2020

Administración de empleados

UNIDAD CURRICULAR: ALGORITMOS Y PROGRAMACIÓN III

INTEGRANTES: BENITEZ MAURICIO BLUR.BENITEZ@GMAIL.COM
JIMENEZ NICOLAS NICOLASDANIELJIMENEZ@HOTMAIL.COM
VILLCA DEBORA DEBYVILLCA95@GMAIL.COM

UNPAZ
Universidad Nacional de José C. Paz

INTRODUCCION

En este informe se pretende describir un programa que permite organizar la nómina de los trabajadores de una empresa, facilitar la liquidación de sueldos, guardar los datos personales de los trabajadores, crear, modificar, dar de baja empleados de distinto tipo, cálculos de salarios, listados de los mismos. Para poder realizarlo se comenzó con la implementación de grafico de clases de las entidades que entran en el contexto y se aplicó conceptos básicos de programación orientada a objetos como son las listas, interfaces, constructores, etc.

Clase: Trabajador

En esta clase se definieron los atributos de la entidad TRABAJADOR, las mismas son datos personales del trabajador, se utilizó un segundo constructor para el agregado de títulos y una lista para el agregado de Trabajadores a cargo. En esta clase también se implementó la interfaz dada por la catedra ITrabajador.

```
public Trabajador (long dni, String nombre, String apellido, TipoCargo
cargo, Fecha fechaIngreso, String titulo,
                        String tituloPostgrado) {
                super();
                this.dni = dni;
                this.nombre = nombre;
                this.apellido = apellido;
                this.cargo = cargo;
                this.fechaIngreso = fechaIngreso;
                this.titulo = titulo;
                this.titulop = tituloPostgrado;
                TrabajadoresACargo=new ArrayList<>();
        }
```

Métodos de la interfaz ITrabajador

```
@Override
        public boolean esEmpleado() {
                return (this.cargo.equals(TipoCargo.JEFES)
                        ||this.cargo.equals(TipoCargo.SUPERVISOR)
                                ||this.cargo.equals(TipoCargo.OPERARIO));
 }
```

```
@Override
        public boolean esDirectivo() {
                return (this.cargo.equals(TipoCargo.DIRECTOR_GENERAL)
                        ||this.cargo.equals(TipoCargo.DIRECTOR_DEPARTAMENTO));
        }
```

Estos métodos de tipo booleano y lanzan verdadero si el trabajador es empleado (se dice ser empleados a trabajadores de tipo operario, supervisor o jefe), o si es directivo (se dice ser directivo a trabajadores de tipo director general o director departamental).

Métodos get y set

getMesesAntiguedad(): devuelve los meses de antigüedad utilizando clase Fecha para el cálculo con su fecha de ingreso y la fecha actual.

public void setTituloUniversitario(String titulo), public String getTituloUniversitario(), devuelve si tiene título universitario o no, public boolean tieneTituloUniversitario() devuelve verdadero si tiene título universitario o false si no lo tiene.

```
@Override
public void setTituloPostgrado(String titulo) throws
ExcepcionOperacionNoPermitida {
                if(!this.tieneTituloUniversitario()) {
                        throw new ExcepcionOperacionNoPermitida ("Debe tener
titulo universitario");
                }
                this.titulop=titulo;}
```

En este método se utilizó excepción si es que no obtiene un previo título universitario public String getTituloPostgrado(), public boolean tieneTituloPostgrado() este método de tipo booleano devuelve verdadero si tiene título de postgrado o false si no lo tiene.

Métodos de los datos personales del trabajador: public String getNombre(), public void setNombre(String pedirTexto); public String getApellido(), public void setApellido(String pedirTexto);public long getDni(), public void setDni(long ingresarDni);

```
@Override
	public void agregarTrabajadorACargo(ITrabajador trabajador) throws
ExcepcionOperacionNoPermitida {
		if(trabajador.getCargo().equals(TipoCargo.OPERARIO)) {
			throw new ExcepcionOperacionNoPermitida ("Un operario no
puede tener trabajador a cargo");}
		TrabajadoresACargo.add(trabajador);}
```

En este método se asigna trabajador a cargo junto con la clase excepción ya que un operario no debe tener otro trabajador a cargo, también se utilizó la clase enum TipoCargo para definir los tipos de cargos que existen en la empresa, finalmente se agrega a la lista TrabajadoresACargo un trabajador que cumple las condiciones.

