**Algoritmos y Programación II**

**Trabajo Práctico 1**

**Empresa**

**Integrantes:**

* **Bellotti, Matias**
* **De Alesandro, Franco**
* **Hansen, Nicolas**
* **Juguera, Matias**

*Introducción*

En este informe se detalla el procedimiento que llevamos a cabo para poder desarrollar el programa a pedir (un administrador de empleados en una empresa). En este podemos contar con varias métodos que se explicaran a continuación (obtener empleado, cuil del empleado, voluntario, etc). Además tendrá presente una consola en donde se podrá utilizar estos métodos de forma interactiva y cómoda. En el desarrollo del programa se tuvo muchos errores al principio de este ya que no se genero un buen UML, además la consola no funcionaba como debía ya que los try y catch no funcionaban, también se generó un método innecesario que fue removido y algunos errores más que se explicarán más adelante como también el uso de las estructuras de datos en el programa.

Al terminar de finalizar una clase, se procedía a realizar un caso de pruebas comprobando cada método para ver si esta clase funciona de manera eficiente y sin ningún error, así como también se comprobó que los errores esperados al ingresar un parámetro de manera incorrecta sean lanzados.

*Desarrollo*

En el desarrollo se tomó varias decisiones como la utilización de un switch en la consola, de esta manera se podría acceder a esta y controlarla de forma más eficiente que utilizar un if else constantemente. Se podrá controlar el MenuConsola (nombre de la clase en donde se encuentra esta) con números que serán mostrados en la consola del programa para poder utilizar métodos queridos y además poder generar empleados como también clase heredadas a esta. Tambien opcion de salir la cual fue programada con un while de manera al seleccionar la opción de salir ya no pueda utilizarse el menú consola.

Se había decidido utilizar try y catch para poder lanzar los errores en la consola pero fue desechado ya que no hacía lo que se esperaba, ya que rompía la consola. Se optó con utilizar un throw error con excepciones ya creadas por nosotros y que estas al lanzarse manden un mensaje a la consola.

Modificamos el UML para que la clase trabajador sea abstracta aunque sus métodos no lo sean para que no sea posible instanciarla ya que hacerlo no tendria ningun sentido, para eso se crearán empleados con cuil y los métodos y atributos heredados del trabajador. Por cada tipo de empleado necesario, se creó una clase distinta con su respectivo constructor. Los atributos “nombre” y “cuil” son compartidos en todas las clases (a excepción de la clase Voluntario, donde se cambió cuil por DNI), además de los atributos necesarios en cada clase (por ejemplo la comisión en el EmpleadoPorHoraAComision, el sueldo por hora en el EmpleadoPorHora, etc).

En nuestro comienzo del programa, realizamos un método llamado “generador de cuil” en donde este generaba los dos últimos dígitos del cuil y además habíamos puesto los dos primeros números como un número fijo. Esto se removió ya que en términos de contexto, nosotros estamos en empresa y estos no tiene que ser los que generen un cuil sino el AFIP, por lo tanto descartamos este método y decidimos que el cuil sea ingresado por parámetro en donde el dni debe pertenecer a este y comprueba que pertenezca a la misma persona, sino se lanzará un error.

Se empezó a construir la consola dentro de la clase Empresa como el administrador de los empleados, pero al ocurrir un error no se obtenía nada, por lo tanto se opto por quitar la consola de empleado (además que esta posee métodos y que esté acompañando con un main no parecía ser lo mas optimo) y se creo una clase aparte ya mencionada, llamada MenuConsola en donde con esta se crea la consola para poder administrar a los trabajadores.

Utilizamos el método CompareTo para comparar los sueldos de los empleados, y si uno es idéntico al otro, tomamos su dni como referencia para comparar.

Cada clase de cada trabajador tiene un método propio de toString en el cual se imprime su nombre, su cuil y cada uno de sus atributos individuales, para poder imprimir al final el listado de todos los trabajadores.

En el primer UML, habíamos cometido el error de diseñar la clase EmpleadoPorHora tal que sus horas trabajadas sean pasadas al ejecutar el método getSueldo. Finalmente, se decidió convertir a horasTrabajadas en un atributo de la clase para que sea pasado por consola las horas trabajadas del empleado, y de esta manera se puede obtener el sueldo correctamente en base a este atributo.

Al principio se utilizó un arrayList para listar a todos los empleados y para poder obtener su dni, se recorría el array viendo el dni de cada uno. Esto se cambió por un hashMap relacionando como key el nombre y el valor su dni, por lo tanto es más óptimos este caso ya que, obtener el dni de cada uno es más rápido y no se aceptan duplicados.

*Conclusión*

En este trabajo se pudo aprender a poder diferenciar algunas estructuras de datos, su eficiencia y ademas cual conviene usar más en ciertas situaciones que en otras. Se pudo crear una consola utilizando los conceptos de entrada y salida aprendidos y se utilizó por primera vez el switch key (que en materias anteriores no se utilizaba) y el do while. Se creó un UML en escala mas real para poder entender el procedimiento que se usa al crearlo. Se pudo entender con mayor eficacia el concepto de abstracto. Se utilizó un herencia dentro de otra es decir, ejecutivo hereda de empleado y este hereda de trabajador, como ejemplo. Llamar a los diferentes tipos de empleados en la lista con un instanceOf y la utilización del compareTo para poder comparar su sueldos entre sí y un sort para poder hacer el ordenamiento ya en la consola.

Como comentario aparte, para mejorar el trabajo en equipo y hacer que la división de tareas sea más rápida y efectiva, comenzamos a trabajar con Git, donde cada miembro del grupo desarrollaba una clase asignada y la subía al directorio en común.

**Lista de archivos .Java:**

**Clases:**

* Ejecutivo.java
* Empleado.java
* EmpleadoPorHora.java
* EmpleadoPorHoraAComision.java
* Empresa.java
* MenuConsola.java
* Trabajador.java
* Voluntario.java

**Excepciones:**

* CuilInvalidoExcepcion.java
* DniDeOtroTipoExcepcion.java
* DniInvalidoExcepcion.java
* ErrorDeComision.java
* ErrorValorEnSueldo.java
* ErrorVentasRealizadas.java
* HorasTrabajadasInvalidasExcepcion.java
* NombreInvalidoExcepcion.java
* PremioInvalidoExcepcion.java
* ValorFueraDeRangoException.java

**Test:**

* EmpresaTest.java