PROGRAMACIÓN – 1º DAM

EXAMEN 1ª convocatoria FINAL (25-MAYO-2022) curso 2021/2022

Nombre y Apellidos:	**Puntuación:**

Se va a crear un programa simple de gestión para una **asociación de adopción de animales**. Se tienen los siguientes datos iniciales y el modelo de clases a desarrollar:

Mascotas

id	nombre	fechanac	idave	idmamifero	idsocio	adoptada
1	Loli	07-jul-18	1	-	1	no
2	María	09-nov-18	2	-	1	no
3	Tari2	11-may-16	-	1	1	sí
4	B0st0n	25-may-21	3	-	2	no
5	Apache	30-ene-20	-	2	2	no
6	Sol	14-feb-19	4	-	2	no
7	Tomoyo	25-may-21	5	-	2	no
8	Jazz	06-jun-17	-	3	-	no

Aves

idave	nombre	salvaje
1	Loli	sí
2	María	no
3	B0st0n	sí
4	Sol	no
5	Tomoyo	no

Mamíferos

idmam	nombre	peso
1	Tari2	6.980
2	Apache	10.500
3	Jazz	4.875

Socios

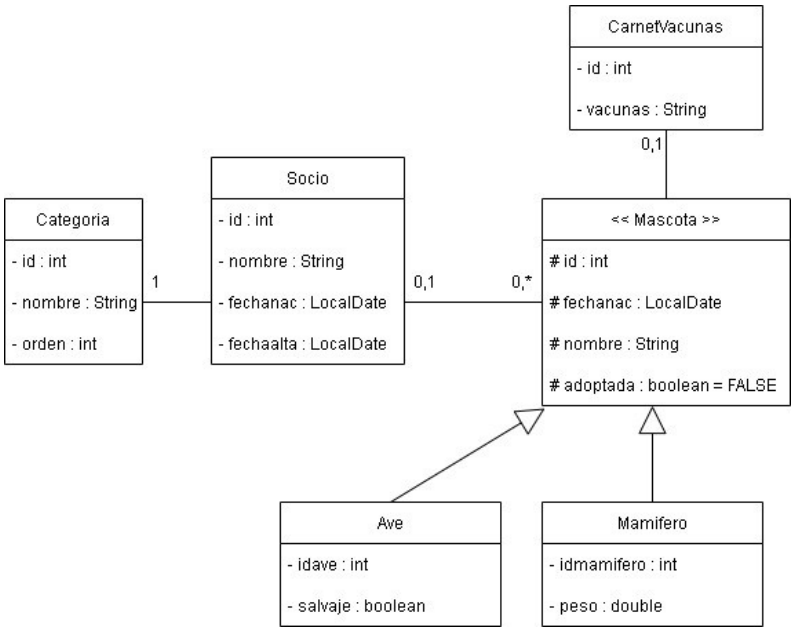
id	nombre	fechanac	fechaalta	categoria	Mascotas
1	José Carlos García	19-nov-88	3	3	1, 2, 3
2	Antonia del Pozo	20-dic-85	3	3	4, 5, 6, 7

Categorias

id	nombre	orden
1	Normal	3
2	Premium	2
3	VIP	1

CarnetsVacunacion

id	idmascota	vacunas
1	3	leptospirosis el 20/06/2016, rabia el 22/07/2016,
2	5	rabia el 13/02/2020,



0. Se tendrá que crear un nuevo proyecto de nombre, `AsociacionXXX` donde XXX es el nombre de cada estudiante. La entrega consistirá en un .ZIP con el proyecto exportando, que incluya todos los ficheros y librerías necesarios para poder ser compilado y ejecutado sin necesidad de ninguna configuración adicional. En caso de encontrar cualquier error sintáctico que imposibilite lo anterior se tomará la prueba como no superada de forma directa. Cada estudiante deberá implementar todos los ficheros que considere oportunos en función de los ejercicios que debe realizar. Sólo se evaluarán los desarrollos correspondientes a los ejercicios pedidos en cada caso particular, por lo que no será necesaria la implementación completa de todo el modelo entero.

