TAREA COLABORATIVA

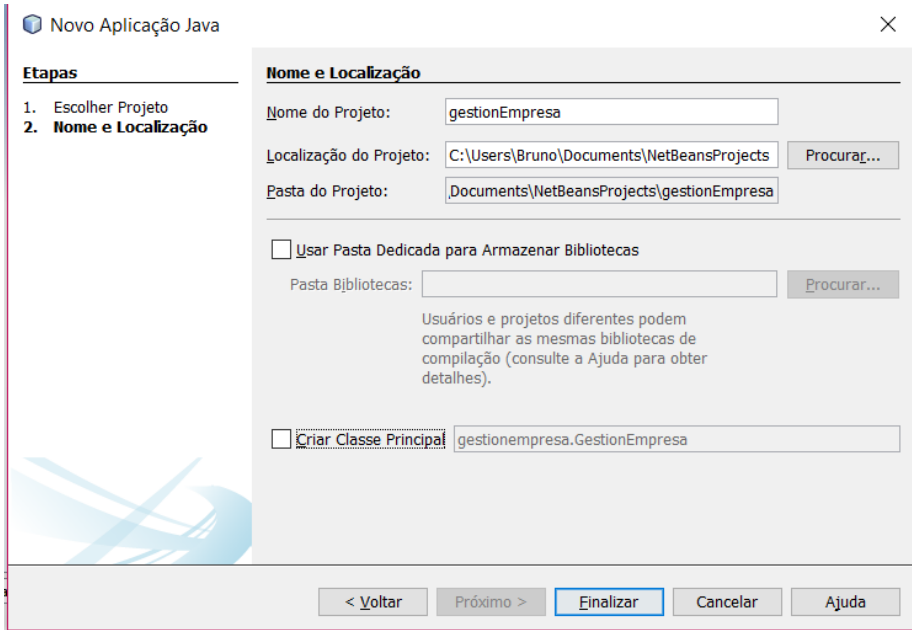
PROGRAMACIÓN

- Miembro 1: Bruno Marenco Cerqueira
- Miembro 2: Elisabeth Guisado Reina
- Grupo de trabajo: 24
- DAW

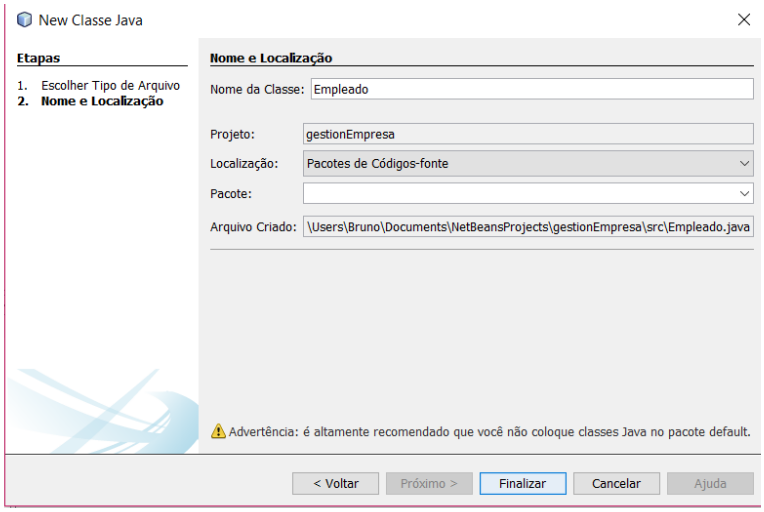
INDICE

MIEMBRO 1 (Bruno Marenco Cerqueira)

-Crear un proyecto en NetBeans denominado, "gestionEmpresa".



