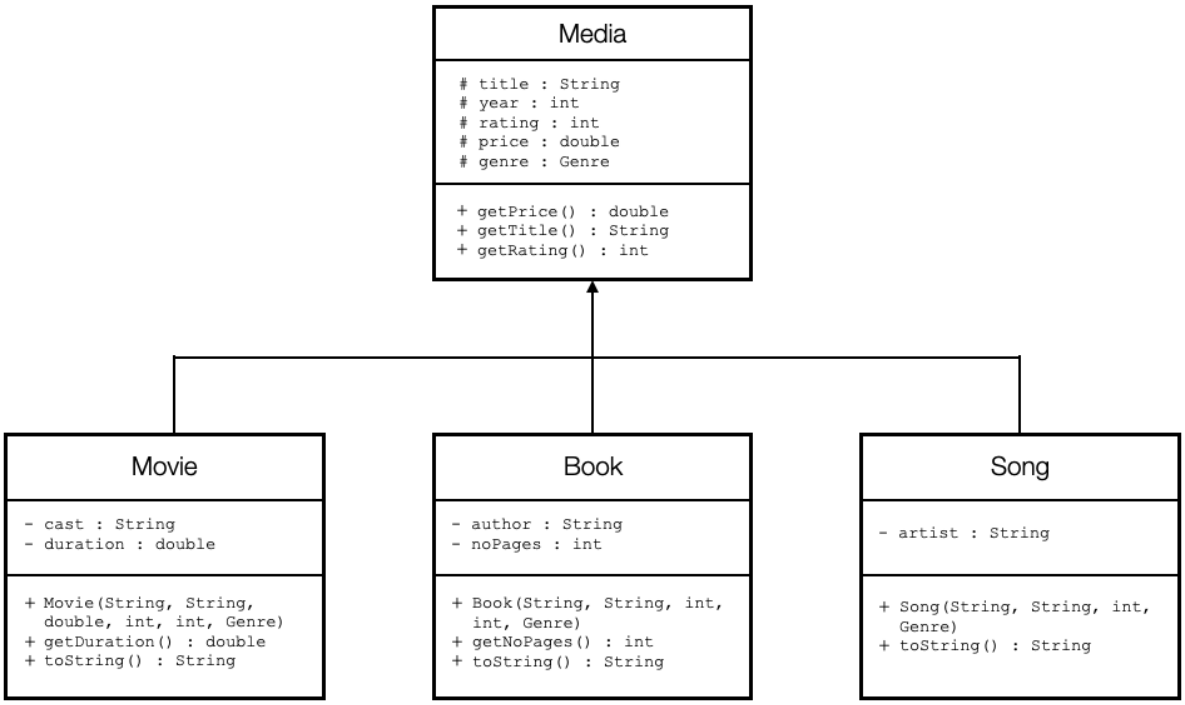
Week 02: Generic Types, more inheritance and object sorting

Challenge 01: YourPrime (Not Amazon) (70 mins)

Well, this is starting to take a familiar tone, right? Fortunately, on Wikipedia, you can find information on the programming languages used in popular websites. If you click this link, you will see that Java dominates the list. As a backend engine, you simply can't run away for using Java, and that's what we going to do.

Amazon has everything (yup, everything), and in order to conquer the digital media streaming market, they've created Prime (yes, it is not new). You also have Netflix, Disney+, …, (well, I think the list goes on for quite a bit here). Typically, if you are developing a media streaming platform, you need to handle various materials. We will limit our type of media (because we are 'gone in 60 minutes not second') for this challenge to **`Movies`**, **`Songs`**, and **`Books`**. You can probably guess where we are going with this, So let's discuss the To-Dos.

TODO#1 – Create all classes as shown in the UML diagram below:



You should have the **Media** superclass, with the **Movie**, **Song**, and **Book** subtypes. OK, we have the **Song** class already from the previous lab session, but you need to refactor some codes to ensure compatibility with our inheritance tree implementation.

** If you refer to the skeleton code, the class declarations have been provided. You are required to complete the definition of methods as stated in the TODO comment available in each method.

