PROYECTO DE PROGRAMACIÓN ORIENTADO A OBJETOS AVANZADA – VERGEFLIX

Descripción del proyecto

En este proyecto tendrás que desarrollar una **aplicación Java en modo consola que permita realizar la gestión de un catálogo audiovisual (con películas, series, etc.)** empleando los fundamentos de la Programación Orientada a Objetos y de los elementos avanzados como herencia y polimorfismo, entre otros. La descripción completa y detallada de la implementación de este proyecto se encuentra en las siguientes secciones.

Debes aplicar todas las buenas prácticas aprendidas, entre otras:

- Implementar los métodos de las clases realizando control de errores.
- Implementar los constructores y métodos de la clase reutilizando todo el código posible.
- Implementar las clases sin modificar la interfaz pública de la misma.
- Validar todos los datos de entrada, tanto los introducidos por teclado por el usuario como los parámetros de las funciones.
- Evitar código duplicado y reutilizar código mediante funciones.
- Nombrar las variables, clases y métodos de acuerdo a las convenciones del lenguaje, y usar identificadores descriptivos y representativos de lo que almacena o realiza la variable/función.
- Documentar las clases con comentarios Javadoc.
- Seguir las especificaciones descritas en este guion sin cambiarlas, salvo justificación y previa consulta con el profesor.

Importante: No se corregirá ningún proyecto en el que haya errores de compilación.

La tarea se califica entre 0 y 10 puntos desglosados de la siguiente manera:

- Clase ***Media*** (0,5 puntos).
- Clase ***Pelicula*** (0,5 puntos).
- Clase ***Serie*** (2 puntos).
- Clase ***Temporada*** (2 puntos).
- Clase ***Capítulo*** (0,5 punto).
- Interfaz ***Valorable*** (0,5 puntos).
- Programa principal ***Vergeflix*** (4 puntos).

Debes crear el proyecto Netbeans ***Vergeflix*** que implementará la gestión de un catálogo audiovisual. El programa principal mostrará un menú con las siguientes opciones:

1. Gestionar películas.
2. Gestionar series.
3. Consultar catálogo.
4. Salir.

Especificación del diseño de las clases

A continuación se detalla el diseño de cada una de las clases de la aplicación.

La clase Media (0,5 puntos)

Esta clase será declarada como **abstracta**. Esta clase modela un elemento del catálogo, de ella heredarán las clases *Pelicula* y *Serie*.

- **Atributos privados**:
    - ***String nombre***. Nombre del elemento del catálogo.
    - ***int calificacionEdad.*** Edad mínima para la que está recomendada. Será un valor entre 0 y 18.
    - ***LocalDate fechaIncorporacionAlCatalogo***. Fecha en la que el elemento catalogable *Media* se incorpora a nuestro catálogo.
    - ***boolean estaDisponible.*** Indica si el elemento está disponible para visualizarse o no.

- Se implementarán los siguientes **constructores y métodos públicos**:

    - **(0,1 puntos)** Constructor con los parámetros del objeto. Si alguno de los parámetros no es válido se lanzará la excepción `IllegalArgumentException`.
      ```
      public Media(String nombre, int calificacionEdad, LocalDate
              fechaIncorporacion, boolean estaDisponible);
      ```
    - **(0,1 puntos)** La clase debe sobrescribir el método ***toString()***.
    - **(0,2 puntos)** Métodos ***getXXX()*** y ***setXXX()***. Se creará un método *get/set* para cada uno de los atributos.
    - **(0,1 puntos)** La clase debe implementar los métodos ***equals()*** y ***compareTo()*** (interfaz ***Comparable).*** *C*onsideraremos que dos elementos Media son iguales si sus atributos *nombre* y *calificacionEdad* lo son. El método ***compareTo()*** se basará en el orden alfabético del nombre, a igualdad de nombre el menor será el que tenga la calificación por edad más baja.

La clase Pelicula (0,5 puntos)

La clase *Pelicula* hereda de *Media* y se compone de director, actor principal, duración y temática (además de los atributos ya heredados).