-Dentro del proyecto denominado "gestionEmpresa", el primer miembro de la pareja deberá crear una clase denominada "Empleado" con las siguientes características:



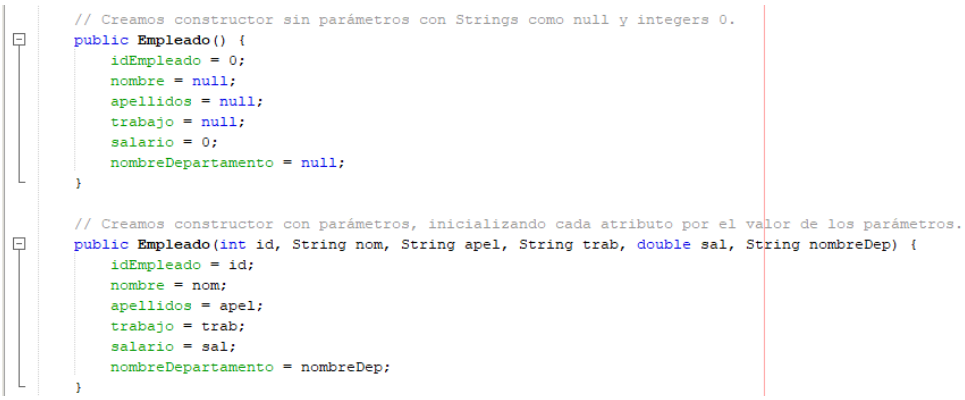
Creamos la clase dentro del proyecto.

- **Atributos** (Todos los atributos de la clase Empleado deben tener visibilidad privada)
- idEmpleado: número entero que representa el identificador único de cada empleado de la empresa.
- nombre: cadena de caracteres que representa el nombre de un determinado Empleado.
- apellidos: cadena de caracteres que representa los apellidos de un determinado Empleado.
- trabajo: cadena de caracteres que representa el tipo de trabajo que desempeña cada Empleado.
- salario: número decimal que representa el salario mensual que recibe cada Empleado.
- nombreDepartamento: cadena de caracteres que representa el nombre identificativo del departamento en el que trabaja el Empleado.

```
*/
public class Empleado {
    //Declaramos los atributosprivados idEmpleado, nombre, apellidos, trabajo, salario, nombreDepartamento

    private int idEmpleado;
    private String nombre;
    private String apellidos;
    private String trabajo;
    private double salario;
    private String nombreDepartamento;
```

Definimos los atributos privados de la clase.

- **Métodos** (Todos los métodos de la clase Empleado deben tener visibilidad pública):
- Constructor sin parámetros: constructor que inicializa todos los atributos de tipo cadenas de caracteres al valor null y los números enteros a 0.
- Constructor con parámetros: constructor que tienen tantos parámetros como atributos tiene la clase, y que inicializa cada uno de los atributos con el valor de los parámetros correspondientes.



```
// Creamos constructor sin parámetros con Strings como null y integers 0.
public Empleado() {
    idEmpleado = 0;
    nombre = null;
    apellidos = null;
    trabajo = null;
    salario = 0;
    nombreDepartamento = null;
}

// Creamos constructor con parámetros, inicializando cada atributo por el valor de los parámetros.
public Empleado(int id, String nom, String apel, String trab, double sal, String nombreDep) {
    idEmpleado = id;
    nombre = nom;
    apellidos = apel;
    trabajo = trab;
    salario = sal;
    nombreDepartamento = nombreDep;
}
```

Creamos los constructores sin parámetros y con parámetros.

- **get y set**: Métodos get y set para poder consultar y modificar cada uno de los atributos desde fuera de la clase,al tener visibilidad privada

```java
// Definimos los métodos get y set de cada atributo.
public int getId() {
    return idEmpleado;
}
public void setId(int id) {
    idEmpleado = id;
}
public String getNombre() {
    return nombre;
}
public void setNombre(String nom) {
    nombre = nom;
}
public String getApellidos() {
    return apellidos;
}
public void setApellidos(String apel) {
    apellidos = apel;
}
public String getTrabajo() {
    return trabajo;
}
public void setTrabajo(String trab) {
    trabajo = trab;
}
public double getSalario() {
    return salario;
}
public void setSalario(double sal) {
    salario = sal;
}
public String getNombreDep() {
    return nombreDepartamento;
}
public void setNombreDep(String nombreDep) {
    nombreDepartamento = nombreDep;
```

Creamos los métodos get y set para cada atributo.