TODO#2 – Notice that all the subtypes have **`Genre`** as variable. Fortunately, we have created our Genre class as abstract, therefore, we just need to add some new genre types into the program. You need to create the **`Fantasy`**, and **`Autobiography`** (which will be genre for book), and **`Comedy`**, **`Action`**, and **`Drama`** (for movie genre).

You might have spotted that in the current design, one can instantiate a new Media subtype and assign it to any Genre (regardless of the category) that we've indicated here. Can you stop this from

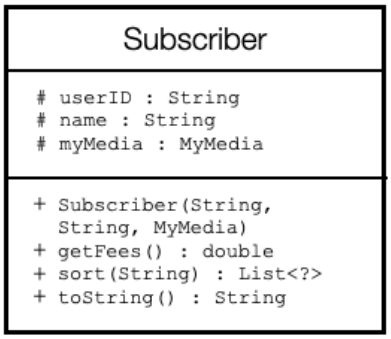
happening? If you can, how would you do it? Don't worry about it now, but it is good to keep this question at the back of your mind, we will revisit this sometime in the future (not so distant future).

TODO#3 – You can define the abstract method for `setPrice()` using the table below:

Genre	Rating > 3	Rating == 3	Rating < 3
Fantasy	1.99	0.99	0.00
Autobiography	2.99	1.99	0.99
Comedy	1.99	0.99	0.00
Action	2.99	1.99	0.99
Drama	1.99	0.99	0.00

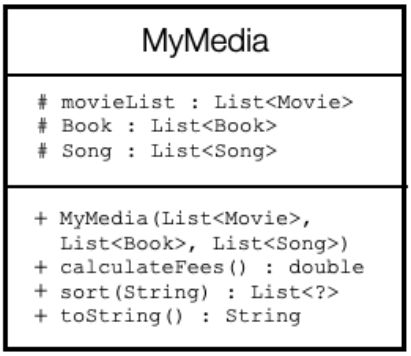
** The genre type related to music will use the same price structure as previous version. So the codes are unchanged. Since, our business model (not a profit-oriented model, it seems) is based on the consumption level (as an ode to the 2000s), you will need to display the monthly charges to our subscriber (more on this next).

TODO#4 – Create a **`Subscriber`** class using the UML diagram given:



There is no point of having a media streaming service if we don't have any subscribers. So, we need a subscriber class to handle our subscribers object. In order to determine the total charge for each subscriber, we need to know the media list, so the method `getFees()` will call a method in `MyMedia` to calculate all media. The overriding `toString()` method will also call a method in `MyMedia` to complete its operation. We will discuss the sorting mechanism in the next challenge (so let it be – o, let it be, at the moment).

TODO#5 – Let's put everything together. You need to create a class called **`MyMedia`** where you will store three (3) lists (playlist of movies, books and songs). You will need to use generics to declare an `ArrayList` that will take the objects corresponding to the respective media types (i.e., a list for each media type). The rest of the class will follow the UML diagram below:



The function `calculateFees()` will calculate the total fees that will be charged to the subscribers. You need to use the `setPrice()` methods for each media type to determine the correct value.

You can refer to the skeleton code for details. The main() method for this challenge is available in the **`YourPrimeTest`** class – you can use the code to help you complete your class definition.

Challenge 02: Sorting (Again?) (20 mins)

Now we have a digital media platform offering a selection of movies, songs and books (yes, the selection is very limited – wait until we get to Week 04). Our subscribers can sort their playlist according to the title, rating, price, number of pages (for books) and duration (for movies), and artist (for songs).
There is a method `List<?> sort(String OrderType)` in the Subscriber class to do this. This method will call the sort() method in `MyMedia` to handle the sorting operation using `Comparator` class according to each media type. You need to complete the definition of this method following:

TODO#1 For each media type, declare the correct sorting method using the Collection sort interface. Given the argument type, you should be able to return a `List<?>` by calling the corresponding sorting-type object.

TODO#2 You need to implement sorting order by title, price and rating for all media types, order by number of pages for the books, and order by duration for the movies. If this is implemented properly in your MyMedia, you should not be facing any reference type issues – and the test code on your `main()` method should run correctly.

Note: If you find the instances lame (and not too your preference), you can change them , or better yet add more objects into the playlist 😊