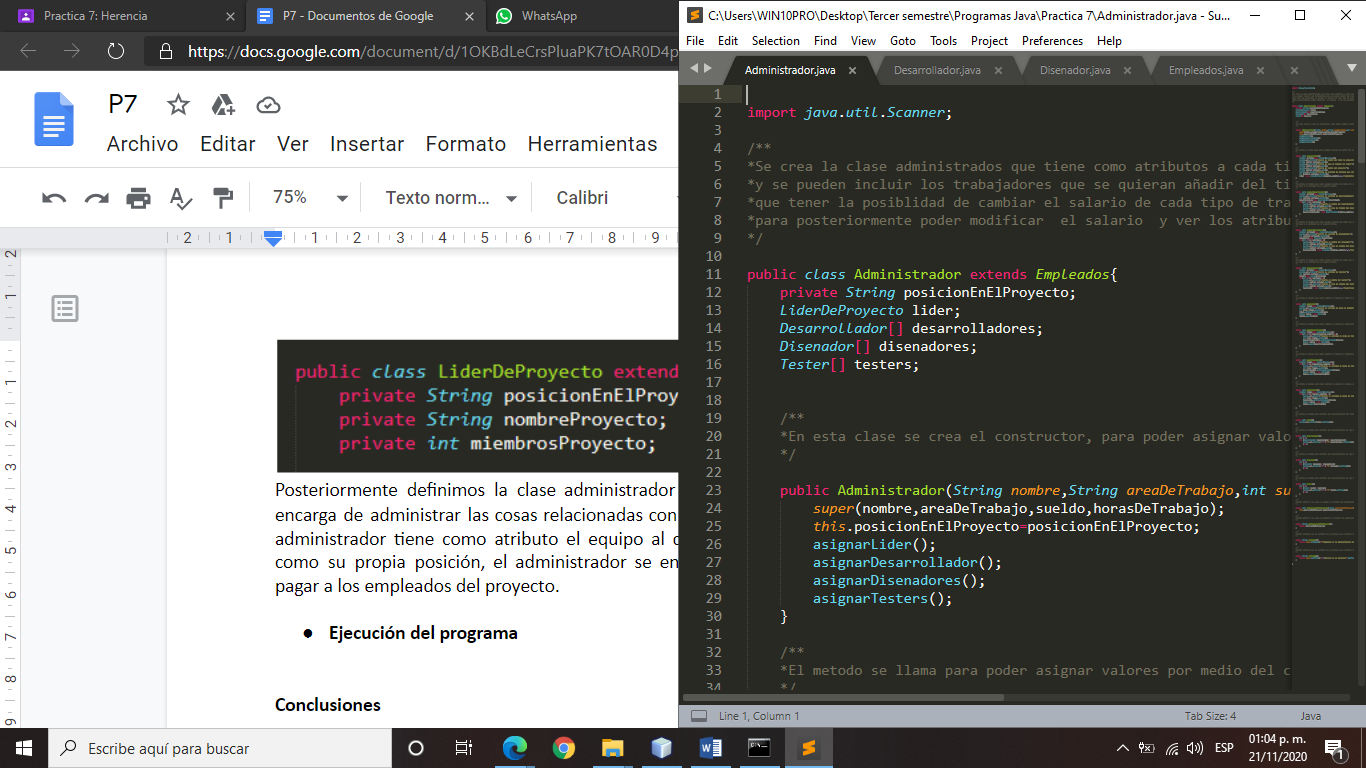
La primera clase en ser desarrollada fue la clase Empleado, esta es la superclase de todas las clases de posiciones, la clase empleado cuenta con los atributos más básicos de un empleado y sus métodos son los más generales y que casi cualquier clase puede utilizar, esta clase así como todas las demás cuentan con un método constructor.