MIEMBRO 2 (Elisabeth Guisado Reina)

-Dentro del proyecto denominado "gestionEmpresa", el segundo miembro de la pareja deberá crear una clase denominada"Departamento" con las siguientes características:

- **Atributos** (Todos los atributos de la clase Departamento deben tener visibilidad privada):
- nombre: cadena de caracteres que representa el nombre identificativo del departamento en la empresa.
- descripción: cadena de caracteres que representará la actividad a la que se dedica el departamento en cuestión.

```java
/**
 *
 * @author Elisabeth
 */
public class Departamento {

    /**
     * Nombre del departamento.
     */
    private String nombre;

    /**
     * Actividad a la que se dedica el departamento.
     */
    private String descripcion;
```

Definimos los atributos.

- **Métodos** (Todos los métodos de la clase Empleado deben tener visibilidad pública):

(los métodos son mostrados em la siguiente página)

- Constructor sin parámetros: constructor que inicializa todos los atributos de tipo cadenas de caracteres al valor null y los números enteros a 0.

```java
/**
 * Constructor sin parámetros.
 */
public Departamento() {
    nombre = null;
    descripcion = null;
}
```

- Constructor con parámetros: constructor que tienen tantos parámetros como atributos tiene la clase, y que inicializa cada uno de los atributos con el valor de los parámetros correspondientes.

```java
/**
 * Constructor con parámetros.
 *
 * @param nombre nombre del departamento.
 * @param descripcion actividad a la que se dedica el departamento.
 */
public Departamento(String nombre, String descripcion) {
    this.nombre = nombre;
    this.descripcion = descripcion;
}
```

- **get y set:** Métodos get y set para poder consultar y modificar cada uno de los atributos desde fuera de la clase, al tener visibilidad privada.

```java
    public String getNombre() {
        return nombre;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    public String getDescripcion() {
        return descripcion;
    }

    public void setDescripcion(String descripcion) {
        this.descripcion = descripcion;
    }

}
```

MIEMBRO 1 (Bruno Marenco Cerqueira, 2º Parte)

Dentro del proyecto denominado "gestionEmpresa", el primer miembro de la pareja creará una clase denominada "listarTrabajadoresEmpresa.java" cuyo cometido sea crear los objetos departamentos y empleados y añadirlos a una colección de la siguiente forma:

- En primer lugar el alumno creará 4 objetos departamentos cuyo nombre identificativo será: "Desarrollo", "Sistemas", "Contabilidad", "Ventas". El resto de valores de los atributos queda a elección del alumno.

- Dentro de la clase **DatosPrueba.java** creada por el miembro 2 (Elisabeth), donde ha creado dos métodos para poder trabajar con estos datos fácilmente, el miembro 1 (Bruno) ha implementado el código para crear los objetos departamentos, (haciendo uso de las constantes creadas por Elisabeth) que se puede ver en la siguiente imagen.

```
// Creamos los 4 departamentos.
Departamento d1 = new Departamento(DESARROLLO, "Departamento centrado en el desarrollo y mantenimiento de software.");
Departamento d2 = new Departamento(SISTEMAS, "Departamento centrado en el desarrollo y mantenimiento de Sistemas.");
Departamento d3 = new Departamento(CONTABILIDAD, "Departamento auxiliar ocupado de la gestión contable y financiera de la empresa.");
Departamento d4 = new Departamento(VENTAS, "Departamento ocupado en marketing, ventas y relaciones institucionales.");
```

- En segundo lugar el alumno creará 12 objetos empleados que tendrán los siguientes trabajos: 3 empleados serán "Progamadores" y trabajarán en el departamento de "Desarrollo", otros 3 empleados serán "Administradores" y trabajarán en el departamento de "Sistemas", otros 3 serán contables y trabajarán en el departamento de "Contabilidad" y los tres últimos serán "Comerciales" y trabajarán en el departamento de "Ventas".

