Actividad 2

**Enunciado**

1. Para que tres números enteros den un triángulo válido, se tiene que dar la circunstancia de que la suma de dos cualquiera de los tiene que ser mayor que la del otro. Por tanto, el ejercicio consta de:
   1. Pedir tres números por consola, e informar si el triángulo es correcto, o no es correcto.
   2. Y si es correcto además decir si es:
      1. Equilátero: los tres lados iguales.
      2. Isósceles: dos iguales y uno desigual
      3. Escaleno: los tres lados distintos
2. Realizar un juego para adivinar un número. Para ello pedir un número N aleatorio de 1 a 40, y luego ir pidiendo números indicando “mayor” o “menor” según sea mayor o menor con respecto a N. El proceso termina cuando el usuario acierta. Y escribimos la cantidad de intentos antes del número exacto.

1. Crear dos Funciones y un Algoritmo que las usa:
   1. La primera función devuelve el literal de sexo, por tanto, recibe un carácter, si el carácter es “H” o “h”, devolvemos “Hombre”; si es “M” o “m” devuelve “Mujer”; si recibe cualquier otra cosa devuelve “Sexo Erróneo”.
   2. La segunda función, recibe un nuero Real que representa el salario de una persona, y devuelve un literal de acuerdo a las siguientes opciones:
      * Salario < de 25.000 : “Salario Bajo”.
      * Salario entre 25.000 y 40.000: “Salario Medio”.
      * Por encima de 45.000: “Salario Alto”

El algoritmo lee nombres de personas sucesivamente hasta que me teclean la palabra FIN (en cualquiera de sus configuraciones de mayúsculas y minúsculas). Si no es FIN, pedimos su salario y la letra de su sexo. El algoritmo informa de su nombre, su literal del salario (llamar a la función correspondiente) y su literal del sexo (llamar a la función).

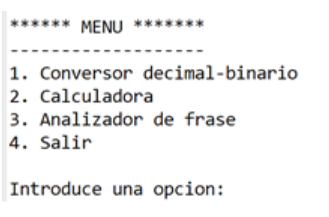
Al final del algoritmo informar:

* Cuántos nombres me han introducido, la suma de los salarios y la media de los salarios.

Actividad 3

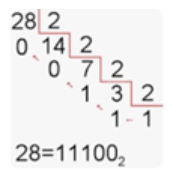
**Enunciado**

 Escribe un programa que muestre el siguiente menú una y otra vez hasta que el usuario escoja la opción “4. Salir” o mientras la opción que introduzca no se encuentre entre las ofrecidas en el menú.

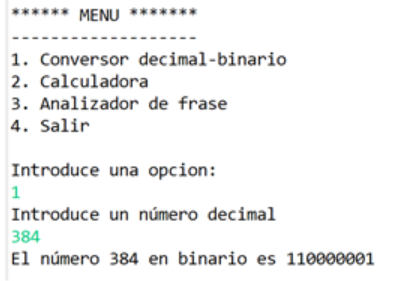


Opción 1:

Se pedirá al usuario por consola un número entero y se mostrará por pantalla dicho número en binario. Recuerda el proceso para pasar a binario:

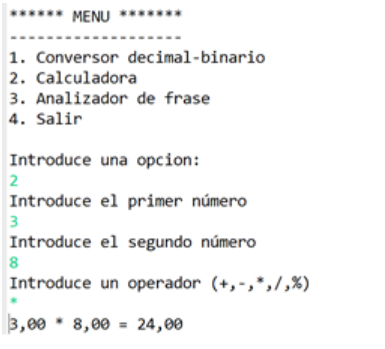
A

El resultado que deberá obtenerse por pantalla es el siguiente:



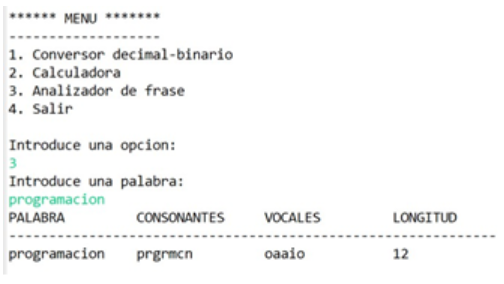
Opción 2:

Se pedirá por consola al usuario dos números decimales y un operador (+, -, \*, /, %). Y se realizará la operación escogida, mostrando el resultado con 2 decimales. El resultado por pantalla será el siguiente:



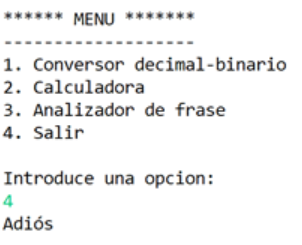
Opción 3:

Se pedirá al usuario por consola una palabra. El programa deberá mostrar todas sus consonantes y todas sus vocales tal y como se muestra a continuación:



Opción 4:

La ejecución del programa terminará diciendo adiós y ya no volverá a mostrarse el menú.



