Ejercicios de la UD05

1. Ejercicios

1.1. Paquete: `UD05._1.gestionEmpleados`

Una empresa quiere hacer una gestión informatizada básica de sus empleados. Para ello, de cada empleado le interesa:

- Nombre (String)
- DNI (String)
- Año de ingreso (número entero)
- Sueldo bruto anual (número real)

2. Diseñar una clase Java `Empleado`, que contenga los atributos (privados) que caracterizan a un empleado e implemente los métodos adecuados para:
    - Crear objetos de la clase: **Constructor** que reciba todos los datos del empleado a crear.
    - Consultar el valor de cada uno de sus atributos. (**Consultores** o **getters**)
    - `public int antiguedad()`. Devuelve el número de años transcurridos desde el ingreso del empleado en la empresa. Si el año de ingreso fuera posterior al de la fecha actual, devolverá 0. Para obtener el año actual puedes usar:
        - ```
          1 int añoActual = Calendar.getInstance().get(Calendar.YEAR);
          ```
    - `public void incrementarSueldo(double porcentaje)`. Incrementa el sueldo del empleado en un porcentaje dado (expresado como una cantidad real entre 0 y 100).
    - `public String toString()`. Devuelve un `String` con los datos del empleado, de la siguiente forma:

      ```
      1 Nombre: Juan González
      2 Dni: 545646556K
      3 Año de ingreso: 1998
      4 Sueldo bruto anual: 20000 €
      ```

    - `public boolean equals(Object o)`. Método para comprobar si dos empleados son iguales. Dos empleados se consideran iguales si tienen el mismo DNI.
    - `public int compareTo(Object o)`. Se considera menor (mayor) el empleado que tiene menor (mayor) DNI.
    - Método estático `public static double calcularIRPF(double salario)`. Determina el % de IRPF que corresponde a un salario (mensual) determinado, según la siguiente tabla:
      ```

Desde salario (incluido)	Hasta salario (no incluido)	% IRPF
0	800	3
800	1000	10
1000	1500	15
1500	2100	20
2100	infinito	30

3. Diseñar una clase Java `TestEmpleado` que permita probar la clase `Empleado` y sus métodos. Para ello se desarrollará el método `main` en el que:
    - Se crearán dos empleados utilizando los datos que introduzca el usuario.
    - Se incrementará el sueldo un 20 % al empleado que menos cobre.
    - Se incrementará el sueldo un 10% al empleando más antiguo.
    - Muestra el IRPF que correspondería a cada empleado.
    - Para comprobar que las operaciones se realizan correctamente, muestra los datos de los empleados tras cada operación.
4. Diseñar una clase `Empresa`, que permita almacenar el nombre de la empresa y la información de los empleados de la misma (máximo 10 empleados) en un array. Para ello, se utilizarán tres atributos: nombre, plantilla (array de empleados) y `numEmpleados` (número de empleados que tiene la empresa) En esta clase, se deben implementar los métodos:
    - `public Empresa (String nombre)`. Constructor de la clase. Crea la empresa con el nombre indicado y sin empleados.
    - `public void contratar(Empleado e) throws PlantillaCompletaException`. Añade el empleado indicado a la plantilla de la empresa, siempre que quepa en el array. Si no cabe, se lanzará la excepción `PlantillaCompletaException`.
    - `public void despedir(Empleado e) throws ElementoNoEncontradoException`. Elimina el empleado indicado de la plantilla. Si no existe en la empresa, se lanza `ElementoNoEncontradoException`.
    - `public void subirTrienio (double porcentaje)` Subir el sueldo, en el porcentaje indicado, a todos los empleados cuya antigüedad sea exactamente tres años.
    - `public String toString()`. Devuelve un `String` con el nombre de la empresa y la información de todos los empleados. La información de los distintos empleados debe estar separada por saltos de línea.
5. Diseñar una clase Java `TestEmpresa` que permita probar la clase `Empresa` y sus métodos. Para ello, desarrolla el método `main` y en él ...:
    - Crea una empresa, de nombre "CataDaw".
    - Contrata a varios empleados (con el nombre, DNI, etc. que quieras).
    - Usa el método `subirTrienio` para subir un 10% el salario de los empleados que cumplen un trienio en el año actual.
    - Despide a alguno de los empleados.
    - Trata de despedir a algún empleado que no exista en la empresa.
    - Muestra los datos de la empresa siempre que sea necesario para comprobar que las operaciones se realizan de forma correcta.

1.2. Paquete: `UD05._2.gestionHospital`

Se desea realizar una aplicación para gestionar el ingreso y el alta de pacientes de un hospital. Una de las clases que participará en la aplicación será la clase `Paciente`, que se detalla a continuación :

1. La clase `Paciente` permite representar un paciente mediante los atributos: `nombre` (cadena), `edad` (entero), estado (entero entre 1 -más grave- y 5 -menos grave-, 6 si está curado), y con las siguientes operaciones:
    - `public Paciente (String n, int e)`. Constructor de un objeto `Paciente` de nombre `n`, de `e` años y cuyo estado es un valor aleatorio entre 1 y 5.
    - `public int getEdad()`. Consultor que devuelve edad.
    - `public int getEstado()`. Consultor que devuelve estado.
    - `public void mejorar()`. Modificador que incrementa en uno el estado del paciente (mejora al paciente)
    - `public void empeorar()`. Modificador que decrementa en uno el estado del paciente (empeora al paciente)
    - `public String toString()`. Transforma el paciente en un `String`. Por ejemplo,

      ```
      Pepe Pérez 46 5
      ```

    - `public int compareTo(Paciente o)`. Permite comparar dos pacientes. Se considera menor el paciente más leve. A igual gravedad, se considera menor el paciente más joven. Ejemplo:
        - Teniendo a `David 40 3`, `Pepe 25 3` y `Juan 35 5`:

