Ejercicio 1. Completar restricciones

En los boletines anteriores, se dejaron algunas restricciones por implementar. El objetivo de este ejercicio es completar los tipos con esas restricciones, para las que necesitamos implementar algunos tratamientos secuenciales. Las restricciones por implementar se muestran en la tabla siguiente:

DOMINIO	TIPO	RESTRICCIÓN
fp.musica	Album	**R1**: El Id está formado por 22 caracteres.
fp.musica	Artista	**R3**: El Id está formado por 22 caracteres.
fp.musica	Cancion	**R1**: El Id está formado por 22 caracteres.
fp.bibliotecas	Persona	**R1:** El DNI debe estar formado por 8 dígitos y una letra. La letra debe ser la que se obtiene con según lo establecido en el artículo 11 del RC 1553/2005.
fp.universidades	Beca	**R1:** El código está formado por tres letras y cuatro dígitos.
fp.universidades	Persona	**R1:** El DNI debe estar formado por 8 dígitos y una letra. La letra debe ser la que se obtiene con según lo establecido en el artículo 11 del RC 1553/2005.

Como puede comprobar todas estas restricciones tienen bastantes similitudes. Para reutilizar código, antes de añadir las restricciones, implemente en una clase de utilidad llamada **Validadores** y situada en el paquete **fp.utiles** los siguientes métodos ***static***:

- ***Boolean sonDigitos (String cadena)***. Devuelve *true* si todos los caracteres de la cadena son dígitos.
- ***Boolean sonLetras (String cadena)***. Devuelve *true* si todos los caracteres de la cadena son letras.
- ***Boolean estaEnBase62 (String cadena)***. Devuelve *true* si todos los caracteres de la cadena son dígitos ([0-9]), letras en minúsculas ([a-z], sin incluir la ñ) o letras en mayúsculas ([A-Z], sin incluir la Ñ).
- ***Boolean esDNIValido(String dni)***[1].Devuelve *true* si la cadena dni que se pasa como parámetro tiene el formato de dni adecuado, es decir, debe estar formado por 8 dígitos, una letra y la letra se debe corresponder a la establecida en el artículo 11 del RC 1553/2005.

Implemente luego las restricciones haciendo uso de estos métodos y de los métodos de la clase ***Checkers***.

Ejercicio 2. Ampliar ListaReproduccion

Se quiere ampliar el tipo **ListaReproduccion** definido en el ejercicio 28 del boletín anterior con las siguientes operaciones:

- ***Boolean esAntologia(String artista)***. Devuelve *true* si todas las canciones de la lista de reproducción son del artista especificado.
- ***Duration getDuracion()***. Devuelve la duración total de la lista de reproducción.
- ***Boolean contieneArtista(String artista)***. Devuelve *true* si existe alguna canción en la lista de reproducción del artista dado como parámetro.
- ***ListaReproduccion getSublistaArtista(String artista)***. Devuelve una nueva lista de reproducción que contenga todas las canciones del artista dado como parámetro.
- ***int getPosicionCancion(String tituloCancion)***. Devuelve la posición (índice) que ocupa en la lista de reproducción la primera canción cuyo título se especifica. Si la canción no está en la lista de reproducción, devuelve -1.

[1] Este método no requiere de un tratamiento secuencial para resolverlo.

- ***Cancion getCancionMasLarga()***. Devuelve la canción con mayor duración de la lista de reproducción. Si no hay ninguna canción en la lista, devuelve null.
- ***Cancion getCancionMasCorta()***. Devuelve la canción con menor duración de la lista de reproducción. Si no hay ninguna canción en la lista, devuelve null.
- ***Set<Artista> getArtistas()***. Devuelve el conjunto de artistas de la lista de reproducción.
- ***void muestraFotosArtistas()***. Muestra la primera foto de cada artista de la lista, para aquellos artistas que tengan alguna foto. Utilice el método funcional `public static void show(String titulo, String url)` de la clase `Imagenes`.

Se pide modificar la interfaz **ListaReproduccion** y la clase **ListaReproduccionImpl** para que el tipo proporcione las nuevas operaciones.