Se debe tener en cuenta que la duración debe ser mayor a 60 minutos y no exceder las 4 horas. Por otro lado la ***Tematica*** se implementará como un ***Enum*** con los siguientes valores: (COMEDIA, DRAMA, TERROR, SUSPENSE, CIENCIA_FICCION, FANTASIA).

tendrá los siguientes atributos y métodos.

- Atributos privados:
    - ***String director***. Nombre del director de la película.
    - ***String actorPrincipal***. Nombre del actor principal de la película.

- ***int duracion.*** Total de minutos que dura la película. Deberá ser un valor mayor a 60 minutos y no exceder las 4 horas.
- ***Tematica categoria***. Variable que contendrá la temática de la película.
- ***ArrayList<Integer> votos***. Lista de votos del 0 al 10 de la película.

- Métodos públicos:
  - **(0,1 puntos)** ***public Pelicula(...)***. Constructor que tendrá los parámetros correspondientes a *Media* más los propios de la clase *Pelicula*, excepto el ArrayList de votos.
  - **(0,1 puntos)** Implementa además el **constructor de copia**.
  - **(0,0 puntos)** Se debe sobrescribir el método *toString()*.
  - **(0,1 puntos)** Métodos ***getXXX()***. Se implementarán los métodos *get* para todos los atributos privados, excepto para el ArrayList de votos.
  - **(0,1 puntos)** Métodos ***setXXX()***. Se implementará los métodos *set* para todos los atributos. excepto para el ArrayList de votos.
  - **(0,1 puntos)** ***boolean votar(int voto).*** El método añade un voto a la lista de votos. Se debe comprobar que el voto es de cero a 10. Si se ha podido añadir el voto se devuelve true.

La clase Capitulo (0,5 puntos)

La clase *Capitulo* modela un capítulo de una serie y tendrá los siguientes atributos y métodos.

- Atributos privados:
  - ***int votosPositivos.*** Cantidad de votos positivos obtenidos por el capítulo.
  - ***int votosNegativos.*** Cantidad de votos negativos obtenidos por el capítulo.
  - ***String titulo.*** Nombre del capítulo.
  - ***LocalDate fechaEmisión.*** Fecha de emisión.

- Métodos públicos:
  - **(0,1 puntos)** ***public Capitulo(...)***. Constructor que tomará por parámetro un nombre para el capítulo y una fecha de emisión e inicializará sus atributos.
  - **(0,1 puntos)** ***Constructor de copia***. Construirá un capítulo que será una copia exacta del capítulo pasado por parámetro.
  - **(0 puntos)** Se debe sobrescribir el método *toString()*.
  - **(0,1 puntos)** Métodos ***getXXX()***. Se implementarán los métodos *get* para todos los atributos privados.
  - **(0,1 puntos)** Métodos **setXXX()**. Se implementará los métodos *set* para el título y la fecha de emisión.
  - **(0,1 puntos)** ***void meGusta(boolean like)***. El método añade un voto positivo o negativo al capítulo dependiendo del parámetro *like*.

La clase Temporada (2 puntos)

La clase *Temporada* modela una temporada de una serie, por lo que tendrá una lista de capítulos y una fecha de estreno:

- Atributos privados:

- ***LocalDate fechaEstreno.*** Fecha de estreno de la temporada.
- ***ArrayList<Capitulo> capitulos.*** Lista de capítulos de la temporada.