          ```
          David.compareTo(Juan) = 2
          Pepe.compareTo(Juan) = -2
          David.compateTo(Pepe) = -15
          ```

2. Diseñar una clase Java `TestPaciente` que permita probar la clase `Paciente` y sus métodos. Para ello se desarrollará el método `main` en el que:
    - Se crearán dos pacientes: "Antonio" de 20 años y "Miguel" de 30 años.
    - Imprimir el estado inicial de los dos pacientes.
    - Mostrar los datos del que se considere menor (según el criterio de `compareTo` de la clase `Paciente`).
    - Aplicar "mejoras" al paciente más grave hasta que los dos pacientes tengan el mismo estado.
    - Imprimir el estado final de los dos pacientes.
3. La clase **Hospital** contiene la información de las camas de un hospital, así como de los pacientes que las ocupan. Un Hospital tiene un número máximo de camas `MAXC` = 200 y para representarlas se utilizará un array (llamado `listaCamas`) de objetos de tipo Paciente junto con un atributo (`numLibres`) que indique el número de camas libres del hospital en un momento dado. El número de cada cama coincide con su posición en el array de pacientes (la posición 0 no se utiliza), de manera que `listaCamas[i]` es el Paciente que ocupa la cama `i` o es `null` si la cama está libre. Las operaciones de esta clase son:
    - `public Hospital()`. Constructor de un hospital. Cuando se crea un hospital, todas las camas están libres.
    - `public int getNumLibres()`. Consultor del número de camas libres.

- `public boolean hayLibres()`. Devuelve true si en el hospital hay camas libres y devuelve false en caso contrario.
- `public int primeraLibre()`. Devuelve el número de la primera cama libre del array `listaCamas` si hay camas libres o devuelve un 0 si no las hay.
- `public void ingresarPaciente(String n, int e) throws HospitalLlenoException` Si hay camas libres, la primera de ellas (la de número menor) pasa a estar ocupada por el paciente de nombre `n` y edad `e`. Si no hay camas libres, lanza una excepción.
- `private void darAltaPaciente(int i)`. La cama i del hospital pasa a estar libre. (Afectará al número de camas libres)
- `public void darAltas()`. Se mejora el estado (método `mejorar()` de `Paciente`) de cada uno de los pacientes del hospital y a aquellos pacientes sanos (cuyo estado es 6) se les da el alta médica (invocando al método `darAltaPaciente`).
- `public String toString()`. Devuelve un `String` con la información de las camas del hospital. Por ejemplo,

```
1 María Medina 30 4
2 Pepe Pérez 46 5
3 libre
4 Juan López 50 1
5 libre
...
200 Andrés Sánchez 29 3
```

4. En la clase `GestorHospital` se probará el comportamiento de las clases anteriores. El programa deberá:
    - Crear un hospital.
    - Ingresar a cinco pacientes con los datos simulados introducidos directamente en el programa.
    - Realizar el proceso de `darAltas` mientras que el número de habitaciones libres del hospital no llegue a una cantidad (por ejemplo 198).
    - Mostrar los datos del hospital cuando se considere oportuno para comprobar la corrección de las operaciones que se hacen.

1.3. Paquete: `UD05.3.contrarreloj`

Se quiere realizar una aplicación para registrar las posiciones y tiempos de llegada en una carrera ciclista contrarreloj.

1. La clase `Corredor` representa a un participante en la carrera. Sus atributos son el dorsal (entero), el nombre (string) y el tiempo en segundos (double) que le ha costado completar el recorrido. Los métodos con los que cuenta son:
    - `public Corredor(int d, String n)`. Constructor a partir del dorsal y el nombre. Por defecto el tiempo tardado es 0
    - `public double getTiempo()`. Devuelve el tiempo tardado por el corredor
    - `public String getNombre()`. Devuelve el nombre del corredor
    - `public void setTiempo( double t)throws IllegalArgumentException`. Establece el tiempo tardado por el corredor. Lanzará la excepción si el tiempo indicado es negativo.

- `public void setTiempo( Tiempo t1, Tiempo t2) throws IllegalArgumentException`. Establece el tiempo tardado por el corredor.

  Los segundos tardados se calculan a partir de dos objetos de tipo Tiempo. Para ello, añadir a la clase `PeriodoDeTiempo` un método `estático` que calcule la diferencia en segundos entre dos objetos de tipo Tiempo dados: `int diferencia(Tiempo t1, Tiempo t2)`. Utilizar dicho método para calcular el tiempo tardado.

  Lanzará la excepción si el tiempo indicado es negativo

- `public String toString()`. Devuelve un String con los datos del corredor, de la forma:

  ```
  (234) - Juan Ramirez - 2597 segundos
  ```

- `public boolean equals(Object o)`. Devuelve true si los corredores tienen el mismo dorsal y false en caso contrario
- `public int compareTo (Object o)`. Un corredor es menor que otro si tiene menor dorsal.
- `public static int generarDorsal()`. Devuelve un número de dorsal generado secuencialmente. Para ello la clase hará uso de un atributo `static int siguienteDorsal` que incrementará cada vez que se genere un nuevo dorsal.

