Ejercicio 1

(0,5 puntos)

Dada la definición del tipo **Libro**, escriba la interfaz correspondiente. Si en alguna de las propiedades necesita un tipo enumerado, escriba también la definición del mismo.

Un libro posee un *código* llamado ISBN, formado por una serie de dígitos y otros caracteres (por ejemplo, "978-0385536516"), que lo identifica de manera inequívoca. Otros datos del libro que necesitamos manejar son el *título* (por ejemplo, "Dexter: El oscuro pasajero"), el *nombre del autor* ("Jeff Lindsay"), el *número de páginas* (por ejemplo, 288), la *fecha de adquisición* (12 de febrero de 2005) y el *precio* que costó en euros (19,90). Además, el sistema almacena el número de *copias vendidas* del libro (por ejemplo, 2500000). La aplicación necesita conocer para cada libro si se trata o no de un *best-seller*. Un libro se considera un *best-seller* si ha vendido más de 500000 copias. Cada libro tiene asignado un *tipo de préstamo*, que indica el periodo por el cual el libro puede ser retirado por los usuarios de la biblioteca. Los tipos de préstamo son exclusivamente diario (DIARIO), semanal (SEMANAL) y mensual (MENSUAL). Una vez creado un libro, tan sólo el número de copias vendidas y el tipo de préstamo pueden ser cambiados. Finalmente, el tipo Libro tiene un orden natural dado por el título del libro y, a igualdad de éste, por su código ISBN.

Ejercicio 2

(0,5 puntos)

El tipo **Album** representa un álbum de fotos. La clase que implementa el tipo, **AlbumImpl**, tiene los siguientes atributos:

```
private String nombre;                  // Nombre del álbum
private Integer numeroPaginas;          // Número de páginas del álbum
private LocalDate fechaCreacion;        // Fecha de creación del álbum
private Set<Foto> fotos;                // Conjunto de fotos del álbum
```

Escriba un constructor de la clase `AlbumImpl` que reciba como parámetros un nombre y un número de páginas e inicialice todos los atributos del objeto. Como valor inicial para la fecha de creación se tomará la fecha del día en que se construye el objeto. Compruebe que se cumple la siguiente restricción: el número de páginas del álbum debe ser igual o superior a 10. Utilice un método independiente para comprobar esta restricción. Si no se cumple, se lanzará la excepción `ExcepcionAlbumNoValido`, cuya clase no es necesario escribir.

Ejercicio 3

(0,5 puntos)

El criterio de igualdad de un tipo **Libro** es el siguiente: *dos libros son iguales si tienen el mismo título y el mismo ISBN*. Suponiendo que en la clase `LibroImpl` tenemos un atributo `titulo` de tipo `String` y un atributo `codigo` de tipo `String`, ambos consultables, escriba los métodos `equals` y `hashCode` de dicha clase.

Ejercicio 4

(0,5 puntos)

Dado un tipo **Foto**, queremos implementar un orden natural dado por el nombre de la foto y, a igualdad de éste, por su tamaño. Suponiendo que en la clase `FotoImpl` tenemos un atributo `nombre` de tipo `String` y un atributo `tamano` de tipo `Double`, ambos consultables, escriba el código que hay que añadir a la clase `FotoImpl` para disponer del criterio de orden indicado.

Ejercicio 5 (1 punto)

El tipo **Video** tiene como propiedades el *nombre* del vídeo, de tipo `String`, el *tamaño*, de tipo `Double`, y el *formato*, del tipo enumerado `FormatoVideo`, que puede tomar los valores AVI, MKV y WEBM. El tipo **VideoYouTube** extiende al tipo Video, al cual añade una propiedad *url* de tipo `String` y dos restricciones: la url debe comenzar por "http://www.youtube.com/" y el formato no puede ser MKV. Escriba un constructor de la clase `VideoYouTubeImpl` que reciba como parámetros todas las propiedades de un VideoYouTube e inicialice los atributos de la clase. Si es necesario lanzar una excepción, suponga definida una de nombre `ExcepcionVideoYouTubeNoValido`.

Ejercicio 6 (0,75 puntos)

El tipo **Biblioteca** tiene una propiedad *videos* que consiste en un conjunto ordenado de vídeos. Un vídeo tiene, entre sus propiedades, una denominada *tamaño*, de tipo `Double`, y el orden natural de los vídeos viene dado por su tamaño (un vídeo es menor que otro si su tamaño es menor). Escriba los siguientes métodos de la clase `BibliotecaImpl`, **sin utilizar bucles**:

- `SortedSet<Video> getVideosMenorTamano(Video video);`

  Devuelve un **nuevo** conjunto ordenado con los vídeos de la biblioteca que tienen un tamaño menor que el del vídeo que se recibe como parámetro, que se supone no nulo.

- `Double getTamanoMayor();`

  Devuelve el tamaño del vídeo de mayor tamaño de la biblioteca, que contiene al menos un vídeo.

Ejercicio 7 (1,5 puntos)

El tipo **App** representa una aplicación para dispositivos móviles. Dados tres conjuntos `ios`, `android` y `windows`, que contienen las apps disponibles para los sistemas operativos iOS, Android y Windows Phone, respectivamente, escriba los siguientes métodos de la clase de utilidad Apps, **sin utilizar bucles**:

- `public static Set<App> iosYOtro(Set<App> ios, Set<App> android, Set<App> windows);`

  Devuelve el conjunto de las apps disponibles en iOS y en otro u otros sistemas operativos más.

- `public static Set<App> alMenosDos(Set<App> ios, Set<App> android, Set<App> windows);`

  Devuelve el conjunto de las apps disponibles al menos en dos sistemas operativos.