- Métodos públicos:
  - **(0,1 puntos)** ***public Temporada(...)***. Constructor que tomará por parámetro la fecha de estreno e inicializará sus atributos (se construirá una lista de capítulos vacía).
  - **(0,05 puntos)** Se debe sobrescribir el método ***toString()***.
  - **(0,05 puntos)** ***LocalDate getFechaEstreno().*** Devolverá la fecha de estreno.
  - **(0,1 puntos)** ***boolean añadirCapitulo(Capitulo capitulo).*** Este método recibirá un capítulo y le hará una copia que añadirá a la temporada sólo en el caso que no esté ya y que su fecha de emisión sea posterior a la fecha de estreno de la temporada. El método devolverá *true* si se ha podido añadir y false en caso contrario.
  - **(0,1 puntos)** ***boolean añadirCapitulo(LocalDate fechaEmision, String titulo).*** Recibirá una fecha de emisión y un título. Con esa información creará un capítulo que añadirá al final de la lista. Se debe controlar que el capítulo no esté ya en la temporada (es decir, no exista otro capítulo con el mismo nombre). Además la fecha de emisión del capítulo deberá ser posterior a la fecha de estreno de la temporada. El método devolverá *true* si se ha podido añadir y *false* en caso contrario.
  - **(0,1 puntos)** ***boolean añadirCapitulo(int posicion, Capitulo capitulo).*** Este método recibirá un capítulo y le hará una copia que añadirá a la temporada en la posición indicada por parámetro. Se debe comprobar que se puede añadir, es decir, que la posición está dentro de los límites y, como en casos anteriores, que el capítulo no esté repetido y la fecha de emisión sea posterior a la fecha de estreno de la temporada. El método devolverá *true* si se ha podido añadir y *false* en caso contrario.
  - **(0,1 puntos)** ***boolean añadirCapitulo(int posicion, LocalDate fechaEmision, String titulo).*** Recibirá una posición, fecha de emisión y un título. Con esa información creará un capítulo que añadirá en la posición indicada. El método devolverá *true* si se ha podido añadir y *false* en caso contrario.
  - **(0,1 puntos)** ***boolean eliminarCapitulo(String titulo).*** Recibirá un título y eliminará el capítulo que contenga ese título. Devolverá *true* si ha eliminado el capítulo.
  - **(0,2 puntos)** ***int eliminarCapítulos(String expresionRegular).*** Recibirá una expresión regular y eliminará todos los capítulos cuyo nombre concuerde con esa expresión regular. Este método devolverá la cantidad de capítulos eliminados.
  - **(0,1 puntos)** ***Capitulo getCapitulo(int posicion).*** Devolverá una copia del capítulo que se encuentra en la posición indicada.
  - **(0,1 puntos)** ***boolean setCapitulo(int posicion, LocalDate fechaEmision, String titulo).*** Modificará, si es posible, el capítulo situado en “posicion” con los datos pasados por parámetro, Si ha sido posible modificarlo devolverá *true*, si no *false*.
  - **(0,1 puntos)** ***boolean setCapitulo(int posicion, LocalDate fechaEmision).*** Modificará, si es posible, el capítulo situado en “posicion” con los datos pasados por parámetro, Si ha sido posible modificarlo devolverá *true*, sino *false*.
  - **(0,1 puntos)** ***boolean setCapitulo(int posicion, String titulo).*** Modificará, si es posible, el capítulo situado en “posicion” con los datos pasados por parámetro, Si ha sido posible modificarlo devolverá true sino false.

- **(0,1 puntos)** ***boolean meGusta(int posicionCapitulo, boolean like).*** Añade si es posible un voto positivo o negativo (dependiendo del booleano like) al capítulo situado en esa posición. Devuelve true si ha sido posible añadir el voto (el capítulo existe) false en caso contrario.
- **(0,1 puntos)** ***boolean meGusta(String titulo, boolean like).*** Añade si es posible un voto positivo o negativo (dependiendo del booleano like) al capítulo con el nombre título. Devuelve true si ha sido posible añadir el voto (el capítulo existe) false en caso contrario.
- **(0,2 puntos)** ***boolean setFechaEstreno(LocalDate fecha).*** Este método cambiará la fecha de estreno de la temporada. Deberá comprobar que es anterior a la fecha de emisión de cada capítulo. Finalmente devolverá *true* si se ha podido cambiar o *false* en caso contrario.
- **(0,3 puntos)** ***ArrayList<Capitulo> capitulosMejorValorados(int n).*** Este método devolverá una lista con los *n* capítulos mejor valorados. Esta lista contendrá copias de capítulos, no referencias. Además, deberán estar ordenadas de mejor a peor.

La clase Serie (2 puntos)

La clase *Serie* modelará una serie de TV. Esta clase heredará de la clase *Media* y tendrá como atributos una lista de temporadas (además de los atributos heredados de *Media*) y una fecha de estreno.

- Atributos privados:
  - ***LocalDate fechaEstreno.*** Fecha de estreno de la serie.
  - ***ArrayList<Temporada> temporadas.*** Lista de temporadas de la serie.