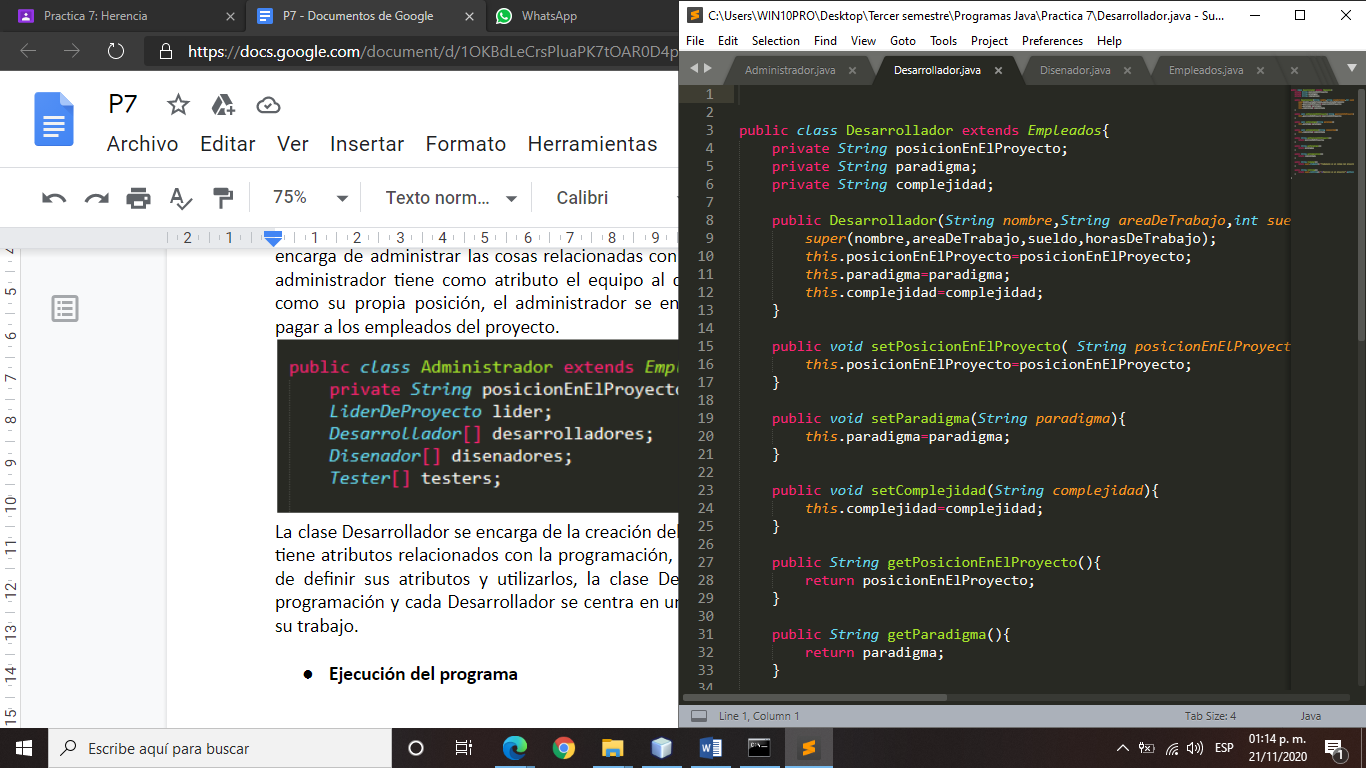
La anterior imagen son los atributos que se definieron que eran los más generales que todos los empleados pueden tener, después se definió la clase líder del proyecto, esta clase contiene atributos específicos de un líder del proyecto así como también los métodos que ayudan a manejar los atributos de la clase específica.



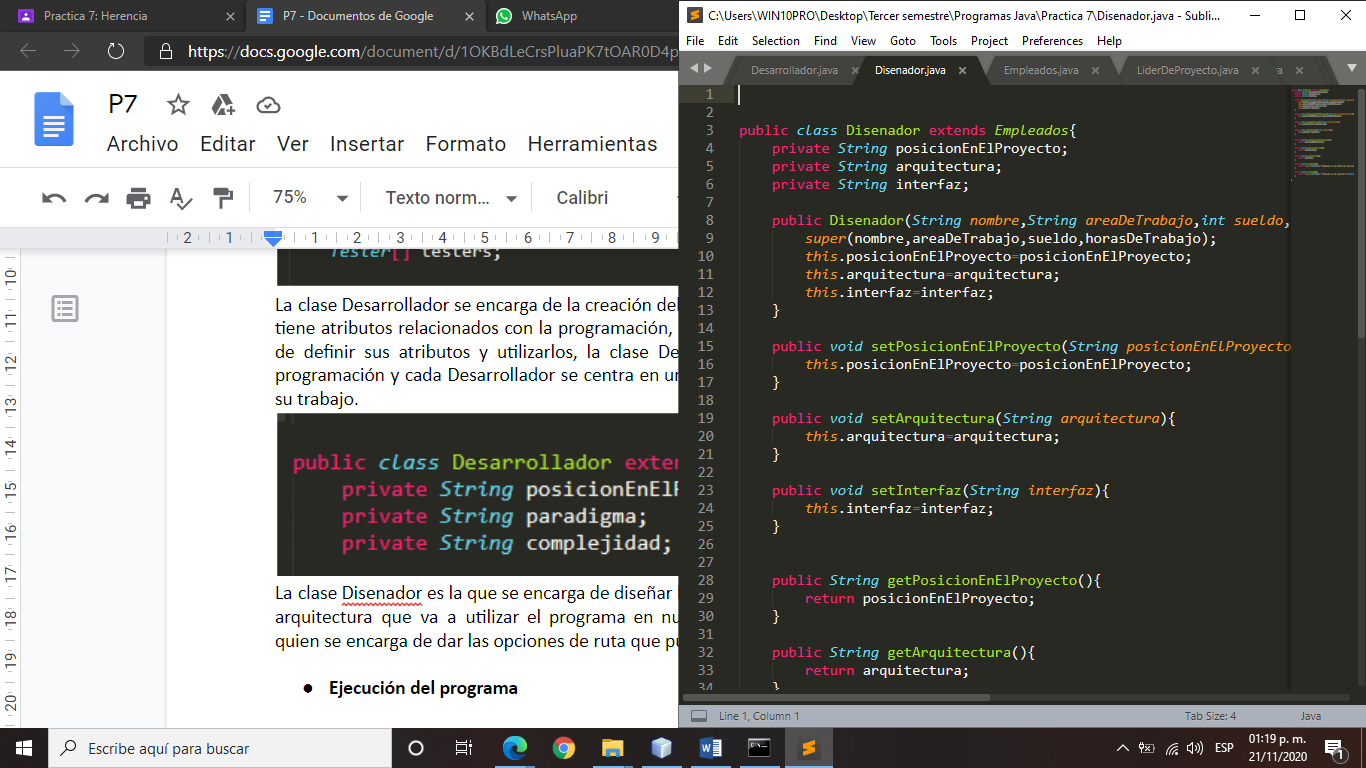
Posteriormente definimos la clase administrador la cual como su nombre dice se encarga de administrar las cosas relacionadas con el equipo, por eso mismo la clase administrador tiene como atributo el equipo al que se encarga de administrar así como su propia posición, el administrador se encarga de asignar las posiciones y pagar a los empleados del proyecto.



