Semester Test – Task 2

Written by: Marella Morad – Student ID: 103076428

For Task 1 of the semester test, I designed an abstract Library Resource class and redesigned the Game and Book classes from the original library design.

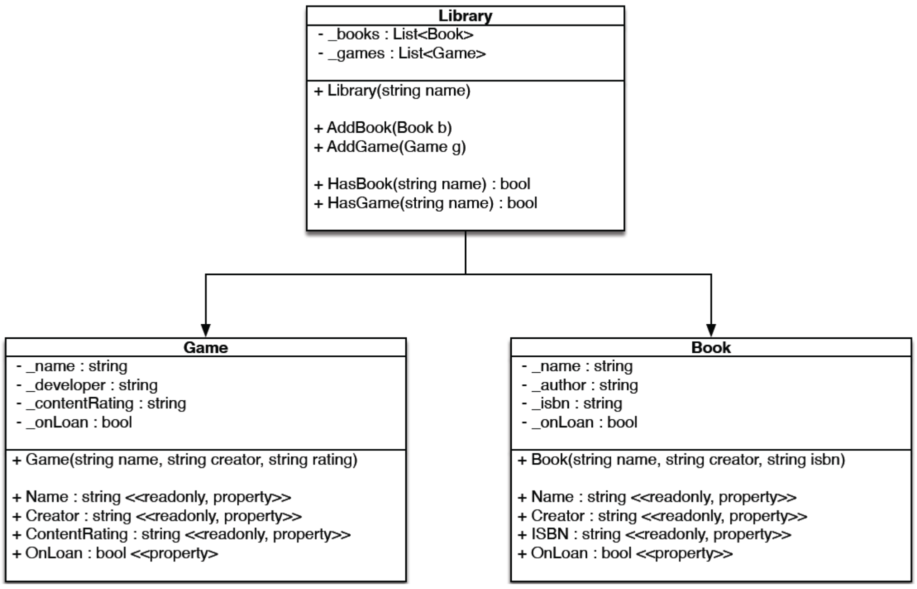

Library
- _books : List<Book>
- _games : List<Game>
+ Library(string name)
+ AddBook(Book b)
+ AddGame(Game g)
+ HasBook(string name) : bool
+ HasGame(string name) : bool
Game
- _name : string
- _developer : string
- _contentRating : string
- _onLoan : bool
+ Game(string name, string creator, string rating)
+ Name : string <<readonly, property>>
+ Creator : string <<readonly, property>>
+ ContentRating : string <<readonly, property>>
+ OnLoan : bool <<property>>
Book
- _name : string
- _author : string
- _isbn : string
- _onLoan : bool
+ Book(string name, string creator, string isbn)
+ Name : string <<readonly, property>>
+ Creator : string <<readonly, property>>
+ ISBN : string <<readonly, property>>
+ OnLoan : bool <<property>>

Figure 1 Original Design of the Library Program

In this report, I will talk about how I applied Abstraction and Polymorphism to achieve a better design.

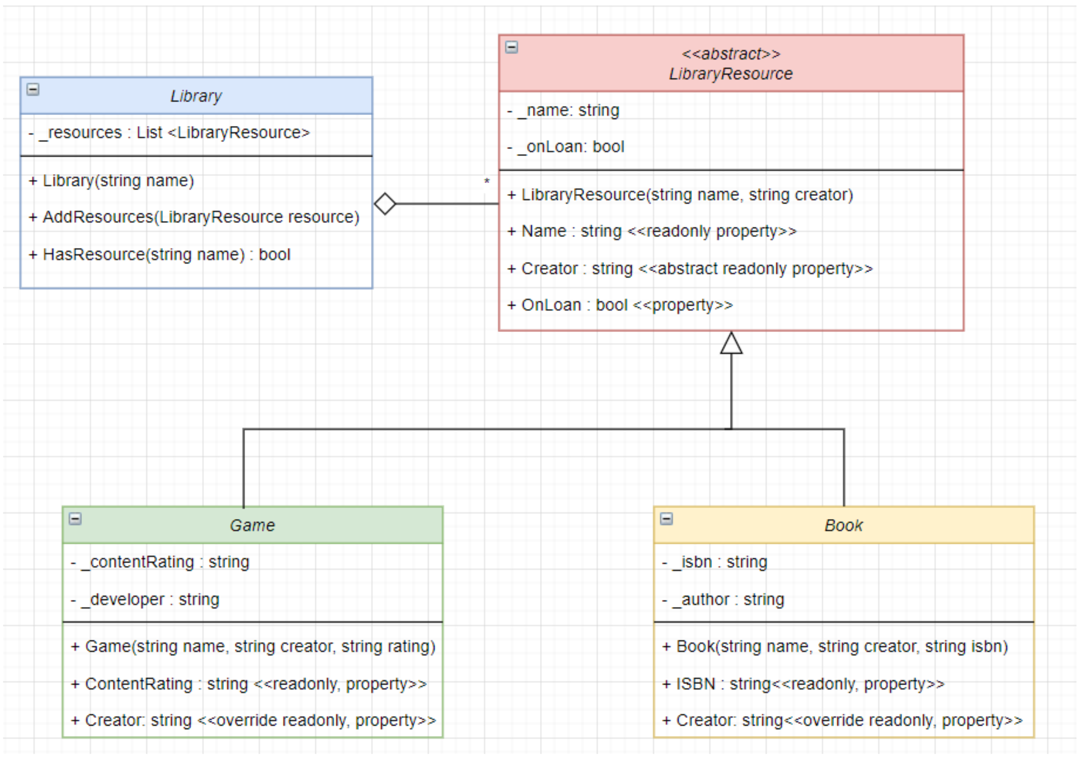
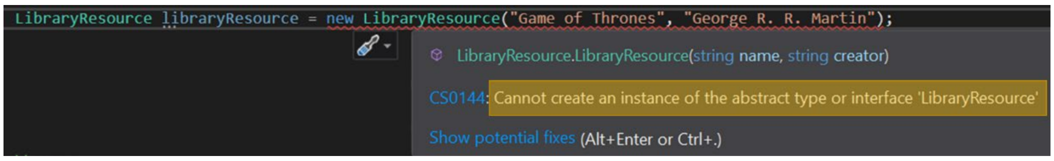

