# Entrega ejercicios 1.3, 1.6, 1.7, 1.4, 1.5

NDGt01e03:	2
NDGt01e04:	4
NDGt01e05:	6
NDGt01e06:	8
NDGt01e07:	9

# NDGt01e03:

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.

Lo que mas me costo de este ejercicio fue recordar los hash que tenían las entidades e java, ya que intentaba devolver objeto de Persona con el toString y me daban solo numeros con letras aleatorias, ya que yo ponía literalmente la entidad, ejemplo yo tenia a la persona mayor indicada de esta manera:

Persona personaMayor = Persona.obtenerPersonaMayor(personas);

con su método que busca a la persona mayor.

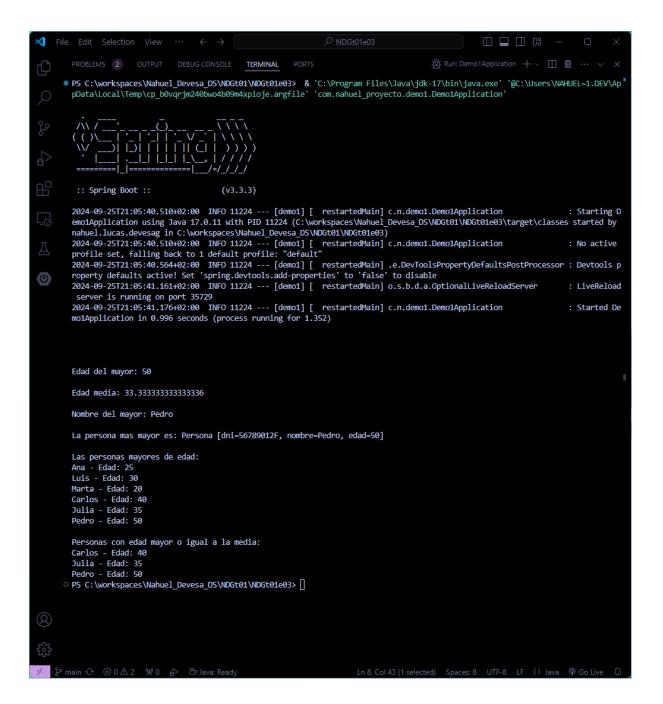
Y en el println vo ponía la entidad tal cual es decir

System.out.println(personaMayor)

claro esto me daba el hash.

Ese fue el mayor problema que tuve.

Algunos problemas secundarios fueron los funcionamientos de los métodos me costo mucho saber como empezar o qué comprobaciones eran esenciales.



# NDGt01e04:

1.4. (Opcional) 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 la habitación como libre y calcular el importe a pagar

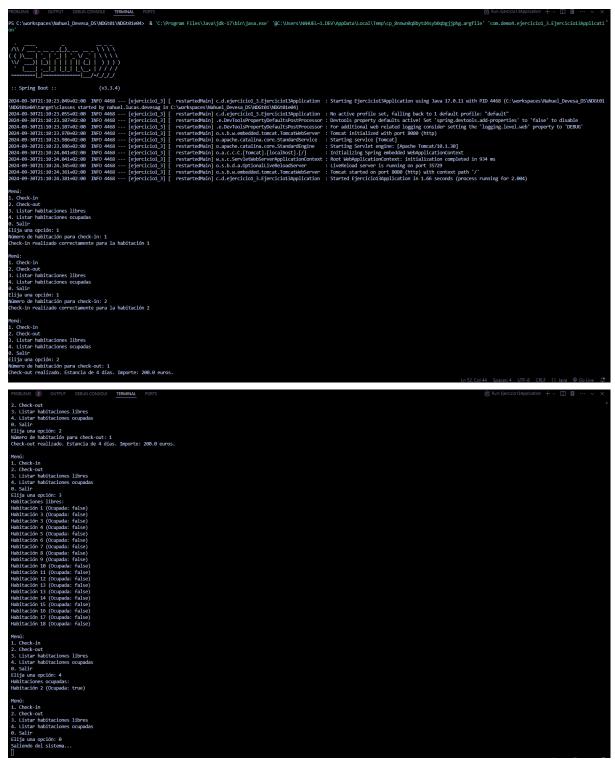
que se calculará en función del tipo de habitación y de los días de estancia (quizás sea necesario almacenar la fecha de llegada en el momento del check-in)

• 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.

Cuestión 1: ¿Habitación debería ser una clase abstracta o una interfaz? ¿Por qué? Clase abstracta ya que todas las habitaciones van a tener algunas variables iguales y métodos iguales..

Cuestión 2: ¿Es útil que el método toString () en la clase Habitación? Sí, es muy útil implementar el método toString() en la clase Habitación. Este método puede proporcionar una representación en forma de cadena de un objeto Habitación, lo que facilitaría la visualización de la información de las habitaciones.



Lo más difícil fue implementar los métodos en el case y no me petaran el programa.

#### NDGt01e05:

1.5. (Opcional) 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.

En este ejercicio no tuve ninguna complicación la verdad.



# NDGt01e06:

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.

En este ejercicio lo que más me dio la lata fue poner los números randoms y comprar los arrays me llevo mas de lo que pensaba lo de los arrays.

```
### SC. Luwrkspaces Valled | Devesta Deviation | Devesta Devesta Deviation | Devesta Devesta Deviation | Devesta Devesta Deviation | Devesta Deviation | Devesta Deviation | Devesta Deves
```

# NDGt01e07:

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. Ejemplo:

```
Introduce una cadena:
zbcabcdddd
{a=1, b=2, c=2, d=4, z=1}
```

```
PROCEASE OUTFOR DEBACOMONI IBMANA DOTS

Schoolspaces/Webball Devesa p5VB0081VB0081407 & "CiVorgram Files\laws\jdc-17\bin\java.cmc" @CiVbers\W48Ei-i.BEV\ppOptra\local\temptrp_B6labfarbespher980sopemox.argfile" 'com.demol.ejerciciol_7.

**Spring Boot :: (0.3.3.4)

2024-09-307213831.2248-289 BYO 5000 -- [ejerciciol_7] [ restarts@Main of .d.ejerciciol_7.Ejerciciol7/pplication : Starting Ejerciciol7/pplication using Java 17.0.11 with PID 5000 (C:Vborkspaces\Value been approximately by maheal.accs.devesag in C:Vborkspaces\Value been approximately been approximately by maheal.accs.devesag in C:Vborkspaces\Value been approximately by maheal.accs.devesag in C:Vborkspaces\Value been approximately been approximately by maheal.accs.devesag in C:Vborkspaces\Value been approximately been approxi
```

En este ejercicio lo que más me conto fue recorrer el String una única vez la verdad, conseguí sacarlo facil recorriendolo muchas veces pero no solo una, este paso me hizo dar algunas vueltas de más.

El tiempo estimado era de unas 8 horas o 9 horas, más o menos. Al final tarde un poquito menos tardando 7 horas y algo.