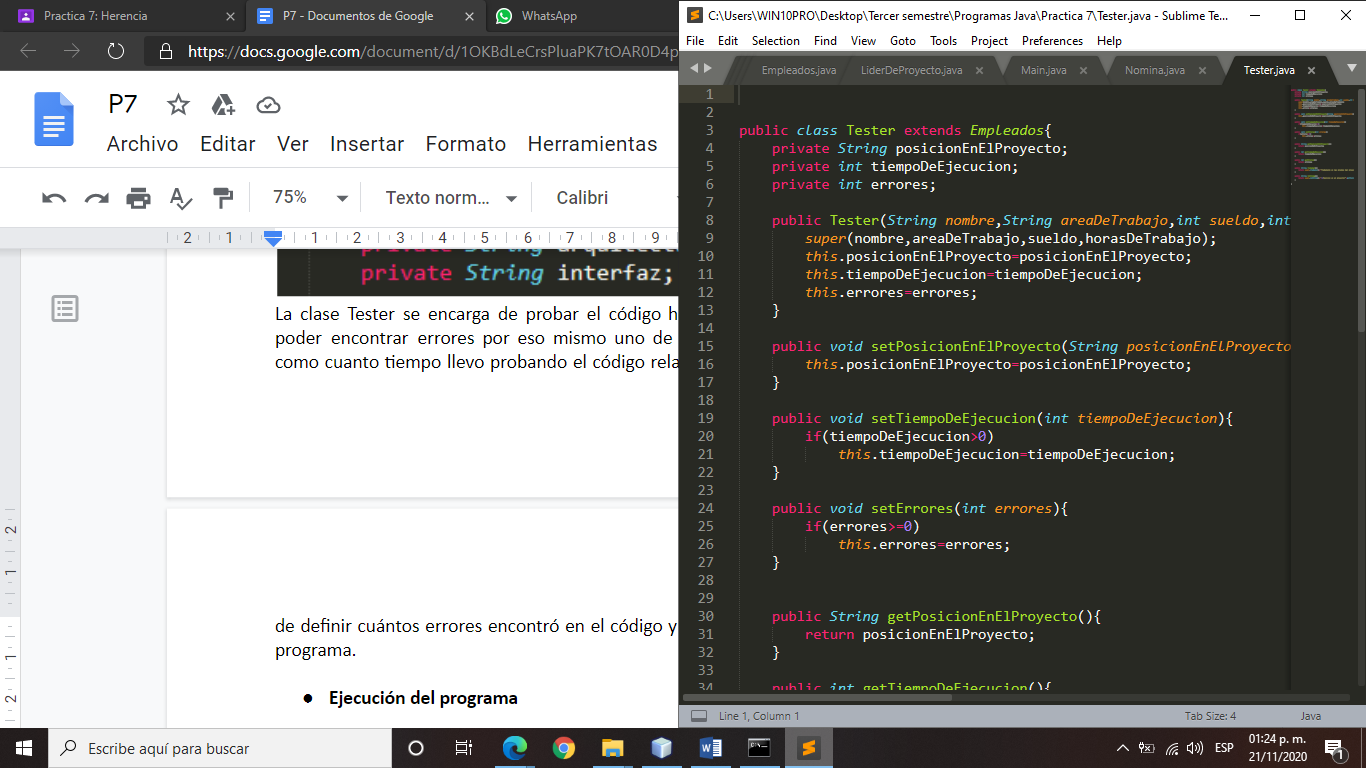
La clase Desarrollador se encarga de la creación del código de la empresa por lo tanto tiene atributos relacionados con la programación, así como sus métodos se encargan de definir sus atributos y utilizarlos, la clase Desarrollador solo se encarga de la programación y cada Desarrollador se centra en un paradigma de programación para su trabajo.



