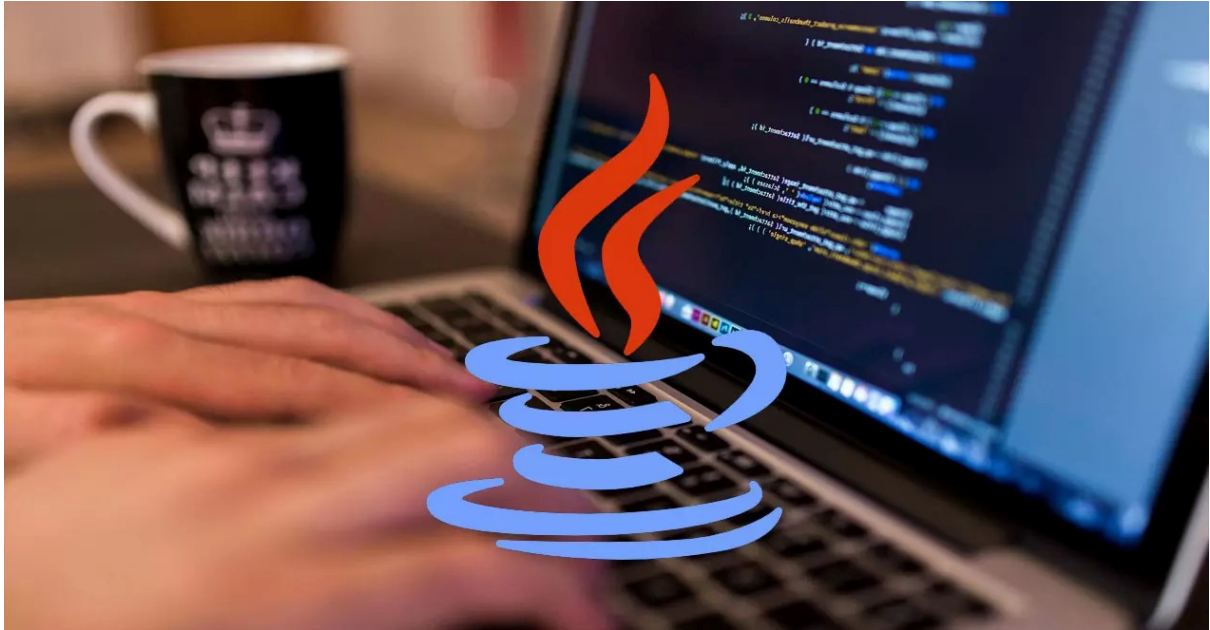


Prácticas Desarrollo en Entorno de Servidor

Sergio Gavela Jiménez

ODAW2



Índice

Portada.....	1 página
Índice.....	2 página
Ejercicio 1.1.....	3 - 4 página
Ejercicio 1.2	4 página
Ejercicio 1.3	5-8 página
Ejercicio 1.4	9-15 página
Ejercicio 1.5	16-22 página
Ejercicio 1.6	23-24 página
Ejercicio 1.7	25-26 página
Ejercicio 1.8	27-29 página

Ejercicio 1.1

1.1. Realiza un pequeño estudio de los principales lenguajes de programación y frameworks de Front-End y Back-End más solicitados en las ofertas de empleo en A Coruña y en España.

Lenguajes de Programación Front-End:

JavaScript: JavaScript es esencial para el desarrollo web Front-End.

HTML/CSS: Aunque no son lenguajes de programación en el sentido tradicional, HTML y CSS son fundamentales para la construcción de la estructura y el diseño de las páginas web.

Frameworks de Front-End:

React.js: React es uno de los frameworks más populares para el desarrollo Front-End. Es mantenido por Facebook y se utiliza ampliamente para crear interfaces de usuario interactivas y reactivas.

Angular: Desarrollado por Google, Angular es otro framework Front-End robusto que se utiliza para crear aplicaciones web complejas.

Vue.js: framework Front-End progresivo que ha ganado popularidad debido a su facilidad de uso y su capacidad para integrarse fácilmente en proyectos existentes.

Lenguajes de Programación Back-End:

Python: Python es ampliamente utilizado en el desarrollo web Back-End debido a su facilidad de lectura y escritura de código.

Java: Java sigue siendo una opción sólida para el desarrollo Back-End, especialmente en empresas grandes y en aplicaciones empresariales.

Frameworks de Back-End:

Django: Un framework de Python que facilita la creación rápida de aplicaciones web robustas y seguras.

Spring : Spring es un framework muy utilizado para el desarrollo Back-End en Java. Proporciona una amplia variedad de herramientas y módulos para crear aplicaciones empresariales escalables y seguras.

1.2. Realiza un pequeño estudio de los principales IDE del mercado, específicos para lenguajes concretos o generalistas.

IDEs Generales:

Visual Studio Code (VS Code):

Lenguajes compatibles: Amplia compatibilidad con una variedad de lenguajes de programación.

Características destacadas: Extensibilidad a través de extensiones, depuración integrada, control de versiones, resaltado de sintaxis, IntelliSense (autocompletado de código), y una comunidad activa de desarrolladores de extensiones.

IntelliJ IDEA:

Lenguajes compatibles: Java, Kotlin, Groovy y otros.

Características destacadas: Refactorización inteligente, análisis estático de código, soporte para frameworks populares (Spring, Android, etc.), y herramientas de desarrollo web.

Eclipse:

Lenguajes compatibles: Principalmente Java, pero admite otros lenguajes a través de complementos.

Características destacadas: Amplia comunidad de desarrollo de complementos, depuración integrada, herramientas de modelado y soporte para desarrollo de aplicaciones empresariales.

IDEs Específicos para Lenguajes:

PyCharm:

Lenguaje específico: Python.

Características destacadas: Soporte avanzado para Python, depuración, análisis de código, y gestión de entornos virtuales.

Android Studio:

Lenguaje específico: Java, Kotlin (desarrollo de aplicaciones Android).

Características destacadas: Diseñador de interfaces de usuario, emuladores Android, herramientas de rendimiento y análisis de código.

Ejercicio 1.3

1.3. Realiza una sencilla aplicación de consola que tenga definida una clase llamada Persona con atributos privados: dni, nombre y edad. Añádele un constructor que incluya todos los atributos, getters, setters, toString y equals y hashCode basado en el dni. Incluye un programa que defina un ArrayList con 6 personas (puedes meter sus valores por hardcode o hacer un sencillo método para que el usuario introduzca sus valores).

Desarrolla distintos métodos en el programa anterior con las siguientes características:

- Método al que se le pasa un ArrayList de Persona y devuelve la edad del mayor.
- Método al que se le pasa un ArrayList de Persona y devuelve la edad media.
- Método al que se le pasa un ArrayList de Persona y devuelve el nombre del mayor.
- Método al que se le pasa un ArrayList de Persona y devuelve la Persona mayor.
- Método al que se le pasa un ArrayList de Persona y devuelve todos los mayores de edad.
- Método al que se le pasa un ArrayList de Persona y devuelve todos los que tienen una edad mayor o igual a la media.

En el main del programa haz llamadas a los métodos anteriores y muestra por pantalla su resultado.

Nota: a la hora de crear los métodos puedes reutilizar código de forma que unos llamen a otros y minimizar el código duplicado.

Solución:

He creado una clase Persona para poner todos tus atributos constructor,getter y setters.

```
public static class Persona {
    private String dni;
    private String nombre;
    private int edad;

    public Persona(String dni, String nombre, int edad) {
        this.dni = dni;
        this.nombre = nombre;
        this.edad = edad;
    }

    public String getDni() {
        return dni;
    }

    public void setDni(String dni) {
        this.dni = dni;
    }

    public String getNombre() {
        return nombre;
    }

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

    public int getEdad() {
        return edad;
    }

    public void setEdad(int edad) {
        this.edad = edad;
    }
}
```

He creado métodos estáticos para cada método requerido para poder llamarlos sin tener que crear ninguna instancia de los mismos desde la propia main.