Ejercicio 3. Ampliar Aeropuerto

Se quiere ampliar el tipo **Aeropuerto** definido en el ejercicio 31 del boletín anterior con las siguientes operaciones nuevas:

- ***Set<Vuelo> seleccionaVuelosFecha(LocalDate fechaSalida)***. Devuelve un conjunto con todos los vuelos del aeropuerto que salen en la fecha dada como parámetro[2]. Si el aeropuerto no tiene vuelos con tal fecha de salida devuelve un conjunto vacío.
- ***Vuelo getVueloMasPasajeros()***. Devuelve el vuelo del aeropuerto con más pasajeros. Si el aeropuerto no tiene vuelos, se eleva la excepción *NoSuchElementException*.
- ***Persona getPasajeroMayor()***. Devuelve el pasajero con más edad de todos los vuelos del aeropuerto. Si el aeropuerto no tiene vuelos eleva la excepción *NoSuchElementException*.
- ***Vuelo getVueloPlazasLibresDestino(String destino)***. Devuelve un vuelo cualquiera (el primero que encuentre) con plazas libres al destino dado como parámetro. Si no hay ningún vuelo con plazas libres a ese destino eleva la excepción *NoSuchElementException*.
- ***Integer calculaTotalPasajerosDestino(String destino)***. Devuelve el número total de pasajeros que han volado a la ciudad destino dada como parámetro. Si el aeropuerto no tiene vuelos dicho destino devuelve cero.
- ***Double calculaMediaPasajerosPorDia()***. Devuelve el número medio de pasajeros que recibe cada día el aeropuerto. Si el aeropuerto no recibe vuelos devuelve cero.

Se pide:

a) Modifique la interfaz Aeropuerto y la clase AeropuertoImpl para que el tipo proporcione las nuevas operaciones.
b) Implemente la restricción **R1** (todos los vuelos de un aeropuerto deben tener su ciudad como origen o destino) del tipo dada en el ejercicio 31 del boletín anterior aplicando un esquema de tratamiento secuencial.

Ejercicio 4. Ampliar Biblioteca

Se desea ampliar el tipo **Biblioteca** del Ejercicio 32 del Bloque 4 con operaciones adicionales para trabajar con tratamientos secuenciales:

- ***Integer cuentaPrestamos(Persona usuario)***. Devuelve el número de préstamos que tiene el usuario dado como parámetro. Lanza IllegalArgumentException si el usuario dado como parámetro no pertenece a la biblioteca.
- ***Integer cuentaPrestamos(Libro libro)***. Devuelve el número de préstamos que tiene el libro dado como parámetro. Lanza IllegalArgumentException si el libro dado como parámetro no pertenece a la biblioteca.
- ***Integer cuentaPrestamos(Month mes)***. Devuelve el número de préstamos que se han realizado en el mes dado como parámetro.

[2] Consulte en la documentación de java el uso del método toLocalDate del tipo LocalDateTime para implementar este método.

- ***Integer [] cuentaPrestamosPorMes ().*** Un array de 12 elementos que tiene el número de préstamos realizados en cada mes. Así en la posición 0 del array estarán el número de préstamos de Enero, en la 1 el de Febrero.
- ***Month getMesConMasPrestamos()***. Devuelve el mes en el que se han realizado más préstamos. Si en ningún mes se han prestado libros, se devuelve el mes de Enero.
- ***List<Libro> seleccionaLibrosSinPrestamos()***. Devuelve una lista con los libros de la biblioteca que no se han prestado nunca.
- ***Set<Persona> seleccionaUsuariosSinPrestamos()***. Devuelve un conjunto con los usuarios que no han solicitado nunca un préstamo.
- ***Boolean tienenTodosLosUsuariosPrestamo()***. Devuelve cierto si todos los usuarios de la biblioteca tienen al menos un préstamo.

Se pide:

a) Modifique la clase BibliotecaImpl para implementar las nuevas operaciones.
b) Complete la clase TestBiblioteca para probar las nuevas operaciones.

Ejercicio 5. Ampliar Cine

Se desea ampliar el tipo **Cine** del Ejercicio 34 del Bloque 4 con operaciones adicionales para trabajar con tratamientos secuenciales. Otras operaciones a añadir:

- ***Boolean estaSalaOcupada(Integer numeroSala, LocalDate fecha, LocalTime hora)***: Devuelve true si la sala que se proporciona como parámetro proyecta una película en la fecha y hora dados como parámetros.
- ***Set<Pelicula> getCartelera(LocalDate fecha)***: Devuelve el conjunto de películas que se emiten en la fecha dada como parámetro en el cine.
- ***LocalTime getSesionMasTemprano(DayOfWeek dia, String pelicula)***: Devuelve la hora de comienzo de la sesión que comienza más temprano de la película cuyo título se da como parámetro y para el día de la semana dado. Si no hay ninguna sesión de la película ese día de la semana, devuelve null.