Nuevamente, dentro de la clase **DatosPrueba.java** creada por el miembro 2 (Elisabeth), el miembro 1 (Bruno) ha implementado el código para crear los objetos empleados, (haciendo uso de las constantes creadas por Elisabeth) que se puede ver en la siguiente imagen.

```
// Creamos los 12 empleados, 3 de cada departamento.
Empleado e1 = new Empleado(1, "Ramón", "Martínez", "Programador", 5000.50, DESARROLLO);
Empleado e2 = new Empleado(2, "Manuel", "Torres", "Programador", 3000, DESARROLLO);
Empleado e3 = new Empleado(3, "Ana", "Torquemada", "Programador", 6000, DESARROLLO);

Empleado e4 = new Empleado(4, "Sara", "Castro", "Administrador", 4000, SISTEMAS);
Empleado e5 = new Empleado(5, "Arancha", "López", "Administrador", 1500, SISTEMAS);
Empleado e6 = new Empleado(6, "Aitor", "Iturriaga", "Administrador", 3000.50, SISTEMAS);

Empleado e7 = new Empleado(7, "Cleide", "Seco", "Contable", 5000, CONTABILIDAD);
Empleado e8 = new Empleado(8, "Celia", "Ruiz", "Contable", 4000, CONTABILIDAD);
Empleado e9 = new Empleado(9, "Juan", "Martínez", "Contable", 1000, CONTABILIDAD);

Empleado e10 = new Empleado(10, "Sol", "López", "Comercial", 2000, VENTAS);
Empleado e11 = new Empleado(11, "Luis", "Batugo", "Comercial", 1500, VENTAS);
Empleado e12 = new Empleado(12, "Pedro", "Yanes", "Comercial", 30000, VENTAS);
```

- A continuación se creará un ArrayList denominado listaDepartamentos donde se añadirán los 4 objetos departamentos creados y un ArrayLis denominado listaEmpleados donde se añadirán los 12 objetos empleados creados.

-Dentro del mismo archivo, hemos importado la biblioteca “import java.util.ArrayList;” para poder crear ArrayLists sin problema. El miembro 1 (Bruno) ha implementado el código para crear los ArrayList pedidos y se han añadido los objetos creados anteriormente con el método “.add”.

```
// ArrayList denominado listaEmpleados donde se añaden los 12 objetos empleados creados.
List<Empleado> listaEmpleados = new ArrayList<>();

listaEmpleados.add(e1);
listaEmpleados.add(e2);
listaEmpleados.add(e3);
listaEmpleados.add(e4);
listaEmpleados.add(e5);
listaEmpleados.add(e6);
listaEmpleados.add(e7);
listaEmpleados.add(e8);
listaEmpleados.add(e9);
listaEmpleados.add(e10);
listaEmpleados.add(e11);
listaEmpleados.add(e12);
```

```
// ArrayList denominado listaDepartamentos donde se añaden los 4 objetos departamentos creados.
List<Departamento> listaDepartamentos = new ArrayList<>();

listaDepartamentos.add(d1);
listaDepartamentos.add(d2);
listaDepartamentos.add(d3);
listaDepartamentos.add(d4);
```

- Por último se listará la información almacenada en los ArrayList creados con anterioridad mostrando en primer lugar la información del departamento y a continuación la información de los empleados que trabajan en dicho departamento.