2. Diseñar una clase Java `TestCorredor` que permita probar la clase Corredor y sus métodos. Para ello se desarrollará el método `main` en el que:
   - Se crearán dos corredores: El nombre lo indicará el usuario mientras que el dorsal se generará utilizando el método `generarDorsal()` de la clase.
   - Se establecerá el tiempo de llegada del primer corredor a 300 segundos y el del segundo a 400.
   - Se mostrarán los datos de ambos corredores (`toString`)
3. La clase `ListaCorredores` permite representar a un conjunto de corredores. En la lista, como máximo habrá 200 corredores, aunque puede haber menos de ese número. Se utilizará un array, llamado lista, de 200 elementos junto con una propiedad `numCorredores` que permita saber cuentos corredores hay realmente. Métodos:
   - `public ListaCorredores ()`. **Construtor**. Crea la lista de corredores, incicialmente vacía.
   - `public void añadir(Corredor c) throws ElementoDuplicadoException`. Añade un corredor al final de la lista de corredores, siempre y cuando el corredor no esté ya en la lista, en cuyo caso se lanzará `ElementoDuplicadoException`
   - `public void insertarOrdenado(Corredor c)`. Inserta un corredor en la posición adecuada de la lista de manera que esta se mantenga ordenada crecientemente por el tiempo de llegada. Para poder realizar la inserción debe averiguarse la posición que debe ocupar el nuevo elemento y, antes de añadirlo al array, desplazar el elemento que ocupa esa posición y todos los posteriores, una posición a la derecha.
   - `public Corredor quitar(int dorsal) throws ElementoNoEncontradoException`. Quita de la lista al corredor cuyo dorsal se indica. El array debe mantenerse compacto, es decir, todos los elementos posteriores al eliminado deben desplazarse una posición a la izquierda. El método devuelve el Corredor quitado de la lista. Si no se encuentra se lanza `ElementoNoEncontradoException`.

`public String toString()` Devuelve un `String` con la información de la lista de corredores. Por ejemplo:

```
Posición: 1
Dorsal: 234
Nombre: Juan Ramirez
Tiempo: 25.97 minutos

Posición: 2
Dorsal: 26
Nombre: José González
Tiempo: 29.7 minutos
```

4. (Clase `ContrarReloj`) Realizar un programa que simule una contrarreloj. Para llevar el control de una carrera contrarreloj se mantienen dos listas de corredores (dos objetos de tipo `ListaCorredores`):
    - (`hanSalido`) Una con los que han salido, que tiene a los corredores por orden de salida. El atributo tiempo de estos corredores será 0. Para que los corredores se mantengan por orden de salida, se añadiran a la lista utilizando el método añadir.
    - (`hanLlegado`) Otra con los corredores que hay llegado a la meta. A medida que los corredores llegan a la meta se les extrae de la primera lista, se les asigna un tiempo y se les inserta ordenadamente en esta segunda lista.

    En el método `main` realizar un programa que muestre un menú con las siguientes opciones:

    1. `Salida`: Para registrar que una corredor ha comenzado la contrarreloj y sale de la línea de salida. Solicita al usuario el nombre de un corredor y su dorsal, y lo añade a la lista de corredores que han salido.
    2. `Llegada`: Para registrar que un corredor ha llegado a la meta. Solicita al usuario el dorsal de un corredor y el tiempo de llegada. Quita al corredor de la lista de corredores que `hanSalido`, le asigna el tiempo que ha tardado y lo inserta (ordenadamente) en la lista de corredores que `hanLlegado`
    3. `Clasificación`: Muestra la lista de corredores que `hanLlegado`. Dado que esta lista está ordenada por tiempo, mostrarla por pantalla nos da la clasificación.
    4. `Salir`: Sale del programa

1.4. Paquete: `UD05._4.reservasLibreria`

Una librería quiere proporcionar a sus clientes el siguiente servicio:

Cuando un cliente pide un libro y la librería no lo tiene, el cliente puede hacer una reserva de manera que cuando lo reciban en la librería le avisen por teléfono.

De cada reserva se almacena:

- `Nif` del cliente (`String`)
- `Nombre` del cliente (`String`)
- `Teléfono` del cliente (`String`)
- `Código` del libro reservado. (`Entero`)
- `Numero` de ejemplares (`entero`)

1. Diseñar la clase `Reserva`, de manera que contemple la información descrita e implementar:

    - `public Reserva(String nif,String nombre, Strint tel, int codigo, int ejemplares)`. Constructor que recibe todos los datos de la reserva.
    - `public Reserva(String nif,String nombre, Strint tel, int codigo)`. Constructor que recibe los datos del cliente y el código del libro. Establece el número de ejemplares a uno.
    - Consultores de todos los atributos.
    - `public int setEjemplares(int ejemplares)`. Modificador del número de ejemplares. Establece el número de ejemplares al valor indicado como parámetro.
    - `public String toString()` que devuelva un `String` con los datos de la reserva
    - `public boolean equals(Object o)`. Dos reservas son iguales si son del mismo cliente y reservan el mismo libro.
    - `public int compareTo(Object o)`. Es menor la reserva cuyo libro es menor. A igual libro es menor aquella cuyo libro es menor.
2. Diseñar una clase Java `TestReservas` que permita probar la clase `Reserva` y sus métodos. Para ello se desarrollará el método `main` en el que:
    - Se creen dos reservas con los datos que introduce el usuario. Las reservas no pueden ser iguales (equals). Si la segunda reserva es igual a la primera se pedirá de nuevo los datos de la segunda al usuario.
    - Se incremente en uno el número de ejemplares de ambas reservas.
    - Se muestre la menor y a continuación la mayor.
3. Diseñar una clase `ListaReservas` que implemente una lista de reservas. Como máximo puede haber 100 reservas en la lista. Se utilizará un array de Reservas que ocuparemos a partir de la posición 0 y un atributo que indique el número de reservas. Las reservas existentes ocuparán las primeras posiciones del array. Implementar los siguientes métodos:
    - `public void reservar(String nif, String nombre, String telefono, int libro, int ejemplares) throws ListaLlenaException, ElementoDuplicadoException`: Crea una reserva y la añade a la lista. Lanza `ElementoDuplicadoException` si la reserva ya estaba en la lista. Lanza `ListaLlenaException` si la lista de reservas está llena.
    - `public void cancelar(String nif, int libro) throws ElementoNoEncontradoException`. Dado un nombre de cliente y un código de libro, anular la reserva correspondiente. Lanzar `ElementoNoEncontradoException` si la reserva no existe.
    - `public String toString()`: Devuelve un `String` con los datos de todas las reservas de la lista.
    - `public int numEjemplaresReservadosLibro(int codigo)`: Devuelve el número de ejemplares que hay reservados en total de un libro determinado.
    - `public void reservasLibro(int codigo)`: Dado un código de libro, muestra el nombre y el teléfono de todos los clientes que han reservado el libro.
4. Realizar un programa `GestionReservas` que, utilizando un menú, permita:
    - Realizar reserva. Permite al usuario realizar una reserva.
    - Anular reserva: Se anula la reserva que indique el usuario (Nif de cliente y código de libro).
    - Pedido: El usuario introduce un código de libro y el programa muestra el nº de reservas que se han hecho del libro. Esta opción de menú le resultará útil al usuario para poder hacer el pedido de un libro determinado.
    - Recepción: Cuando el usuario recibe un libro quiere llamar por teléfono a los clientes que lo reservaron. Solicitar al usuario un código de libro y mostrar los