		EVAL1				EVAL 2				EVAL 3			
	Ejercicios	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
TODO		**X**	**X**	**X**	**X**	**X**	**X**	**X**		**X**	**X**		
Ejercicios del 1 al 10 excepto 8	**Ptos. / Ejercicio**	**0,5**	**0,5**	**0,5**	**1,5**	**0,5**	**0,5**	**1,5**		**1,5**	**3**		
Eval2 + Eval3			**X**			**X**		**X**	**X**	**X**	**X**		
Ejercicios 2, 5, y del 7 al 10	**Ptos. / Ejercicio**		**0,5**			**0,5**		**1,5**	**3**	**1,5**	**3**		
sólo Eval3			**X**			**X**				**X**	**X**	**X**	**X**
Ejercicios 2, 5 y del 9 al 12	**Ptos. / Ejercicio**		**0,5**			**0,5**				**1,5**	**3**	**2,5**	**2**

1. (Máx 0,5ptos.) Implementar la enumeración *`Categoria`* que tiene todo socio, cuyos elementos se corresponden con los datos de la siguiente tabla:

	Enum Categoria		
Value	**int id**	**String nombre**	**int orden**
NORMAL	1	normal	3
PREMIUM	2	premium	2
VIP	3	V.I.P.	1

   Implementar un constructor para la enumeración con esos 3 parámetros, métodos para la obtención de cada campo por separado, y otro método también público y estático que muestre por la pantalla todos los valores de esa enumeración, cada uno en una línea distinta y separando cada campo por un tabulador.

2. (Máx 0,5ptos.) Implementar 2 métodos públicos estáticos `data()` y `mostrarCompleto()` en la clase `Socio.java` para devolver una cadena de caracteres en ambos casos. En el primero, se mostrarán los campos: el id del socio seguido del nombre, luego la fecha de nacimiento y la fecha de alta (ambas en formato dd/MM/yyyy) y finalmente el id de la categoría, todos ellos separados mediante el carácter '`|`'.
   El otro método, además, se mostrará la colección de `Mascotas` del socio (si es que tiene alguna), indicando para cada una de ellas, su id seguido del nombre y si se trata de un ave o de un mamífero (indicando el idave o idmamifero respectivamente).

3. (Máx 0,5ptos.) Se quiere identificar en todo momento del programa cuáles son los identificadores de las mascotas registradas más vieja y más joven. Añadir a la clase `Mascota` un array de 2 posiciones donde la 1ª posición corresponderá al idMascota más longevo y la 2ª posición el de la que sea más joven.

4. (Máx 1,5ptos.) Completar las clases `Socio` y `Mascota`, junto a las relaciones con el resto de entidades del diagrama de clases, de forma que:
   - Todo `Socio` tiene obligatoriamente una `Categoria` y solo una. Además, tiene una colección de `Mascotas` (que puede ser vacía). Por su parte, cada `Mascota` pertenece a siempre a 1 solo

`Socio` y algunas mascotas ya tienen su propio `Carnet` de vacunación. Las mascotas o bien son un `Ave` o bien son un `Mamifero`, en ambos casos con un id propio dentro de cada subtipo.

- Todas las clases dispondrán de un constructor por defecto, otro de copia y otro constructor con todos los campos obligatorios, así como métodos getters y setters para cada campo siguiente:
  - un identificador propio y único para cada clase. Y, además,:
  - Para la clase `Socio`:
    - Un nombre que es una cadena de caracteres de entre 3 y 150 caracteres, siendo válidos sólo los caracteres alfabéticos (letras).
    - Una fecha de nacimiento que ha de ser validada para que sea mayor de 16 años.
    - La fecha de alta del socio corresponde con el momento de creación del socio en el sistema.
    - La `Categoria` a la que pertenece el socio.
    - La colección de sus `Mascotas` (que puede ser vacía).
  - Para la clase `Mascota`:
    - Un nombre que es una cadena de caracteres de entre 2 y 30 caracteres, siendo válidos tanto los caracteres alfabéticos (letras) como los caracteres numéricos (dígitos), pero no otro tipo de caracteres.
    - La fecha de nacimiento de la mascota. No puede ser posterior a hoy.
    - Un campo que indica si la mascota es adoptada o no (por defecto no lo es).
    - El idsocio al que pertenece la mascota. Puede darse que una mascota no pertenezca aún a nadie, así queda disponible para su adopción posterior por parte de un socio.
    - El idcarnet de vacunación si es que lo tiene ya la mascota. El carnet simplemente tiene un identificador y luego se concatena en una cadena de cualesquiera caracteres (hasta 500 como máximo) las vacunas del animal, separadas por comas, e indicando el nombre de la vacuna y la fecha en que se inyectó.
  - Para la clase `Ave`:
    - Se tendrá un identificador propio para las aves.
    - Además, se almacena si el ave es salvaje o no.
  - Para la clase `Mamifero`:
    - Se tendrá un identificador propio para los mamíferos.
    - Además, se almacena el peso del animal, expresado en Kilogramos (con 3 decimales).
- Implementar los métodos públicos y estáticos `nuevaMascota()` y `nuevoSocio()` para obtener sendos objetos completos validados desde la entrada estándar por parte de un usuario.
  - En el caso de la mascota, preguntar siempre por el nombre, fecha de nacimiento, si es adoptada o no, el tipo de mascota (ave o mamífero) junto a los datos propios de cada subtipo y dar la posibilidad opcional al usuario de introducir un nuevo carnet de vacunación y/o los datos de un nuevo socio (que sería el dueño de la mascota a crear).
  - Para el caso del socio, preguntar por el nombre, su fecha de nacimiento, la categoría a la que pertenece (de entre las posibles) y la lista de sus mascotas.