public double getSalario(), public void setSalario(double salario): devuelve el salario del trabajador

```
@Override
	public double getPremio() {
		boolean esDirectivo = cargo.equals(TipoCargo.DIRECTOR_DEPARTAMENTO
)||cargo.equals(TipoCargo.DIRECTOR_GENERAL);
		boolean esEmpleado = cargo.equals(TipoCargo.JEFES) ||
cargo.equals(TipoCargo.SUPERVISOR) || cargo.equals(TipoCargo.OPERARIO);
		double premio = 0;
		if (esDirectivo && this.getMesesAntiguedad() >= 12) {
			premio += 100000;}
		if (esDirectivo && this.getMesesAntiguedad() < 12) {
			premio += 50000;
		}
		if (esEmpleado && this.getMesesAntiguedad() >= 12) {
			premio += 60000;
		}
		if (esEmpleado && this.getMesesAntiguedad() < 12) {
			premio += 30000;
		}
		if (this.tieneTituloUniversitario()) {
			premio += 15000;
		}
		if (this.tieneTituloPostgrado()) {
			premio += 25000;
		}
		if (this.getCantidadEmpleadosACargoDirecto() > 0) {
			premio += (1000 * this.getCantidadEmpleadosACargoDirecto());
		}
		if (this.getCantidadEmpleadosACargoTotal() > 0) {
			premio += (500 * this.getCantidadEmpleadosACargoTotal());
		}
		return premio;}
```

En este método se calcula y retorna su premio según su cargo y sus meses de antigüedad

```
@Override
	public double getMontoACobrar() {
		return cargo.getSalario()+this.getPremio();}
```

En este método devuelve el monto a cobrar del trabajador, calculado su salario junto con su premio correspondiente.

```java
@Override
	public int getCantidadEmpleadosACargoDirecto() {
		if(cargo.equals(TipoCargo.OPERARIO)) {
			return 0;
		}
		return this.TrabajadoresACargo.size();
	}
```

Para devolver la cantidad de empleados a cargo directo se tiene que cumplir la condición que el trabajador no debe ser un operario, de esa manera retorna el tamaño de la lista de los trabajadores a cargo, estos son directores, jefes y supervisores.

```java
@Override
	public int getCantidadEmpleadosACargoTotal() {

		return this.cantidad();
	}
public int cantidad() {
		List<ITrabajador> listaTrabajadores = new ArrayList<>();
		for(ITrabajador t: this.TrabajadoresACargo) {
			// Acá recorro los directores de departamento, en caso
de que sea un director general esta hecho
		if(!listaTrabajadores.contains(t)) {
				listaTrabajadores.add(t);
			}
			for(ITrabajador t2: t.getListaACargo()) {
				// Acá recorro los jefes
				if(!listaTrabajadores.contains(t2)) {
					listaTrabajadores.add(t2);
				}
				for(ITrabajador t3: t2.getListaACargo()) {
					// Acá recorro los supervisores
					if(!listaTrabajadores.contains(t3)) {
						listaTrabajadores.add(t3);
					}
					for(ITrabajador t4: t3.getListaACargo()) {
						// Acá recorro los operarios
						if(!listaTrabajadores.contains(t4))
{
							listaTrabajadores.add(t4);
						}
					}
				}
			}
		}
		return listaTrabajadores.size();}
```

Para el método de cantidad de empleados total se utilizó una lista auxiliar donde se almacenan trabajadores que están a cargo, mientras realiza el recorrido por todos los cargos de la empresa, finalmente retorna el tamaño de la lista auxiliar.

Clase:Empresa

En esta clase empresa se implementan los métodos que llevaran a cabo las tareas de administración de la nómina de empleados.

Los trabajadores se agregarán a una estructura de lista, donde se diferenciarán según su jerarquía. En esta lista se guardan los datos relevantes de cada trabajador.

Como algunos trabajadores tendrían personal a cargo, determinamos que la mejor forma de implementar este requerimiento era crear una sublista de “trabajadores a cargo” asignada al trabajador “jefe”.

Colocamos comentarios en cada método para facilitar la lectura e interpretación del código.

Al final de la clase Empresa situamos los métodos que no estaban explícitamente pedidos en la consigna pero que nos resultaron muy útiles para el funcionamiento global del código.

Clase: Principal

Por mi parte me centré en el desarrollo de la clase Principal que va a ser un contacto con la clase Empresa que es donde se maneja el sistema.,

Al ejecutar la clase Principal se llama a un menú con el que se podrá interactuar la clase Empresa (y por lo tanto podremos dar de alta un trabajador, darlo de baja, saber su sueldo y demás opciones).

A medida que iba desarrollando esta clase, tenía muy en cuenta el evitar usar muchos bloques de código iguales por ello creé un método para ingresar el DNI de un trabajador ya que esta acción era requerida muchas veces en las distintas opciones que ofrece el sistema. Por la misma razón también fueron creados los métodos para ingresar la fecha y seleccionar un cargo para el trabajador.

Por último, la clase Principal tiene un método que se utiliza para pruebas llamado anadirTrabajadoresPredeterminados que su función era cargar una lista de trabajadores al sistema y así evitar la carga manual.

También me encargué de crear la clase NodoTrabajador que estaba pensado para implementar una lista doblemente enlazada.