Actividad 4

**Enunciado**

Crear una clase Empleado con los siguientes atributos privados:

* idEmpleado: int
* nombre, apellidos, email : String
* salario: double
* complemento: double
* sexo: char
* idDepartamento: int

Crear un constructor por defecto, y otro con todos los parámetros.

Crear los getter and setter para cada atributo privado.

Redefinir el método toString()

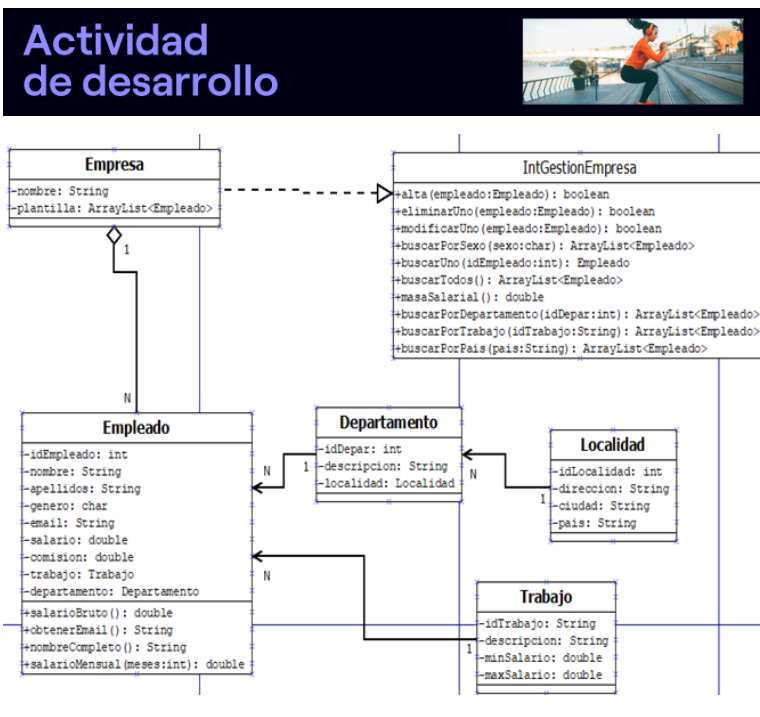
 La clase Empleado, tendrá los siguientes métodos propios:

* salarioBruto() : double. Devolver la suma del salario y el complemento
* salarioMensual(int meses) : double
* literalSexo() : String. H -> “Hombre”, M -> “Mujer”
* obtenerEmail() : String. Devolver un String con la primera letra del nombre + primer apellido, en minúsculas.
* nombreCompleto(): String. Devolver la concatenación del nombre + “ “ + apellidos

Crear una clase Principal con el método main y:

* Crear un objeto de la clase Empleado llamando al constructor con todos los parámetros y asignándoles valor a cada atributo.
* Escribir por consola(syso), los datos de esta empleado llamando a los métodos de la clase que hemos propuesto; un syso por cada método de la clase.
* Crear un objeto vacío, y rellenar cada dato usando los métodos set. Invocar en el syso a la variable de este empleado.

Actividad 5



**Enunciado**

Tomamos como base la actividad 4 de la Unidad Formativa 7. Pero hay que tener en cuenta que los atributos han cambiado un poco por la Asociación de Empleado con la clase Trabajo y con la clase Departamento, como ves en la figura del esquema de clases de la Actividad.

Los métodos de Empleado son los mismos que los de las actividades por lo que los puedes copiar.

* La clase Empleado, tendrá los siguientes métodos propios:
* salarioBruto() : double. Devolver la suma del salario y el complemento
* salarioMensual(int meses) : double
* literalSexo() : String. H -> “Hombre”, M -> “Mujer”
* obtenerEmail() : String. Devolver un String con la primera letra del nombre + primer apellido, en minúsculas.
* nombreCompleto(): String. Devolver la concatenación del nombre + “ “ + apellidos
* Hay que crear de cada clase Localidad, Departamento y Empleado:
* Los datos privados indicados en el esquema.
* Un constructor sin parámetros y un constructor con todo.
* Los getter and setter de cada atributo privado.
* Redefinir toString en cada clase.
* Redefinir equals y hashcode de cada clase:
  + Dos empleados son iguales si su idEmpleado es el mismo.
  + Dos departamentos son iguales si su idDepar es el mismo.
  + Dos localidades son iguales si su idLocalidad es la misma.