Cada método creado retorna cada valor que pide el ejercicio.

```

package com.example;

import java.util.ArrayList;

public class Main {
    Run | Debug
    public static void main(String[] args) {
        ArrayList<Persona> personas = new ArrayList<Persona>();

        personas.add(new Persona(dni:"02553236M", nombre:"carlos", edad:20));
        personas.add(new Persona(dni:"02553237M", nombre:"pepe", edad:22));
        personas.add(new Persona(dni:"02553238M", nombre:"juan", edad:40));
        personas.add(new Persona(dni:"02553239M", nombre:"sergio", edad:76));
        personas.add(new Persona(dni:"02553249M", nombre:"sara", edad:66));
        personas.add(new Persona(dni:"02553230M", nombre:"alex", edad:4));

        System.out.println(devuelveMasEdad(personas));
        System.out.println(devuelveEdadMedia(personas));
        System.out.println(devuelveNombreMayor(personas));
        //devuelven objetos Personas
        System.out.println(devuelvePersonaMayor(personas));
        System.out.println(devuelveMayoresEdad(personas));
        System.out.println(devuelveMayoreQueMedia(personas));
    }
}

```

Inicializo el Array de personas y le introduzco valores.

Desde main llamo a las diferentes funciones que he ido creando para resolver el problema.

```

public static int devuelveMasEdad(ArrayList<Persona> personas) {
    Persona personaMayor = null;
    for (int i = 0; i < personas.size(); i++) {
        if (personaMayor == null || personaMayor.getEdad() < personas.get(i).getEdad()) {
            personaMayor = personas.get(i);
        }
    }

    return personaMayor.getEdad();
}

public static float devuelveEdadMedia(ArrayList<Persona> personas) {
    float sumaEdades = 0;
    for (int i = 0; i < personas.size(); i++) {
        sumaEdades += personas.get(i).getEdad();
    }
    return (sumaEdades/(float)personas.size());
}

public static String devuelveNombreMayor(ArrayList<Persona> personas) {
    Persona personaMayor = null;
    int i = com.example.Main.devuelveNombreMayor(ArrayList<Persona>)
    for (int i = 0; i < personas.size(); i++) {
        if (personaMayor == null || personaMayor.getEdad() < personas.get(i).getEdad()) {
            personaMayor = personas.get(i);
        }
    }
    return personaMayor.getNombre();
}

```

Podemos ver que con estas tres funciones devolvemos la edad de la persona mas mayor, la media de edad y también el nombre de la persona más mayor.

```
PS D:\DAW2\SERVIDOR> d:;; cd 'd:\DAW2\SERVIDOR'; & 'C:\Program Files\Java\jre-1.8\bin\java.exe' '-cp' 'D:\DAW2\SERVIDOR\actividad 1.3\demo\target\classes' 'com.example.Main'
76
38.0
sergio
PS D:\DAW2\SERVIDOR>
```

Como podemos ver lo hace sin ningún tipo de problema.

```
public static Persona devuelvePersonaMayor(ArrayList<Persona> personas) {
    Persona personaMayor = null;
    for (int i = 0; i < personas.size(); i++) {
        if (personaMayor == null || personaMayor.getEdad() < personas.get(i).getEdad()) {
            personaMayor = personas.get(i);
        }
    }
    return personaMayor;
}

public static ArrayList<Persona> devuelveMayoresEdad(ArrayList<Persona> personas) {
    ArrayList<Persona> mayoresEdad = new ArrayList<>(personas);
    for (int i = 0; i < personas.size(); i++) {
        if ( (personas.get(i).getEdad())>=18) {
            mayoresEdad.add(personas.get(i));
        }
    }
    return mayoresEdad;
}

public static ArrayList<Persona> devuelveMayoreQueMedia(ArrayList<Persona> personas) {
    ArrayList<Persona> mayoresQueMedia = new ArrayList<>(personas);
    float media = devuelveEdadMedia(personas);
    for (int i = 0; i < personas.size(); i++) {
        if (personas.get(i).getEdad()>media) {
            mayoresQueMedia.add(personas.get(i));
        }
    }
    return mayoresQueMedia;
}
```

En cambio en estos tres últimos funciones nos devuelven objetos Persona o ArrayList, entonces al imprimirlos por consola no los podemos ver con claridad.

```
PS D:\DAW2\SERVIDOR> d:;; cd 'd:\DAW2\SERVIDOR'; & 'C:\Program Files\Java\jre-1.8\bin\java.exe' '-cp' 'D:\DAW2\SERVIDOR\actividad 1.3\demo\target\classes' 'com.example.Main'
com.example.Main$Persona@15db9742
[com.example.Main$Persona@6d06d69c, com.example.Main$Persona@7852e922, com.example.Main$Persona@4e25154f, com.example.Main$Persona@15db9742, com.example.Main$Persona@70dea4e, com.example.Main$Persona@5c647e05, com.example.Main$Persona@6d06d69c, com.example.Main$Persona@7852e922, com.example.Main$Persona@4e25154f, com.example.Main$Persona@15db9742, com.example.Main$Persona@70dea4e]
[com.example.Main$Persona@6d06d69c, com.example.Main$Persona@7852e922, com.example.Main$Persona@4e25154f, com.example.Main$Persona@15db9742, com.example.Main$Persona@70dea4e, com.example.Main$Persona@5c647e05, com.example.Main$Persona@4e25154f, com.example.Main$Persona@15db9742, com.example.Main$Persona@70dea4e]
PS D:\DAW2\SERVIDOR>
```


Ejercicio 1.4

1.4. Se desea hacer la gestión de las habitaciones de un hotel. Todas las habitaciones tienen un número

de habitación y un proceso de check-in y check-out. En el hotel hay tres tipos de habitaciones, aunque

podría haber más en el futuro:

- 3 habitaciones Lowcost (cuesta 50 euros/día todo el año).

- 10 habitaciones dobles. Tienen una tarifa normal de 100 euros/día y un incremento del 20%

si el día de salida es abril, julio o agosto.

- 5 habitaciones Suite. 200 euros/día con 20% de descuento para estancias de 10 o más días.

- Debes crear una o más clases para las habitaciones y una clase para el Hotel. La clase Hotel

tendrá las 18 habitaciones en un ArrayList y las marcará inicialmente como “no ocupadas”.

- El programa tendrá un menú para hacer check-in entre las habitaciones libres, check-out entre

las ocupadas y listar habitaciones libres y ocupadas.

- El check-in es común para todas las habitaciones, consiste en marcar la habitación como ocupada. El check-out consiste en marcar ción como libre la habitación y calcular el importe a pagar

que se calculará en función del tipo de habitación y de los días de estancia (quizás sea necesario almacenar la fecha de llegada en el momento del check-in)

- Sugerencia: Para probar el programa, al hacer el check-out, puedes considerar cada día como

un segundo, así, si han pasado 3 segundos, considerar 3 días.

Solución:

Para el siguiente problema he creado una clase abstracta padre que se llama “Habitación” que tendrá los atributos y métodos comunes necesarios para las clases hijas como pueden ser el checkIn() o el checkOut().

Después he creado tres clases hijas que heredan de Habitación que son “Doble”, “LowCost” y “Suite”.

