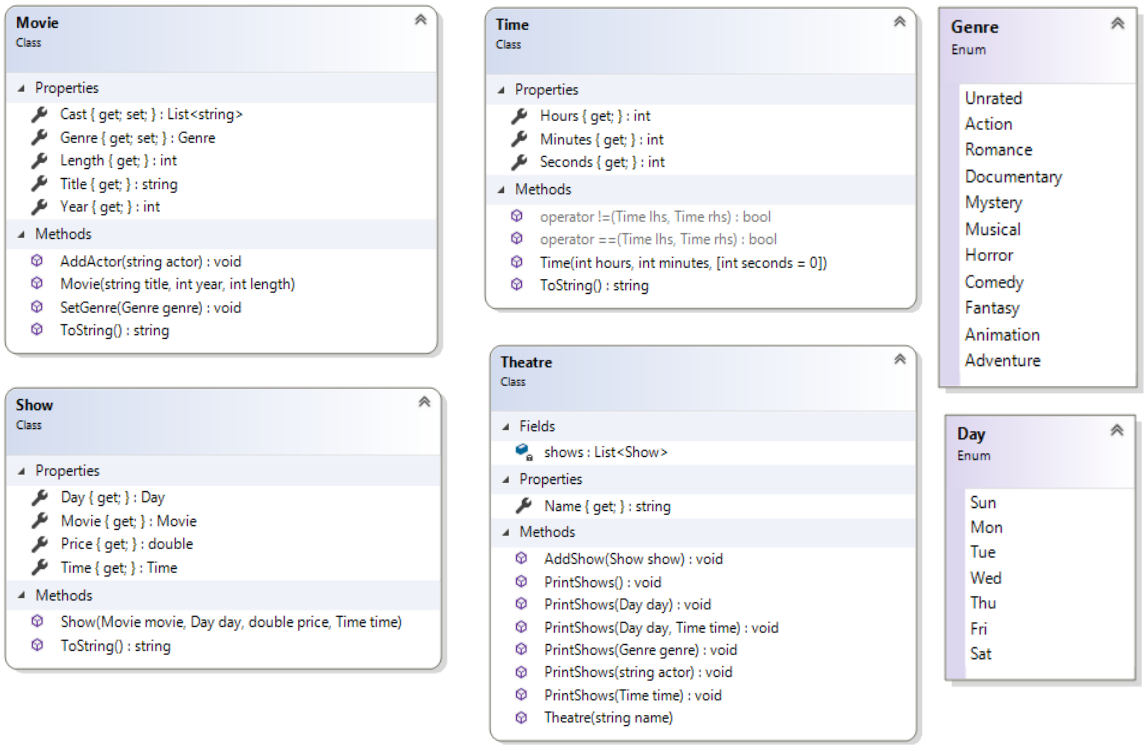
Programming II

Assignment 3 – Implement a Theatre Application

Once you are done with your program, upload it to eCentennial under Assessments / Assignment / Assignment 3

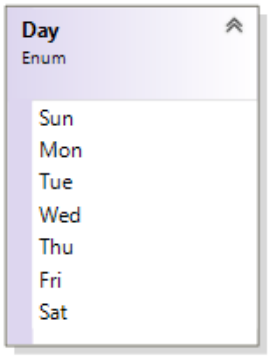
This is the most complex application that you have seen so far. It comprises of six user defined types: four classes and two enums. This assignment attempts to simulate a simple version of a Theater application. In this application the user is able to search for a movie based on genre, name of actor, the time of the day and the day of the week.

You should define your types in the following order: the two enums, Time, Movie, Show and then Theater. In the test harness, you will create a theater, some movies and some shows. You will add movies to the show and then add the shows to the theater. Then, you will perform some simple queries on this collection.



The MovieDay Enum

This type represents the days of the week and is comprised of seven constants. The constants are the first three letters of the day of the week.

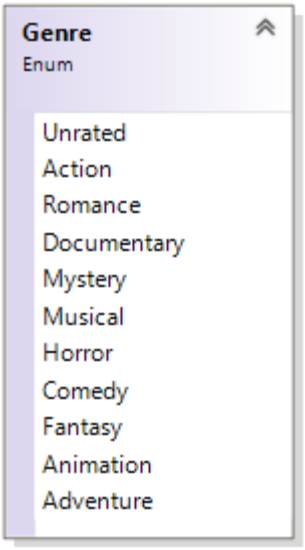


It is not necessary to assign values to the members.

Also, you do not have to use the `[Flags]` attribute to decorate this type.

The MovieGenre Enum

This type represents the various categories of movie. Because a movie may fall under multiple categories, you will have to use the `[Flags]` to decorate this enum.



