Static Books

Here is a video walkthrough of the solutions for all parts.

Suppose we have the following Book and Library classes.

```java
class Book {
    public String title;
    public Library library;
    public static Book last = null;

    public Book(String name) {
        title = name;
        last = this;
        library = null;
    }

    public static String lastBookTitle() {
        return last.title;
    }
    public String getTitle() {
        return title;
    }
}
```

```java
class Library {
    public Book[] books;
    public int index;
    public static int totalBooks = 0;

    public Library(int size) {
        books = new Book[size];
        index = 0;
    }

    public void addBook(Book book) {
        books[index] = book;
        index++;
        totalBooks++;
        book.library = this;
    }
}
```

(a) For each modification below, determine whether the code of the `Library` and `Book` classes will compile or error if we **only** made that modification, i.e. treat each modification independently.

1. Change the `totalBooks` variable to **non `static`**
2. Change the `lastBookTitle` method to **non `static`**
3. Change the `addBook` method to **`static`**
4. Change the `last` variable to **non `static`**
5. Change the `library` variable to **`static`**

Solution:

1. Compile
   `totalBooks` is only used inside of a nonstatic function, so changing it to nonstatic would not cause compilation errors (although note that it no longer counts the total number of books correctly).
2. Compile
   Both static and nonstatic methods can access static variables, so changing `lastBookTitle` to be static would still allow it to access `last.title`.
3. Error
   Static methods cannot access instance variables, so changing `addBook` to be static would cause it to be unable to find the `books` or `index` variables.

4. Error
   Again, static methods cannot access instance variables, so changing `last` to be static would cause `lastBookTitle` to fail.

5. Compile
   Constructors are allowed to modify static variables; similarly, instances of a class can access that class's static variables. Thus, changing `library` to be static would not affect the `Book` constructor or `book.library` in `addBook`.

(b) Using the Book and Library classes from before, write the output of the main method below. If a line errors, put the precise reason it errors and continue execution.

```
public class Main {
    public static void main(String[] args) {
        System.out.println(Library.totalBooks);                ______________________
        System.out.println(Book.lastBookTitle());              ______________________
        System.out.println(Book.getTitle());                   ______________________

        Book goneGirl = new Book("Gone Girl");
        Book fightClub = new Book("Fight Club");

        System.out.println(goneGirl.title);                    ______________________
        System.out.println(Book.lastBookTitle());              ______________________
        System.out.println(fightClub.lastBookTitle());         ______________________
        System.out.println(goneGirl.last.title);               ______________________

        Library libraryA = new Library(1);
        Library libraryB = new Library(2);
        libraryA.addBook(goneGirl);

        System.out.println(libraryA.index);                    ______________________
        System.out.println(libraryA.totalBooks);               ______________________

        libraryA.totalBooks = 0;
        libraryB.addBook(fightClub);
        libraryB.addBook(goneGirl);

        System.out.println(libraryB.index);                    ______________________
        System.out.println(Library.totalBooks);                ______________________
        System.out.println(goneGirl.library.books[0].title);   ______________________
    }
}
```

Solution:

```
public class Main {
    public static void main(String[] args) {
        System.out.println(Library.totalBooks);                0
        System.out.println(Book.lastBookTitle());              Error, NullPointerException
        System.out.println(Book.getTitle());                   Error, does not compile

        Book goneGirl = new Book("Gone Girl");
        Book fightClub = new Book("Fight Club");

        System.out.println(goneGirl.title);                    Gone Girl
        System.out.println(Book.lastBookTitle());              Fight Club
12         System.out.println(fightClub.lastBookTitle());          Fight Club
13         System.out.println(goneGirl.last.title);                Fight Club
14
15         Library libraryA = new Library(1);
16         Library libraryB = new Library(2);
17         libraryA.addBook(goneGirl);
18
19         System.out.println(libraryA.index);                     1
20         System.out.println(libraryA.totalBooks);                1
21
22         libraryA.totalBooks = 0;
23         libraryB.addBook(fightClub);
24         libraryB.addBook(goneGirl);
25
26         System.out.println(libraryB.index);                     2
27         System.out.println(Library.totalBooks);                 2
28         System.out.println(goneGirl.library.books[0].title);    Fight Club
29     }
30 }
```

Explanation:
Line 3: The static variable `totalBooks` is initialized to 0.
Line 4: We haven't created any books yet, so the `Book` constructor has never been called, and `last` is `null`. When we attempt to call `lastBookTitle`, we access the `title` property of a `null` object, which results in a `NullPointerException`.
Line 5: You cannot call a nonstatic method using the class name; only instances of the class can call their instance methods.
Line 10: The string `"Gone Girl"` was passed into the constructor of the `goneGirl` object, so its title is `Gone Girl` (printing removes quotes).
Line 11: Whenever a new book is created, the static variable `last` points to it. Thus, `last` points to the most recently created book, `fightClub`.
Line 12: Instances of a class can access static variables.
`goneGirl.last` is the same as `Book.last`, which is `fightClub`.
Line 19: `index` gets incremented each time we call `addBook`, so after adding `goneGirl` to `libraryA`, its index is 1.
Line 20: `totalBooks` gets incremented each time we call `addBook`, so after adding `goneGirl` to `libraryA`, its `totalBooks` is 1. (Remember, instances can access a class's static variables).
Line 26: `index` gets incremented each time we call `addBook`, and it is an instance variable, so each library has its own copy of `index`. After adding `goneGirl` and `fightClub` to `libraryB`, its index is 2.
Line 27: `totalBooks` is a static variable, so on line 22, `totalBooks` gets reset to 0 for the entire class. Then, it gets incremented twice in `addBook` for a total of 2.
Line 28: In `addBook`, we set `book.library` equal to the library to which that book was *most recently added to*. `goneGirl` was most recently added

to `libraryB`, so its `library` is `libraryB`. Each library has its own `books` array which tracks books from oldest to newest addition. The first book added to `libraryB` was `fightClub`.