Se pide:

a) Modifique la interfaz Cine y la clase CineImpl para implementar las nuevas operaciones
b) Complete la clase TestCine para probar las nuevas operaciones

Ejercicio 6. NubePuntos

Dada el tipo Punto con las operaciones y métodos definidos en la interfaz Punto que se define a continuación, implemente el tipo **NubePuntos** en el paquete **fp.geometria**.

```
public interface Punto extends Comparable<Punto>{
   Double getX();
   Double getY();
   void setX(Double nuevaX);
   void setY(Double nuevaY);
   Double getDistancia (Punto p);
   Cuadrante getCuadrante();

}
```

NubePuntos
Propiedades:

- **puntos:** List<Punto>. Básica, consultable.
- **Número de puntos**: Integer. Derivada. Representa el número de puntos de la nube.

Constructores:

- **C1:** Constructor sin parámetros con el que se crea una nube de puntos vacía.
- **C2:** Constructor con una colección de puntos como parámetro que crea una nube de punto que contiene los puntos de esa colección.
- **C3:** Constructor copia, que toma como parámetro otra nube de puntos y crea una copia de la misma.

Restricciones:
- **R1:** La lista de puntos no puede ser nula y tampoco puede contener ningún punto nulo.

Representación como cadena:
- Entre corchetes, los puntos de la nube. Por ejemplo, [(2.0, 1.0), (-1.0, 0.0)]

Criterio de igualdad:
- Dos nubes de puntos son iguales si tienen los mismos puntos y en la misma posición.

Otras operaciones:
- ***void incorporaPunto (Punto p)*:** Incorpora el punto p a la nube de puntos, siempre que no sea nulo.
- ***void incorporaPuntos (Collection<Punto> colPuntos)*:** Incorpora la colección de puntos dada como parámetro a la nube, siempre que la colección no sea nula.
- ***void incorporaPuntos (NubePuntos nube):*** Incorpora los puntos de la nube dada como parámetro a la nube que invoca a la operación, siempre que la nube de puntos no sea nula.
- ***void eliminaPunto (Punto p):*** Elimina el punto dado como parámetro de la nube de puntos. Si no está en la nube o es nulo la operación no tiene efecto.
- ***Double calculaMediaX()***: Devuelve la media de las coordenadas X de los puntos de la nube. Si no hay ningún punto en la nube, se eleva IllegalArgumentException.
- ***Double calculaMediaY()***: Devuelve la media de las coordenadas Y de los puntos de la nube. Si no hay ningún punto en la nube, se eleva IllegalArgumentException.
- ***Boolean hayAlgunPuntoEnEjeX()***: Devuelve cierto si existe algún punto de la nube que esté en el eje X.
- ***Boolean estanTodosADistanciaMenorQue(Double distancia)***: Devuelve cierto si todos los puntos de la nube están a una distancia del origen de coordenadas menor que d.
- ***Integer cuentaPuntosConXMayorA(Double umbral)***: Devuelve el número de puntos de la nube con coordenada X mayor que el umbral dado como parámetro.
- ***Integer [] cuentaPuntosPorCuadrante():*** Devuelve un array de cinco elementos que contendrá el número de puntos de la nube que hay en cada cuadrante y en los ejes.
- ***NubePuntos seleccionaPuntos (Double distancia):*** Devuelve una NubePuntos con aquellos puntos que estén a una distancia del origen menor que la dada como parámetro.
- ***Punto getPuntoMasDistante():*** Devuelve el punto de la nube que está más lejos del origen de coordenadas. Si no hay puntos en la nube, devuelve null.
- ***Punto getPuntoMasCercano(Punto p):*** Devuelve el punto de la nube cuya distancia al punto p dado como parámetro es menor. Si no hay puntos en la nube, devuelve null.
- ***void aplicaSimetriaEjeX():*** Le cambia el signo a la coordenada X de todos los puntos de la nube.
- ***Double sumaCoordenadasXPrimerCuadrante():*** Suma las coordenadas X de los puntos del primer cuadrante.
- **Double multiplicaSumaCoordenadasXeY (Cuadrante c):** Calcula el producto de la suma de las coordenadas X e Y de los puntos del cuadrante dado como parámetro.
- **List<Punto> getPuntosBisectriz():** Devuelve los puntos situados en la bisectriz.
- **int getIndicePunto(Cuadrante c):** Devuelve el índice del primer punto situado en el cuadrante dado como parámetro.