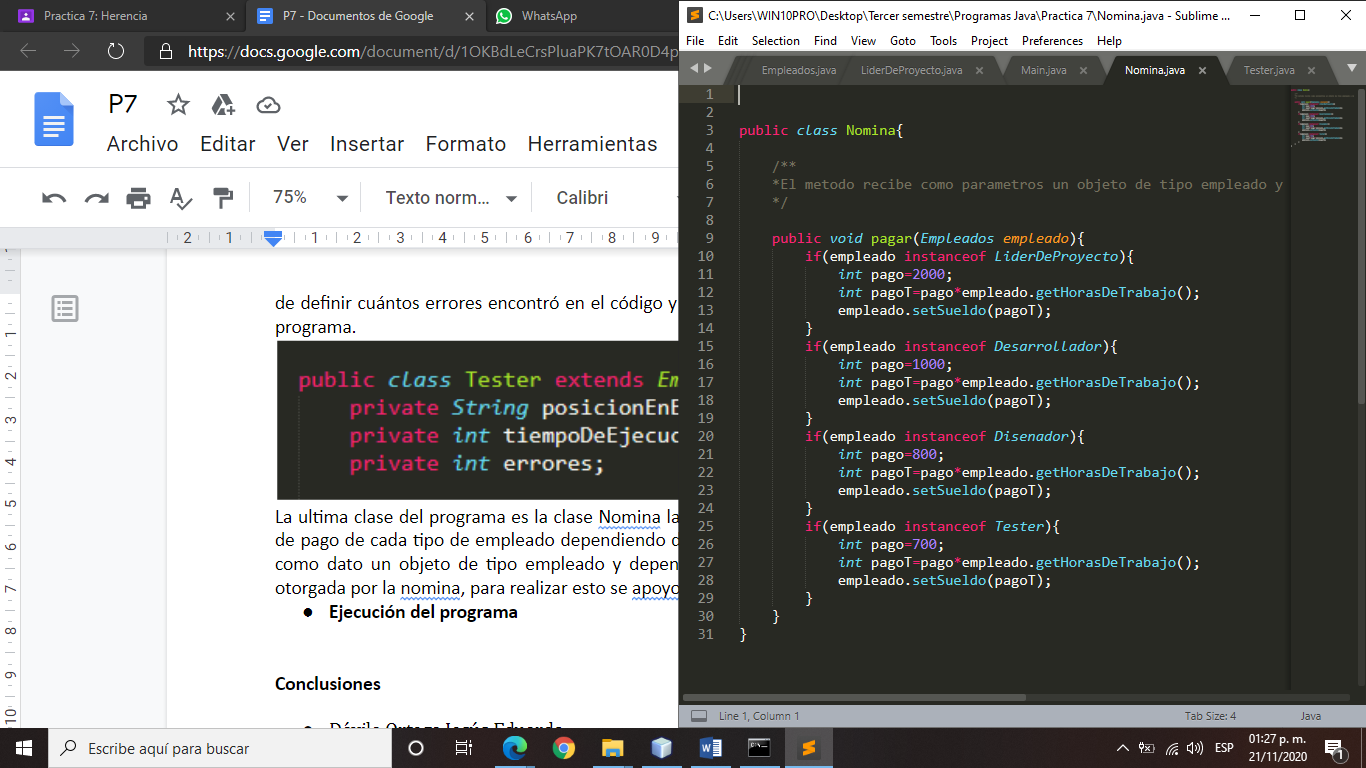
La clase diseñador es la que se encarga de diseñar la interfaz del proyecto así como la arquitectura que va a utilizar el programa en nuestro caso, en pocas palabras es quien se encarga de dar las opciones de ruta que puede seguir el proyecto.



La clase Tester se encarga de probar el código hecho por los desarrolladores para poder encontrar errores por eso mismo uno de sus atributos son los errores, así como cuanto tiempo llevo probando el código relacionado, sus métodos se encargan de definir cuántos errores encontró en el código y cuánto tiempo llevo ejecutando el programa.