- Métodos públicos:
  - **(0,1 puntos)** ***public Serie(...)***. Constructor que tomará por parámetro la fecha de estreno, los atributos que hereda de Media e inicializará la serie. (se construirá una lista de temporadas vacía)
  - **(0,1 puntos)** ***Constructor de copia***. Construirá una serie que será una copia exacta de la serie pasada por parámetro.
  - **(0,05 puntos)** ***LocalDate getFechaEstreno().*** Devolverá la fecha de estreno.
  - **(0,05 puntos)** Se debe sobrescribir el método ***toString()***.
  - **(0,1 puntos)** ***boolean añadirTemporada(LocalDate* fechaEstreno*).*** Creará una temporada con la fecha de estreno pasada por parámetro y la añadirá a la lista de temporadas. Se deberá comprobar que la temporada se puede añadir. No debe haber dos temporadas en un mismo mes. Devolverá true si se ha podido añadir o false en caso contrario.
  - **(0,1 puntos)** ***boolean añadirTemporada(Temporada temporada).*** Añadirá una copia de la temporada pasada por parámetro a la lista de temporadas de la serie. Se deberá comprobar que la temporada se puede añadir. No debe haber dos temporadas en un mismo mes. Devolverá true si se ha podido añadir o false en caso contrario.
  - **(0,1 puntos)** ***boolean eliminarTemporada(int n).*** Eliminará la temporada situada en la posición n. Devolverá true si se ha podido eliminar, false en caso contrario.
  - **(0,1 puntos)** ***boolean añadirCapitulo(int nTemporada, Capitulo capitulo).*** Este método recibirá un número de temporada y un capítulo. El método añadirá un la copia de ese

capítulo a la temporada. Se debe comprobar que es posible hacerlo. El método devolverá true si se ha podido añadir y false en caso contrario.

- **(0,1 puntos)** ***boolean añadirCapitulo(int nTemporada, LocalDate fechaEmision, String titulo).*** Recibirá un número de temporada, una fecha de emisión y un título. Con esa información creará un capítulo que añadirá a la temporada al final de la lista. Se debe controlar que el capítulo no esté ya en la temporada (es decir, no exista otro capítulo con el mismo nombre). Además la fecha de emisión del capítulo deberá ser posterior a la fecha de estreno de la temporada. El método devolverá true si se ha podido añadir y false en caso contrario.
- **(0,1 puntos)** ***boolean añadirCapitulo(int nTemporada, int posicion, Capitulo capitulo).*** Este método recibirá un número de temporada, un capítulo y una posición dentro de la temporada. Se añadirá una copia de ese capítulo en la temporada y posición especificada. Se debe comprobar que se puede añadir. El método devolverá true si se ha podido añadir y false en caso contrario.
- **(0,1 puntos)** ***boolean añadirCapitulo(int nTemporada, int posicion, LocalDate fechaEmision, String titulo).*** Recibirá un número de temporada, una posición, fecha de emisión y un título. Con esa información creará un capítulo que añadirá en la posición indicada y en la temporada indicada. Se debe comprobar que es posible hacerlo. El método devolverá true si se ha podido añadir y false en caso contrario.
- **(0,1 puntos)** ***boolean eliminarCapitulo(int nTemporada, String titulo).*** Recibirá un número de temporada y un título del capítulo. El método eliminará el capítulo de esa temporada que contenga ese título. Devolverá true si ha eliminado el capítulo.
- **(0,1 puntos)** ***int eliminarCapítulos(String expresionRegular).*** Recibirá una expresión regular y eliminará todos los capítulos de todas las temporadas cuyo nombre concuerde con esa expresión regular. Este método devolverá la cantidad de capítulos eliminados.
- **(0,1 puntos)** ***Capitulo getCapitulo(int nTemporada, int posicion).*** Devolverá una copia del capítulo de la temporada nTemporada que se encuentra en la posición indicada. Se debe comprobar que existe, sino se devuelve null.
- **(0,1 puntos)** ***boolean setCapitulo(int nTemporada, int posicion, LocalDate fechaEmision, String titulo).*** Modificará, si es posible, el capítulo de la temporada nTemporada situado en “posicion” con los datos pasados por parámetro, Si ha sido posible modificarlo devolverá true sino false.
- **(0,1 puntos)** ***boolean setCapitulo(int nTemporada, int posicion, LocalDate fechaEmision).*** Modificará, si es posible, el capítulo de la temporada nTemporada situado en “posicion” con los datos pasados por parámetro, Si ha sido posible modificarlo devolverá true sino false.
- **(0,1 puntos)** ***boolean setCapitulo(int nTemporada, int posicion, String titulo).*** Modificará, si es posible, el capítulo de la temporada nTemporada situado en “posicion” con los datos pasados por parámetro, Si ha sido posible modificarlo devolverá true sino false.
- **(0,1 puntos)** ***boolean meGusta(int nTemporada, int posicionCapitulo, boolean like).*** Añade si es posible un voto positivo o negativo (dependiendo del booleano like) al capítulo de la temporada nTemporada y situado en esa posición. Devuelve true si se ha añadido.

