LTP – Ejercicios de Java

Bloque 1: Ejercicios de Herencia

1. Escribe una clase *Multimedia* para almacenar información de objetos de tipo multimedia (películas, discos, sonido en mp3, vídeos en mp4…). Esta clase contiene los siguientes atributos: *título*, *autor*, *formato* y *duración*. El formato puede ser uno de los siguientes[1]: wav, mp3, midi, avi, mov, mpg, cd de audio y dvd. El valor de todos los atributos se pasa por parámetro en el momento de crear el objeto. Esta clase tiene, además, un método consultor para devolver cada uno de los atributos y un método *toString()* que devuelve en un *String* la información del objeto. Por último, un método *equals()* que recibe un objeto de tipo *Multimedia* y devuelve *true* en caso de que el título y el autor sean iguales y *false* en caso contrario.

---

2. Escribe una clase *Película* que herede de la clase *Multimedia* anterior. La clase *Película* tiene, además de los atributos heredados, un actor principal y una actriz principal. La clase debe tener dos métodos consultores para obtener los nuevos atributos y debe sobrescribir el método *toString()* para reflejar la nueva información.

---

3. Escribe una clase *Disco*, que herede de la clase *Multimedia* ya realizada. La clase *Disco* tiene, aparte de los elementos heredados, un atributo para almacenar el género al que pertenece (con valores posibles rock, pop y ópera). La clase debe tener un método para obtener el nuevo atributo y debe sobrescribir el método *toString()* para que devuelva toda la información.

---

4. Escribe una clase *ListaMultimedia* para almacenar objetos de tipo multimedia. La clase debe tener un atributo que sea un array de objetos *Multimedia* y un entero para contar cuántos objetos hay almacenados. Además, tiene un constructor y los siguientes métodos:

- el constructor recibe un entero por parámetro indicando el número máximo de objetos que va a almacenar.
- *talla()*: devuelve el número de objetos que hay en la lista.
- *agregar(Multimedia m)*: añade el objeto *m* al final de la lista, y devuelve *true*; en caso de que la lista esté llena, devuelve *false*.
- *Multimedia obtener(int pos)*: devuelve el objeto situado en la posición especificada.
- *String toString()*: devuelve la información de los objetos que están en la lista.

---

5. Escribe una aplicación donde:

- Se cree un objeto de tipo *ListaMultimedia* de tamaño 10.
- Se creen dos películas y se añadan a la lista.

---

[1] Para restringir un atributo a un conjunto finito de valores constantes, puedes utilizar las *clases enumeradas* de Java. Más información aquí: https://docs.oracle.com/javase/tutorial/java/javaOO/enum.html

- Se creen dos discos y se añadan a la lista.
- Se muestre la lista por pantalla.

---

6. Escribe una clase *Lavadora* con las siguientes características:

- Sus atributos son *precio base*, *color*, *consumo energético* (letras entre A y F), *peso* y *carga máxima*. Dichos atributos deben ser privados.
- La clase contendrá tres constructores. Uno sin parámetros que inicialice todos los atributos a sus valores por defecto; otro que reciba el precio base y el peso (el resto de atributos se inicializarán por defecto); y un último constructor con todos los atributos.
- Por defecto, el color será blanco, el consumo energético será F, el precio base es de 100 €, el peso de 5 kg y la carga de 5 kg.
- Los colores disponibles son: blanco, negro, rojo, azul y gris. Usa una clase enumerada para representarlos.

Además, la clase debe implementar los siguientes métodos:

- Métodos consultores para todos los atributos.
- *comprobarPrecioBase(double precio)*: comprueba que el precio es válido (no es un número negativo) y, si lo es, lo devuelve. En caso contrario, devolverá el precio base por defecto. Será un método privado a invocar en el segundo y tercer constructor.
- *comprobarConsumoEnergetico(char letra)*: comprueba que la letra es válida (está entre la A y la F) y, si lo es, la devuelve. En caso contrario, devolverá la letra por defecto. Será un método privado a invocar en el tercer constructor (el que recibe argumentos para todos los atributos).
- *precioFinal()*: aplica un incremento al precio base según el consumo energético, el peso y la carga. Este incremento no debe afectar al valor del atributo *precioBase*, simplemente se devolverá el valor incrementado como resultado del método. Los incrementos se aplican de acuerdo a las siguientes tablas:

CONSUMO ENERGÉTICO	INCREMENTO DE PRECIO
A	+100 €
B	+80 €
C	+60 €
D	+50 €
E	+30 €
F	+10 €

PESO	INCREMENTO DE PRECIO
> 80 kg	+100 €
>50 kg y <=79 kg	+80 €
>20 kg y <=49 kg	+50 €
>=0 kg y <=19 kg	+10 €

CARGA MÁXIMA	INCREMENTO DE PRECIO
>30 kg	+50 €

---

7. Escribe una clase *Televisor* con las siguientes características:

- Sus atributos son *precio base*, *color*, *consumo energético* (letras entre A y F), *peso*, *resolución* (en pulgadas) y si es compatible con *imagen en 3D*. Dichos atributos deben ser privados.
- La clase contendrá tres constructores. Uno sin parámetros que inicialice todos los atributos a sus valores por defecto; otro que reciba el precio base y el peso (el resto de atributos se inicializarán por defecto); y un último constructor con todos los atributos.
- Por defecto, el color será blanco, el consumo energético será F, el precio base es de 100 €, el peso de 5 kg, la resolución de 20 pulgadas y no es compatible con 3D.
- Los colores disponibles son: blanco, negro, rojo, azul y gris. Usa una clase enumerada para representarlos.