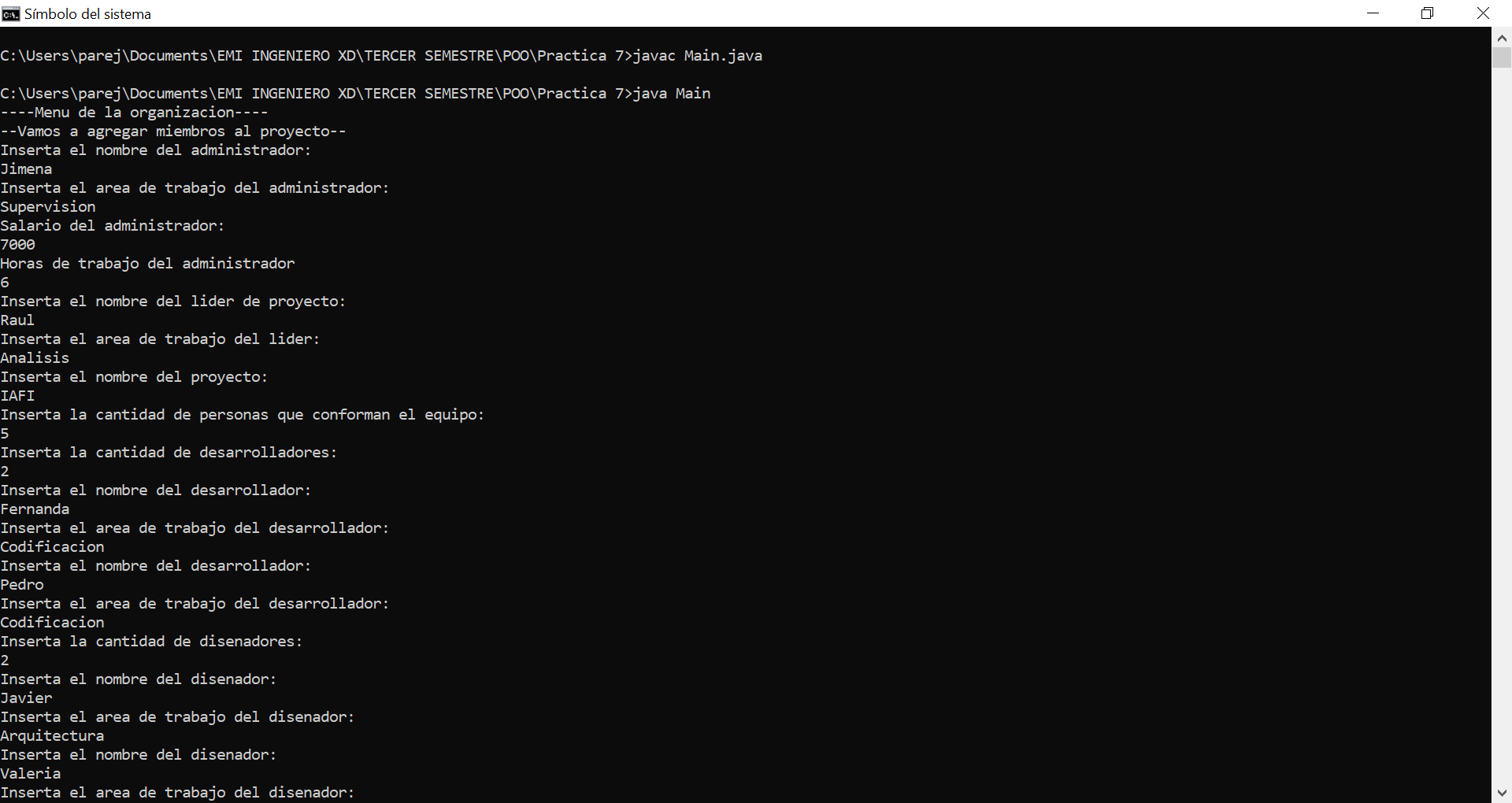


La última clase del programa es la clase Nomina la cual tiene como método la forma de pago de cada tipo de empleado dependiendo de su posición, este método recibe como dato un objeto de tipo empleado y dependiendo de su posición es la paga otorgada por la nómina, para realizar esto se apoyó en la sentencia intanceOf.

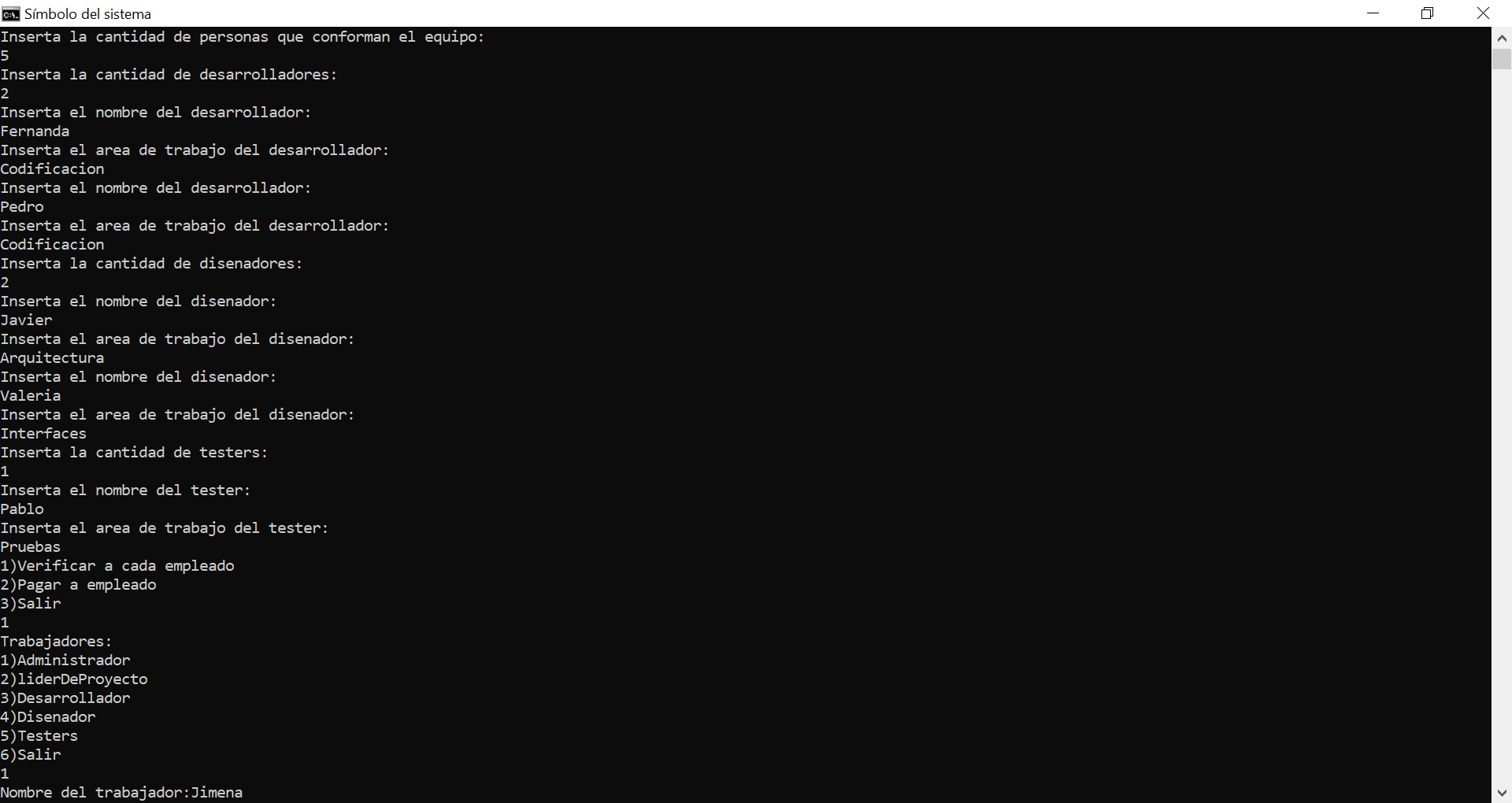


* **Ejecución del programa**

Al correr el programa, lo primero que ingresamos es el nombre del administrador, su área de trabajo, su salario y sus horas de trabajo, después se hace lo mismo con el líder del proyecto, solo que no se pide al usuario que ingrese su sueldo, puesto a que nosotros ya lo preestablecimos en el programa, también se pide al usuario que ingrese el nombre del proyecto del cual es líder el trabajador.