5. (Máx 0,5 ptos.) (apartado A) Crear una nueva interfaz de nombre `Exportable` que trabaje con genéricos. Contendrá 2 métodos: uno para exportar hacia un fichero de caracteres y otro para exportar hacia un fichero de bytes. Ambos métodos reciben 2 parámetros cada uno: el primer parámetro será del tipo genérico y corresponderá al elemento a exportar; el 2º parámetro será la localización del fichero donde se exportará el elemento. Ambos métodos devolverán un valor booleano para indicar el éxito de la exportación del elemento hacia el fichero de la ruta indicada.
(apartado B) Crear una nueva interfaz de nombre `OperacionesCRUD` que también trabaje con genéricos. Contendrá 3 métodos: un primer método para insertar un elemento de tipo genérico en la BD (en la tabla/s que corresponda) y que devuelva el identificador creado tras la inserción o -1 si

hubo cualquier error durante ese proceso. Otro segundo método para buscar en la BD un elemento a través de su identificador que se pasa como parámetro. Si lo encuentra devolverá el elemento o null en caso contrario (o si hubiese cualquier problema en la búsqueda). Un último tercer método que devuelva la colección de todos los elementos (si los hay) de un tipo recuperados desde la BD.

6. (Máx 0,5 ptos.) Añadir a los métodos `nuevoSocio()` y `nuevaMascota()` del ejercicio 4 una gestión adecuada de todas las posibles excepciones que puedan darse al introducir cualesquiera de los valores de los campos desde la entrada estándar. Mostrar, en caso de producirse, un mensaje por la salida estándar indicando que hubo un error, de cuál se trata, y la pila de llamadas que lo produjeron a través del método `printStackTrace()`.

7. (Máx 1,5 ptos.) (apartado A → 0,5ptos) Implementar la interfaz `Exportable` del ejercicio 5.A para la clase `Socio`. Para el método del fichero de caracteres, utilizar la función `mostrarCompleto()` del ejercicio 2. Posteriormente, en la función principal, declarar una variable de tipo `Socio` para un nuevo socio y llamar a los métodos de la interfaz para ejemplificar su uso.
   (apartado B → 1pto): Crear una nueva clase `ComparadorMascotas` que implemente la interfaz `Comparator` para las mascotas, de forma que se ordenen en función de su fecha de nacimiento y, si coinciden, desempatar por orden alfabético del nombre. Incluir en la función principal un ejemplo de uso, en el que se declare una variable para una lista de Mascotas, inicialmente vacía, y se solicite al usuario que introduzca los datos de varias nuevas mascotas. Recorrer mediante un iterador esa lista y mostrar por la salida estándar el idMascota, nombre y de fecha nacimiento. Ordenar luego la lista mediante el comparador implementado y recorrerla de nuevo a través de un iterador, volviendo a mostrar los mismos datos de cada mascota.

8. (máx 3 ptos) (apartado A → 0,5ptos) Implementar la interfaz `Comparable` para la clase `Mascota`, de forma que se ordenen en función de su fecha de nacimiento (de más joven a más vieja) y, si coinciden, desempatar por orden alfabético del nombre.
   (apartado B → 1pto.) Implementar un método que toma los datos desde 2 ficheros de caracteres de nombres `aves.txt` y `mamiferos.txt` que contienen los datos de nuevas mascotas de ambos tipos para adoptar, con las siguientes estructuras:
   - En el fichero de las aves con el siguiente orden:
     <idave> | <nombre> | <fecha_nacimiento (dd/MM/yyyy)> | <salvaje> | <vacunas>
   - En el fichero de los mamíferos con el siguiente orden:
     <idmamifero> | <nombre> | <fecha_nacimiento (dd/MM/yyyy)> | <peso> | <vacunas>

   Recorrer ambos ficheros para crear objetos completos de los tipos `Ave` y `Mamifero` con dichos datos y almacenarlos en 2 conjuntos, que se puedan ordenar según el criterio anterior.
   (apartado C → 1,5ptos.) Completar el método del apartado B de manera que se cree, con los elementos de los 2 conjuntos, una lista de mascotas ordenada (según el criterio del apartado A) con aquellas mascotas cuya fecha de nacimiento sea de los últimos 6 meses. Exportar luego esa lista resultante hacia un fichero binario de nombre `neonatos.dat`.