datos (nombre y teléfono) de los clientes que lo tienen reservado.

1.5. Paquete: `UD05._5.gestorCorreoElectronico`

Queremos realizar la parte de un programa de correo electrónico que gestiona la organización de los mensajes en distintas carpetas. Para ello desarrollaremos:

1. La clase `Mensaje`. De un mensaje conocemos:
    - `Codigo (int)` Número que permite identificar a los mensajes.
    - `Emisor (String)`: email del emisor.
    - `Destinatario (String)`: email del destinatario.
    - `Asunto (String)`
    - `Texto (String)`

    Desarrollar los siguientes métodos:

    - Constructor que reciba todos los datos, excepto el código, que se generará automáticamente (nº consecutivo. Ayuda: utiliza una variable de clase (`static`))
    - Consultores de todos los atributos.
    - `public boolean equals(Object o)`. Dos mensajes son iguales si tienen el mismo código.
    - `public static boolean validarEMail(String email)`: Método estático que devuelve true o false indicando si la dirección de correo indicada es válida o no. Una dirección es válida si tiene la forma `direccion@subdominio.dominio`
    - `public String toString()`
2. Con la clase `TestCorreo` probaremos las clases y métodos desarrollados.
    - Crea varios mensajes con los datos que introduzca el usuario y muéstralos por pantalla.
    - Prueba el método `validarEMail` de la clase Mensaje con las direcciones siguientes (solo la primera es correcta):
        - `tuCorreo@gmail.com`
        - `tuCorreogmail.com`
        - `tuCorreo@gmail`
        - `tuCorreo.com@gmail`
3. La clase `Carpeta`, cada carpeta tiene un nombre y una lista de Mensajes. Para ello usaremos un array con capacidad para 100 mensajes y un atributo que indique el número de mensajes que contiene la carpeta. Además se implementarán los siguientes métodos:
    - `public Carpeta(String nombre)`: Constructor. Dado un nombre, crea la carpeta sin mensajes.
    - `public void añadir(Mensaje m)`: Añade a la carpeta el mensaje indicado.
    - `public void borrar(Mensaje m) throws ElementoNoEncotradoException`: Borra de la carpeta el mensaje indicado. Lanza la excepción si el mensaje no existe.
    - `public Mensaje buscar(int codigo) throws ElementoNoEncontradoException`: Busca el mensaje cuyo código se indica. Si lo encuentra devuelve el mensaje, en caso contrario lanza la excepción.
    - `public String toString()` que devuelva un `String` con el nombre de la carpeta y sus mensajes
    - `public static void moverMensaje(Carpeta origen, Carpeta destino, int codigo) throws ElementoNoEncontradoException`: Método estático. Recibe dos Carpetas de correo y un código de mensaje y mueve el mensaje indicado de

una carpeta a otra. Para ello buscará el mensaje en la carpeta origen. Si existe lo eliminará y lo añadirá a la carpeta de destino. Si el mensaje indicado no está en la carpeta de origen lanza `ElementoNoEncontradoException`.

4. Con la clase `TestCarpetas` probaremos las clases y métodos desarrollados:
    - Crea dos carpetas de correo de nombre Mensajes recibidos y Mensajes eliminados respectivamente.
    - Crea varios mensajes y añádelos Mensajes recibidos.
    - Mueve el mensaje de código 1 desde la Mensajes recibidos a Mensajes elimiminados.
    - Muestra el contenido de las carpetas antes y después de cada operación (añadir, mover,...)

1.6. Paquete: `UD05._6.juegoDeCartas`

Se está desarrollando una aplicación que usa una baraja de cartas. Para ello, se implementarán en Java las clases necesarias.

1. Una de ellas es la clase `Carta` que permite representar una carta de la baraja española. La información requerida para identificar una `Carta` es:
    - su palo (oros, copas, espadas o bastos) y
    - su valor (un entero entre 1 y 12).