También una clase hotel en la que va a contener una ArrayList de objetos Habitación usando polimorfismo y creando objetos que realmente van a ser objetos que hereden de ello.

```

1 package com.example;
2
3
4 public abstract class Habitacion {
5     protected int precioDia;
6     protected boolean ocupada;
7     protected long momentoEntrada;
8     protected int numero;
9     protected String tipo;
10
11
12     protected Habitacion(int numero){
13         this.numero = numero;
14         this.ocupada = false;
15     }
16
17
18     protected void checkIn(){
19         if(this.ocupada == false){
20             this.ocupada = true;
21             momentoEntrada= System.currentTimeMillis();
22             System.out.println(x:"Check In realizado con exito");
23         }else{
24             System.out.println(x:"Habitacion no disponible ");
25         }
26     }
27
28 }
29
30 protected abstract void checkOut();
31
32
33
34

```

La clase Padre Habitacion con todos los atributos requeridos y los métodos necesarios.

```
J App.java  J Hotel.java 2  J Doble.java  J Suite.java X  J LowCost.java  J Habitacion.java
demo > src > main > java > com > example > J Suite.java > Suite > checkOut()
1  package com.example;
2
3  public class Suite extends Habitacion {
4
5      public Suite(int numero) {
6          super(numero);
7          this.precioDia = 200;
8          this.tipo = "suite";
9      }
10
11      @Override
12      protected void checkOut() {
13          if (this.ocupada = true) {
14              this.ocupada = false;
15              long momentoSalida = System.currentTimeMillis();
16              int segundosTranscurridos = ((int) (momentoSalida - momentoEntrada) / 1000);
17
18              if (segundosTranscurridos >= 10) {
19
20                  System.out.println("El importe por " + segundosTranscurridos + " dias seria de "
21                      + this.precioDia * segundosTranscurridos + " pero con el 20% de descuento se quedaria en "
22                      + this.precioDia * segundosTranscurridos * 0.8);
23
24              } else {
25
26                  System.out.println("El importe por " + segundosTranscurridos + " dias seria de "
27                      + this.precioDia * segundosTranscurridos);
28              }
29
30          } else {
31              System.out.println(x:"La habitacion no se encuentra disponible");
32          }
33      }
34
35  }
36
37
```

La clase Suite en la que definimos el método checkOut() ya que en la clase padre hemos especificado que todos las hijas han de tenerlo.

También captamos el momento en el que se hace el checkIn y lo comparamos con el momento que se hace el checkOut así podemos saber cuantos segundos han pasado para poder usarlo en nuestra aplicación.

```
App.java  Hotel.java 2  Doble.java  Suite.java  LowCost.java X  Habitacion.java

demo > src > main > java > com > example > LowCost.java > ...
1  package com.example;
2
3  public class LowCost extends Habitacion {
4      public LowCost(int numero) {
5          super(numero);
6          this.precioDia = 50;
7          this.tipo = "lowCost";
8      }
9
10     public void checkOut() {
11         if (this.ocupada == true) {
12             this.ocupada = false;
13             long momentoSalida = System.currentTimeMillis();
14             int segundosTranscurridos = ((int) (momentoSalida - momentoEntrada) / 1000);
15             System.out.println("El importe por " + segundosTranscurridos + " dias seria de "
16                 + this.precioDia * segundosTranscurridos);
17         } else {
18             System.out.println(x:"La habitacion no se encuentra disponible");
19         }
20     }
21 }
22
23
```

La clase LowCost

```
App.java  Hotel.java 2  Doble.java X  Suite.java  LowCost.java  Habitacion.java

demo > src > main > java > com > example > Doble.java > Doble > checkOut()
1  package com.example;
2
3  import java.util.Calendar;
4
5  public class Doble extends Habitacion {
6      public Doble(int numero) {
7          super(numero);
8          this.precioDia = 100;
9          this.tipo = "doble";
10     }
11
12     @Override
13     protected void checkOut() {
14         if (this.ocupada == true) {
15             this.ocupada = false;
16             long momentoSalida = System.currentTimeMillis();
17             int segundosTranscurridos = ((int) (momentoSalida - momentoEntrada) / 1000);
18             // funcion que devuelve el mes actual
19             int mes = Calendar.getInstance().get(Calendar.MONTH);
20
21             if (mes == 3 || mes == 6 || mes == 7) {
22                 System.out.println("El importe por " + segundosTranscurridos + " dias seria de "
23                     + ((float) (this.precioDia * segundosTranscurridos) * 1.2));
24             } else {
25                 int segundosTranscurridos - com.example.Doble.checkOut()
26                 System.out.println("El importe por " + segundosTranscurridos + " dias seria de "+this.precioDia * segundosTranscurridos);
27             }
28         }
29         else {
30             System.out.println(x:"La habitacion no se encuentra disponible");
31         }
32     }
33 }
34
35
```

La clase Doble

```

J App.java  J Hotel.java  X  J Doble.java  J Suite.java  J LowCost.java  J Habitacion.java
demo > src > main > java > com > example > J Hotel.java > Hotel > Hotel()
1  package com.example;
2
3  import java.util.ArrayList;
4  import java.util.Scanner;
5  import java.util.concurrent.TimeUnit;
6
7  public class Hotel {
8      protected ArrayList<Habitacion> habitaciones;
9
10     public Hotel() {
11
12         habitaciones = new ArrayList<>();
13     }
14
15     public void CreaHabitaciones() {
16         // Creacion de LowCost
17         for (int i = 0; i < 3; i++) {
18             habitaciones.add(new LowCost(i + 1));
19         }
20
21         // Creacion de Dobles
22         for (int i = 3; i < 13; i++) {
23             habitaciones.add(new Doble(i + 1));
24         }
25
26         // Creacion de Suite
27         for (int i = 13; i < 18; i++) {
28             habitaciones.add(new Suite(i + 1));
29         }
30     }
31
32     public void menu() {
33         int respuesta;
34         System.out.print(s:"Bienvenido al hotel, ");
35         do {
36             Scanner teclado = new Scanner(System.in);
37             int habitacionSeleccionada;
38             System.out.println(x:"que desea hacer? \n [0] Salir \n [1] Check In \n [2] Check Out ");
39             respuesta = teclado.nextInt();
40             if (respuesta == 1) {
41                 muestraHabitacionesDisponibles();
42                 System.out.println(x:"Elija el numero de la habitacion en el que le gustaria hacer check in");
43                 habitacionSeleccionada = teclado.nextInt();
44                 if(habitacionSeleccionada>0 && habitacionSeleccionada<18){
45                     // hacemos check in de la habitacion seleccionada
46                     habitaciones.get(habitacionSeleccionada - 1).checkIn();
47                 }
48             }
49         } while (respuesta != 0);
50     }
51 }

```

La clase Hotel en la que se guarda el ArrayList de habitaciones y contiene los métodos para mostrar el menu y para mostrar las habitaciones ocupadas y las disponibles.

```

J App.java  X  J Hotel.java  J Doble.java  J Suite.java  J LowCost.java
demo > src > main > java > com > example > J App.java > ...
1  package com.example;
2
3
4  public class App
5  {
6      Run | Debug
7      public static void main( String[] args )
8      {
9          Hotel miHotel = new Hotel();
10         miHotel.CreaHabitaciones();
11         miHotel.menu();
12     }
13 }
14
15

```

Aquí está mi main, en la que creo el hotel y llamo a sus métodos principales.

```
PS D:\DAW2\SERVIDOR\ejercicio1.4\demo> & 'C:\
Bienvenido al hotel, que desea hacer?
[0] Salir
[1] Check In
[2] Check Out
█
```

Menú principal

```
Habitacion doble numero 9
Habitacion doble numero 10
Habitacion doble numero 11
Habitacion doble numero 12
Habitacion doble numero 13
Habitacion suite numero 14
Habitacion suite numero 15
Habitacion suite numero 16
Habitacion suite numero 17
Habitacion suite numero 18
Elija el numero de la habitacion en el que le gustaria hacer check in
█
```

Al elegir check in nos salen las habitaciones disponibles para poder hacerlo y nos pide que marquemos una.

```
14
Check In realizado con exito
que desea hacer?
[0] Salir
[1] Check In
[2] Check Out
█
```

Nos confirma que ha sido realizado con éxito.

```
que desea hacer?
[0] Salir
[1] Check In
[2] Check Out
2
Habitaciones Ocupadas:
Habitacion suite numero 14
Elija el numero de la habitacion que le gustaria hacer check out
█
```

ahora al pulsar checkout nos sale la habitación número 14 disponible para realizarlo

```
El importe por 89 días seria de 17800 pero con el 20% de descuento se quedaria en 14240.0
que desea hacer?
[0] Salir
[1] Check In
[2] Check Out
```

Al pasar 89 segundos desde que se realizó el check in nos cobra 89 días y al ser más de 10 días nos realiza el descuento especial que tienen las habitaciones suite.