Antes de crear el Interface y la clase Empresa debemos probar que las clases creadas están operativas.

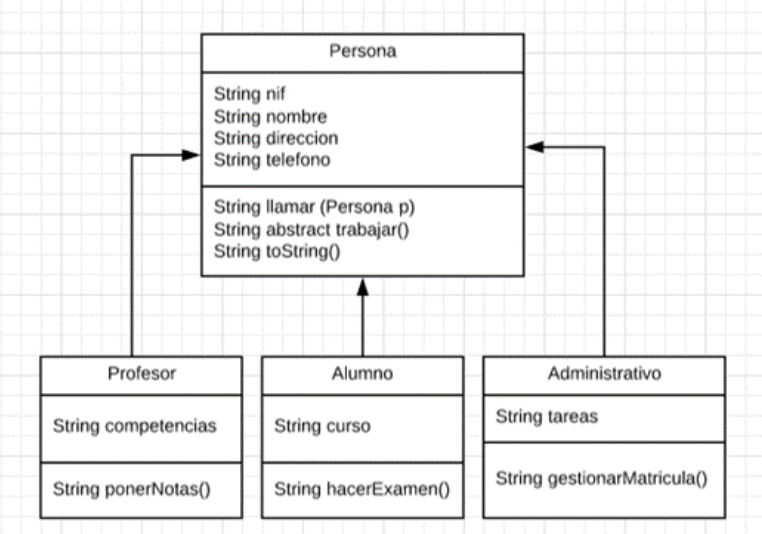
* Para ello, crear una clase TestingEmpleado que muestre al menos:
* Crear al menos, un objeto Localización, otro de departamento con esa Localización, un Trabajo, y un Empleado con su departamento y su Trabajo.
* La información completa de un empleado SIN usar toString:
  + Nombre, apellidos, género, nombre del departamento, nombre de su trabajo, dirección , ciudad y país.
* Extraer del empleado su trabajo y su departamento en variables de su tipo. Y pintarlas con syso.
* Crear el Interface IntGestionEmpresa.
* Crear la Clase Empresa, con los datos privados del diagrama, y en esta clase definir:
* Un constructor con un parámetro tipo String con nombreEmpresa, y en él hacer el new ArrayList.
* Hacer geter an settr SOLO de nombre.
* Implementar los métodos del interface e implementar el contenido de cada uno de ellos.
* Crear un método privado:
  + cargarDatos(): void. En donde creamos Empleados para probar, de distintos trabajos, departamentos con sus localidades.
* Crear una clase Principal con main static, con una prueba en secuencia que muestre cada información que sale de la prueba de cada método de la gestión de la Empresa. Separar cada prueba con un syso que indique claramente qué se va a probar, y el resultado.

Actividad 6

Enunciado

Para la gestión de un centro educativo, un programa necesita contemplar las entidades profesor, alumno y administrativo, teniendo en cuenta que todos, a su vez, son personas.

Crea la estructura de clases que se corresponda con el siguiente diagrama:

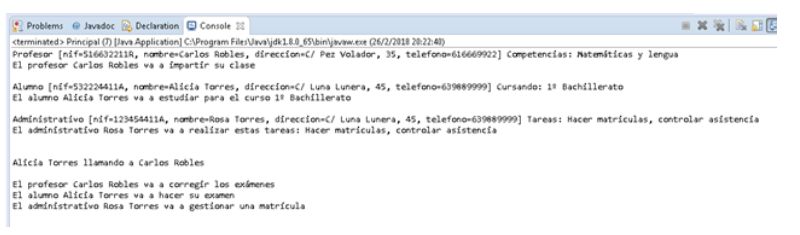


* La clase *Persona* será abstracta.
* Todas las propiedades serán privadas y accesibles mediante métodos *get*/*set*.
* El método *llamar()* devolverá una cadena similar a esta: “Alicia Torres llamando a Carlos Robles”. Como parte de la cadena se utilizará la propiedad *nombre* del objeto al que se aplica y del objeto pasado como argumento.
* El método *trabajar()* se especializará en cada subclase para devolver una cadena que indique el tipo de tarea que realiza cada perfil de persona. Por ejemplo:
  + *Profesor*: “El profesor Carlos Robles va a impartir su clase”. Como parte de la cadena usarás la propiedad *nombre*.
  + *Alumno*: “El alumno Alicia Torres va a estudiar para el curso 1º Bachillerato”. Como parte de la cadena usarás las propiedades *nombre* y *curso*.
  + *Administrativo*: “El administrativo Rosa Torres va a realizar estas tareas: hacer matrículas, controlar asistencia”. Como parte de la cadena utilizarás las propiedades *nombre* y *tareas*.
* Además, las clases derivadas ampliarán su especialización con métodos propios como *ponerNotas()*, *hacerExamen()*o *gestionarMatricula()*. Cada uno de estos métodos devolverá una cadena de este tipo:
  + *ponerNotas()*: “El profesor Carlos Robles va a corregir los exámenes”. Como parte de la cadena tienes que utilizar la propiedad *nombre*.
  + *hacerExamen()*: “El alumno Alicia Torres va a hacer su examen”. Como parte de la cadena debes utilizar la propiedad *nombre*.
  + *gestionarMatricula()*: “El administrativo Rosa Torres va a gestionar una matrícula”. Como parte de la cadena utilizarás la propiedad *nombre*.
* Además de lo especificado en el diagrama de clases, debes sobrescribir el método *toString()* en cada una de las clases para que muestre toda la información posible. Puedes hacerlo de forma automática con ayuda de Eclipse (clic derecho / Source / Generate toString…).

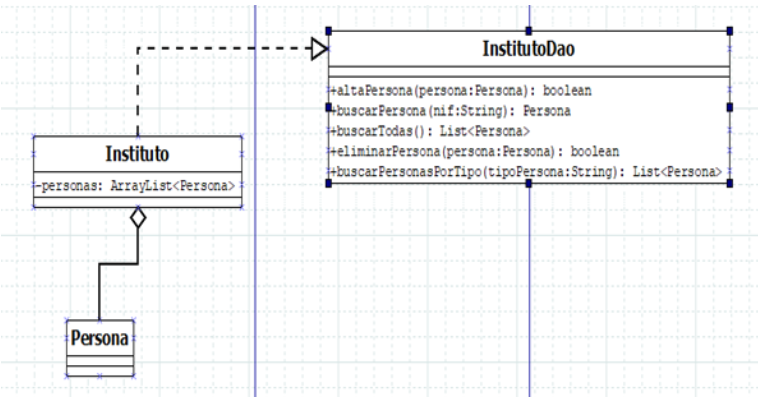
1. Para terminar crea la clase *Principal* con método *main* y realiza las siguientes acciones:

* Crea un objeto *Profesor*, un objeto *Alumno* y un objeto *Administrativo*.
* Ejecuta el método *toString()* sobre cada uno de ellos.
* Ejecuta el método *trabajar()* sobre cada uno de ellos.
* Ejecuta el método *llamar(Persona p)* para que el alumno llame al profesor.
* Ejecuta el método *ponerNotas()* del objeto *Profesor*.
* Ejecuta el método *hacerExamen()* del objeto *Alumno*.
* Ejecuta el método *gestionarMatricula()*del objeto *Administrativo*.

Ejemplo de ejecución:



Segunda Parte,



Crear el Interface **InstitutoDao** con los métodos que vienen en el esquema, y una clase **Instituto**, que tiene como dato privado una Lista de Personas, que implementa los métodos del interface que implementa.

Todos los métodos son independientes del tipo de Persona que hay dentro: Administrativo, Profesor o Alumno. Excepto buscarPersonasPorTipo, que devuelve las Personas que hay dentro de la lista que sean de ese tipo (recuerda “*instanceof*”).