Library
- _resources : List <LibraryResource>
+ Library(string name)
+ AddResources(LibraryResource resource)
+ HasResource(string name) : bool
*
<<abstract>>
LibraryResource
- _name: string
- _onLoan: bool
+ LibraryResource(string name, string creator)
+ Name : string <<readonly property>>
+ Creator : string <<abstract readonly property>>
+ OnLoan : bool <<property>>
Game
- _contentRating : string
- _developer : string
+ Game(string name, string creator, string rating)
+ ContentRating : string <<readonly, property>>
+ Creator: string <<override readonly, property>>
Book
- _isbn : string
- _author : string
+ Book(string name, string creator, string isbn)
+ ISBN : string<<readonly, property>>
+ Creator: string<<override readonly, property>>

Figure 2 Redesign applying Abstraction and Polymorphism

Explanation

The concept of **Abstraction** is defined as the process of designing an object based on its Roles, Responsibilities, Classifications and Collaborations with other objects. For this task, I was required to design the Library Resource abstract class (see Figure 2). As outlined in the document, the `LibraryResource` class is an abstract class, which means it cannot instantiate an object. Meaning, `LibraryResource resource = new LibraryResource()` does not work (See Figure 3).



```
LibraryResource libraryResource = new LibraryResource("Game of Thrones", "George R. R. Martin");
```

LibraryResource.LibraryResource(string name, string creator)

CS0144: Cannot create an instance of the abstract type or interface 'LibraryResource'

Show potential fixes (Alt+Enter or Ctrl+.)

Figure 3 Evidence of how abstract classes cannot be instantiated

To design the `LibraryResource` class, I analysed the original design (Figure 1) to find similarities in the `Game` and `Book` classes which I will include as part of the `Library Resource` class.

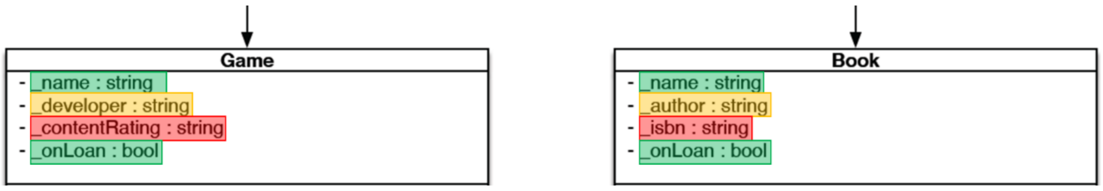
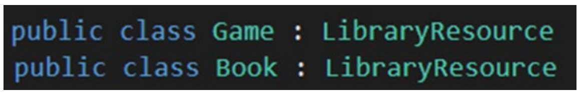


Figure 4 Comparing Game and Book classes {Same fields (Green), Similar fields (Yellow), Different Fields (Red)}

As shown in Figure 4, the `name` and `onLoan` fields are the exact same in both classes. While the `isbn` and `contentRating` are different fields. Leaving the author to be a shared but not the exact same field. Therefore, I decided to include the `name` and `onLoan` fields in the `LibraryResource` class and remove them from the `game` and `book` classes to reduce duplication of code and hence increase efficiency. I also added the `creator` property to the `LibraryResource` class, however, I made it an abstract property which will be overridden in the children classes (`Game` and `Book`). This concept of overriding elements comes from the inheritance relationship between these classes, where the `LibraryResource` class is the parent class, and the `Game` and `Book` classes are the children classes which inherit features from their parent class but also have their own specific features. Eventually the concept of polymorphism is applied by implementing multiple inheritance relationships between objects.



```
public class Game : LibraryResource
public class Book : LibraryResource
```