Cuestión 1: ¿Habitación debería ser una clase abstracta o una interfaz? ¿Por qué?

Debe de ser Clase abstracta porque el método checkIn() es común para todas las clases hijas y podemos definirlo en la clase padre.

Cuestión 2: ¿Es útil que el método toString () en la clase Habitación?

Es útil para mostrar el los datos de cada objeto instanciado aunque yo lo he hecho con métodos propios.

Ejercicio 1.5

1.5. Se desea desarrollar un programa gestione los dispositivos domóticos de un edificio.

Para ello

tendremos un ArrayList que contenga en principio 3 elementos, uno para el termostato de la calefacción, otro para el ascensor y otro para el dial de la radio del hilo musical, pero en el futuro

podríamos tener más elementos.

El termostato tiene una fecha de última revisión, un valor entero en grados centígrados: mínimo 15,

máximo 80 y la temperatura inicial es 20. El ascensor tiene una planta en la que se encuentra, pudiendo

ser desde 0 a 8. La planta inicial es la cero. El dial de radio va desde 88.0 a 104.0

avanzando de décima

en décima, siendo el valor inicial 88.0.

De cada elemento (y los futuros que aparezcan) deben ser capaces de realizar las siguientes funciones:

- subir(), incrementa una unidad el elemento domótico. Devuelve true si la operación se realiza

correctamente o false si no se puede hacer por estar al máximo.

- bajar(): decrementa una unidad el elemento domótico. Devuelve true si la operación se realiza

correctamente o false si no se puede hacer por estar al mínimo.

- reset(): pone en la situación inicial el elemento domótico. No devuelve nada.

- verEstado(): Devuelve un String con el tipo de dispositivo y su estado actual.

Además, el termostato debe incluir un nuevo método:

- revisar(). Fija a la fecha de revisión a la fecha actual. No devuelve nada.

Una vez definido el sistema, crea un programa que inicie un ArrayList con una instancia de cada uno

de los 3 dispositivos y luego mediante un menú nos permita hacer operaciones, primero qué operación

queremos realizar (0:Salir, 1:subir un dispositivo, 2:bajar un dispositivo, 3: resetear un dispositivo,

4:revisar termostato) y luego seleccionar sobre qué elemento queremos trabajar (verificando que sea

un valor entre 0 y el tamaño del ArrayList -1)

- El menú, además de las opciones nos mostrará siempre el estado de todos los dispositivos.

Solución:

Para este ejercicio he creado una interfaz de Elemento que engloba a todos los elementos que se citan en el ejercicio.


```
package com.example;  
  
public interface Elemento {  
    public boolean subir();  
    public boolean bajar();  
    public void reset();  
    public String verEstado();  
    public void revisar();  
}
```

En la interfaz Elemento creo todos los comportamientos que deben tener todas sus clases hijas.

Luego en cada clase en particular implemento los métodos que definí en la interfaz.

```
1 package com.example;
2
3 import java.time.LocalDate;
4
5 public class Ascensor implements Elemento{
6     private int planta ;
7     private LocalDate revision;
8
9     public Ascensor() {
10         revision = LocalDate.now();
11         planta = 0;
12     }
13
14
15
16     public boolean subir(){
17
18         if(planta<8 && planta>=0){
19             this.planta += 1;
20             return true;
21         }else{
22             System.out.println(x:"Se ha alcanzado la planta maxima");
23             return false;
24         }
25     }
26
27
28     public boolean bajar(){
29
30         if(planta<=8 && planta>0){
31             this.planta -= 1;
32             return true;
33         }else{
34             System.out.println(x:"Se ha alcanzado la planta minima");
35             return false;
36         }
37     }
38
39
40     public void reset(){
41         planta =0;
42     }
43
44     public String verEstado(){
45         return "Este dispositivo es un Ascensor se encuentra en la planta "+ plan
46     }
```

Clase ascensor.

```
1 package com.example;
2
3 import java.time.LocalDate;
4
5 public class Radio implements Elemento{
6     private float dial ;
7     private LocalDate revision;
8
9     public Radio() {
10         dial = 88;
11         revision = LocalDate.now();
12     }
13
14     public boolean subir(){
15
16         if(dial>=88 && dial<104){
17             dial += 0.1;
18             return true;
19         }else{
20             System.out.println(x:"Error. Dial maximo alcanzado");
21             return false;
22         }
23     }
24
25     public boolean bajar(){
26
27         if(dial>88 && dial<=104){
28             dial -= 0.1;
29             return true;
30         }else{
31             System.out.println(x:"Error. Dial minimo alcanzado");
32             return false;
33         }
34     }
35
36     public void reset(){
37         dial =0;
38     }
39
40     public String verEstado(){
41         return "Este dispositivo es una radio se encuentra en el dial "+ di
42     }
43
44
45
46
```

Clase Radio.

The screenshot shows an IDE with the following components:

- EXPLORER:** Displays the project structure. The 'demo' folder contains a 'src' folder, which in turn contains a 'main' folder. Inside 'main', there is a 'java' folder, which contains a 'com' folder. The 'com' folder contains an 'example' folder, which contains the 'Termostato.java' file. Other files in the 'main' folder include 'App.java', 'Ascensor.java', 'Elemento.java', 'Menu.java', 'Radio.java', 'pom.xml', 'test', and 'target'.
- EDITOR:** Shows the code for 'Termostato.java'. The code is as follows:

```
1 package com.example;
2
3 import java.time.LocalDate;
4
5 public class Termostato implements Elemento {
6     private LocalDate revision;
7     private int temperatura ;
8
9
10    public Termostato() {
11        temperatura = 20;
12        this.revision = LocalDate.now();
13    }
14
15    public boolean subir(){
16
17        if(temperatura>=15 && temperatura<80){
18            temperatura += 1;
19            return true;
20        }else{
21            System.out.println(x:"Error. Maximo de grados alcanzados");
22            return false;
23        }
24    }
25
26    public boolean bajar(){
27
28        if(temperatura>15 && temperatura<=80){
29            temperatura -= 1;
30            return true;
31        }else{
32            System.out.println(x:"Error. Minimo de grados alcanzados");
33            return false;
34        }
35    }
36
37    public void reset(){
38        temperatura = 0;
39    }
40
41    public String verEstado(){
42        return "Este dispositivo es un Termostato y se encuentra a "+ temper
43    }
44
45
46
47
48
```

Clase Termostato.

```
demo > src > main > java > com > example > J Menu.java > Menu > start(ArrayList<Elemento>)  
1 package com.example;  
2  
3 import java.util.ArrayList;  
4 import java.util.Scanner;  
5  
6 public class Menu {  
7  
8     public static void start(ArrayList<Elemento> elementos){  
9         Scanner teclado = new Scanner(System.in);  
10        int respuestaAccion;  
11  
12        do{  
13            System.out.println(x:"Introduce que operacion quiere hacer:\n 0--  
14            respuestaAccion = teclado.nextInt();  
15            try {  
16                if(respuestaAccion!= 0){  
17                    int respuestaAccion = com.example.Menu.start(ArrayList<Elemento>)+  
18                }  
19                if(respuestaAccion==1){  
20                    elementos.get(respuestaIndex-1).subir();  
21                }else if(respuestaAccion == 2){  
22                    elementos.get(respuestaIndex-1).bajar();  
23                }else if(respuestaAccion == 3){  
24                    elementos.get(respuestaIndex-1).reset();  
25                }  
26                System.out.println(elementos.get(respuestaIndex-1).verEstado());  
27            }  
28            } catch (Exception e) {  
29                System.out.println(x:"Valores no validos. Intentelo de nuevo"  
30            }  
31  
32  
33  
34  
35        } while (respuestaAccion!= 0);  
36        System.out.println(x:"Hasta la vista.");  
37  
38        //list.forEach((n) -> System.out.println(n));  
39  
40  
41  
42  
43    }  
44  
45 }  
46
```

```
demo > src > main > java > com > example > J App.java > ...  
1 package com.example;  
2  
3 import java.util.ArrayList;  
4 import java.util.Arrays;  
5  
6 public class App {  
7  
8     Run | Debug  
9     public static void main(String[] args) {  
10  
11        Ascensor miAscensor = new Ascensor();  
12        Radio miRadio = new Radio();  
13        Termostato miTermostato = new Termostato();  
14        ArrayList<Elemento> elementos = new ArrayList<>(Arrays.asList(miAscensor...  
15        Menu.start(elementos);  
16    }  
17  
18 }  
19
```