- **(0,1 puntos)** ***boolean meGusta(int nTemporada, String titulo, boolean like).*** Añade si es posible un voto positivo o negativo o negativo (dependiendo del booleano like) al capítulo de la temporada nTemporada con el nombre título. Devuelve true si se ha añadido.
- **(0,1 puntos)** ***boolean setFechaEstreno(LocalDate fecha).*** Este método cambiará la fecha de estreno de la serie. Deberá comprobar que es anterior o igual a la fecha de estreno de cada temporada. Finalmente devolverá true si se ha podido cambia o false en caso contrario.
- **(0,1 puntos)** ***Temporada getCopiaTemporada(int nTemporada).*** Devuelve, si es posible una copia de la temporada número "nTemporada", sino devuelve null.

La interfaz Valorable (0,5 puntos)

Interfaz *Valorable*. Obligarán a todas las clases que la implementen a implementar el método ***calcularPuntuacion()***.

- **(0,1 puntos)** Declaración de la interfaz y método ***int calcularPuntuacion()***. Se calculará como un valor entre cero y diez.

Las clases que implementarán la interfaz *Valorable* y por tanto están obligadas a implementar el método *calcularPuntuacion()* serán: *Capítulo*, *Temporada* y *Media*. En la clase Media no se ofrecerá implementación ya que se tendrá que implementar en sus subclases (*Serie* y *Pelicula)* para poder hacer polimorfismo sobre esta interfaz. Para cada uno de los casos, el cálculo se hará de diferente manera:

- *Capitulo:*
  - **(0,1 puntos)** La puntuación del capítulo se hará teniendo en cuenta el número de votos positivos y el número de votos negativos y se calculará dividiendo el número de votos positivos por el número de votos total. Finalmente, a ese número que estará entre cero y uno, lo multiplicaremos por 10.
- *Temporada*:
  - **(0,1 puntos)** La valoración de la temporada será la media de la valoración de cada uno de sus capítulos.
- *Serie*:
  - **(0,1 puntos)** La puntuación de la serie será la media de la valoración de las temporadas.
- *Pelicula*:
  - **(0,1 puntos)** Media de la lista de votos obtenida de la película.

La clase Vergeflix (2 puntos) Programa principal (contendrá el *main*)

El programa principal gestionará un catálogo de series y películas. Para ello se creará un *ArrayList* que contendrá objetos de tipo *Media* (podrá contener tanto objetos *Serie* como *Pelicula*).

El menú con las distintas opciones disponibles será el siguiente:

1. Gestionar películas.
2. Gestionar series.
3. Consultar catálogo.
4. Salir.

Opción 1. Gestionar películas

Esta opción nos mostrará un submenú con todas las opciones para gestionar las películas del catálogo. En concreto serán las siguientes:

1. Añadir Película **(0,2 puntos)**
2. Modificar Película **(0,2 puntos)**
3. Eliminar Película **(0,2 puntos)**
4. Listar películas **(0,2 puntos)**
5. Valorar Película **(0,2 puntos)**
6. Volver

Para cada una de las opciones se deja propuesto que se pidan los datos necesarios para realizar la acción concreta. El programa deberá, además de realizar la acción, mostrar un mensaje con el resultado.