Además, la clase debe implementar los siguientes métodos:

- Métodos consultores para todos los atributos.
- *comprobarPrecioBase(double precio)*: comprueba que el precio es válido (no es un número negativo) y, si lo es, lo devuelve. En caso contrario, devolverá el precio base por defecto. Será un método privado a invocar en el segundo y tercer constructor.
- *comprobarConsumoEnergetico(char letra)*: comprueba que la letra es correcta (está entre la A y la F) y, si lo es, la devuelve. En caso contrario, devolverá la letra por defecto. Será un método privado a invocar en el tercer constructor (el que recibe argumentos para todos los atributos).
- *precioFinal()*: aplica un incremento al precio base según el consumo energético, el peso, la resolución y si emite imagen en 3D. Este incremento no debe afectar al valor del atributo *precioBase*, simplemente se devolverá el valor incrementado como resultado del método. Los incrementos se aplican de acuerdo a las siguientes tablas:

CONSUMO ENERGÉTICO	INCREMENTO DE PRECIO
A	+100 €
B	+80 €
C	+60 €
D	+50 €
E	+30 €
F	+10 €

PESO	INCREMENTO DE PRECIO
> 80 kg	+100 €
>50 kg y <=79 kg	+80 €
>20 kg y <=49 kg	+50 €
>=0 kg y <=19 kg	+10 €

RESOLUCIÓN	INCREMENTO DE PRECIO
>40 pulgadas	+30%

COMPATIBLE CON 3D	INCREMENTO DE PRECIO
Sí	+50 €

---

8. Escribe una aplicación donde:

- Se cree un array de objetos de 10 posiciones.
- Se creen 2 objetos de la clase Lavadora y otros 2 de la clase Televisor (valores arbitrarios). Estos objetos se tienen que insertar intercalados en el array (una lavadora, un televisor, una lavadora y un televisor o a la inversa).
- Se recorra el array (sólo las posiciones que tengan objetos) y se muestren los precios finales de cada uno de los objetos.

---

9. Observa que tanto las *Lavadoras* como los *Televisores* comparten características comunes (atributos, métodos, etc.). Empleando la herencia, reescribe el programa de los ejercicios 6 y 7 de forma que dicha información común quede condensada en una clase *Electrodoméstico*, y se reutilice la mayor porción de código posible. No se deben poder instanciar objetos de la clase *Electrodoméstico*.

---

10. Reescribe la aplicación del ejercicio 8 adaptándola a la nueva jerarquía de herencia.

---

11. Reflexiona sobre las implicaciones de adoptar la solución del ejercicio 8 o la del ejercicio 10 desde el punto de vista del mantenimiento del software. ¿Qué ocurriría si se añadiera una tercera clase que herede de *Electrodoméstico*? ¿Qué habría que modificar de ambas soluciones para que continúe funcionando igual? Observa también que en el ejercicio 10 hemos dejado de imprimir la clase del objeto que hemos encontrado, puesto que para mantener esa información tendríamos que seguir efectuando los cástings sobre los objetos del array. ¿Hay alguna manera de obtener un resultado por pantalla idéntico al del ejercicio 8 sin recurrir a polimorfismo *ad-hoc*?

Bloque 2: Ejercicios de Interfaces

1. Escribe una interfaz, llamada *ColeccionInterfaz*, que declare los siguientes métodos:

- *estaVacia()*: método booleano que devuelve *true* si la colección está vacía y *false* en caso contrario.
- *extraer()*: devuelve y elimina el primer elemento (Object) de la colección.
- *primero()*: devuelve el primer elemento (Object) de la colección, sin eliminarlo.
- *agregar(Object)*: método booleano que añade un objeto por el extremo que corresponda, y devuelve *true* si se ha añadido y *false* en caso contrario.

A continuación, escribe una clase *Pila*, que implemente esta interfaz, utilizando para ello un array de *Object* y un contador de objetos.

---

2. Escribe una clase, de nombre *PruebaPila*, en la que se implementen dos métodos:

- *rellenar()*: recibe por parámetro un objeto de tipo *ColeccionInterfaz*, y añade los números del 1 al 10.
- *imprimirYVaciar()*: recibe por parámetro un objeto de tipo *ColeccionInterfaz* y va extrayendo e imprimiendo los datos de la colección hasta que se quede vacía.
- Crea un objeto de tipo *Pila* y pruébalo.