Creo una clase menu para gestionar la salida por pantalla y los mensajes y lo lanzo todo desde mi clase App.

```

Bienvenido.
=====
Estado de los dispositivos:
El Ascensor se encuentra en la planta 0 y su fecha de revision es 2023-10-01
La radio se encuentra en el dial 88.0 y su fecha de revision es 2023-10-01
El Termostato y se encuentra a 20º y su fecha de revision es 2023-10-01
=====
Introduce que operacion quiere hacer:
0: Salir
1: Subir dispositivo
2: Bajar dispositivo
3: Resetear un dispositivo
4: Hacer Revision
1
Que elemento de los 3 quieres modificar? [1]Ascensor [2]Radio [3]Termostato
1
Planta ascendida correctamente.
=====

```

El menú te pide que introduzcas una acción que quieras realizar y después el dispositivo que quieres actualizar.

como podemos ver el ascensor ha subido correctamente.

```

=====
Introduce que operacion quiere hacer:
0: Salir
1: Subir dispositivo
2: Bajar dispositivo
3: Resetear un dispositivo
4: Hacer Revision
2
Que elemento de los 3 quieres modificar? [1]Ascensor [2]Radio [3]Termostato
1
Se ha alcanzado la planta minima
=====

```

Al intentar bajar cuando esta en el minimo no se realiza y te avisa de que se ha alcanzado la planta mínima.

Ejercicio 1.6

1.6. Realizar una clase llamada Primitiva que tenga definido una colección de 6 elementos con el resultado de un sorteo de la primitiva (serán 6 enteros con valores comprendidos entre 1 y 49 y sin repetidos). Los números se deberán mostrar ordenados ascendentemente así que decide cual es la colección que mejor se adapta a estos requisitos. La clase dispondrá de un constructor en el que se generan y almacenen esos números al azar, también tendrá un método al que se le pase una combinación jugada como parámetro (no necesariamente ordenada) y devuelva el número de aciertos. Realiza a continuación un programa en el que el usuario introduzca boletos (6 números sin repetidos) y le diga cuantos acertó. Realizar control de errores, tanto si el usuario introduce valores no numéricos, números repetidos o valores no comprendidos entre 1 y 49.

Solución:

Para este ejercicio he creado una clase Primitiva en la que he creado diferentes métodos para dividir el problema en problemas más pequeños y menos complejos y resolverlos poco a poco.

```

demo > src > main > java > com > example > Primitiva.java > Primitiva > preguntarNumeros()
9      public Primitiva() {
10         numeros = generaAleatorios();
11     }
12
13
14
15     public int jugada(int numero1, int numero2, int numero3 , int numero4, int nu
16
17         int numeroAciertos = 0;
18         ArrayList<Integer> intentos = new ArrayList<>();
19         intentos.add(numero1);
20         intentos.add(numero2);
21         intentos.add(numero3);
22         intentos.add(numero4);
23         intentos.add(numero5);
24         intentos.add(numero6);
25
26
27         for(int i = 0; i< this.numeros.size();i++){
28             if(intentos.contains(this.numeros.get(i))){
29                 numeroAciertos++;
30             }
31         }
32         return numeroAciertos;
33     }
34
35     public boolean repetido(ArrayList<Integer> numeros, int numero){
36         boolean estaRepetido = false;
37         for(int i = 0;i< numeros.size(); i++){
38             if(numero == numeros.get(i)){
39                 estaRepetido = true;
40             }
41         }
42         return estaRepetido;
43     }
44 }
45
46     public ArrayList<Integer> generaAleatorios(){
47         int numero;
48         ArrayList<Integer> numeros = new ArrayList<>();
49         for(int i = 0 ;i <6 ;i++){
50             do {
51                 numero = (int) (Math.random() * 49) + 1;
52             } while (repetido(numeros, numero));
53             numeros.add(numero);
54         }
55         return numeros;
56     }

```

Clase Primitiva

Primero genero los 6 números ganadores de forma aleatoria para que cada vez que la aplicación sea ejecutada sean diferentes.

Después pido al usuario que introduzca 6 números y los comparo con los premiados.

```

s\Java\jdk-15\bin\java.exe' '-XX:+ShowCodeDetails
\ejercicio1.6\demo\target\classes' 'com.example.4
Introduce un numero entre 1 y 49
1
Introduce un numero entre 1 y 49
3
Introduce un numero entre 1 y 49
4
Introduce un numero entre 1 y 49
5
Introduce un numero entre 1 y 49
6
Introduce un numero entre 1 y 49
7
El numero de aciertos es 1
Los numeros premiados son 27 30 5 41 45 22

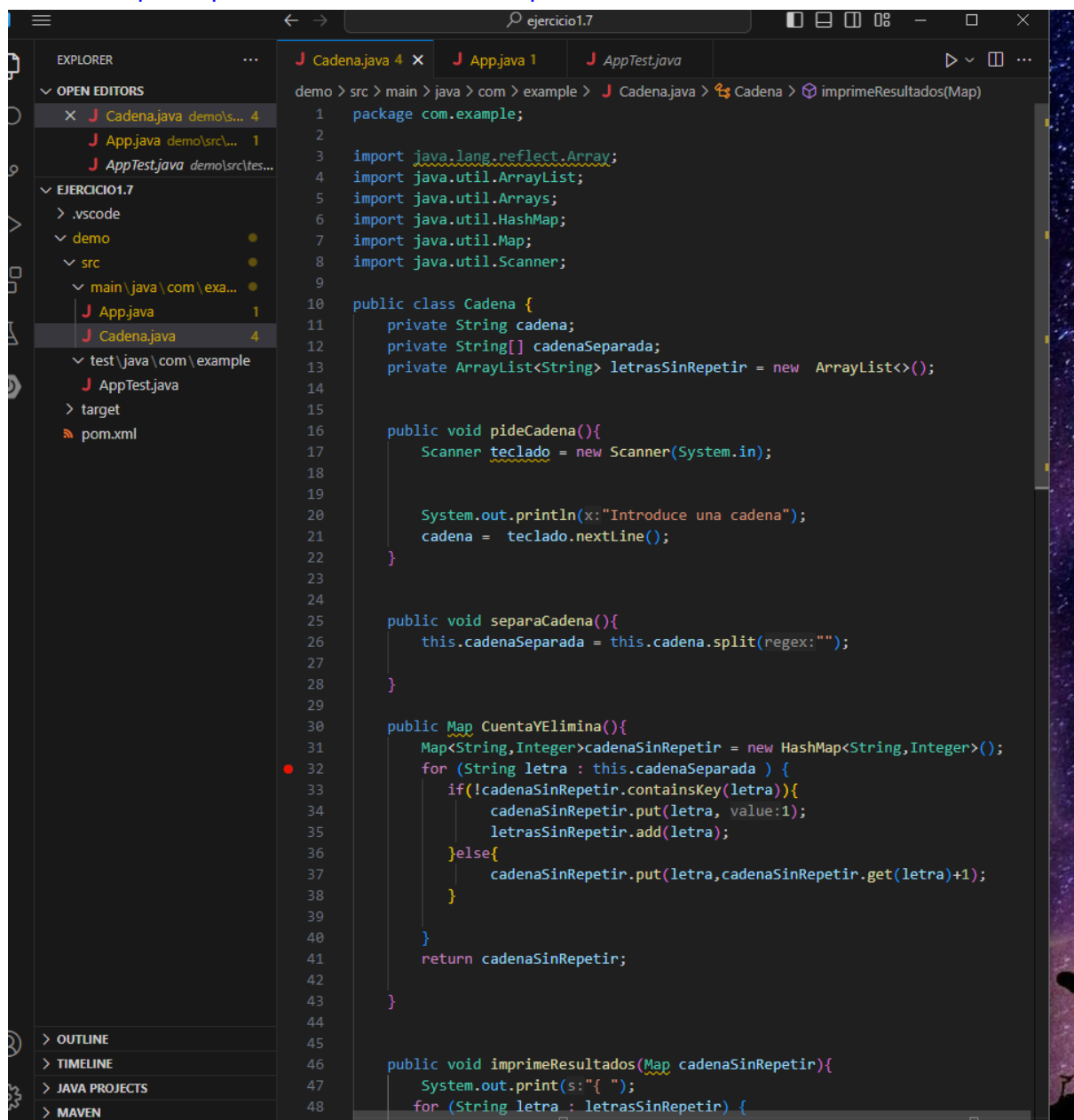
```