-Se ha creado una clase **ListarTrabajadoresEmpresa.java,** para poder listar los ArrayList y mostrarlos por pantalla, para ello se ha utilizado un bucle for, que itera por el objeto ArrayList con los departamentos e imprime las informaciones de ese departamento, antes de ir al segundo elemento del ArrayList se hace otro bucle for a través del ArrayList con los empleados y se imprimen todos los empleados que cumplen la condición de pertenecer al departamento en el que nos encontramos en el primer bucle, para ello usamos una condición if, de esta forma se muestra por pantalla cada departamento y una lista con todos los empleados y sus informaciones de cada departamento.

```java
public class ListarTrabajadoresEmpresa {

    /**
     * Lista los departamentos y empleados recibidos por parámetro. Para cada
     * departamento se muestran los empleados que trabajan en él.
     *
     * @param listaDepartamentos lista de departamentos.
     * @param listaEmpleados lista de empleados.
     */
    public static void listarEmpleadosPorDepartamentos(List<Departamento> listaDepartamentos, List<Empleado> listaEmpleados) {
        for (Departamento dep : listaDepartamentos) {
            System.out.println("El departamento " + dep.getNombre() + " se define como: " + dep.getDescripcion());
            System.out.println("Aquí trabajan:");
            for (Empleado emp : listaEmpleados) {
                if (emp.getNombreDep().equals(dep.getNombre())) {
                    System.out.println(emp.getNombre() + " " + emp.getApellidos() + ", trabaja como " + emp.getTrabajo() + " y gana " + emp.getSalario() + " euros.");
                }
            }
        }
    }
}
```

-En la siguiente imagen se puede ver el resultado de esta parte.

```
El departamento Desarrollo se define como: Departamento centrado en el desarrollo y mantenimiento de software.
Aquí trabajan:
Ramón Martínez, trabaja como Programador y gana 5000.5 euros.
Manuel Torres, trabaja como Programador y gana 3000.0 euros.
Ana Torquemada, trabaja como Programador y gana 6000.0 euros.
El departamento Sistemas se define como: Departamento centrado en el desarrollo y mantenimiento de Sistemas.
Aquí trabajan:
Sara Castro, trabaja como Administrador y gana 4000.0 euros.
Arancha López, trabaja como Administrador y gana 1500.0 euros.
Aitor Iturriaga, trabaja como Administrador y gana 3000.5 euros.
El departamento Contabilidad se define como: Departamento auxiliar ocupado de la gestión contable y financiera de la empresa.
Aquí trabajan:
Cleide Seco, trabaja como Contable y gana 5000.0 euros.
Celia Ruiz, trabaja como Contable y gana 4000.0 euros.
Juan Martínez, trabaja como Contable y gana 1000.0 euros.
El departamento Ventas se define como: Departamento ocupado en marketing, ventas y relaciones institucionales.
Aquí trabajan:
Sol López, trabaja como Comercial y gana 2000.0 euros.
Luis Batugo, trabaja como Comercial y gana 1500.0 euros.
Pedro Yanes, trabaja como Comercial y gana 30000.0 euros.
```

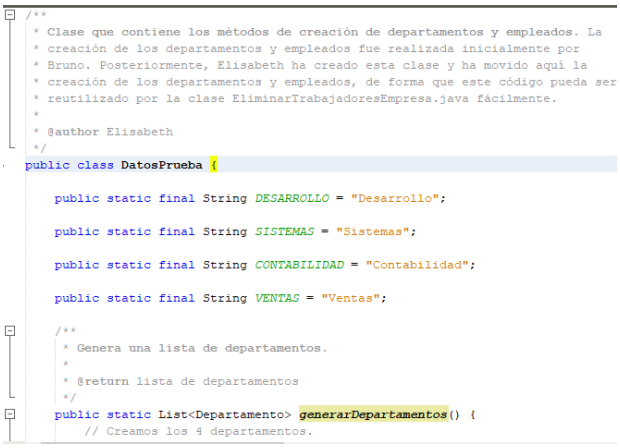
MIEMBRO 2 (Elisabeth Guisado Reina, 2º Parte)

Dentro del proyecto denominado "gestionEmpresa", el segundo miembro de la pareja creará una clase denominada "EliminarTrabajadoresEmpresa.java" cuyo cometido sea el de borrar aquellos empleados cuyo sueldo sea mayor que 2000 € de la lista de empleados. Para ello se deben seguir los siguiente pasos:

- Partiendo de las mismas especificaciones de los ArrayList listaDepartamentos y listaEmpleados creados con anterioiridad, se deben ir recorriendo los ArrayList de Empleados y determinar aquellos que ganan un salario mayor a 2000 €, y en caso de que ganen más de 2000€ eliminar dicho empleado de listaEmpleados.