Después para poder conocer cuántos trabajadores se modelarán se pide al usuario que ingrese cuántas personas hay en el equipo de trabajo, en este caso nosotros establecimos que 5. Y después de ingresar el número, ahora se pregunta cuántos de estos trabajadores son desarrolladores y se hace la lectura de sus datos, preguntando al usuario por su nombre y área de trabajo, después se pregunta al usuario por el número de desarrolladores y sus nombres y áreas de trabajo y por último se pregunta al usuario por el número de testers en el equipo de trabajo, y se hacen las mismas lecturas, su nombre y área de trabajo.



Después de concluir toda la lectura de los datos, ahora pasamos al menú principal, en el cual tenemos 2 opciones, verificar a un empleado, mostrando sus datos o pagarle. Escogimos primero la opción uno, para explicar cómo funciona nuestro programa.

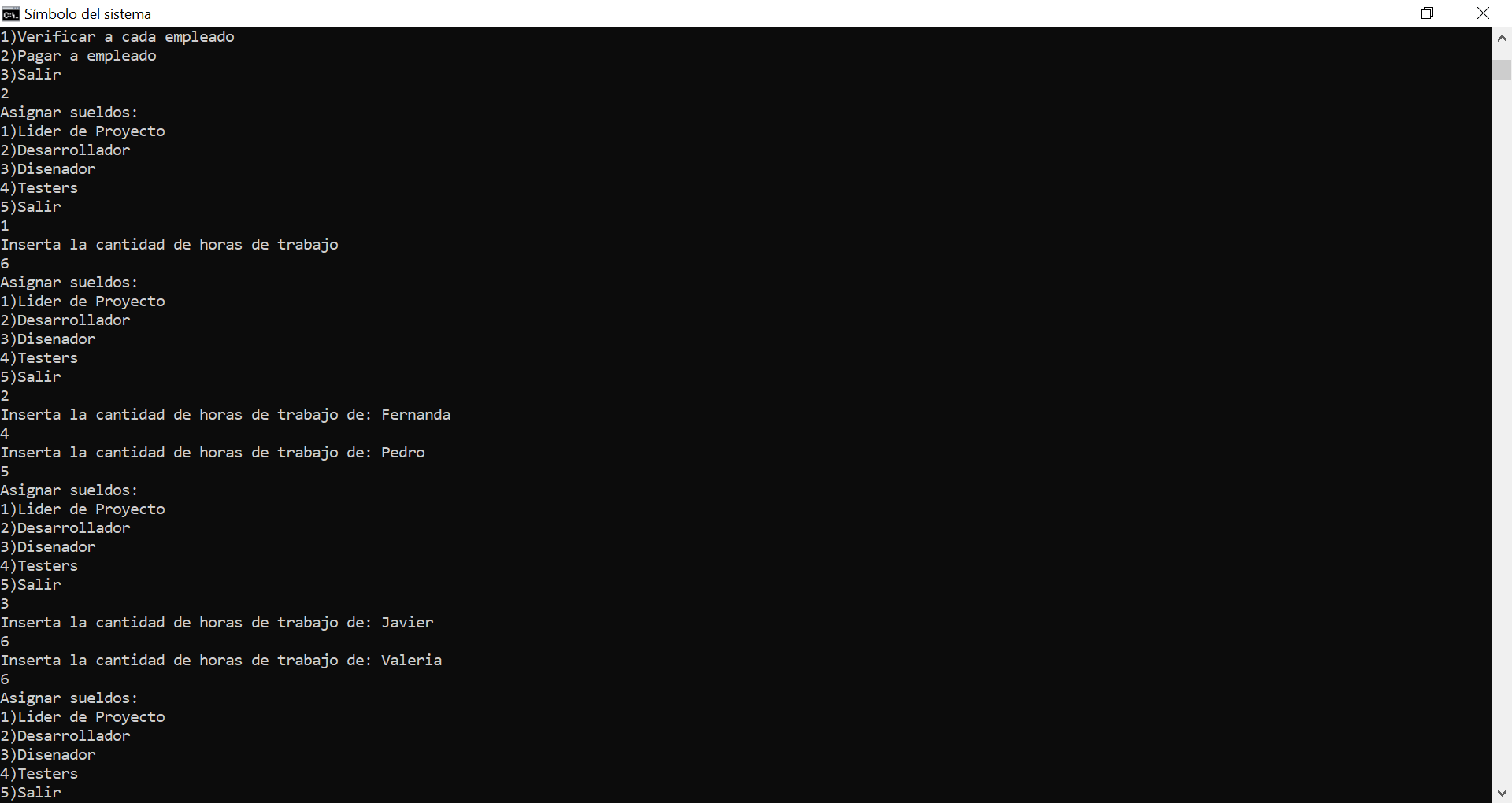
Al escoger la opción 1, y en el submenú de verificar a un empleado, se imprimen los datos del administrador, y observamos sus atributos. Después decidimos imprimir a pantalla los datos del líder del proyecto, desarrolladores, diseñadores y testers, sin embargo marca que su sueldo es 0, esto se debe a que aún consideramos que no han trabajado, por tanto no han recibido sueldo o pago aún.