The bit-wise operator is a single bar "|" and it is used to combine more than one genre. E.g. `Genre genre = Genre.Action | Genre.Romance | Genre.Comedy;`

For this to work correctly, you will have to assign appropriate values for each name. Appropriate values are 0, 1, 2, 4, 8, 16, 32 etc.

The Time Struct

This type represents a time when a movie is schedule to be shown.

Time Struct
Fields
- hours : int - minutes : int
Properties
Methods
+ «Constructor» Time(hours : int, minutes = 0 : int) + ToString() : string +$ operator ==(lhs : Time, rhs : Time) : bool +$ operator !=(lhs : Time, rhs : Time) : bool

Fields:

All of the properties have public getters and private setters and are self-explanatory.

1. **hours** – this int represents the private field hours.
2. **minutes** – this int represents the private field minutes.

Properties:

There are no properties.

Methods:

1. **public Time(int hours, int minutes = 0)** – This public constructor takes two int parameters and assigns them to the appropriate properties. The second parameter is optional
2. **public override string ToString()** – This method overrides the same method of the Object class. It does not take any parameter but return a string representation of itself. You decide on the format for the output.
3. **public static operator bool ==(Time lhs, Time rhs)** – This method overrides the equality operator. This method will return true if the difference between the two arguments is less than 15 minutes, otherwise it returns false. This method is used in the **PrintShows(Time time)** method of the Theater class.

If you cannot figure out the 15 minutes difference, then check for equality of the fields **`hours`** and **`minutes`**.
One way of figuring out the 15 minutes difference is to convert the hours and minutes fields to just a minute value. If you do the same to both objects, you can determine how close the two objects are.

4. `public static operator bool !=(Time lhs, Time rhs)` – This method overrides the not equality operator. This method is required by the compiler when the above method is implemented. It does the reverse of the above. This method is not used directly be any class.

The Movie Class

This class will model a movie.

Movie Class
Properties
+ «C# property, setter absent» Length : int + «C# property, setter absent» Year : int + «C# property, setter absent» Title : string + «C# property, private setter» Genre : Genre + «C# property, setter absent» Cast : List<string>
Methods
+ «Constructor» Movie(title : string, year : int, length : int) + AddActor(actor : string) : void + SetGenre(genre : MovieGenre) : void + ToString() : string

Properties:

All of the properties have public getters and missing setters and are self-explanatory.

1. **Length** – this property is an int representing the length of the movie in minutes. The getter is public and the setter is absent.
2. **Year** – this property is an int representing the year that this movie was released. The getter is public and the setter is absent.
3. **Title** – this property is a string representing the title of the movie. The getter is public and the setter is absent.
4. **Genre** – this property is an enum representing the genre of this movie. The getter is public and the setter is private. This property is also modified by the **SetGenre(Genre genre)** method
5. **Cast** – this property is a list of string representing the names of the actors in this movie. The getter is public and the setter is private.

Methods:

1. **public Movie(string name, int year, int length)** – This public constructor takes one string and two int parameters. It does the following:
   a. Assigns the arguments to the appropriate properties.
   b. Initialize the Cast property to an empty list of string.

2. **`public void AddActor(string actor)`** – This public method takes a single a string argument and adds it to the collection of actors (**`Cast`**).
3. **`public void SetGenre(Genre genre)`** – This public method takes a single enum argument and assigns it to the property of the same name. This argument maybe a combination of multiple genres.
4. **`public override string ToString()`** – This method overrides the same method of the Object class. It does not take any parameter but return a string representation of itself. You will need to show all the properties of this object.
   To show the actors, you need to build a single string comprising all of the elements in this movie. Use the `string.Join(", ", Cast)`
   You decide on the format for the output.

The Show Class

This class models a movie show. You will also implement this in Visual Studio. The objects from this class are immutable i.e. it does not change. It simply captures the four required values for an object. A short description of the class members is given below:

Show Class
Properties
+ «C# property, setter absent» Price : double + «C# property, setter absent» Day : Day + «C# property, setter absent» Movie: Movie + «C# property, setter absent» Time: Time
Methods
+ «constructor» Show(movie : Movie , day : Day , price : double , time : Time) + ToString() : string

Properties:

All the properties have public getter and the setters are absent.

1. **Price** – this property is a double representing the price of admission to this show. The getter is public and the setter is absent.
2. **Day** – this property is an enum representing the day of the week of this show. The getter is public and the setter is absent.
3. **Movie** – this property is an object reference of the movie class. The getter is public and the setter is absent.
4. **Time** – this property is an object of the Time class representing the time of this show. The getter is public and the setter is absent.

Methods:

1. **`public Show(Movie movie, Day day, double price, Time time)`** – This is the public constructor that takes four arguments and assigns them to the appropriate properties.
2. **`public override string ToString()`** – This is the public method overrides the method of the same name in the object class to return a meaningful description of this object.