Ejercicio 1.7

1.7. Realizar un programa donde el usuario introduce un String y se muestre la cantidad de veces que aparece cada letra (ordenadas alfabéticamente, no por orden de aparición). Para tener un rendimiento óptimo, se debe recorrer el String solo una vez. Elige la colección óptima para minimizar el código necesario.

Solución:

Para este ejercicio he creado una clase Cadena que va a contener todos los métodos necesarios para operar con la cadena inicial que nos introduce el usuario.



```
demo > src > main > java > com > example > J Cadena.java > Cadena > imprimeResultados(Map)
1  package com.example;
2
3  import java.lang.reflect.Array;
4  import java.util.ArrayList;
5  import java.util.Arrays;
6  import java.util.HashMap;
7  import java.util.Map;
8  import java.util.Scanner;
9
10 public class Cadena {
11     private String cadena;
12     private String[] cadenaSeparada;
13     private ArrayList<String> letrasSinRepetir = new ArrayList<>();
14
15
16     public void pideCadena(){
17         Scanner teclado = new Scanner(System.in);
18
19
20         System.out.println(x:"Introduce una cadena");
21         cadena = teclado.nextLine();
22     }
23
24
25     public void separaCadena(){
26         this.cadenaSeparada = this.cadena.split(regex:"");
27     }
28
29
30     public Map CuentaYElimina(){
31         Map<String,Integer>cadenaSinRepetir = new HashMap<String,Integer>();
32         for (String letra : this.cadenaSeparada ) {
33             if(!cadenaSinRepetir.containsKey(letra)){
34                 cadenaSinRepetir.put(letra, value:1);
35                 letrasSinRepetir.add(letra);
36             }else{
37                 cadenaSinRepetir.put(letra,cadenaSinRepetir.get(letra)+1);
38             }
39         }
40         return cadenaSinRepetir;
41     }
42
43
44
45
46     public void imprimeResultados(Map cadenaSinRepetir){
47         System.out.print(s:"{ ");
48         for (String letra : letrasSinRepetir) {
```

Primero le pido la cadena al usuario, luego la separo para después poder ordenarla y contar cuantas veces se ha repetido cada letra.

```
Introduce una cadena
holaquetalestasyomuybien
{ a = 3    b = 1    e = 3    h = 1    i = 1    l = 2    m = 1    n = 1    o
= 2    q = 1    s = 2    t = 2    u = 2    y = 2    }
```

Ejercicio 1.8

1.8. Realiza un programa de consola para la gestión de los empleados de una empresa sobre una base de datos SQLite. De los empleados mantenemos un identificador único para cada empleado, su nombre completo y su salario. El programa inicialmente cargará la tabla de empleados desde un fichero csv y luego presentará un menú para realizar las siguientes operaciones:

- Alta: solicita el id, nombre y salario. Opcional: puedes validar que no exista el id.
- Baja: solicita un id y si lo encuentra elimina el empleado correspondiente.
- Modificación: solicita un id y si lo encuentra, solicita nuevo nombre y salario y lo modifica.
- Consulta: solicita salario mínimo y máximo, y muestra los empleados con salario comprendido entre los valores introducidos.

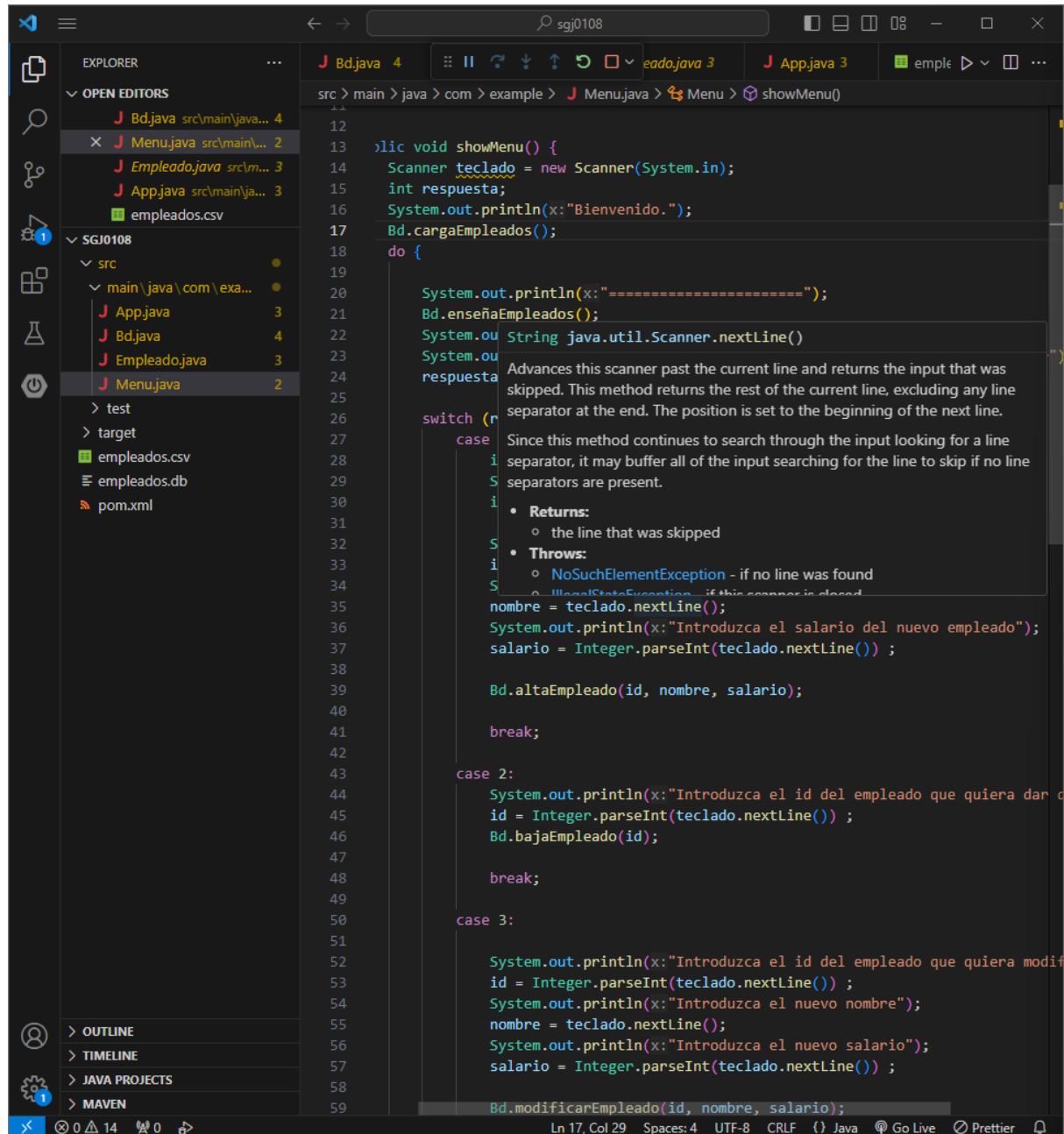
El menú, además de las acciones anteriores, siempre mostrará el conjunto de empleados existentes en la tabla en cada momento.