- He creado la clase ***DatosPrueba.java*** para generar los datos de prueba de forma que puedan ser utilizados facilmente tanto desde ***ListarTrabajadoresEmpresa.java*** como desde ***EliminarTrabajadoresEmpresa.java***.

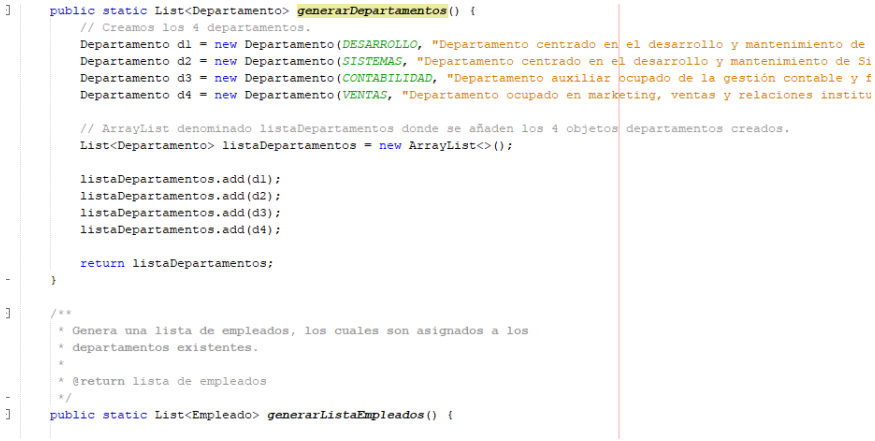
Adjunto las capturas de pantalla:



```java
/**
 * Clase que contiene los métodos de creación de departamentos y empleados. La
 * creación de los departamentos y empleados fue realizada inicialmente por
 * Bruno. Posteriormente, Elisabeth ha creado esta clase y ha movido aquí la
 * creación de los departamentos y empleados, de forma que este código pueda ser
 * reutilizado por la clase EliminarTrabajadoresEmpresa.java fácilmente.
 *
 * @author Elisabeth
 */
public class DatosPrueba {

    public static final String DESARROLLO = "Desarrollo";

    public static final String SISTEMAS = "Sistemas";

    public static final String CONTABILIDAD = "Contabilidad";

    public static final String VENTAS = "Ventas";

    /**
     * Genera una lista de departamentos.
     *
     * @return lista de departamentos
     */
    public static List<Departamento> generarDepartamentos() {
        // Creamos los 4 departamentos.
```

Como puede observarse, he definido unas constantes para el nombre de los departamentos, de forma que éstas puedan ser usadas tanto en la creación de los propios departamentos como desde la creación de los empleados.

A continuación se muestra el método utilizado para generar el listado de departamentos. La definición de esta clase y este método fue realizada por el miembro 2 (Elisabeth), mientras que el contenido de este método fue implementado por el miembro 1 (Bruno).



```
public static List<Departamento> generarDepartamentos() {
    // Creamos los 4 departamentos.
    Departamento d1 = new Departamento(DESARROLLO, "Departamento centrado en el desarrollo y mantenimiento de
    Departamento d2 = new Departamento(SISTEMAS, "Departamento centrado en el desarrollo y mantenimiento de Si
    Departamento d3 = new Departamento(CONTABILIDAD, "Departamento auxiliar ocupado de la gestión contable y f
    Departamento d4 = new Departamento(VENTAS, "Departamento ocupado en marketing, ventas y relaciones institu

    // ArrayList denominado listaDepartamentos donde se añaden los 4 objetos departamentos creados.
    List<Departamento> listaDepartamentos = new ArrayList<>();

    listaDepartamentos.add(d1);
    listaDepartamentos.add(d2);
    listaDepartamentos.add(d3);
    listaDepartamentos.add(d4);

    return listaDepartamentos;
}

/**
 * Genera una lista de empleados, los cuales son asignados a los
 * departamentos existentes.
 *
 * @return lista de empleados
 */
public static List<Empleado> generarListaEmpleados() {
```

