**Build a Library**

Congratulations, you’ve passed the grueling rigmarole of librarian school and have become head librarian at your local Books-‘N-Stuff.

Just as you sit down, eager to utilize all those skills you learned in “Lib 203 - Shushing: How to Maintain Order While Spitting”, you realize you’re still using index cards to handle everything.

But no matter, you know some JavaScript, so let’s get to work modernizing your new digs.

Books-‘N-Stuff carries three different types of media: books, CDs, and movies. In this project you will create a parent class named Media with three subclasses: Book, Movie, and CD. These three subclasses have the following properties and methods:

**Book**

* **Properties**: author (string), title (string), pages (number), isCheckedOut (boolean, initially false), and ratings (array, initially empty).
* **Getters**: all properties have a getter
* **Methods**: .getAverageRating(), .toggleCheckOutStatus(), and .addRating()

**Movie**

* **Properties**: director (string), title (string), runTime (number), isCheckedOut (boolean, initially false), and ratings (array, initially empty)
* **Getters**: all properties have a getter
* **Methods**: .getAverageRating(), .toggleCheckOutStatus(), and .addRating()

**CD**

* **Properties**: artist (string), title (string), isCheckedOut (boolean, initially false), and ratings (array, initially empty), songs (array of strings)
* **Getters**: all properties have a getter
* **Methods**: .getAverageRating(), .toggleCheckOutStatus(), and .addRating()

If you’re looking for a challenge, try to create the four classes without using the steps below.

If you get stuck during this project or would like to see an experienced developer work through it, click “**Get Help**“ to see a **project walkthrough video**.

**Tasks**

**0/25Complete**

Mark the tasks as complete by checking them off

**Library Classes**

**1.**

Let’s start by making a parent class for our Book, CD, and Movie classes.

Create an empty class called Media.

Hint

Use the following syntax to create an empty class:

class emptyClass {

}

**2.**

Inside the Media class, create an empty constructor() method that takes one parameter.

This argument will set the one property that is in all three subclasses of Media, and does not have a default value.

Hint

Each of the three classes has properties for title, isCheckedOut, and ratings. However, the title property is the only one that does not have a default value.

Use the following syntax:

class emptyClass {

constructor(keyOne) {

}

}

**3.**

Inside the constructor, set the values for Media properties that Book, CD, and Movie share. Prepend all of the property names with an underscore (\_).

Hint

Set the \_title property to title, set the \_isCheckedOut property to false, and set the \_ratings property to an empty array. Use the following syntax:

class emptyClass {

constructor(keyOne) {

this.\_keyOne = keyOne;

this.\_keyTwo = 33;

this.\_keyThree = true;

}

}

**4.**

Create getters for the title, isCheckedOut, and ratings properties. Each getter should return the value saved to the matching property.

Hint

Use the following syntax to create getters:

class emptyClass {

constructor(keyOne) {

...

}

get keyTwo() {

return this.\_keyTwo;

}

}

**5.**

Create a setter for the isCheckedOut property.

Hint

Use the following syntax to create setters:

class emptyClass {

constructor(keyOne) {

...

}

set keyTwo(newKeyTwo) {

this.\_keyTwo = newKeyTwo;

}

}

**6.**

Under your getters, create a method named toggleCheckOutStatus that changes the value saved to the \_isCheckedOut property.

If the value is true, then change it to false. If the value is false, then change it to true.

Hint

You can use the following syntax to negate the value saved to a boolean.

methodName() {

this.\_booleanProperty = !this.\_booleanProperty;

}

**7.**

Under .toggleCheckOutStatus(), create a method named getAverageRating. Return the average value of the ratings array.

Use the [reduce](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/Reduce?v=example) method to find the sum of the ratings array. Divide this sum by the length of the ratings array, and return the result.

Take a look at the hint if you need help with the syntax for finding the sum of the ratings array.

Hint

The code below calculates the sum of the numbers saved to the ratings array.

let ratingsSum = this.ratings.reduce((currentSum, rating) => currentSum + rating, 0);

You can access the length of an array using .length (see below).

const lengthOfArray = myArray.length;

**8.**

Let’s add a method named addRating that accepts one argument and uses .push() to add it to the end of the ratings array.