9. (Máx 1,5ptos.) Asumir que se tiene el siguiente modelo relacional para persistir la información de las entidades manejadas por la aplicación en las siguientes tablas.

CATEGORIAS
id int(11) PK AUTOINCREMENT **nombre** VARCHAR(20) UNIQUE NOT NULL **orden** int NOT NULL

SOCIOS
id int(11) PK AUTOINCREMENT **nombre** VARCHAR(150) NOT NULL **fechanac** DATE NOT NULL **fechaalta** DATE NOT NULL **idcategoria** INT(11) FK REFERENCES categorias.id

CARNETS
id INT(11) PK AUTOINCREMENT **vacunas** VARCHAR(500) NOT NULL

MASCOTAS
id INT(11) PK AUTOINCREMENT **nombre** VARCHAR(30) NOT NULL **fechanac** DATE NOT NULL **adoptada** BOOLEAN NOT NULL **idave** INT(11) **idmamifero** INT(11) **salvaje** BOOLEAN **peso** DECIMAL(6,2) **idcarnet** INT(11) FK REFERENCES carnets.id **idsocio** INT(11) FK REFERENCES socios.id

(apartado A → 1pto.) Crear una clase `SocioDAO` que implemente la interfaz `OperacionesCRUD` del ejercicio 5.B para la clase `Socio`, de forma que se codifiquen los 3 métodos:

- un primer método para insertar un elemento de tipo `Socio` en la BD (en las tablas que corresponda) y que devuelva el identificador de socio creado tras la inserción o -1 si hubo cualquier error durante el proceso.
- Otro segundo método para buscar en la BD un `Socio` a través de su identificador que se pasa como parámetro. Si lo encuentra, devolverá un objeto socio (solo con los datos de las tablas socios y categorías) o devolverá `null` en caso no encontrarlo o de error.
- Un último tercer método que devuelva una colección de objetos `Socio` con todos los socios del sistema recuperados desde las tablas socios y categorias de la BD.

(apartado B → 0,5ptos) Estructurar el proyecto para que trabaje en varias capas/paquetes, distinguiéndose según la funcionalidad propia de cada clase. Añadir una clase para realizar una conexión a una BD MySQL de nombre **bdasociacion** y que trabaje en el propio servidor local en el puerto 3306. Esa clase deberá contener también un método para poder cerrar una conexión existente. Luego, en la función principal, crear 2 variables de tipo `Socio` con los datos que se muestran al comienzo de esta prueba y una variable `SocioDAO` para conectar con la BD. Insertar en la BD a través del objeto DAO los 2 socios, recuperar después todos los socios desde la BD y finalmente buscar el socio de identificador 3.

10. (Máx:3ptos) (apartado A → 1,5ptos.) Implementar una GUI de nombre `FrameNuevoSocio.java` para obtener todos los datos básicos de un nuevo objeto `Socio`, es decir, su nombre, fecha de nacimiento y categoría a la que pertenece (de entre las 3 marcadas en el ejercicio 1). Se incluirá en ella un botón '*Aceptar*' que, al ser pulsado, activará un escuchador para su procesamiento. Éste consiste en tomar todos y cada uno de los valores de los campos de la GUI para posteriormente validarlos (de acuerdo a lo expuesto en el ejercicio 4 para la clase `Socio`). En caso de no ser válidos se avisará al usuario mediante una ventana emergente y se le informará de su/s error/es.

(apartado B → 0,75ptos.) Si todo va bien tras la validación se procederá a la inserción en la BD sobre la tabla socios mediante un objeto `SocioDAO` (del ejercicio 11.A), para posteriormente informar al usuario del éxito de la inserción (indicando el nuevo `idsocio` asignado) o del error.
(apartado C → 0,75ptos.) Por último, si hubo éxito al insertar los datos del nuevo socio en la BD, exportar la información de éste hacia un fichero de caracteres de nombre `carnetsocio_<idsocio>.txt` con el formato siguiente:

```
Asociación "Mascotas 4ever" de Oviedo
Carnet de socio Nº: <idsocio> para <nombre>.
Fecha nacimiento: <dd/MM/YYYY>                Categoría: <categoria>
-------------------------------------
En Oviedo, a <fecha_alta(dd/MM/yyyy)>.
```

Avisar al usuario mediante una ventana emergente del resultado de la exportación hacia el fichero, tanto si fue exitoso como si hubo cualquier problema (indicando en ese caso lo que ocurrió).