El profesor te entregará un esqueleto con el código del programa (carpeta CrudEmpleadoJdbc) para que lo completes y también el archivo con la base de datos SQLite, aunque este último se puede crear desde el plugin de Chrome: 'Sqlite Manager'. En la siguiente imagen se ve el proceso: se crearía la tabla con el nombre que queramos (por ejemplo: empleados) y luego en el botón inferior "Save" guardamos la base de datos, que contiene esa única tabla, con el nombre que queramos. La base de datos se guardará en un solo fichero.

Cuestiones:

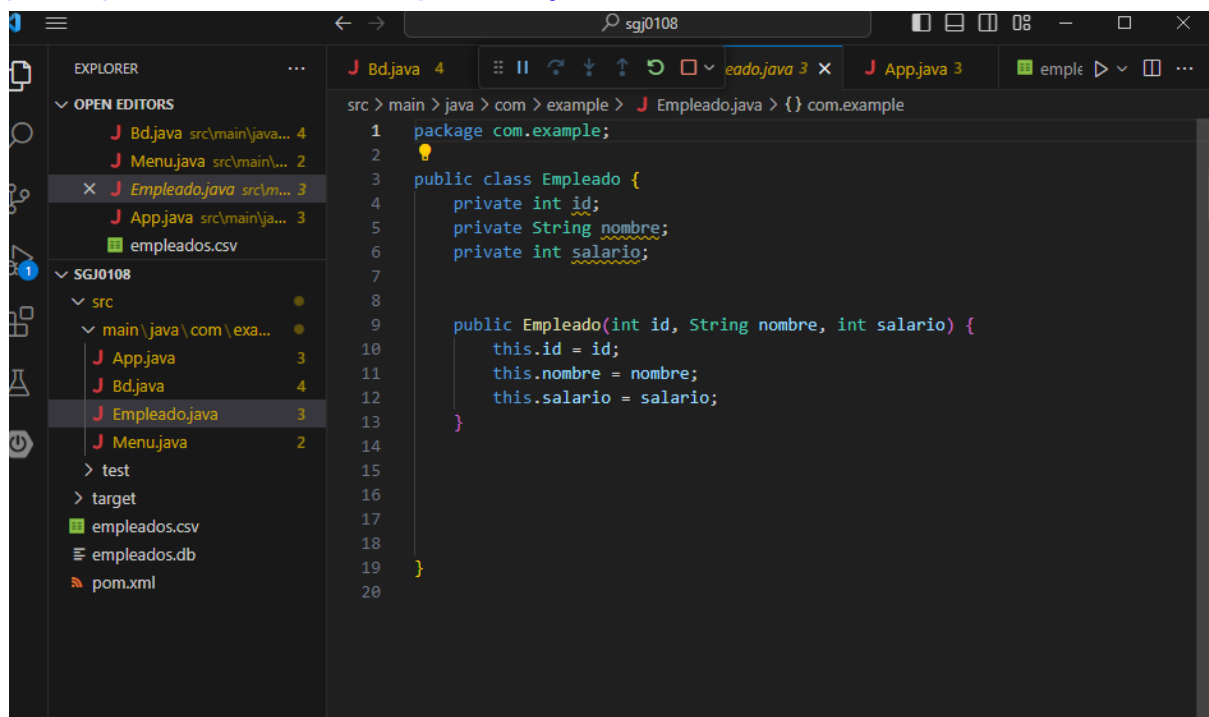
Solución: En este ejercicio he tenido bastantes problemas y me he encontrado con bastantes dificultades pero al final he conseguido que funcione.

Primero he creado una clase Menu que va a ser la que muestre el menú principal y con el que el usuario interactúe con la consola.



```
11
12
13 public void showMenu() {
14     Scanner teclado = new Scanner(System.in);
15     int respuesta;
16     System.out.println(x: "Bienvenido.");
17     Bd.cargaEmpleados();
18     do {
19
20
21         System.out.println(x: "=====");
22         Bd.enseñaEmpleados();
23         System.out.print(x: "Seleccione una opción: ");
24         String java.util.Scanner.nextLine()
25         Advances this scanner past the current line and returns the input that was
26         skipped. This method returns the rest of the current line, excluding any line
27         separator at the end. The position is set to the beginning of the next line.
28         Since this method continues to search through the input looking for a line
29         separator, it may buffer all of the input searching for the line to skip if no line
30         separators are present.
31         • Returns:
32             ◦ the line that was skipped
33         • Throws:
34             ◦ NoSuchElementException - if no line was found
35             ◦ IllegalStateException - if this scanner is closed
36         nombre = teclado.nextLine();
37         System.out.println(x: "Introduzca el salario del nuevo empleado");
38         salario = Integer.parseInt(teclado.nextLine());
39
40         Bd.altaEmpleado(id, nombre, salario);
41
42         break;
43     case 2:
44         System.out.println(x: "Introduzca el id del empleado que quiera dar de baja");
45         id = Integer.parseInt(teclado.nextLine());
46         Bd.bajaEmpleado(id);
47
48         break;
49     case 3:
50
51         System.out.println(x: "Introduzca el id del empleado que quiera modificar");
52         id = Integer.parseInt(teclado.nextLine());
53         System.out.println(x: "Introduzca el nuevo nombre");
54         nombre = teclado.nextLine();
55         System.out.println(x: "Introduzca el nuevo salario");
56         salario = Integer.parseInt(teclado.nextLine());
57
58         Bd.modificarEmpleado(id, nombre, salario);
59     }
```

Después una clase Empleado que va contener los atributos principales de cada empleado y su constructor.