    Para dicha clase, se pide:
    - Definir 4 constantes, atributos de clase (estáticos) públicos enteros, para representar cada uno de los palos de la baraja (`OROS` será el valor 0, `COPAS` el 1, `ESPADAS` el 2 y `BASTOS` el 3).
    - Definir los atributos (privados): palo y valor.
    - Escribir dos constructores: uno para construir una carta de forma aleatoria (sin parámetros) y otro para construir una carta de acuerdo a dos datos: su palo y su valor (si los datos son incorrectos se lanzará `IllegalArgumentException`)
    - Escribir dos métodos consultores y dos métodos modificadores de los valores de los atributos.
    - Escribir un método `compareTo` para comprobar si la carta actual es menor que otra carta dada. El criterio de ordenación es por palos (el menor es oros, después copas, a continuación espadas y, finalmente, bastos) y dentro de cada palo por valor (1, 2, ..., 12).
    - Escribir un método `equals` para comprobar la igualdad de dos cartas. Dos cartas son iguales si tienen el mismo palo y valor.
    - Escribir un método `sigPalo` para devolver una nueva carta con el mismo valor que el de la carta actual pero del palo siguiente, según la ordenación anterior y sabiendo que el siguiente al palo bastos es oros.
    - Escribir un método `toString` para transformar en `String` la carta actual, con el siguiente formato: "valor de palo"; por ejemplo, "4 de oros" o "1 de bastos" (sobrescritura del método `toString` de `Object`).
2. Implementar una clase `JuegoCartas` con los métodos siguientes:
    - Un método de clase (estático) `public static int ganadora( Carta c1, Carta c2)` que dados dos objetos `Carta` y un número entero representando el palo de triunfo (o palo ganador), determine cuál es la carta ganadora. El método debe devolver 0 si las dos cartas son iguales. En caso contrario, devolverá -1 cuando la primera carta es la ganadora y 1 si la segunda carta es la ganadora.

      Para determinar la carta ganadora se aplicarán las siguientes reglas:

        - Si las dos cartas son del mismo palo, la carta ganadora es el as (valor 1) y, en el resto de casos, la carta ganadora es la de valor más alto (por ejemplo, "1 de oros" gana a "7 de oros", "5 de copas" gana a "2 de copas", "11 de bastos" gana a "7 de bastos").
        - Si las dos cartas son de palos diferentes:
            - Si el palo de alguna carta es el palo de triunfo, dicha carta es la ganadora.
            - En otro caso, la primera carta siempre gana a la segunda.
    - Un método `main` en el que se debe:
        - Crear una `Carta` a partir de un palo y un valor dados (solicitados al usuario desde teclado), y mostrar sus datos por pantalla.
        - Crear una `Carta` aleatoriamente y mostrar sus datos por pantalla.
        - Generar aleatoriamente un entero en el rango [0..3] representando el palo de triunfo, y mostrar por pantalla a qué palo corresponde.
        - Mostrar por pantalla la carta ganadora (invocando al método del apartado anterior con el objeto `Carta` del usuario.

1.7. Paquete: `UD05._7.gestorVuelos`

Se desea realizar una aplicación `GestorVuelos` para gestionar la reserva y cancelación de vuelos en una agencia de viajes. Dicha agencia trabaja únicamente con la compañía aérea Iberia, que ofrece vuelos desde/hacia varias ciudades de Europa. Se deben definir las clases que siguen, teniendo en cuenta que sus atributos serán privados y sus métodos sólo los que se indican en cada clase.

1. Implementación de la clase **Vuelo**, que permite representar un vuelo mediante los atributos:
    - `identificador` (`String`)
    - `origen` (`String`)
    - `destino` (`String`)
    - `hSalida` (`Time`)
    - `hLlegada` (`Time`)
    - Además, cada vuelo dispone de 50 asientos, es decir, pueden viajar, como mucho, 50 pasajeros en cada vuelo. Para representarlos, se hará uso de `asiento`, un array de `String` (nombres de los pasajeros) junto con un atributo `numP` que indique el número actual de asientos reservados. Si el asiento `i` está reservado, `asiento[i]` contendrá el nombre del pasajero que lo ha reservado. Si no lo está, `asiento[i]` será `null`. En el array `asiento`, las posiciones impares pertenecen a asientos de ventanilla y las posiciones pares, a asientos de pasillo (la posición 0 no se utilizará).

   En esta clase, se deben implementar los siguientes métodos:

    - `public Vuelo(String id, String orig, String dest, Time hsal, Time hlleg)`. **Constructor** que crea un vuelo con identificador, ciudad de origen, ciudad de destino, hora de salida y hora de llegada indicados en los respectivos parámetros, y sin pasajeros.
    - `public String getIdenificador()`. Devuelve el `identificador`
    - `public String getOrigen()`. Devuelve `origen`.
    - `public String getDestino()`. Devuelve `destino`.

- `public boolean hayLibres()`. Devuelve `true` si quedan asientos libres y `false` si no quedan.
- `public boolean equals(Object o)`. Dos vuelos son iguales si tienen el mismo identificador.
- `public int reservarAsiento(String pas, char pref) throws VueloCompletoException`. Si el vuelo ya está completo se lanza una excepción. Si no está completo, se reserva al pasajero `pas` el primer asiento libre en `pref`. El carácter `pref` será '`P`' o '`V`' en función de que el pasajero desee un asiento de pasillo o de ventanilla. En caso de que no quede ningún asiento libre en la preferencia indicada (`pref`), se reservará el primer asiento libre de la otra preferencia. El método devolverá el número de asiento que se le ha reservado. Este método hace uso del método privado `asientoLibre`, que se explica a continuación.
- `private int asientoLibre(char pref)`. Dado un tipo de asiento `pref` (pasillo '`P`' o ventanilla '`V`'), devuelve el primer asiento libre (el de menor numero) que encuentre de ese tipo. O devuelve `0` si no quedan asientos libres de tipo `pref`.
- `public void cancelarReserva(int numasiento)`. Se cancela la reserva del asiento `numasiento`.
- `public String toString()`. Devuelve una `String` con los datos del vuelo y los nombres de los pasajeros, con el siguiente formato:

```
IB101 Valencia París 19:05:00 21:00:00
Pasajeros:
Asiento 1: Sonia Dominguez
…
Asiento 23: Fernando Romero
```