Hint

Use the following syntax to add a value to the end of an array.

newValue(inputValue) {

this.values.push(inputValue);

}

**9.**

Next, we’’ll build a Book class that extends Media. If you feel comfortable building the Book class on your own, give it a shot. If not, use the steps below to help you along the way.

Whether you want to follow the steps or not, use the list of properties, getters, and methods as a reference.

**Book**

* **Properties**: author (string), title (string), pages (number), isCheckedOut (boolean, initially false), and ratings (array, initially empty).
* **Getters**: all properties have a getter
* **Methods**: .getAverageRating(), .toggleCheckOutStatus(), and .addRating()

Create an empty Book class that extends Media.

Hint

Use the following syntax to extend a class:

class ChildClass extends ParentClass {

}

**10.**

Inside the Book class, create a constructor that accepts three arguments. These arguments are used to set properties that do not have default values.

Hint

The constructor method must accept arguments for author, title, and pages.

**11.**

Call super on the first line of the Books‘s constructor method. Pass it any arguments that the parent constructor uses.

Hint

Pass title to the parent constructor. The parent constructor uses title to set the \_title property.

**12.**

Use the remaining arguments to set the author and pages properties in Book.

Hint

Use this.\_propertyName to set property values.

**13.**

Since our Book class inherits Media‘s properties and getters, we only need to create two new getters in the Book class.

Add two new getters to the Book class. Each getter should return the value saved to its matching property.

Hint

Use the following syntax to add getters for the author and pages properties.

get myProperty() {

return this.\_myProperty;

}

**14.**

Let’s see if you can create an entire Movie class using only the property, getter, and method specifications below:

**Movie**

* **properties**: director (string), title (string), runTime (number), isCheckedOut (boolean, initially false), and ratings (array, initially empty)
* **getters**: all properties have a getter
* **methods**: .getAverageRating(), .toggleCheckOutStatus(), and .addRating()

Take a look at the Hint to see step-by-step instructions.

Hint

Follow these steps to create a Movie class that extends Media:

* Create an empty Movie class that extends Media.
* Create a constructor that accepts arguments for director, title, and runTime.
* On the first line of the constructor, call super and pass it title.
* On the next two lines, set your \_director and \_runTime properties.
* Under the constructor, create getters for director and runTime. Each getter should return the value saved to that property.

**15.**

Create a Book instance with the following properties:

* Author: 'Bill Bryson'
* Title: 'A Short History of Nearly Everything'
* pages: 544

Save the instance to a constant variable named historyOfEverything.

Hint

Use the following syntax to create a new instance:

const myInstance = new myClass(arg1, arg2, arg3);

**16.**

Call .toggleCheckOutStatus() on the historyOfEverything instance.

**17.**

Log the value saved to the isCheckedOut property in the historyOfEverything instance.

**18.**

Call .addRating() three times on historyOfEverything with inputs of 4, 5, and 5.

**19.**

Call .getAverageRating() on historyOfEverything. Log the result to the console.

**20.**

Create a Movie instance with the following properties:

* Director: 'Jan de Bont'
* Title: 'Speed'
* Runtime: 116

Save the instance to a constant variable named speed.

Hint

Use the following syntax to create a new instance:

const myInstance = new myClass(arg1, arg2, arg3);

**21.**

Call .toggleCheckOutStatus() on the speed instance.

**22.**

Log the value saved to the isCheckedOut property in the speed instance.

**23.**

Call .addRating() three times on speed with inputs of 1, 1, and 5.

**24.**

Call .getAverageRating() on speed. Log the result to the console.

**25.**

Nice work! If you’ve made it this far, you clearly have a strong understanding of class syntax and inheritance.

If you would like to continue working on this project, we have listed some avenues to build on your existing progress.

* Add more properties to each class (movieCast, songTitles, etc.)
* Create a CD class that extends Media.
* In .addRating(), make sure input is between 1 and 5.
* Create a method called shuffle for the CD class. The method returns a randomly sorted array of all the songs in the songs property.
* Create class called Catalog that holds all of the Media items in our library.