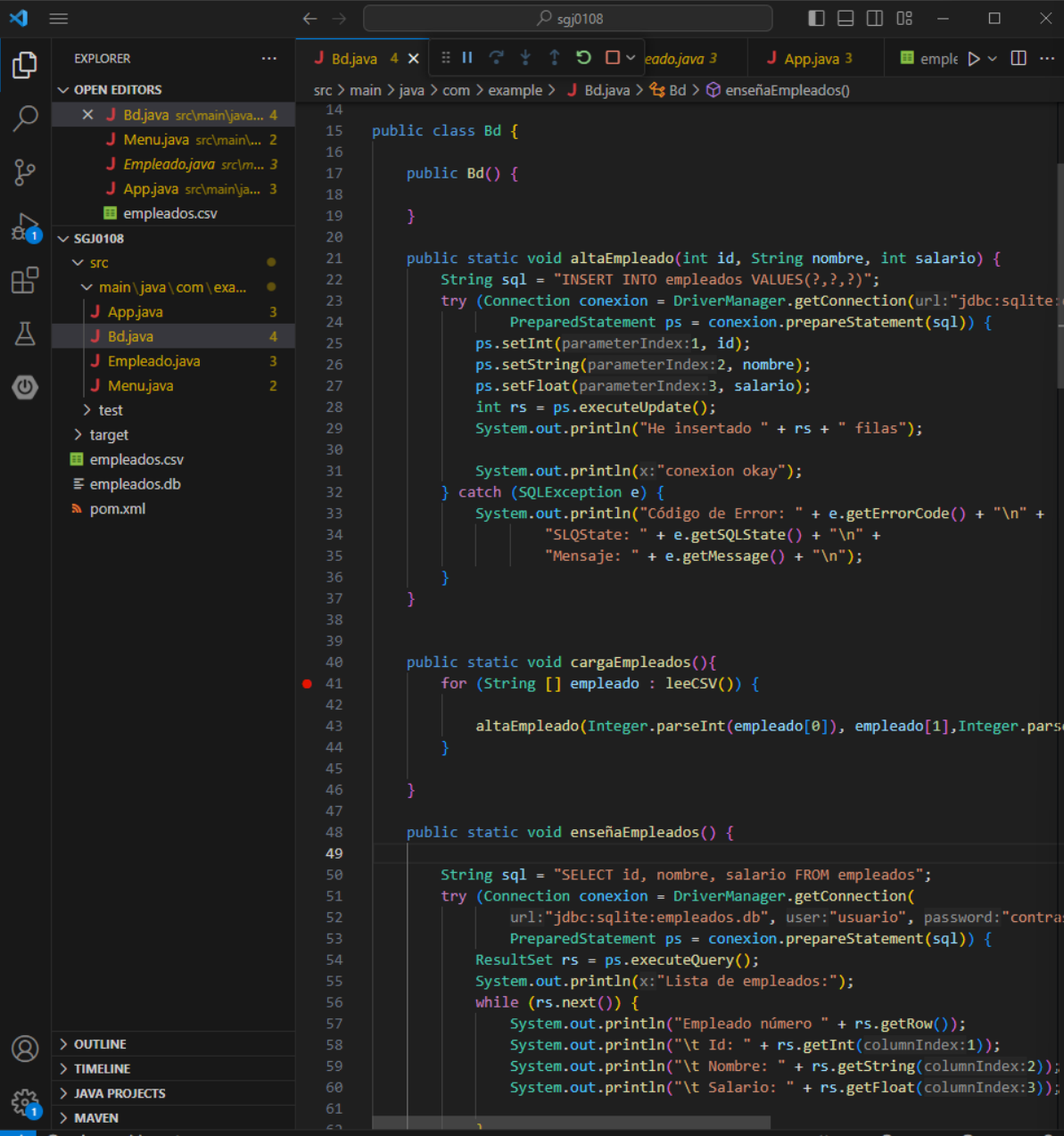


The screenshot shows an IDE with the following components:

- EXPLORER:** Displays the project structure. The 'src' folder is expanded, showing 'main' > 'java' > 'com' > 'example'. The 'Empleado.java' file is selected.
- Editor:** Shows the code for 'Empleado.java'. The code is as follows:

```
1 package com.example;
2
3 public class Empleado {
4     private int id;
5     private String nombre;
6     private int salario;
7
8
9     public Empleado(int id, String nombre, int salario) {
10         this.id = id;
11         this.nombre = nombre;
12         this.salario = salario;
13     }
14
15
16
17
18
19 }
20
```

Finalmente he creado la clase estática Bd para poder usarla sin ser instanciada que tiene todos los métodos necesarios para leer el csv, insertar usuarios, borrarlos, hacer consultas etc...



```
14 public class Bd {
15
16     public Bd() {
17
18     }
19
20
21     public static void altaEmpleado(int id, String nombre, int salario) {
22         String sql = "INSERT INTO empleados VALUES(?, ?, ?)";
23         try (Connection conexion = DriverManager.getConnection(url: "jdbc:sqlite:empleados.db", user: "usuario", password: "contrasena");
24             PreparedStatement ps = conexion.prepareStatement(sql)) {
25             ps.setInt(parameterIndex: 1, id);
26             ps.setString(parameterIndex: 2, nombre);
27             ps.setFloat(parameterIndex: 3, salario);
28             int rs = ps.executeUpdate();
29             System.out.println("Se insertado " + rs + " filas");
30
31             System.out.println(x: "conexion okay");
32         } catch (SQLException e) {
33             System.out.println("Código de Error: " + e.getErrorCode() + "\n" +
34                 "SQLState: " + e.getSQLState() + "\n" +
35                 "Mensaje: " + e.getMessage() + "\n");
36         }
37     }
38
39
40     public static void cargaEmpleados(){
41         for (String [] empleado : leeCSV()) {
42
43             altaEmpleado(Integer.parseInt(empleado[0]), empleado[1], Integer.parseInt(empleado[2]));
44         }
45     }
46
47
48     public static void enseñaEmpleados() {
49
50         String sql = "SELECT id, nombre, salario FROM empleados";
51         try (Connection conexion = DriverManager.getConnection(
52             url: "jdbc:sqlite:empleados.db", user: "usuario", password: "contrasena");
53             PreparedStatement ps = conexion.prepareStatement(sql)) {
54             ResultSet rs = ps.executeQuery();
55             System.out.println(x: "Lista de empleados:");
56             while (rs.next()) {
57                 System.out.println("Empleado número " + rs.getRow());
58                 System.out.println("\t Id: " + rs.getInt(columnIndex: 1));
59                 System.out.println("\t Nombre: " + rs.getString(columnIndex: 2));
60                 System.out.println("\t Salario: " + rs.getFloat(columnIndex: 3));
61             }
62         }
```

con la función cargaEmpleados consulto el archivo csv y los inserta en la tabla para que tenga valores iniciales en ella.

```
PROBLEMS 14 OUTPUT TERMINAL PORTS DEBUG CONSOLE

Lista de empleados:
Empleado número 1
    Id: 0
    Nombre: pepe perez
    Salario: 1000.0
Empleado número 2
    Id: 1
    Nombre: ana lopez
    Salario: 2000.0
Empleado número 3
    Id: 2
    Nombre: jose garcia
```

como podemos ver lo hace correctamente.
Damos de alta un usuario nuevo.

```
=====
[1]Alta [2]Baja [3]Modificación [4]Consulta [0]Salir
1
Introduzca el id del nuevo empleado
3
Introduzca el nombre del nuevo empleado
gonzalo garcia
Introduzca el salario del nuevo empleado
1800
He insertado 1 filas
```

```

    Id: 2
    Nombre: jose garcia
    Salario: 1500.0
Empleado número 4
    Id: 3
    Nombre: gonzalo garcia
    Salario: 1800.0
conexion okay
=====
[1]Alta [2]Baja [3]Modificación [4]Consulta [0]Salir
█
```

Ahora le damos de baja

```
=====
[1]Alta [2]Baja [3]Modificación [4]Consulta [0]Salir
2
Introduzca el id del empleado que quiera dar de baja
3
Se ha borrado correctamente
=====
```

```
=====
Lista de empleados:
Empleado número 1
    Id: 0
    Nombre: pepe perez
    Salario: 1000.0
Empleado número 2
    Id: 1
    Nombre: ana lopez
    Salario: 2000.0
Empleado número 3
    Id: 2
    Nombre: jose garcia
    Salario: 1500.0
conexion okay
=====
[1]Alta [2]Baja [3]Modificación [4]Consulta [0]Salir
```

Hacemos una modificación.

```
=====
[1]Alta [2]Baja [3]Modificación [4]Consulta [0]Salir
3
Introduzca el id del empleado que quiera modificar
2
Introduzca el nuevo nombre
juancho perez
Introduzca el nuevo salario
1200
conexion okay
=====
Lista de empleados:
Empleado número 1
    Id: 0
    Nombre: pepe perez
    Salario: 1000.0
Empleado número 2
    Id: 1
    Nombre: ana lopez
    Salario: 2000.0
Empleado número 3
    Id: 2
    Nombre: juancho perez
    Salario: 1200.0
```


Una consulta.

```
Conexion Okay
=====
[1]Alta [2]Baja [3]Modificación [4]Consulta [0]Salir
4
Introduzca el salario minimo
1500
Introduzca el salario maximo
7000
Lista de empleados:
Empleado número 1
    Id: 1
    Nombre: ana lopez
    Salario: 2000.0
```

- Estudia qué es el patrón de diseño “Repository” y comprueba si en este ejercicio se cumple.

El patrón de diseño Repository separa la lógica de acceso a datos de la lógica de negocio en una aplicación. Proporciona una interfaz común para acceder a los datos y oculta los detalles de cómo se almacenan y recuperan. Esto mejora la modularidad, el mantenimiento y la prueba de la aplicación.

La lógica principal de negocio se encuentra en la clase bd, pero también la uso para imprimir algunas cosas por pantalla asique estaría mezclada y no se cumpliría.

- ¿Qué ventajas aporta el uso de una clase BdManagerImpl y una interfaz BdManager?

El uso de una clase BdManagerImpl y una interfaz BdManager proporciona flexibilidad y separación de responsabilidades en una aplicación al separar la lógica de acceso a datos de la lógica de negocio.