The Theatre Class

This class models a theater. You will also implement this in Visual Studio. A short description of the class members is given below:

Theatre Class
Fields
- shows : List<Show>
Properties
+ «C# property, setter absent» Name : string
Methods
+ Theater(name : string) + AddShow(show : Show) : void + PrintShows() : void + PrintShows(genre : Genre) : void + PrintShows(day : Day) : void + PrintShows(time : Time) : void + PrintShows(actor : string) : void + PrintShows(day : Day, time : Time) : void

Fields:

1. **shows** – this private field is a list of Show objects.

Properties:

2. **Name** – this property is a string representing the name of the theater. Getter is public and the setter is absent

Methods:

1. `public Theater(string name)` – This is the public constructor that takes the name of the theater. This constructor does the following:
   a. Assigns the argument to the appropriate property.
   b. Initialize the **Shows** property to a new list of show

2. **`public void AddShow(Show show)`** – This public method takes a show object and adds it to the collection of shows.

> Again, these six methods illustrate the benefits of method overloading.

All of the following overloaded methods below will display the following:

- Name of the theater.
- The value of the filter, the first method has no filter, so instead display **`"All shows".`**
- A numbered list of the movies displayed.

3. **`public void PrintShows()`** – This public method does not take any argument neither does it return a value. It displays all the shows that is available in the shows collection.
4. **`public void PrintShows(Genre genre)`** – This public method takes a genre as an argument and display all the shows that contains the flag of this genre.
   Use the instance method **`HasFlags()`**.
5. **`public void PrintShows(Day day)`** – This public method takes a day object as an argument and display all the shows matching this day object.
6. **`public void PrintShows(Time time)`** – This public method takes a time object as an argument and display all the shows matching the hour value of this time object.
   This will only work correctly if the **`==`** operator is implemented correctly in the Time class. The `Time` object that is associated with a show is not the same as the `Time` object that will be passed as the argument, therefore comparison with the usual == operator will not work because the object reference are not the same. Hence you have to supply a custom == operator that compares the numeric values of the fields of the two objects must be compared.
7. **`public void PrintShows(string actor)`** – This public method takes a string representing the name of an actor as an argument and display all the shows that this actor appears in.
   Remember that all the actors are stored as a List<string>, so you will have to use the instance method **`Contains()`** to check for the presence of an actor.
8. **`public void PrintShows(Day day, Time time)`** – This public method takes a day value and a time value as arguments and display all the shows matching the day and the time value
   The `Time` object that is associated with a show is not the same as the `Time` object that will be passed as the argument, therefore the numeric values of the fields of the two objects must be compared.