Al igual que con la creación de los departamentos, el miembro 2 (Elisabeth) realizó la definición del método, mientras que el cuerpo del método en sí fue implementado por el miembro 1 (Bruno).

```
public static List<Empleado> generarListaEmpleados() {

    // Creamos los 12 empleados, 3 de cada departamento.
    Empleado e1 = new Empleado(1, "Ramón", "Martínez", "Programador", 5000.50, DESARROLLO);
    Empleado e2 = new Empleado(2, "Manuel", "Torres", "Programador", 3000, DESARROLLO);
    Empleado e3 = new Empleado(3, "Ana", "Torquemada", "Programador", 6000, DESARROLLO);

    Empleado e4 = new Empleado(4, "Sara", "Castro", "Administrador", 4000, SISTEMAS);
    Empleado e5 = new Empleado(5, "Arancha", "López", "Administrador", 1500, SISTEMAS);
    Empleado e6 = new Empleado(6, "Aitor", "Iturriaga", "Administrador", 3000.50, SISTEMAS);

    Empleado e7 = new Empleado(7, "Cleide", "Seco", "Contable", 5000, CONTABILIDAD);
    Empleado e8 = new Empleado(8, "Celia", "Ruiz", "Contable", 4000, CONTABILIDAD);
    Empleado e9 = new Empleado(9, "Juan", "Martínez", "Contable", 1000, CONTABILIDAD);

    Empleado e10 = new Empleado(10, "Sol", "López", "Comercial", 2000, VENTAS);
    Empleado e11 = new Empleado(11, "Luis", "Batugo", "Comercial", 1500, VENTAS);
    Empleado e12 = new Empleado(12, "Pedro", "Yanes", "Comercial", 30000, VENTAS);

    // ArrayList denominado listaEmpleados donde se añaden los 12 objetos empleados creados.
    List<Empleado> listaEmpleados = new ArrayList<>();

    listaEmpleados.add(e1);
    listaEmpleados.add(e2);
    listaEmpleados.add(e3);
    listaEmpleados.add(e4);
```

- Por último se debe mostrar el número de Empleados que hay en cada departamento una vez se hayan realizado los borrados.

Para este apartado hemos implementado un método estático que recibe la lista de empleados y el salario límite. El método recorre la lista de empleados, utilizando un ListIterator, ya que es necesario cuando vamos a realizar borrado de elementos, y elimina aquellos empleados que tengan un salario mayor al recibido por parámetro. Cada una de las acciones realizadas son notificadas al usuario mediante su impresión en consola.

```java
/**
 * Clase cuya funcionalidad es eliminar empleados con sueldo mayor a uno dado y
 * realizar el conteo de los empleados disponibles de cada departamento.
 *
 * @author Elisabeth
 */
public class EliminarTrabajadoresEmpresa {

    /**
     * Recorre todos los empleados y elimina aquellos que tengan un sueldo
     * superior al recibido por parámetro.
     *
     * @param listaEmpleados lista de empleados.
     * @param sueldo sueldo a partir del cual un empleado debe ser eliminado de
     * la lista recibida como parámetro.
     */
    public static void eliminarTrabajadoresSegunSueldo(List<Empleado> listaEmpleados, double sueldo) {
        System.out.println("*** Eliminando los trabajadores cuyo sueldo es mayor de " + sueldo + " euros ***");
        ListIterator<Empleado> empleadosIterator = listaEmpleados.listIterator();
        while (empleadosIterator.hasNext()) {
            Empleado e = empleadosIterator.next();
            if (e.getSalario() > sueldo) {
                System.out.println("Eliminando el empleado: " + e.getNombre()
                        + " " + e.getApellidos() + " (sueldo = " + e.getSalario() + " €)");
                empleadosIterator.remove();
            }
        }
        System.out.println("********** Fin del proceso de eliminación **********");
```