Opción 2. Gestionar series

Esta opción nos mostrará un submenú con todas las opciones para gestionar las series del catálogo. En concreto serán las siguientes:

1. Añadir serie **(0,1 puntos)**
2. Modificar Serie **(0,1 puntos)**
3. Eliminar Serie **(0,1 puntos)**
4. Listar Series **(0,1 puntos)**
5. Gestionar Serie concreta.
6. Volver

Para cada una de las opciones se deja propuesto que se pidan los datos necesarios para realizar la acción concreta. El programa deberá, además de realizar la acción, mostrar un mensaje con el resultado.

- **Opción 5. Gestionar serie concreta.** (dentro del menú gestionar series) Esta opción deberá gestionar las distintas temporadas y capítulos de las series por lo que mostrará el siguiente submenú.

- 1. Añadir Temporada **(0,1 puntos)**
- 2. Modificar Temporada (se puede modificar la fecha de estreno añadiendo un método para ello a la clase Serie, o modificar algún capítulo de la temporada a través de los métodos setCapitulo que hay en serie) **(0,1 puntos)**
- 3. Eliminar Temporada **(0,1 puntos)**
- 4. Añadir capítulos **(0,1 puntos)**
- 5. Eliminar capítulos **(0,1 puntos)**
- 6. Listar temporadas y capítulos **(0,1 puntos)**
- 7. Valorar Capítulo **(0,1 puntos)**
- 8. Volver

Para cada una de las opciones se deja propuesto que se pidan los datos necesarios para realizar la acción concreta. El programa deberá, además de realizar la acción, mostrar un mensaje con el resultado.

Opción 3. Consultar catálogo.

En esta opción se ofrecerá distintas opciones de consulta de los elementos almacenados. En concreto se permitirán las siguientes opciones:

1. Mostrar Series o películas según su calificación de edad **(0,3 puntos)**
2. Listar las n mejores películas por temática **(0,2 puntos)**
3. Listar las n mejores películas por autor **(0,2 puntos)**
4. Listar las n mejores películas por director **(0,2 puntos)**
5. Listar las n mejores series **(0,5 puntos)**
6. Listar los n mejores elementos disponibles del catálogo (serie o película) **(0,5 puntos)**
7. Volver

Para cada una de las opciones se deja propuesto que se pidan los datos necesarios para realizar la acción concreta. El programa deberá, además de realizar la acción, mostrar un mensaje con el resultado.

Importante: Para realizar el programa principal se deberá pensar y realizar métodos y funciones que modularicen el código. Como por ejemplo:

- public static void imprimirLista(ArrayList<Media> catalogo)
- public static Media mejorElementoCatalogo(ArrayList<Media> catalogo)
- public static ArrayList<Media> mejoresElementosCatalogo(int topn, ArrayList<Media>

catalogo)

- public static ArrayList<Media> listaMediaSegunEdad(ArrayList<Media> catalogo, int calificacionEdad)
- public static void imprimirMenuPrincipal()
- public static void imprimirSubMenuGestionarSeries()
- public static ArrayList<Pelicula> peliculasDelCatalogo(ArrayList<Media> catalogo)
- public static ArrayList<Serie> seriesDelCatalogo(ArrayList<Media> catalogo)
- etc.

Importante
Para poder probar el programa el proyecto deberá realizar una precarga de datos previa que rellene el ArrayList catálogo con datos de prueba.

Se deben comprobar y manejar los errores de forma adecuada para evitar que el programa termine de forma inesperada. Por tanto se comprobará que el usuario introduce valores correctos en todo momento.

Se valorará el diseño modular y estructurado, la claridad y simplicidad del código y su documentación. Todas las validaciones de las cadenas de caracteres se realizarán mediante expresiones regulares. Para ello, se reutilizará lo máximo posible el código creando funciones en los lugares donde creas que son más adecuados para realizar esta tarea.

Entrega de la tarea

El proyecto se desarrollará con el IDE Netbeans. El nombre del proyecto y de las clases deben llamarse de la forma indicada en la descripción de la tarea. Se comprimirá la carpeta del proyecto Netbeans en un único fichero en formato .ZIP o 7z que se subirá al buzón de la tarea en la plataforma Moodle.