Se pide:

a) Implementar la interfaz **NubePuntos** y la clase **NubePuntosImpl**.
b) Escribir una clase **TestNubePuntos** para probar las operaciones del tipo.

Ejercicio 7. GestorApuestas

Se quiere ampliar el ejercicio 11 del boletín 2 con la introducción de un nuevo tipo llamado GestorApuestas, con las siguientes características:

Propiedades:
- **Id**, de tipo String, consultable. Representa el id del gestor de apuestas
- **Apuestas**, de tipo SortedSet<Apuesta>, consultable. Es el conjunto de apuestas

Constructores:
- **C1:** recibe el *id* como parámetro, e inicializa las apuestas con un conjunto ordenado vacío.

Criterio de igualdad:
- Dos gestores de apuestas son iguales si tienen el mismo id.

Representación como cadena:
- El *id*, seguido de una coma y de la representación como cadena de las apuestas, todo entre corchetes.

Operaciones:
- ***void nuevaApuesta(Apuesta a)***. Añade la apuesta dada como parámetro al conjunto de apuestas. La apuesta solo se añade si el partido para el que se hace la apuesta aún no se ha jugado. En caso contrario, se lanzará IllegalArgumentException.
- ***void eliminaApuesta(Apuesta a)***. Elimina la apuesta dada como parámetro al conjunto de apuestas. La apuesta solo se elimina si el partido para el que se hace la apuesta aún no se ha jugado. En caso contrario, se lanzará IllegalArgumentException.
- ***Double calculaRecaudacion(PartidoFutbol partido)***. Devuelve la cantidad total de dinero apostado en el partido dado como parámetro.
- ***Double calculaRecaudacionGanadores(PartidoFutbol partido)***. Devuelve la cantidad total de dinero apostado en el partido dado como parámetro solo en las apuestas ganadoras.
- ***Double getGanancias(Apuesta apuesta)***. Devuelve las ganancias de la apuesta dada como parámetro. Para ello aplique la siguiente fórmula, siempre que haya recaudación de apuestas ganadoras:

  dinero a repartir=recaudación del partido objeto de la apuesta/2
  ganancia=(dinero a repartir*cantidad apostada)/recaudación de apuestas ganadoras para el partido objeto de la apuesta.

  Si no hay apuestas ganadoras, no habrá ganancias (el método devuelve 0).
- ***Set<String> seleccionaUsuariosGanadores(PartidoFutbol partido)***. Devuelve un conjunto con los id de los usuarios que han acertado el resultado del partido dado como parámetro.
- ***String getIdMaximoAcertante()***. Devuelve el id del usuario con más apuestas ganadoras. Si no hay ningún partido con apuestas ganadoras se elevará NoSuchElementException.
- ***Apuesta getApuestaMaximoBeneficio()***.Devuelve la apuesta que ha obtenido más beneficios. Si no hay ninguna, devuelve null.
- ***Double calculaTotalApostado(String idUsuario)***. Devuelve el total de dinero que ha apostado el usuario cuyo id es el dado como parámetro.
- ***PartidoFutbol getPartidoMasApuestas()***. Devuelve el partido de fútbol para el que se han hecho más apuestas. Si no hay ninguno devuelve null.
- ***PartidoFutbol getPartidoMasAcertantes()***. Devuelve el partido para que ha habido más apuestas ganadoras. Si no hay ninguno devuelve null.

Se pide implementar en **fp.apuestas**, la interfaz **GestorApuestas** y la clase **GestorApuestasImpl**. Implemente también en **fp.apuestas.test** una clase **TestApuestas** para probar las operaciones del tipo.

Ejercicio 8. GestorPeliculasFavoritas

Se quiere añadir un nuevo tipo ***GestorPeliculasFavoritas***, a partir del tipo ***Pelicula*** definido en boletines anteriores con las siguientes propiedades y operaciones:

Propiedades:
- **Id**, de tipo String, consultable. Representa el id del gestor de películas.
- **Nombre**, de tipo String, consultable. Representa el nombre que se le da a la lista de películas favoritas.
- **Películas**, de tipo Set<Pelicula>, consultable. Es el conjunto de películas favoritas del usuario.

