EJERCICIO 1 (3 puntos)

Se dispone de un tipo **Libro** con la siguiente interfaz:

```
public interface Libro extends Comparable<Libro> {
	String getCodigo();
	String getTitulo();
	String getAutor();
	Integer getNumeroPaginas();
	TipoPrestamo getTipoPrestamo();
	Set<String> getGeneros();
	void nuevoGenero(String genero);
}
```

La clase que implementa el tipo, **LibroImpl**, tiene los siguientes atributos:

```
private String codigo;                  // Código ISBN
private String titulo;                  // Título
private String autor;                   // Autor
private Integer numeroPaginas;          // Número de páginas
private TipoPrestamo tipoPrestamo;      // Tipo de préstamo: DIARIO, SEMANAL, MENSUAL
private Set<String> generos;            // Géneros del libro. Ej.: "Ficción", "Ensayo"…
```

Apartado a)

Escriba un constructor de la clase `LibroImpl` que permita construir un objeto de tipo Libro a partir de una representación del mismo en forma de cadena de caracteres. Ejemplo de formato de la cadena:

```
"978-0385536516# Dexter: El oscuro pasajero# Jeff Lindsay#288 #MENSUAL"
```

El número de páginas debe ser mayor que 0. Suponga definido un método `checkNumeroPaginas` que comprueba esta restricción y utilícelo donde proceda. En la representación como cadena pueden aparecer espacios en blanco entre los separadores y los valores de las propiedades.

Apartado b)

Se desea implementar una factoría de objetos de tipo Libro. Para ello, en una clase de utilidad `Libros`, implemente las siguientes propiedades poblacionales:

- Número de libros distintos creados (dos libros son iguales si tienen el mismo ISBN).
- Conjunto de todos los libros distintos creados.
- Códigos ISBN de todos los libros creados. La implementación de esta propiedad debe posibilitar el acceso a cualquier libro creado a partir de su ISBN.

Utilice los siguientes nombres para los métodos: `getNumLibrosCreados`, `getLibrosCreados`, `getCodigosLibrosCreados` y `getLibroCreado`. Este último obtiene un libro a partir de su ISBN, **sin usar bucles ni streams**.

En la misma clase `Libros`, implemente dos métodos creacionales para el tipo Libro: un método creacional copia y un método creacional a partir de `String`. Llame a ambos métodos `createLibro`. Suponga que la clase `LibroImpl` dispone de un constructor con parámetros que recibe valores para todas las propiedades del libro excepto los géneros.

EJERCICIO 2 (4 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 vueloMasPasajerosAnyo(Integer anyo);
	Boolean todosVuelosDestinoMismaFecha(String destino);
	Long numeroPasajerosConMaletaFacturada(LocalDate fecha);
	SortedMap<String, Vuelo> vueloConMasPasajerosPorDestino();
}
```

Implemente los siguientes métodos de la clase `AeropuertoImpl` **utilizando exclusivamente Java 8**:

a) `Vuelo vueloMasPasajerosAnyo(Integer anyo);`

Devuelve el vuelo con mayor número de pasajeros que salió en el año indicado. A igualdad de número de pasajeros, devuelve el que tiene fecha de salida anterior. Si no hay vuelos en dicho año, el método devolverá `null` (nota: el orden natural del tipo Vuelo viene dado por su código)

b) `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.

c) `Long 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.

d) `SortedMap<String, Vuelo> vueloConMasPasajerosPorDestino();`

Devuelve un `SortedMap` que relaciona cada destino con el vuelo a dicho destino con mayor número de pasajeros.

EJERCICIO 3 (3 puntos)

El tipo **Video** tiene las siguientes propiedades:

- **Nombre** del vídeo, cadena de caracteres que puede contener espacios en blanco y tiene un máximo de 255 caracteres.
- **Tamaño**, de tipo real.
- **Formato**, que puede tomar los valores AVI, MKV y WEBM.

a) Defina los siguientes tipos en C:
   - Un tipo `Cadena` capaz de almacenar un máximo de 255 caracteres (este valor podría cambiar en el futuro)
   - Un tipo enumerado `Formato` con los valores AVI, MKV y WEBM.
   - Un tipo struct `Video`.
   - Un tipo `PVideo` que representa un puntero a vídeo.
   - Un tipo `ArrayVideos` capaz de almacenar un máximo de 100 vídeos (este valor podría cambiar en el futuro)

b) Escriba una función que lea un vídeo del teclado. La función devolverá un valor 0 si el vídeo se ha leído correctamente, y -1 en caso contrario. La lectura puede fallar si el tamaño es menor o igual que 0 o el formato no es uno de los formatos admitidos. Puede usar una función auxiliar si lo considera necesario. Nota: no es necesario que utilice la función `fflush`. Un ejemplo de salida en pantalla de esta función puede ser el siguiente:

```
Nombre del vídeo: Juego de Tronos 5x03
Tamaño: 1.89
Formato: MKV
```

Realice un test en el cual declare una variable de tipo vídeo, escriba una llamada a la función anterior para leer dicha variable desde el teclado y muestre en pantalla el vídeo leído o un mensaje de error si no se ha leído correctamente. Suponga definida una función `muestraVideo` que recibe un vídeo y lo muestra en pantalla.

c) Escriba una función que, a partir de un array de vídeos y un formato de vídeo, devuelva el número de vídeos de dicho formato que hay en el array.

Realice un test en el cual declare un array de vídeos, lea varios vídeos desde el teclado y llame a la función anterior para obtener el número de vídeos en formato AVI que contiene el array. Para leer el array suponga definida una función `leeArrayVideos` que lee un conjunto de vídeos desde el teclado y devuelve el número de vídeos leídos.

Anexo: clases e interfaces de la API de Java

```
public interface Map<K,V> {
  V put(K key, V value);
  V get(Object key);
  boolean containsKey(Object key);
  Set<K> keySet();
  Collection<V> values();
  int size();
}

public class LocalDate {
  static LocalDate now();
  static LocalDate of(int year, int month, int dayOfMonth);
  int getYear();
}

public interface Comparator<T> {
  int compare(T o1, T o2);
  static <T,U extends Comparable<? super U>> Comparator<T> comparing(Function<? super T,
    ? extends U> keyExtractor);
  static <T extends Comparable<? super T>> Comparator<T> naturalOrder();
  default Comparator<T> reversed();
  default Comparator<T> thenComparing(Comparator<? super T> other);
}

public class Optional<T> {
  T get();
  boolean isPresent();
  static <T> Optional<T> of(T value);
  T orElse (T other);
}

public interface Stream<T> {
  T reduce(T identity, BinaryOperator<T> accumulator);
  Optional<T> reduce(BinaryOperator<T> accumulator);
  <U> U reduce(U identity, BiFunction<U,? super T,U> accumulator, BinaryOperator<U> combiner);
  <R> R collect(Supplier<R> resultFactory, BiConsumer<R,? super T> accumulator, BiConsumer<R,R> combiner);
  <R,A> R collect(Collector<? super T,A,R> collector);
  boolean allMatch(Predicate<? super T> predicate);
  boolean anyMatch(Predicate<? super T> predicate);
  boolean noneMatch(Predicate<? super T> predicate);
  Optional<T> min(Comparator<? super T> comparator);
  Optional<T> max(Comparator<? super T> comparator);
  long count();
  Optional<T> findFirst();
  Optional<T> findAny();
  void forEach(Consumer<? super T> action);
  Stream<T> filter(Predicate<? super T> predicate);
  <R> Stream<R> map(Function<? super T,? extends R> mapper);
  IntStream mapToInt(ToIntFunction<? super T> mapper);
  DoubleStream mapToDouble(ToDoubleFunction<? super T> mapper);
  <R> Stream<R> flatMap(Function<? super T, ? extends Stream<? extends R>> mapper);
  Stream<T> distinct();
  Stream<T> sorted();
  Stream<T> sorted(Comparator<? super T> comparator);
}
```