Figure 5 Inheritance in C#

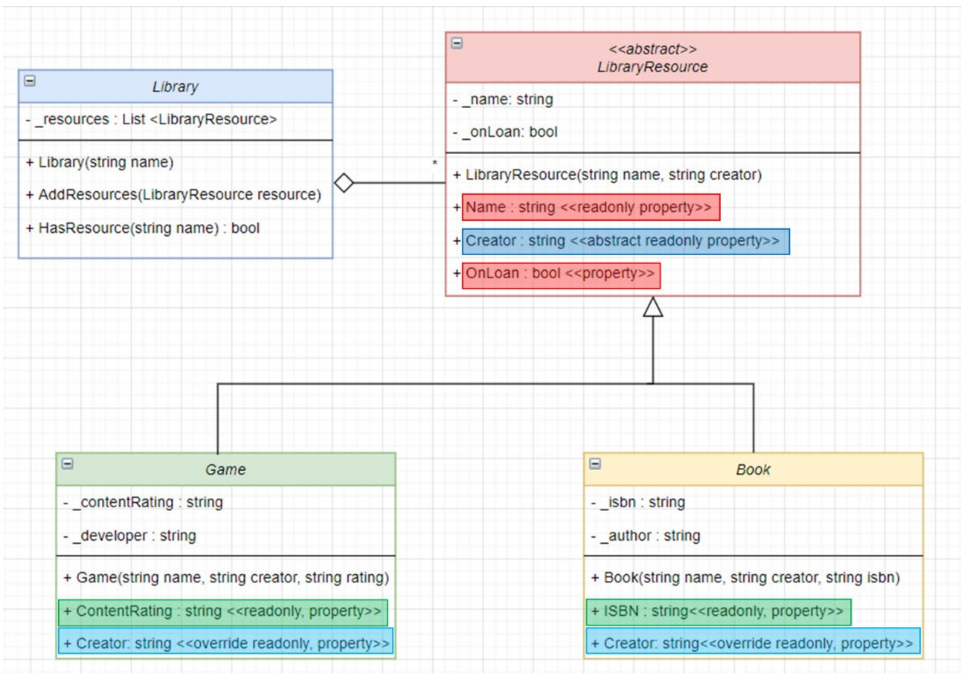
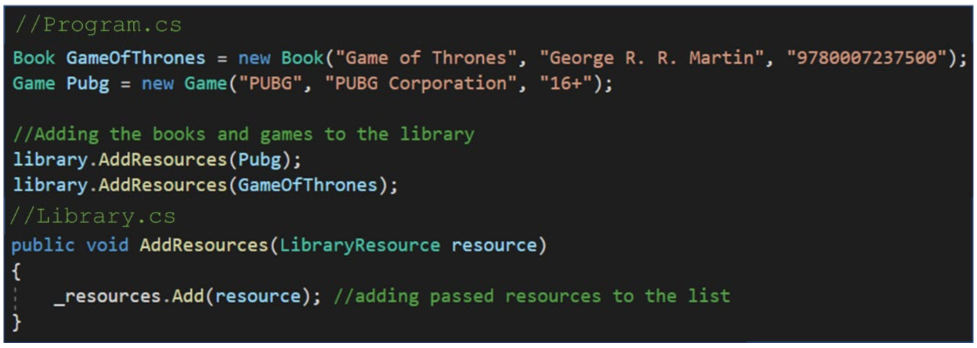

Library
- _resources : List <LibraryResource>
+ Library(string name)
+ AddResources(LibraryResource resource)
+ HasResource(string name) : bool
*
<<abstract>>
LibraryResource
- _name: string
- _onLoan: bool
+ LibraryResource(string name, string creator)
+ Name : string <<readonly property>>
+ Creator : string <<abstract readonly property>>
+ OnLoan : bool <<property>>
Game
- _contentRating : string
- _developer : string
+ Game(string name, string creator, string rating)
+ ContentRating : string <<readonly, property>>
+ Creator: string <<override readonly, property>>
Book
- _isbn : string
- _author : string
+ Book(string name, string creator, string isbn)
+ ISBN : string<<readonly, property>>
+ Creator: string<<override readonly, property>>

Figure 6 Inherited features (Name and OnLoan), Specific features (ContentRating and ISBN), and inherited but slightly changed features (Creator)

In addition, even though the abstract class cannot instantiate an object, it performs a powerful role in generalising other resources (`Game` and `Book`). The implementation of this abstract class initialises **Polymorphism** in our design. **Polymorphism** basically means many forms, and, in this case, it will generalise the `Game` and `Book` classes as being a `LibraryResource` so when we call one of the `AddResources` method from the `Library` class, we pass it either a Game or a Book object, and it will treat it as a `LibraryResource` object.



```
//Program.cs
Book GameOfThrones = new Book("Game of Thrones", "George R. R. Martin", "9780007237500");
Game Pubg = new Game("PUBG", "PUBG Corporation", "16+");

//Adding the books and games to the library
library.AddResources(Pubg);
library.AddResources(GameOfThrones);
//Library.cs
public void AddResources(LibraryResource resource)
{
    _resources.Add(resource); //adding passed resources to the list
}
```

Figure 7 Polymorphism example – AddResources() method from the Library class