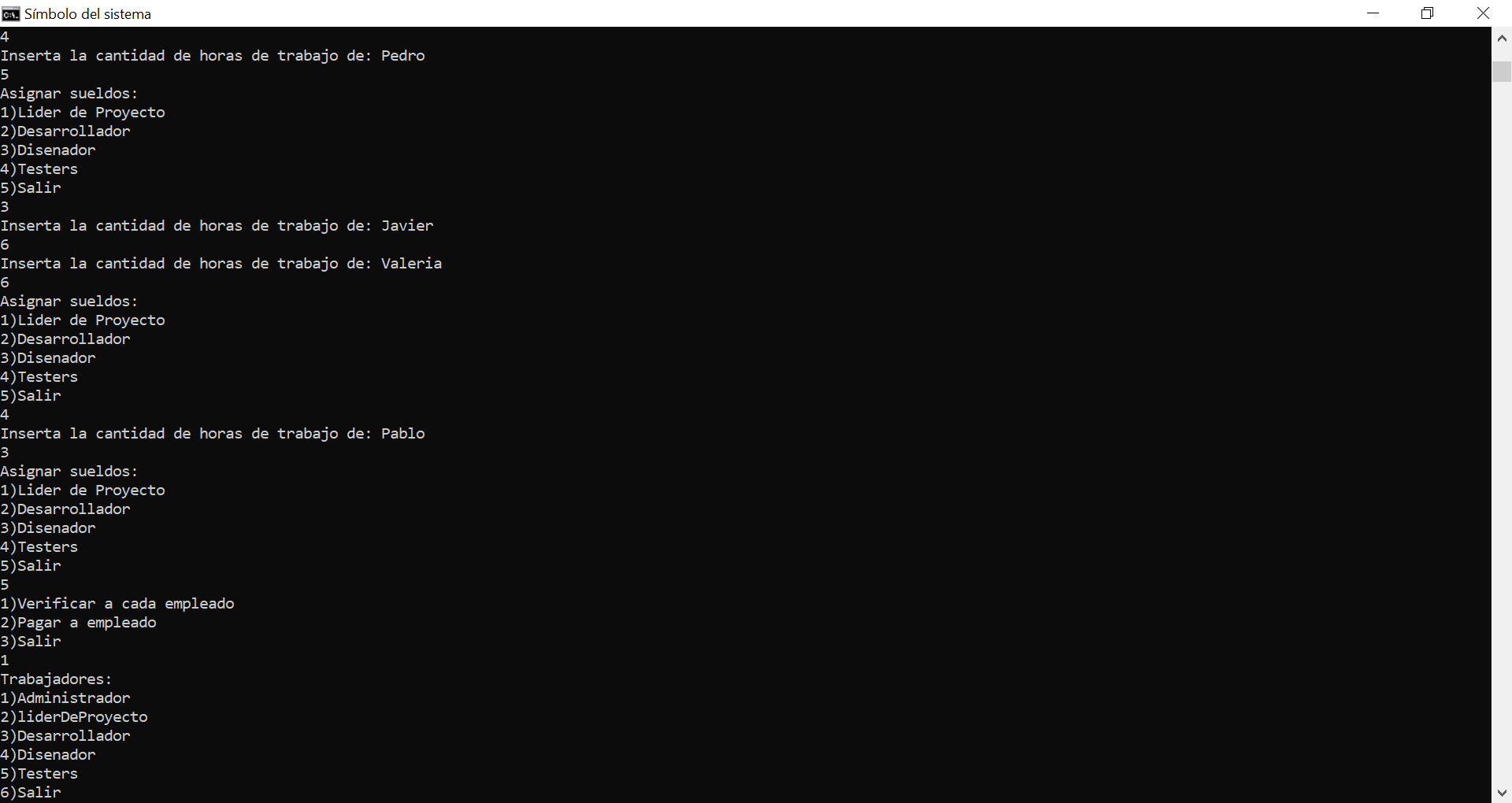






Entonces para poder establecer sus sueldos por horas de trabajo, salimos del menú de verificar trabajadores, y pasamos al menú principal y seleccionamos la opción 2 que es pagar a los trabajadores, dentro, hay 4 opciones que podemos seleccionar, la opción 1 es pagar al líder del proyecto, la 2 pagar los desarrolladores, la 3 pagar a los diseñadores y la 4 pagar a los testers. Entonces escogimos las 4 opciones y por cada uno se pregunta cuántas horas trabajaron, ya que eso influirá en su sueldo. En la opción 1, al solo haber un líder de proyecto no se especifica su nombre, pero en las demás opciones si se pregunta al usuario uno por uno las horas trabajadas por los miembros del equipo.