---

3. Escribe una clase *Cola* que implemente la interfaz *ColeccionInterfaz*, usando un objeto de la clase *LinkedList*.

---

4. Escribe un programa para una biblioteca que contenga libros y revistas. Las **características comunes** que se almacenan tanto para las revistas como para los libros son el *código*, el *título*, y el *año de publicación*. Estas tres características se pasan por parámetro en el momento de crear los objetos. Los libros tienen además un atributo *prestado*. Los libros, cuando se crean, no están prestados. Las revistas tienen un número. En el momento de crear las revistas se pasa el número por parámetro. Tanto las revistas como los libros deben tener (aparte de los constructores) un método *toString()* que devuelve el valor de todos los atributos en una cadena de caracteres. También tienen un método que devuelve el año de publicación, y otro el código. Para prevenir posibles cambios en el programa se tiene que implementar una interfaz *Prestable* con los métodos *prestar()*, *devolver()* y prestado. La clase *Libro* implementa esta interfaz.

---

5. Escribe una aplicación en la que se implementen dos métodos:

- *cuentaPrestados()*: recibe por parámetro un array de objetos, y devuelve cuántos de ellos están prestados.
- *publicacionesAnteriores A()*: recibe por parámetro un array de *Publicaciones* y un año, y devuelve cuántas publicaciones tienen fecha anterior al año recibido por parámetro.

- En el método *main()*, crear un array de *Publicaciones*, con 2 libros y 2 revistas, prestar uno de los libros, mostrar por pantalla los datos almacenados en el array y mostrar por pantalla cuántas hay prestadas y cuantas hay anteriores a 1990.

Bloque 3: Ejercicios de Genericidad

1. Escribe una clase *Pila* genérica usando para ello un atributo del tipo *LinkedList*. La clase *Pila* tendrá los siguientes métodos:

- *estaVacia()*: devuelve *true* si la pila está vacía y *false* en caso contrario.
- *extraer()*: devuelve y elimina el primer elemento de la colección.
- *primero()*: devuelve el primer elemento de la colección
- *agregar()*: añade un objeto por el extremo que corresponda.
- *toString()*: devuelve en forma de *String* la información de la colección

---

2. Implementa una pila utilizando como atributos un array genérico y un entero que cuente el número de objetos insertados. La clase se debe llamar *PilaArray* y tiene los mismos métodos que la pila del ejercicio anterior.

---

3. Escribe una clase *Matriz* genérica con los siguientes métodos:

- constructor que recibe por parámetro el número de filas y columnas de la matriz.
- *set()* recibe por parámetro la fila, la columna y el elemento a insertar. El elemento es de tipo genérico. Este método inserta el elemento en la posición indicada.
- *get()* recibe por parámetro la fila y la columna. Devuelve el elemento en esa posición. El elemento devuelto es genérico.
- *columnas()* devuelve el número de columnas de la matriz.
- *filas()* devuelve el número de filas de las matriz.
- *toString()* devuelve en forma de *String* la información de la matriz.

---

4. Escribe una aplicación que:

- Cree una matriz de enteros de 4 filas y 2 columnas
- Rellénala con números consecutivos comenzando por el 1.
- Muestra por pantalla la matriz.
- Muestra por pantalla el contenido de la celda en la fila 1, columna 2

---

5. Escribe una interfaz *ColeccionSimpleGenerica*, que como su propio nombre indica, es genérica, con los siguientes métodos:

- *estaVacia()*: devuelve true si la pila está vacía y false en caso contrario
- *extraer()*: devuelve y elimina el primer elemento de la colección.
- *primero()*: devuelve el primer elemento de la colección.
- *agregar()*: añade un objeto por el extremo que corresponda.

---

6. Escribe una clase genérica *ListaOrdenada* con un tipo parametrizado *E* que sea *Comparable* (genericidad restringida). La clase debe tener lo siguiente:

- Un constructor

- *boolean add(E o)*: agrega el elemento *o* en la posición que le corresponda, garantizando que la lista se mantiene ordenada.
- *E get(int index)*: devuelve el element en la posición *index*.
- *int size()*: devuelve la cantidad de elementos actual de la lista
- *boolean isEmpty()*: devuelve true si la lista está vacía y false en caso contrario
- *boolean remove(E o)*: elimina el elemento *o* y devuelve *true*, si está en la lista. En caso contrario, devuelve *false*.
- *int indexOf(E o)*: busca el elemento *o* en la lista y devuelve su posición.
- *String toString():* imprime los elementos de la lista.

---

7. Escribe una clase, de nombre *ArrayListOrdenado*, que herede de *ArrayList*. Esta clase solo debe aceptar, como parámetro genérico de tipo, clases que implementen *Comparable*. La clase *ArrayListOrdenado* debe sobrescribir el método *add(E objeto)* para que añada los elementos en orden.

---

8. Escribe una interfaz genérica *Operable*, que sea genérica y que declare las cuatro operaciones básicas: suma, resta, producto y división. Cada operación se debe definir entre el objeto actual (`this`) y un parámetro `par`.