Testing 1 Mark

In your test harness (the Main() method in the Program Class), copy and paste the following code:

```
Movie terminator = new Movie("Terminator 2: Judgement Day", 1991, 105);
terminator.AddActor("Arnold Schwarzenegger");
terminator.SetGenre(Genre.Horror | Genre.Action);
terminator.AddActor("Linda Hamilton");
Show s1 = new Show(terminator, Day.Mon, 5.95, new Time(11, 35));

Console.WriteLine(s1);              //displays one object
Theatre eglinton = new Theatre("Cineplex");
eglinton.AddShow(s1);
eglinton.PrintShows();              //displays one object

Movie godzilla = new Movie("Godzilla 2014", 2014, 123);
godzilla.AddActor("Aaron Johnson");
godzilla.AddActor("Ken Watanabe");
godzilla.AddActor("Elizabeth Olsen");
godzilla.SetGenre(Genre.Action | Genre.Documentary | Genre.Comedy);

Movie trancendence = new Movie("Transendence", 2014, 120);
trancendence.AddActor("Johnny Depp");
trancendence.AddActor("Morgan Freeman");
trancendence.SetGenre(Genre.Comedy);
eglinton.AddShow(new Show(trancendence, Day.Sun, 7.87, new Time(18, 5)));

Movie m1 = new Movie("The Shawshank Redemption", 1994, 120);
m1.AddActor("Tim Robbins");
m1.AddActor("Morgan Freeman");
m1.SetGenre(Genre.Action);
eglinton.AddShow(new Show(m1, Day.Sun, 8.87, new Time(14, 5)));

Movie avengers = new Movie("Avengers: Endgame", 2019, 120);
avengers.AddActor("Robert Downey Jr.");
avengers.AddActor("Chris Evans");
avengers.AddActor("Chris Hemsworth");
avengers.AddActor("Scarlett Johansson");
avengers.AddActor("Mark Ruffalo");
avengers.SetGenre(Genre.Action | Genre.Fantasy | Genre.Adventure);
eglinton.AddShow(new Show(avengers, Day.Sat, 12.25, new Time(21, 5)));

m1 = new Movie("Olympus Has Fallen", 2013, 120);
m1.AddActor("Gerard Butler");
m1.AddActor("Morgan Freeman");
m1.SetGenre(Genre.Action);
eglinton.AddShow(new Show(m1, Day.Sun, 8.87, new Time(16, 5)));

m1 = new Movie("The Mask of Zorro", 1998, 136);
m1.AddActor("Antonio Banderas");
m1.AddActor("Anthony Hopkins");
m1.AddActor(" Catherine Zeta-Jones");
m1.SetGenre(Genre.Action | Genre.Romance);
eglinton.AddShow(new Show(m1, Day.Sun, 8.87, new Time(16, 5)));

m1 = new Movie("Four Weddings and a Funeral", 1994, 117);
m1.AddActor("Hugh Grant");
m1.AddActor("Andie MacDowell");
```

```
m1.AddActor("Kristin Scott Thomas");
m1.SetGenre(Genre.Comedy | Genre.Romance);
eglinton.AddShow(new Show(m1, Day.Sat, 8.87, new Time(15, 5)));
eglinton.AddShow(new Show(m1, Day.Tue, 6.50, new Time(16, 5)));

m1 = new Movie("You've Got Mail", 1998, 119);
m1.AddActor("Tom Hanks");
m1.AddActor("Meg Ryan");
m1.SetGenre(Genre.Comedy | Genre.Romance);
eglinton.AddShow(new Show(m1, Day.Sat, 8.87, new Time(15, 5)));

m1 = new Movie("The Poison Rose", 2019, 98);
m1.AddActor("John Travolta");
m1.AddActor("Morgan Freeman");
m1.AddActor("Brendan Fraser");
m1.SetGenre(Genre.Action | Genre.Romance);
eglinton.AddShow(new Show(m1, Day.Sun, 10.25, new Time(22, 5)));

Movie car3 = new Movie("Cars 3", 2017, 109);
car3.AddActor("Owen Williams");
car3.AddActor("Cristela Alonzo");
car3.AddActor("Arnie Hammer");
car3.AddActor("Chris Cooper");
car3.SetGenre(Genre.Comedy | Genre.Animation | Genre.Romance);
eglinton.AddShow(new Show(car3, Day.Sat, 6.40, new Time(09, 55)));
eglinton.AddShow(new Show(car3, Day.Sat, 6.50, new Time(11, 05)));

Movie toys4 = new Movie("Toys Story 4", 2019,  89);
toys4.AddActor("Keanu Reeves");
toys4.AddActor("Christina Hendricks");
toys4.AddActor("Tom Hanks");
toys4.AddActor("Tim Allen");
toys4.SetGenre(Genre.Comedy | Genre.Fantasy | Genre.Animation);
eglinton.AddShow(new Show(toys4, Day.Sat, 6.40, new Time(14, 10)));

eglinton.AddShow(new Show(godzilla, Day.Mon, 6.89, new Time(13, 55)));
eglinton.AddShow(new Show(avengers, Day.Sat, 12.25, new Time(21, 5)));
eglinton.AddShow(new Show(godzilla, Day.Sun, 6.89, new Time(14)));
eglinton.AddShow(new Show(toys4, Day.Sat, 6.40, new Time(14, 10)));
eglinton.AddShow(new Show(avengers, Day.Sat, 12.25, new Time(21, 5)));
eglinton.AddShow(new Show(godzilla, Day.Sun, 6.89, new Time(16, 55)));
eglinton.AddShow(new Show(avengers, Day.Sat, 12.25, new Time(21, 5)));
eglinton.AddShow(new Show(m1, Day.Sat, 10.25, new Time(20, 35)));
eglinton.AddShow(new Show(godzilla, Day.Wed, 8.50, new Time(22, 5)));
eglinton.AddShow(new Show(avengers, Day.Tue, 10.75, new Time(20, 30)));
eglinton.AddShow(new Show(godzilla, Day.Thu, 8.50, new Time(20, 15)));
eglinton.AddShow(new Show(avengers, Day.Wed, 10.75, new Time(20, 30)));
eglinton.AddShow(new Show(godzilla, Day.Fri, 8.50, new Time(18, 25)));
eglinton.AddShow(new Show(avengers, Day.Sun, 10.75, new Time(14, 15)));

eglinton.PrintShows();                              //displays 27 objects
eglinton.PrintShows(Day.Sun);                       //displays 8 objects
eglinton.PrintShows(Genre.Action);                  //displays 19 objects
eglinton.PrintShows(Genre.Romance);                 //displays 8 objects
eglinton.PrintShows(Genre.Action | Genre.Romance);  //displays 3 objects
eglinton.PrintShows("Morgan Freeman");              //displays 5 objects

Time time = new Time(14, 05);
```

```
eglinton.PrintShows(time);                          //displays 6 objects

eglinton.PrintShows(Day.Sun, time);                 //displays 3 objects
```