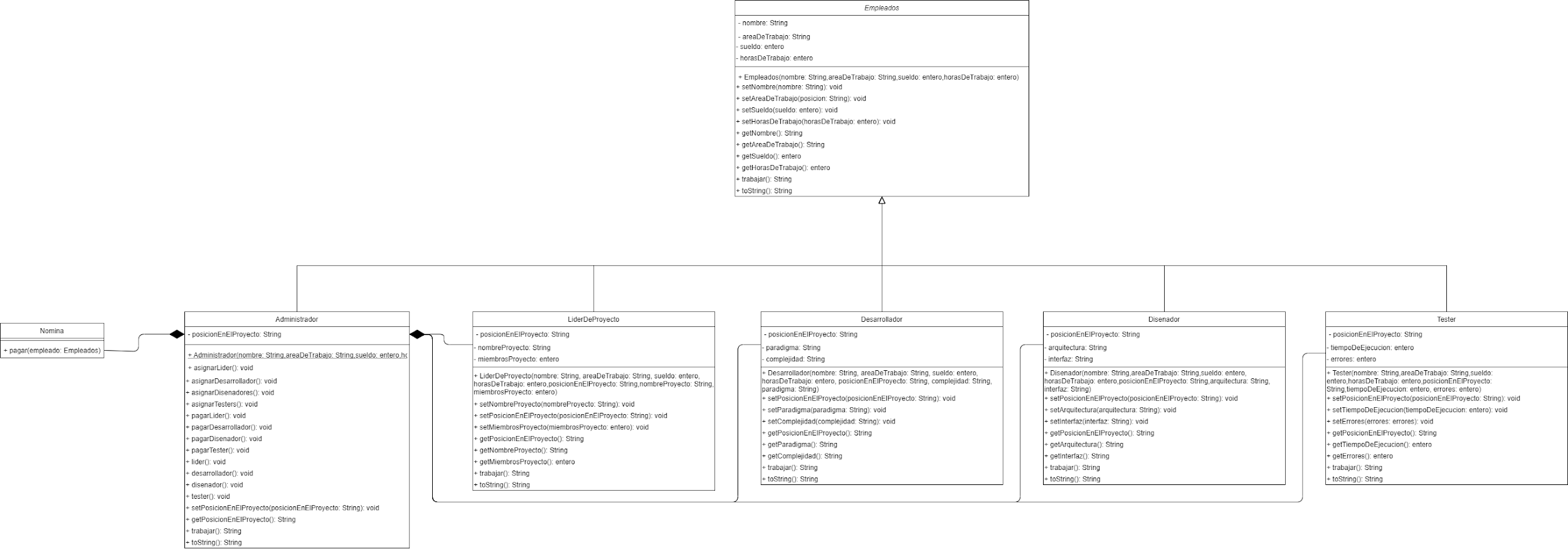


Salimos del menú de pagar a empleados para verificar que ahora los empleados, después de sus horas laboradas, ya tienen un sueldo. Y en el menú principal volvemos a seleccionar la opción de verificar empleados, y dentro de este menú imprimimos todos los datos de todos los empleados involucrados y observamos que ya tienen un sueldo por sus horas trabajadas. Por tanto verificamos que nuestro programa funciona.





**Diagrama UML de las clases Desarrollada**



**Conclusiones**

* Dávila Ortega Jesús Eduardo:

Durante la práctica entendí la necesidad de la herencia dentro del paradigma orientado a objetos, ya que esto facilita de gran manera la programación y el tratamiento de clases así como la reutilización de código debido a la herencia, y como esta herencia genera una jerarquía de clases, aunque al momento de realizar un análisis de cómo se debía de hacer la herencia llegaba a ser complejo por cómo se debía de establecer las relaciones y como se define la superclase, ya que esta debía de encapsular los atributos generales de cada subclase y llevar a cabo esta abstracción puede llegar a ser muy difícil si no se definen los parámetros de las clases de manera correcta y puede llegar a cometerse un error, pero con cuidado se puede llegar a tener una correcta abstracción y pensamiento que facilite el código y facilite su implementación

* Díaz Hernández Marcos Bryan:

En esta práctica aprendí a implementar, por medio de la realización del ejercicio, la herencia que implica el uso de la jerarquía de clase, y la utilización de conceptos vistos anteriormente, además de realizar el proceso de análisis que conlleva utilizar la herencia y la implementación de las clases, el proceso del análisis fue algo bastante difuso por el hecho de que no tenía estructurada la forma al cien por ciento pero bajo la marcha y con ayuda de mis compañeros pude entender mejor el concepto y ponerlo en práctica, cumpliendo así con el objetivo de la práctica.

* Pareja Ávila Emiliano:

Se cumplieron los objetivos de la práctica, ya que se aplicaron todos los conceptos que involucran la herencia, como la sobrecarga de métodos o sobreescritura, y no solo de las clases heredadas de la clase padre Empleado, sino también de la clase Object, ya que todas las clases que creamos en nuestro programa heredan los métodos de dicha clase, es por eso por lo que pudimos sobre escribir el método toString. También aplicamos otro concepto que es la composición de clases y a opinión personal, esta ha sido la práctica más completa que hemos tenido, ya que pudimos aplicar todo lo visto en clase.

* Vázquez Zavala Oliver Alex:

Esta práctica me ha dejado de aprendizaje conceptos importantes relacionados con la herencia en el paradigma orientado a objetos, además me ayudó a reforzar otros conceptos previamente conocidos, haciendo uso en el desarrollo del programa de la herencia y así permitirnos reutilizar código, además de los conceptos de sobrecarga y sobreescritura de gran ayuda en el desarrollo de la práctica, dejándome ver cuál es la forma correcta de implementar estas propiedades en Java y cómo se relacionan con otros conceptos previamente estudiados.

**Bibliografía**

- Barnes David, Kölling Michael Programación Orientada a Objetos con Java. Tercera Edición, Pearson, 2007