Ejercicio 8 (1,5 puntos)

El tipo **Farmacia** tiene una propiedad *medicamentos* de tipo `List<Medicamento>` que representa los medicamentos dispensados en la farmacia, y otra propiedad *principios* de tipo `List<String>` que representa los principios activos de los medicamentos. Ambas propiedades están relacionadas, de forma que cada elemento de la segunda lista representa el principio que contiene el medicamento que ocupa la misma posición en la primera lista. Por ejemplo, dadas las listas

```
medicamentos = ["Algidol", "Bisolfren", "Amitron"]
principios   = ["Paracetamol", "Ibuprofeno", "Amoxicilina"]
```

entonces el medicamento "Algidol" contiene el principio activo "Paracetamol", el medicamento "Bisolfren" contiene el principio activo "Ibuprofeno" y el medicamento "Amitron" contiene el principio activo "Amoxicilina".

Teniendo en cuenta que ninguna de las listas puede contener elementos repetidos, implemente los métodos siguientes en la clase `FarmaciaImpl`, **sin utilizar bucles**:

- `Medicamento recetaMedicamento(String principio);`

  Devuelve el medicamento que contiene el principio activo indicado como parámetro, o `null` si no existe ningún medicamento con dicho principio.

- `void nuevoMedicamento(Medicamento medicamento, String principio);`

  Añade el medicamento y su principio activo al final de sus respectivas listas. No obstante, si ya existe un medicamento asociado a dicho principio activo, se debe cambiar por el medicamento recibido como parámetro.

Ejercicio 9 (1 punto)

El tipo **Biblioteca** tiene una propiedad *videos* que consiste en un conjunto ordenado de vídeos. Un vídeo tiene una propiedad *formato*, del tipo enumerado `FormatoVideo`, que puede tomar los valores AVI, MKV y WEBM. Escriba un método de la clase `BibliotecaImpl` que cuente el número de vídeos de cada tipo de formato que contiene la biblioteca. El método devolverá un array de tres elementos de tipo `Integer` que representan, respectivamente, el número de vídeos en formato AVI, MKV y WEBM.

Ejercicio 10 (2,25 puntos)

El tipo **Vuelo** está definido por la siguiente interfaz:

```
public interface Vuelo extends Comparable<Vuelo> {
	String getCodigo();
	String getOrigen();
	String getDestino();
	LocalDate getFechaSalida();
	void setFechaSalida (LocalDate fsal);
	LocalDate getFechaLlegada ();
	void setFechaLlegada (LocalDate flleg);
	List<Pasajero> getPasajeros();
}
```

A su vez, el tipo **Pasajero** viene dado por la siguiente interfaz:

```
public interface Pasajero extends Persona {
	Integer getNumFila();           // Número de fila que ocupa el pasajero
	Character getLetraAsiento();    // Letra del asiento que ocupa el pasajero
	Boolean getMaletaFacturada();   // true si el pasajero factura una maleta
}
```

El tipo **Aeropuerto** tiene una propiedad *vuelos* que representa los vuelos que parten del aeropuerto. Su interfaz es:

```
public interface Aeropuerto {
	List<Vuelo> getVuelos();
	Vuelo vueloMasPasajerosDestino(String destino);
	Boolean todosVuelosDestinoMismaFecha(String destino);
	Integer numeroPasajerosConMaletaFacturada(LocalDate fecha);
}
```

Implemente los siguientes métodos de la clase `AeropuertoImpl`:

1. `Vuelo vueloMasPasajerosDestino(String destino);`

   Devuelve el vuelo con mayor número de pasajeros cuyo destino es el indicado. Si no hay vuelos con dicho destino, se lanzará la excepción `NoSuchElementException`.

2. `Boolean todosVuelosDestinoMismaFecha(String destino);`

   Devuelve `true` si todos los vuelos que tienen el destino indicado llegan a su destino el mismo día que salen, y `false` en caso contrario.

3. `Integer numeroPasajerosConMaletaFacturada(LocalDate fecha);`

   Devuelve el número de pasajeros que facturan maletas en los vuelos que llegan en la fecha indicada, o 0 si no llegan vuelos en dicha fecha.

Anexo: clases e interfaces de la API de Java

```
public interface Collection<E> extends Iterable<E> {
  boolean add(E e);
  boolean addAll(Collection<? extends E> c);
  void clear();
  boolean contains(Object o);
  boolean containsAll(Collection<?> c);
  boolean isEmpty();
  boolean remove(Object o);
  boolean removeAll(Collection<?> c);
  boolean retainAll(Collection<?> c);
  int size();
}

public interface List<E> extends Collection<E> {
  void add(int index, E element);
  boolean addAll(int index, Collection<? extends E> c);
  E get(int index);
  int indexOf(Object o);
  int lastIndexOf(Object o);
  E remove(int index);
  E set(int index, E element);
  List<E> subList(int fromIndex, int toIndex);
}
```

```
public interface SortedSet<E> extends Set<E> {
  SortedSet<E> headSet(E toElement);
  SortedSet<E> tailSet(E fromElement);
  SortedSet<E> subSet(E fromElement, E toElement);
  E first();
  E last();
}

public class String {
  char charAt (int index) {..}
  boolean contains (CharSequence s) {..}
  int indexOf (char c) {..}
  boolean isEmpty() {..}
  int length() {..}
  Boolean startsWith(String s) {..}
}

public class LocalDate {
  LocalDate now() {..}
  LocalDate of(int year, int month, int dayOfMonth) {..}
}
```