Constructores:
- **C1:** Recibe un parámetro por cada propiedad básica del tipo.
- **C2:** Recibe un parámetro por cada propiedad básica del tipo, excepto el conjunto de películas, que se inicializará como un conjunto vacío.

Criterio de igualdad:
- Dos gestores de películas son iguales si tienen el mismo id.

Representación como cadena:
- El nombre, seguido del id entre paréntesis. Por ejemplo, “Películas de FP (1234)”.

Operaciones:

- ***void nuevaPelicula(Pelicula pelicula)***: Añade la película dada como parámetro al conjunto de películas del gestor. Si el parámetro es nulo eleva la excepción IllegalArgumentException.
- ***void nuevasPeliculas(Collection<Pelicula> peliculas)***: Añade la colección de películas dada como parámetro al conjunto de películas del gestor. Si el parámetro es nulo eleva la excepción IllegalArgumentException.
- ***void eliminaPelicula(Pelicula pelicula)***: Elimina la película dada como parámetro del conjunto de películas del gestor. Si el parámetro es nulo eleva la excepción IllegalArgumentException.
- ***Integer cuentaPeliculas(String nombreActor)***: Proporciona el número de películas en las que ha participado la actriz o el actor cuyo nombre se da como parámetro. Si el parámetro es nulo eleva la excepción IllegalArgumentException.
- ***MiembroStaff getActorMasPeliculas()***: Proporciona el actor que ha participado en más películas entre las favoritas del gestor. Si no hay películas devuelve ***null***.
- ***Boolean hayAlgunaPeliculaDirigidaPor(String nombreDirector)***: Proporciona un valor lógico que indica si existe alguna película entre las películas del gestor que haya sido dirigida o codirigida por el director cuyo nombre se da como parámetro. Si el parámetro es nulo eleva la excepción IllegalArgumentException.
- ***Pelicula getPeliculaMasActores()***: Proporciona la película en la que han participado más actores.
- ***Set<MiembroStaff> seleccionaActoresParticipantesTodas()***: Proporciona el conjunto de actores que han participado en todas las películas del gestor. Si no hay ninguna actriz o actor que haya participado en todas las películas devuelve el conjunto vacío.
- ***Set<Pelicula> getPeliculasDirigidasPor(String nombreDirector)***: Proporciona el conjunto de películas dirigidas o codirigidas por el director cuyo nombre se da como parámetro. Si no hay ningún director que haya participado en alguna de las películas devuelve el conjunto vacío. Si el parámetro es nulo eleva la excepción IllegalArgumentException.
- ***Set<String> getGeneros()***: Proporciona el conjunto de géneros de todas las películas del gestor. Si no hay películas devuelve el conjunto vacío.
- ***Set<Pelicula> getPeliculasAnyo(Integer anyo)***: Proporciona el conjunto de películas estrenadas el año dado como parámetro. Si no hay ninguna película con dicho año en su fecha de estreno devuelve el conjunto vacío. Si el parámetro es nulo eleva la excepción IllegalArgumentException.
- ***Set<Pelicula> getPeliculasDeActor(String nombreActor)***: Proporciona el conjunto de películas en las que ha participado la actriz o el actor cuyo nombre se da como parámetro. Si no hay ninguna película con dicho intérprete devuelve el conjunto vacío. Si el parámetro es nulo eleva la excepción IllegalArgumentException.
- ***Set<String> getPaises()***: Proporciona el conjunto de países de todas las películas del gestor. Si no hay películas devuelve el conjunto vacío.

Para facilitar la implementación de algunas de las operaciones anteriores amplíe el tipo ***Pelicula*** definido en boletines anteriores con las dos siguientes **operaciones**:

- ***Boolean esActor(String nombreActor):*** Proporciona un valor lógico que indica si existe alguna actriz o actor con el nombre dado como parámetro que haya participado en la película. Si el parámetro es nulo eleva la excepción IllegalArgumentException.
- ***Boolean esDirector(String nombreDirector):*** Proporciona un valor lógico que indica si existe algún director con el nombre dado como parámetro que haya dirigido o codirigido la película. Si el parámetro es nulo eleva la excepción IllegalArgumentException.

Se pide:

a) Modifique la interfaz Pelicula y la clase PeliculaImpl para que el tipo proporcione las nuevas propiedades.
b) Implemente la interfaz GestorPeliculasFavoritas y la clase GestorPeliculasFavoritasImpl con las propiedades y operaciones descritas.
c) Implemente una clase TestGestorPeliculasFavoritas para probar las operaciones del tipo.