Llegados a este punto, la siguiente captura de pantalla muestra como recorremos la lista de empleados para contabilidar el número de empleados de cada departamento. Para ello definimos 4 variables enteras, una por cada departamento, y las iremos incrementando cuando sea oportuno conforme vamos recorriendo la lista de empleados recibida.

```java
    /**
     * Realiza un conteo de todos los trabajadores mostrando el número total de
     * empleados por cada departamento.
     *
     * @param listaEmpleados lista de empleados.
     */
    public static void contarTrabajadores(List<Empleado> listaEmpleados) {
        System.out.println("*** Calculando cuántos trabajadores hay disponibles en cada departamento: ***");
        int numEmpDepDesarrollo = 0;
        int numEmpDepSistemas = 0;
        int numEmpDepContabilidad = 0;
        int numEmpDepVentas = 0;

        for (Empleado e : listaEmpleados) {
            switch (e.getNombreDep()) {
                case DESARROLLO:
                    numEmpDepDesarrollo++;
                    break;
                case SISTEMAS:
                    numEmpDepSistemas++;
                    break;
                case CONTABILIDAD:
                    numEmpDepContabilidad++;
                    break;
                case VENTAS:
                    numEmpDepVentas++;
                    break;
```

Por último, una vez recorridos y contabilizados todos los empleados, mostramos la información en consola:

```
                    case VENTAS:
                        numEmpDepVentas++;
                        break;
                    default:
                        System.out.println("Departamento inválido: " + e.getNombreDep());
                }
            }
            System.out.println("*****************************************************");
            System.out.println("**** NÚMERO TOTAL DE EMPLEADOS POR DEPARTAMENTO ****");
            System.out.println("*****************************************************");
            System.out.println("** " + DESARROLLO + ": " + numEmpDepDesarrollo);
            System.out.println("** " + SISTEMAS + ": " + numEmpDepSistemas);
            System.out.println("** " + CONTABILIDAD + ": " + numEmpDepContabilidad);
            System.out.println("** " + VENTAS + ": " + numEmpDepVentas);
            System.out.println("*****************************************************");
            System.out.println("********** Fin del proceso **********");
        }
    }
```

-Clase PracticaColaborativa.java (Realizada por los dos miembros: Elisabeth Guisado y Bruno Marenco)

Esta clase la hemos realizado entre los dos miembros para que sea el punto de entrada de la ejecución del proyecto JAVA. Contiene un metodo main desde el cual se generan los empleados y departamentos, asi como también se realizan las diferentes acciones que se piden a lo largo de la práctica.
Adjuntamos la captura de pantalla.

```
/**
 * Esta clase representa el punto de entrada de ejecución de la práctica. A
 * través del método main definido en esta clase se generan los departamentos y
 * empleados, así como las diferentes acciones que deben realizarse sobre éstos:
 * <p>
 * <ul>
 * <li>Generación de departamentos y empleados</li>
 * <li>Listado de empleados por departamento</li>
 * <li>Eliminación de ciertos empleados</li>
 * <li>Recuento de los empleados de cada departamento</li>
 * </ul>
 * </p>
 *
 * @author Elisabeth / Bruno
 */
public class PracticaColaborativa {

    public static void main(String[] args) {
        List<Departamento> listaDepartamentos = DatosPrueba.generarDepartamentos();
        List<Empleado> listaEmpleados = DatosPrueba.generarListaEmpleados();

        ListarTrabajadoresEmpresa.listarEmpleadosPorDepartamentos(listaDepartamentos, listaEmpleados);
        EliminarTrabajadoresEmpresa.eliminarTrabajadoresSegunSueldo(listaEmpleados, 2000);
        EliminarTrabajadoresEmpresa.contarTrabajadores(listaEmpleados);
    }
}
```