2. Diseñar e implementar una clase Java `TestVuelo` que permita probar la clase `Vuelo` y sus métodos. Para ello se desarrollará el método `main` en el que:
   - Se cree el vuelo IB101 de Valencia a París, que sale a las 19:05 y llega a las 21:00
   - Reservar:
     - Un asiento de ventanilla a "Miguel Fernández"
     - Un asiento de ventanilla a "Ana Folgado"
     - Un asiento de pasillo a "David Más"
   - Mostrar el vuelo por pantalla
   - Cancelar la reserva del asiento que indique el usuario.
3. Implementación de la clase `Compañía` para representar todos los vuelos de una compañía aérea. Una Compañía tiene un nombre nombre y puede ofrecer, como mucho, 10 vuelos distintos. Para representarlos se utilizará `listaVuelos`, un array de objetos `Vuelo` junto con un atributo `numVuelos` que indique el número de vuelos que la compañía ofrece en un momento dado. Las operaciones de esta clase son:
   - `public Compania(String n) throws FileNotFoundException`. Constructor de una compañía de nombre `n`. Cuando se crea una compañía, se invoca al método privado `leeVuelos()` para cargar la información de vuelos desde un fichero. Si el fichero no existe, se propaga la excepción `FileNotFoundException`

- `private void leeVuelos() throws FileNotFoundException`. Lee desde un fichero toda la información de los vuelos que ofrece la compañía y los va almacenando en el array de vuelos `listaVuelos`. El nombre del fichero coincide con el nombre de la compañía y tiene extensión `.txt`. La información de cada vuelo se estructura en el fichero como sigue:

```
<Identificador>
<Origen>
<Destino>
<Hora de salida>
<Minuto de salida>
<Hora de llegada>
<Minuto de llegada>
...
...
```

  Si el fichero no existe, se propaga la excepción `FileNotFoundException`.

- `public Vuelo buscarVuelo(String id) throws ElementoNoEncontradoException`. Dado un identificador de vuelo `id`, busca dicho vuelo en el array de vuelos `listaVuelos`. Si lo encuentra, lo devuelve. Si no, lanza `ElementoNoEncontradoException`.
- `public void mostrarVuelosIncompletos(String o, String d)`. Muestra por pantalla los vuelos con origen `o` y destino `d`, y que tengan asientos libres. Por ejemplo, vuelos con asientos libres de la compañía Iberia con origen Milán y destino Valencia:

```
Iberia IB201 Milán Valencia 14:25:00 16:20:00
Iberia IB202 Mílan Valencia 21:40:00 23:35:00
```

4. En la clase `GestorVuelos` se probará el comportamiento de las clases anteriores. En esta clase se debe implementar el método `main` en el que, por simplificar, se pide únicamente:
   - la creación de la compañía aérea `Iberia`. Se dispone de un fichero de texto "`Iberia.txt`", con la información de los vuelos que ofrece.
   - Reserva de un asiento de ventanilla en un vuelo de Valencia a París por parte de Manuel Soler Roca. Para ello:
     - Mostraremos vuelos con origen Valencia y destino París, que no estén completos.
     - Pediremos al usuario el identificador del vuelo en que quiere hacer la reserva.
     - Buscaremos el vuelo que tiene el identificador indicado. Si existe realizaremos la reserva y mostraremos un mensaje por pantalla. En caso contrario mostraremos un mensaje de error por pantalla.

1.8. Paquete: UD05._8.`maquinaExpendedora`

Se desea simular el funcionamiento de una máquina expendedora. Se trata de una expendedora sencilla que, por el momento, será capaz de dispensar únicamente un producto.

Su funcionamiento, a grandes rasgos, es el siguiente:

1. El cliente introduce dinero en la máquina. Al dinero introducido lo llamaremos `credito`.
2. Selecciona el artículo que quiere comprar (ya hemos comentado que por el momento habrá un solo artículo).
3. Si hay stock del artículo seleccionado, la máquina dispensa el artículo elegido y devuelve el importe sobrante (diferencia entre el crédito introducido y el precio del artículo).

Durante el proceso se pueden producir diversas incidencias, como por ejemplo, que el cliente no haya introducido suficiente crédito para comprar el producto, que no quede producto o que no haya cambio suficiente para la devolución. La máquina también da la posibilidad de solicitar la devolución del crédito sin realizar la compra.

1. Diseñar la clase `Expendedora` (proyecto `Expendedora`) con los atributos y métodos que se describen a continuación.
    - Atributos (privados)
        - `credito`: Cantidad de dinero (en euros) introducida por el cliente.
        - `stock`: Número de unidades que quedan en la máquina disponibles para la venta. Se reducirá con cada nueva venta.
        - `precio`: Precio del único artículo que dispensa la máquina (en euros).
        - `cambio`: Cambio del que dispone la máquina. El cambio disponible se reduce cada vez que se devuelve al cliente la diferencia entre el crédito introducido y el precio del producto comprado. El cambio nunca se ve incrementado por las compras de los clientes.
        - `recaudación`: Representa la suma de las ventas realizadas por la máquina (en euros). Se ve incrementada con cada nueva compra.
    - Métodos:
        - Constructor: `public Expendedora (double cambio, int stock, double precio)`. Crea la expendedora inicializando los atributos cambio, stock y precio con los valores indicados en los parámetros). El crédito y la recaudación serán cero.
        - Consultores:
            - Métodos consultores para los atributos crédito, cambio, y recaudación
            - Los consultores para el stock y el precio los haremos previendo que en el futuro la máquina pueda expender más de un tipo de producto. Para consultar el stock y el precio se indicará como parámetro el número de producto que se quiere consultar aunque, por el momento se ignorará el valor de dicho atributo.
            - `public getStock (int producto)` Devuelve el stock disponible del producto indicado. En esta versión simplificada se devolverá el valor del atributo stock, sea cual sea el valor de producto.
            - `public getPrecio (int producto)` Devuelve el precio del producto indicado. En esta versión simplificada se devolverá el valor del atributo precio, , sea cual sea el valor de producto.
        - Modificadores: Para simplificar, consideramos que los atributos de la máquina solo van a cambiar por operaciones derivadas de su funcionamiento, por lo que no proporcionamos modificadores públicos