Ejercicio 9. GestorSeriesFavoritas

Se quiere añadir un nuevo tipo ***GestorSeriesFavoritas***, a partir del tipo ***SerieTV*** definido en boletines anteriores con las siguientes propiedades y operaciones:

Propiedades:
- **Id**, de tipo String, consultable. Representa el id del gestor de series.
- **Nombre**, de tipo String, consultable. Representa el nombre que se le da a la lista de series favoritas.
- **SeriesTV**, de tipo Set<SerieTV>, consultable. Es el conjunto de películas favoritas del usuario.

Constructores:
- **C1:** Recibe un parámetro por cada propiedad básica del tipo.
- **C2:** Recibe un parámetro por cada propiedad básica del tipo, excepto el conjunto de series, que se inicializará como un conjunto vacío.

Criterio de igualdad:
- Dos gestores de series son iguales si tienen el mismo id.

Representación como cadena:
- El nombre, seguido del id entre paréntesis. Por ejemplo, “Series de Informatica (1234)”.

Operaciones:

- ***void nuevaSerie(SerieTV serie)***: Añade la serie dada como parámetro al conjunto de series del gestor. Si el parámetro es nulo la operación no tiene efecto.
- ***void nuevasSeries(Collection<SerieTV> series)***: Añade la colección de series dada como parámetro al conjunto de películas del gestor. Si el parámetro es nulo la operación no tiene efecto. Si la colección tiene algún elemento nulo, éste no se añadirá a las series del gestor.
- ***void eliminaSerie(SerieTV serie)***: Elimina la serie dada como parámetro. Si la serie no está en el gestor, la operación no tiene efecto.
- ***void eliminaSerie(SerieTV serie)***: Elimina la serie con el nombre dado como parámetro. Si la serie no está en el gestor, la operación no tiene efecto.
- ***SerieTV getSerieMasTemporadas ()***: Devuelve la serie del gestor que tiene más temporadas. Si no hay ninguna serie en el gestor, devuelve null.
- ***SerieTV getSerieMasEpisodios()***: Devuelve la serie del gestor que tiene más episodios. Si no hay ninguna serie en el gestor, devuelve *null*.

- ***Set<SerieTV> getSeriesDe(MiembroStaff persona)***: Devuelve el conjunto de series en las que ha trabajado en cualquier rol (creador, actor o arstista invitado) la persona dada como parámetro.
- ***Integer[] contarSeriesSegunEstado()***: Devuelve un *array* con el número de series con estado FINALIZADA, EN_CURSO y EN_PRODUCCION del conjunto de series favoritas.
- ***Boolean haySeriesDeCadenaTV(String cadenaTV)***: Devuelve cierto si hay alguna serie de la cadena de televisión pasada como parámetro.
- ***SerieTV getSerieMasPopular(String genero)***: Devuelve la serie más popular del género dado como parámetro. Si no hay ninguna serie de ese género se eleva NoSuchElementException.
- ***Boolean seEmitieronTodasEn(Integer anyo)***: Devuelve cierto si todas las series del gestor se emitieron en el año dado como parámetro. Se considera que una serie se emitió en un año si alguno de sus episodios se ha emitido en ese año.

Para facilitar la implementación de algunas de las operaciones anteriores amplíe el tipo ***SerieTV*** definido en boletines anteriores con las siguientes **operaciones**:

- ***Boolean esCreador(MiembroStaff persona):*** Devuelve cierto si la persona dada como parámetro forma parte del equipo de creadores de la serie.
- ***Boolean esActor(MiembroStaff persona):*** Devuelve cierto si la persona dada como parámetro ha trabajado como actor o actriz en alguno de los episodios de la serie.
- ***Boolean esArtistaInvitado(MiembroStaff persona):*** Devuelve cierto si la persona dada como parámetro ha trabajado como artista invitado en alguno de los episodios de la serie.

Se pide:

a) Modifique la interfaz SerieTV y la clase SerieTVImpl para que el tipo proporcione las nuevas operaciones.
b) Implemente la interfaz GestorSeriesFavoritas y la clase GestorSeriesFavoritasImpl con las propiedades y operaciones descritas.
c) Implemente una clase TestGestorSeriesFavoritas para probar las operaciones del tipo.