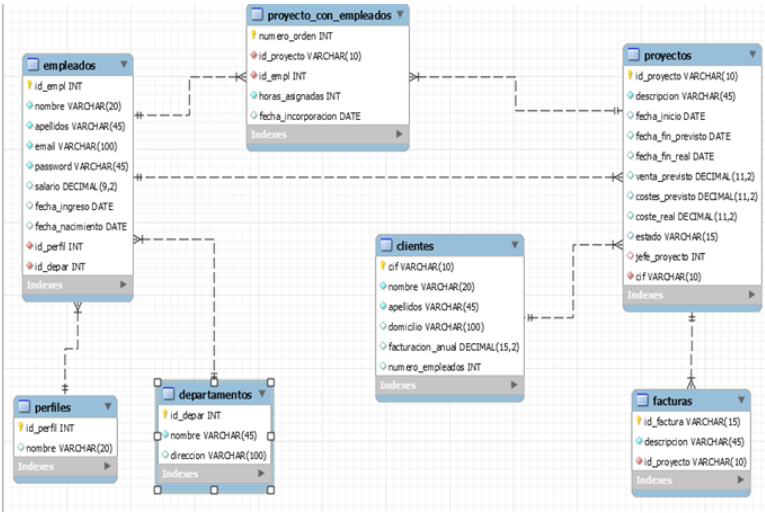
Para probar crea Personas de distinto tipo y haz uno o más **Testing** que pruebe todos los métodos de **InstitutoDao**.

Para la prueba de buscar por tipo, cuando a la Clase **TestBuscar**… le llega la lista, recorre esta lista y llama al método que es propio de cada clase, Alumno, Profesor o Administrativo (“recuerda el casting de persona a la Clase correspondiente).

Actividad 7

**Enunciado**

Realizar una Aplicación de gestión de proyectos, de acuerdo con la siguiente base de Datos:



La estructura de paquetes del proyecto es:

1. javabeans
2. daos
3. conexion
4. testing
5. principales

1. *CLIENTES*

Consideraciones:

Hacer una clase con main estático GestionClientes con un  menú con las siguientes opciones:

1. Alta del Cliente
2. Buscar un Cliente
3. Mostrar Todos.
4. Eliminar un cliente
5. Salir

Definir el javabean de Cliente, el interface para CRUD de cliente incluido findAll(), y el Interface de ClienteDaoImpl, para implementar los métodos del Interface, accediendo a bases de datos.

1. *RECURSOS HUMANOS*

Crear los javabean Departamento , Perfil y Empleado, los interface y las clases que implementan el interface.

CONSIDERACIONES:

* La clase Empleado, tendrá los siguientes métodos propios:
  + salarioBruto() : double
  + salarioMensual(int meses) : double
  + literalSexo() : String. H -> Hombre, M -> mujer
  + obtenerEmail() : String. Primera letra del nombre + primer apellido, en minúsculas
  + nombreCompleto(): String. Nombre + “ “ + apellidos
* Métodos de EmpleadoDao, además de los CRUD, y findAll():
  + empleadosByDepartamento(int idDepar): List<Empleado>
  + empleadosBySexo(char sexo): List<Empleado>
  + empleadosByApellido(String subcadena): List<Empleado>
  + salarioTotal(): double
  + salarioTotal(int idDepar): double. Salario Total para un

1. *PROYECTOS*– Empleados en proyecto

Métodos propios de Proyecto:

* margenPrevisto():double. Importe de venta – coste previsto
* margenReal(): double Importe de venta – gastos reales
* diferenciaGastos(): double. Gasto real – gasto previsto
* diferenciaFinPrevistoReal(): int . Días entre fin previsto y fin real

Métodos ProyectoDao, además de CRUD y findAll(), implementados con acceso a Base de datos:

* proyectosByEstado(String estado): List<Proyecto>
* proyectosByCliente(String cif) : List<Proyecto>
* proyectosByJefeProyectoAndByEstado(int jefeProyecto, String estado): List<Proyecto>
* importesVentaProyectosTerminados(): double
* margenBrutoProyectosTerminados():double Diferencia suma Importes venta y gastos reales.
* diasATerminoProyectoActivo(String codigoProyecto): int. Cuantos días quedan para terminar el proyecto (diferencia entre fecha\_fin\_previsto y la fecha de hoy

Métodos EmpleadosEnProyecto:

* costeHorasAsignadas(): double . Horas \* precio/hora

Métodos EmpleadosEnProyectoDao, si quieres  CRUD y findAll() y obligatorio, implementado con acceso a bases de datos:

* **empleadosByProyecto(String codigoProyecto): List<EmpleadosEnProyecto>**
* **asignarEmpleadosAProyecto(List<EmpleadosEnProyecto> empleados): int**
* horasAsignadasAProyecto(String codigoProyecto): int. Suma de las horas de los empleados asignados al proyecto.
* costeActualDeProyecto(String codigoProyecto): double . horas\*coste-hora de cada empleado asignado al proyecto.
* margenActualProyecto(String codigoProyecto): double. Importe\_venta del proyecto – costeActual del Proyecto

**Hacer clases de prueba por cada Javabean-Interface-ImplMy8, para probar la funcionalidad.**