11. (Máx 2,5ptos.) (apartado A → 1,5ptos.) Crear una clase `MascotaDAO` que implemente la interfaz `OperacionesCRUD` del ejercicio 5.B para la clase `Mascota`, para que se puedan trabajar objetos tanto de la entidad `Ave` como de la entidad `Mamifero`. Se podrán reimplementar los métodos de la interfaz como se desee, pero se pide, al menos:
    - Un método para insertar un objeto de tipo Mascota, que se encargará de averiguar el subtipo correspondiente para persistir adecuadamente la información sobre la tabla mascotas de la BD, y que devuelva el nuevo idmascota generado tras la inserción o -1.
    - Un método para buscar una Mascota que, dado un identificador de mascota, devuelva el objeto desde la tabla mascotas con su subtipo adecuado y los datos propios del subtipo, así como todos los datos de la propia clase Mascota.
    - Un método para obtener la colección de todas las mascotas desde la tabla mascotas de la BD, generando por cada fila de esa tabla un objeto nuevo del subtipo de mascota adecuado y que se añada a la colección a devolver.

    (apartado B → 1pto.) Añadir un método público `Mamiferos[] buscarSoloMamiferos()` a la clase anterior para que devuelva un array de objetos de tipo `Mamifero` con la colección de las filas de la tabla mascotas de la BD que corresponden a mamíferos únicamente. Luego, probar este método desde la función principal, recorriendo cada elemento del array resultante mediante un bucle de tipo `for` y mostrando por la pantalla los siguientes datos para cada uno en una línea distinta:
    idMamifero=<idmamifero> . <nombre>, <fechanac(dd/MM/yyyy)>, peso=<peso(xx.xxx)> Kg.
    Además, si la mascota tiene carnet de vacunación, añadir:
    idcarnet=<idcarnet> . vacunas=<vacunas>

12. (Máx 2ptos.) (apartado A→0,75ptos.) Implementar la GUI `FrameNuevaMascota.java` para la inserción de los datos de una nueva `Mascota` en el sistema por parte del usuario. Se le solicitará el nombre de la mascota y la fecha de nacimiento y se dará la opción de marcar si es adoptada o no. Luego se permitirá la opción al usuario de seleccionar el `Socio` al que pertenece la nueva mascota (de entre los ya existentes en la tabla socios de la BD) o "ninguno" si se trata de una mascota que estará disponible para adoptar posteriormente. Así mismo, el usuario determinará el subtipo concreto de la mascota (ave o mamífero) y se le presentará la manera de introducir los datos propios de cada uno. De forma opcional, el usuario podrá incluir desde un componente de tipo `TextArea` la colección de vacunas de la nueva mascota.
    (apartado B→ 0,75ptos.) Añadir un botón "Aceptar" que active un escuchador para el evento click. En éste se recogerán todos los datos del formulario del apartado anterior para componer un objeto `Mascota` y se validará éste de acuerdo a lo expresado en el ejercicio 4. Además, lo siguiente:

- Si el usuario introdujo un valor válido en el campo de vacunas, se generará un carnet de vacunación también nuevo asociado a la nueva mascota unívocamente.
- El usuario obligatoriamente debe marcar si la nueva mascota es un ave o un Mamífero y se deberá validar que sólo se hayan introducido valores en los campos que corresponde según el subtipo elegido, además de su validación particular.
- Por defecto se marcará “ninguno” como elemento para seleccionar el socio al que pertenezca la nueva mascota. Si se eligió otro elemento distinto a éste deberá enlazarse el nuevo objeto `Mascota` con el objeto `Socio` correspondiente.

(apartado C→0,5ptos.) Si la validación del objeto `Mascota` en el apartado anterior fue correcta, proceder a la persistencia de toda la información de la nueva mascota sobre la BD en la tabla mascotas (y, opcionalmente también en la tabla carnets). Para ello, utilizar el mecanismo que se haya implementado en los ejercicios anteriores. Notar que el idave/idmamifero es un dato que se debería autocalcular al persistir en la BD). Informar al usuario del resultado de la operación, indicando mediante una ventana emergente si hubo éxito o no. En el primer caso mostrando entonces el nuevo idMascota generado, el nuevo idAve ó idMamifero asignado, los datos del socio y de la vacunación (si es que los tiene) y el resto de los otros datos de la mascota recién insertada.