El archivo se nombrará siguiendo las siguientes pautas:

Apellido1_Apellido2_Nombre_PROG_ProyectoVergeflix.zip

Resultados de aprendizaje y criterios de evaluación relacionados

En esta actividad se evalúan los siguientes resultados de aprendizaje con los criterios de evaluación que se relacionan para cada uno de ellos:

- RA 3. Escribe y depura código, analizando y utilizando las estructuras de control del lenguaje.
  a) Se ha escrito y probado código que haga uso de estructuras de selección.
  b) Se han utilizado estructuras de repetición.
  c) Se han reconocido las posibilidades de las sentencias de salto.
  d) Se ha escrito código utilizando control de excepciones.
  e) Se han creado programas ejecutables utilizando diferentes estructuras de control.
  f) Se han probado y depurado los programas.

g) Se ha comentado y documentado el código.

- RA 4. Desarrolla programas organizados en clases analizando y aplicando los principios de la programación orientada a objetos.
  a) Se ha reconocido la sintaxis, estructura y componentes típicos de una clase.
  b) Se han definido clases.
  c) Se han definido propiedades y métodos.
  d) Se han creado constructores.
  e) Se han desarrollado programas que instancien y utilicen objetos de las clases creadas anteriormente.
  f) Se han utilizado mecanismos para controlar la visibilidad de las clases y de sus miembros.
  g) Se han definido y utilizado clases heredadas.
  h) Se han creado y utilizado métodos estáticos.
  i) Se han definido y utilizado interfaces.
- RA 5. Realiza operaciones de entrada y salida de información, utilizando procedimientos específicos del lenguaje y librerías de clases.
  a) Se ha utilizado la consola para realizar operaciones de entrada y salida de información.
  b) Se han aplicado formatos en la visualización de la información.
  c) Se han reconocido las posibilidades de entrada / salida del lenguaje y las librerías asociadas.
- RA 6. Escribe programas que manipulen información seleccionando y utilizando tipos avanzados de datos.
  b) Se han reconocido las librerías de clases relacionadas con tipos de datos avanzados.
  c) Se han utilizado listas para almacenar y procesar información.
  d) Se han utilizado iteradores para recorrer los elementos de las listas.
  e) Se han reconocido las características y ventajas de cada una de la colecciones de datos disponibles.
  g) Se han utilizado expresiones regulares en la búsqueda de patrones en cadenas de texto.
- RA 7. Desarrolla programas aplicando características avanzadas de los lenguajes orientados a objetos y del entorno de programación.
  a) Se han identificado los conceptos de herencia, superclase y subclase.
  b) Se han utilizado modificadores para bloquear y forzar la herencia de clases y métodos.
  c) Se ha reconocido la incidencia de los constructores en la herencia.
  d) Se han credo clases heredadas que sobrescriban la implementación de métodos de la superclase.
  e) Se han diseñado y aplicado jerarquías de clases.
  f) Se han probado y depurado jerarquías de clases.
  g) Se han realizado programas que implementen y utilicen jerarquías de clases.
  h) Se ha comentado y documentado el código.

Rúbrica de evaluación

Cada apartado puntuable del proyecto se valorará con la siguiente rúbrica.

#	Criterio	Porcentaje
1	El programa/función implementado no cumple los requisitos, no soluciona de forma algorítmica el ejercicio o las soluciones obtenidas por el programa no son las esperadas, el código no compila o contiene errores.	0%
2	El programa/función implementado soluciona de forma algorítmica el ejercicio pero falla con datos de entrada no permitidos.	25%
3	El programa/función implementado tiene fallos inesperados en situaciones específicas o concretas, es decir, falla para un determinado caso o valor de entrada, pero en general el resultado obtenido es válido.	50%
4	El programa/función implementado cumple los requerimientos pero: • El código no es legible o no está bien estructurado.	75%
5	El programa/función implementado se ajusta perfectamente a la especificación: • Se validan los datos de entrada. • El resultado obtenido es válido para cualquier dato de entrada. • El código es modular y se emplean funciones/métodos adecuadamente. • El código es legible y usa comentarios relevantes y/o Javadoc.	100%