- Otros métodos:
- `public String toString()` Devuelve un `String` de la forma:

```
Credito: 3.0 euros
Cambio: 12.73 euros
Stock: 12 unidades:
Recaudación: 127.87 euros
```

- `public void introducirDinero(double importe)` Representa la operación mediante la cual el cliente añade dinero (crédito) a la máquina. Esta operación incrementa el crédito introducido por el cliente en el importe indicado como parámetro.
- `public double solicitarDevolucion()` Representa la operación mediante la cual el cliente solicita la devolución del crédito introducido sin realizar la compra. El método devuelve la cantidad de dinero que se devuelve al cliente.
- `public double comprarProducto(int producto) throws NoHayCambioException, NoHayProductoException, CreditoInsuficienteException`. Representa la operación mediante la cual el cliente selecciona un producto para su compra. El método devuelve la cantidad de dinero que se devuelve al cliente.

  Si no se produce ninguna situación inesperada, se reduce el stock del producto, se devuelve el cambio, se pone el crédito a cero y se incrementa la recaudación.

  Si la venta no es posible se lanzará la excepción correspondiente a la situación que impide completar la venta.

2. La clase `Producto` permite representar uno de los artículos de los que vende una máquina expendedora. Para ello utilizaremos tres atributos privados `nombre` (`String`), `precio` (`double`) y `stock` (`int`), y los siguientes métodos:
   - `public Producto(String nombre, double precio, int stock)` Constructor que inicializa el producto con los parámetros indicados
   - Consultores de los tres atributos: `getNombre`, `getPrecio` y `getStock`
   - `public int decrementarStock()`: Decrementa en 1 el stock del producto y devuelve el stock resultante.
3. La clase `Surtido` representa una colección de productos. Para ello se usará un atributo `listaProductos`, array de `Productos`. El array se rellenará con los datos de productos extraidos de un fichero de texto y, una vez creado el surtido no será posible añadir o quitar productos. Así, el array de productos estará siempre completo y no es necesario ningún atributo que indique cuantos productos hay en el array.

   Se implementarán los siguientes métodos:

   - `public Surtido() throws FileNotFoundException` Crea el surtido con los datos de los productos que se encuentran en el fichero `productos.txt`. El fichero tiene el siguiente formato:

```
<nº de productos>
<nombre de producto> <precio> <stock>
<nombre de producto> <precio> <stock>
<nombre de producto> <precio> <stock>
...
```

Como vemos, la primera línea del fichero indica el número de productos que contiene el surtido. Este dato lo usaremos para dar al array de productos el tamaño adecuado.

- `public int numProductos()` Devuelve el número de productos que componen el surtido
- `public Producto getProducto(int numProducto)`: Devuelve el producto que ocupa la posición `numProducto` del surtido. La primera posición válida es la `1`. La posición `0` no se utiliza.
- `public String[] getNombresProductos()` Devuelve un array con los nombres de los productos. La posición `0` del array no se utilizará (será `null`)

4. Revisar la clase `Expendedora`. Añadir los atributos y hacer los cambios necesarios en la clase para que sea capaz de dispensar varios productos.
5. Añadir a la clase el método public Surtido `getSurtido()`, que devuelva el surtido de la máquina (`Objeto` de la clase `Surtido`)
6. Modificar la clase `TestExpendedora` para adaptarla a los cambios hechos en la clase `Expendedora`.

2. Actividades

1. Introducir por teclado un valor de tipo `double` y convertirlo en Wrapper.
2. Introducir por teclado un valor numérico en un `String` y convertirlo en entero.
3. Introducir por teclado un valor numérico entero en un `String` y convertirlo en un `Wrapper`.
4. Introducir por teclado dos valores numéricos enteros y la operación que queremos realizar (`suma`, `resta` o `multiplicación`). Realizar la operación y mostrar el resultado en `Binario`, `Hexadecimal` y `Octal`.
5. Mostrar los segundos transcurridos desde el `1 de Enero de 1970 a las 0:00:00` hasta `hoy`.
6. Mostrar la `fecha` y `hora` de hoy con los siguientes formatos (para todos los ejemplos se supone que hoy es 26 de agosto de 2021 a las 17 horas 16 minutos y 8 segundos):

   a) `August 26, 2021, 5:16 pm`
   b) `08.26.21`
   c) `26, 8, 2021`
   d) `20210826`
   e) `05-16-08, 26-08-21`
   f) `Thu Aug 26 17:16:08`
   g) `17:16:08`
7. Introducir un día, un mes y un año y verificar si es una fecha correcta.
8. Introducir dos fechas e indicar los días transcurridos entre las dos fechas.
9. Introducir una fecha y devolver las fecha de los pagos a 30, 60 y 90 días.
10. Introducir tres fechas e indicar la mayor y a menor.
11. Introducir el día, mes, año. Crear una fecha a partir de los datos introducidos y comprobar e indicar si se trata de la fecha actual, si es una fecha pasada o una fecha futura.
12. Introducir una fecha de nacimiento de un empelado e indicar cuántos años tiene el empleado.
13. Introducir la fecha de caducidad de un producto e indicar si el producto está o no caducado. El valor por defecto será la fecha actual y solo se podrán introducir fechas del año en curso.
14. Mostrar una fecha con formato dd/mm/aaaa utilizando 0 delante de los días o meses de 1 dígito.
15. Mostrar una fecha con formato `DiaSemana`, `DiaMes` de `Mes` del `Año` a las `horas:minutos:segundos`. Por ejemplo: Miercoles, 9 de Diciembre del 2015 a las 18:45:32
16. Suma 10 años, 4 meses y 5 días a la fecha actual.
17. Resta 5 años, 11 meses y 18 días a la fecha actual.
18. Introducir el número de horas trabajadas por un empleado y la fecha en las que las trabajo. Si el día fue sábado o domingo el precio hora trabajada es 20€ en caso contrario 15€. Calcula la cantidad de dinero que habrá que pagar al empleado por las horas trabajadas.

19. Introducir la fecha inicial y final de una nómina y calcular lo que debe cobrar el empleado sabiendo que cada día trabajado recibe 55 € y tiene una retención del 12% sobre el sueldo.
20. Crear una clase `Alumno` con los atributos `codigo`, `nombre`, `apellidos`, `fecha_nacimiento`, `calificacion`. La fecha de nacimiento deberá introducirse como una fecha. Crear constructor, métodos `setter` y `getter` y `toString`. Crear una instancia con los siguientes valores `1`, `'Luis'`, `'Mas Ros'`, `05/10/1990`, `7.5`. Mostrar los datos del alumno además de su edad.
21. Introducir la fecha de entrega de un documento y nos diga si está dentro o fuera de plazo teniendo en cuenta que la fecha de entrega límite es la fecha actual.
22. Introducir en un array `nombre`, `apellidos` y `sueldo` de varios trabajadores y la `fecha de alta` en la empresa. Las fechas deberán introducirse como fechas. Recorrer el array y mostrar para cada trabajador la retención que debe aplicarse sobre el sueldo teniendo en cuenta que los trabajadores incorporados antes de 1980 tienen una retención del 20%, los trabajadores con fecha entre 1980 y 2000 una retención del 15% y los trabajadores con fecha posterior al 2000 la retención que aplicaremos será el 5% del sueldo.
23. Realizar una aplicación para la gestión de la información de las personas vinculadas a una `Facultad`, que se pueden clasificar en tres tipos: estudiantes, profesores y personal de servicio.
    A continuación, se detalla qué tipo de información debe gestionar esta aplicación:
    - Por cada `Persona`, se debe conocer, al menos, su `nombre` y `apellidos`, su `número de identificación` y su `estado civil`.
    - Con respecto a los `Empleados`, sean del tipo que sean, hay que saber su `año de incorporación` a la facultad y qué `número de despacho` tienen asignado.
    - En cuanto a los `Estudiantes`, se requiere almacenar el `curso` en el que están matriculados.
    - Por lo que se refiere a los `Profesores`, es necesario gestionar a qué `departamento` pertenecen (`lenguajes`, `matemáticas`, `arquitectura`, ...).
    - Sobre el `Personal de servicio`, hay que conocer a qué `sección` están asignados (`biblioteca`, `decanato`, `secretaría`, ...).

    El ejercicio consiste, en primer lugar, en definir la jerarquía de clases de esta aplicación. A continuación, debe programar las clases definidas en las que, además de los constructores, hay que desarrollar los métodos correspondientes a las siguientes acciones:
    - Cambio del estado civil de una persona.
    - Reasignación de despacho a un empleado.
    - Matriculación de un estudiante en un nuevo curso.
    - Cambio de departamento de un profesor.
    - Traslado de sección de un empleado del personal de servicio.
    - Imprimir toda la información de cada tipo de individuo.

    En el método `main` crear un array de `personas`. Crear diferentes instancias de las subclases e insertarlas en el array. Probar los diferentes métodos desarrollados.
24. Crea una clase `Empleado` y una subclase `Encargado`. Los encargados reciben un 10% más de sueldo base que un empleado normal aunque realicen el mismo trabajo. Implementa dichas clases en el paquete objetos y sobrescribe el método `getSueldo()` para ambas clases.

25. Realiza un método estático que dada la `fecha de nacimiento` de una persona indique si es mayor de edad.
26. Crear la clase `Dado`, la cual desciende de la clase `Sorteo`. La clase `Dado`, en la llamada `lanzar()` mostrará un número aleatorio del 1 al 6. Crear la clase `Moneda`, la cual desciende de la clase `Sorteo`. Esta clase en la llamada al método `lanzar()` mostrará las palabras cara o cruz.
27. Realiza una clase `Conversor` que tenga las siguientes características: Toma como parámetro en el constructor un valor entero. Tiene un método `getNumero` que dependiendo del parámetro devolverá el mismo número en el siguiente `B Binario`, H Hexadecimal, `O Octal`. Realiza un método `main` en la clase para probar todo lo anterior.
28. Realiza una clase `ConversorFechas` que tenga los siguientes métodos:
    - `String normalToAmericano(String)`. Este método convierte una fecha en formato normal `dd/mm/yyyy` a formato americano `mm/dd/yyyy`
    - `String americanoToNormal(String)`. Este método realiza el paso contrario, convierte fechas en formato americano a formato normal.
29. Realiza una clase `Huevo` con un atributo `tamaño` (`S`, `M`, `L`, `XL`) con el método `toString`. La clase `Huevo` está compuesta por dos clases internas, una `Clara` y otra `Yema`. Ambas clases tiene un atributo `color` y el método `toString`. Realiza un método `main` en el que se cree un objeto de tipo `Huevo`, `Clara` y `Yema`. Se le asigne valor a sus atributos y se muestren dichos valores.

3. Fuentes de información

- Wikipedia
- Programación (Grado Superior) - Juan Carlos Moreno Pérez (Ed. Ra-ma)
- Apuntes IES Henri Matisse (Javi García Jimenez?)
- Apuntes AulaCampus
- Apuntes José Luis Comesaña
- Apuntes IOC Programació bàsica (Joan Arnedo Moreno)
- Apuntes IOC Programació Orientada a Objectes (Joan Arnedo Moreno)