COMSC-205 SPRING 2024 ASSIGNMENT 2

Due March 3 at 11:55pm

Submission Checklist

Your final submission to Gradescope must have the following files:

- **Resource.java** - a parent class to base a library system off of
- **Book.java** - a child class
- **CD.java** - a child class
- **Library.java** - a class holding methods that manipulates arrays of Resources
- A **README.txt file** that clearly states:
  - Your name and the name of the assignment
  - Attribution for any sources used
  - Any notes on questions, difficulties, or bugs
- An **ImpactsReflection.pdf file** which contains your answers to the impacts reflection.

Background

You've been hired on a temporary basis by LITS, the Library, Information, and Technical Services department here at Mt. Holyoke. LITS runs the library system, using databases to organize all the different items that can be borrowed.

For your first job, they are asking you to make a database for lending items in their catalog. To start, they want you to make two types of lendable objects: Books and CDs. Each has a number of different pieces of information related to them. The database needs to be able to hold all those different pieces of information. Some information is the same: for example, every item has a title, and every item should be able to be checked out. Some of the information is different: Books have an Author, while CDs have an Artist.

We've provided you with example test code in the Main class, but no other starting code. Use the instructions to guide your planning, and use the example file to check your work. Read through the entire assignment and use the hints that follow to suggest an approach to developing your program.

The Assignment

This assignment is to write several classes and supporting code to represent a basic library that holds these two kinds of materials. Make use of inheritance to model the real-world relationship between the objects and their properties, which also will minimize repeated code.

Read through this. At the end, the Hints and Advice section gives ideas about how to

write the code a little at a time, testing as you go.

Resource

Resource.java should be an **abstract class** and a parent class for the two kinds of materials. It should have instance variables for:

- Title
- Publisher
- Year of publication
- Is it electronic or not
- How many copies the library has
- How many copies are currently available
- Who is currently borrowing it

Use your judgment about what type each field should be.

Resource should have the following methods:

- A **constructor** with this signature:
  ```
  public Resource (String title, String publisher, int year,
                   boolean electronic, int numCopies)
  ```
  - Be sure that all instance variables are initialized before the constructor returns
  - You can assume all parameter values are reasonable. For example, the number of copies will be >= 1.

- Three getters: for Title, Publisher, and Borrowers:
  ```
  public String getTitle()
  public String getPublisher()
  public String[] getBorrowers()
  ```

- Three is-methods:
  ```
  public boolean isElectronic()
  public boolean isAvailable()
  public boolean isBorrowedBy (String person)
  ```

- a **toString** method that returns a string containing the title and year separated by a comma, like “Data Structures, 2021”. This should override the method inherited from the Object class.

- a **checkOut** method to check out a resource. Checking out a resource requires as an argument a string of who it’s going to (the borrower). If the item is not available or no name was provided for the borrower, it should just return false. Otherwise, it should update the appropriate information and return true.
  ```
  public boolean checkOut (String borrower)
  ```

To check if the borrower name is provided you need to check two things: borrower is not null and not the empty string “”. To compare two strings, you use the .equals method. So, your if condition to see if the borrower value is valid should be:

```
if (borrower != null && !borrower.equals(“”))
```

To remember the name of the borrower, you will need to walk the array of borrowers to find an empty slot to put the new borrower in.

- a **checkIn** method to check in a resource. Since there may be multiple copies checked out, checking in a resource requires as an argument a string of the borrower who is checking in the resource. If the borrower passed as an argument does not have the resource checked out, just return false. Otherwise, update the appropriate information and return true.

  ```
  public boolean checkIn (String borrower)
  ```

Book, CD

Each of these classes should inherit from Resource. Their constructors should call the parent constructor.

Each class has different instance variables.

The *Book* specific variables are:

- Author
- Number of pages

Its constructor should have this signature:

```
public Book (String title, String publisher, int year,
    boolean electronic, int numCopies, String author, int numPages)
```

The *CD* specific variables are:

- Artist
- Length

Its constructor should have this signature:

```
public CD (String title, String publisher, int year, int numCopies,
           String artist, int length)
```

Assumptions about electronic:

- Books can be electronic or not electronic.

- Assume all CDs ARE NOT electronic.

These assumptions mean that when users create a Book, they will need to specify whether or not it is electronic, but for CDs, they do NOT need to specify this information.

Each class should override the toString method, printing out a short description. An example is below. The first example is for a book. The second is for a CD.

Ocean Vuong, On Earth We're Briefly Gorgeous, 2019, 256

Beyonce, Renaissance, 2022, 62

The information that toString returns should be formatted as shown here.

Library

This class should contain **one** instance variable to hold an array of Resource objects representing all of the books and CDs in the library. Keep in mind that a Book ***is a*** Resource, and a CD ***is a*** Resource. This means that CDs and Books can be kept in the same array.

Library should have a constructor that is passed in the maximum number of resources the library can have. It should have this signature:

```
public Library (int numResources)
```

Write the following functions:

- **addResource**, which takes a resource and adds it to the array of the library's resources. If the array is full, this method should return false. If it is not full, it should add it to the array and return true. It should have this signature:
  ```
  public boolean addResource (Resource r)
  ```

- **getAllResources**, which returns an array of all the resources in the library

- **getUnavailable**, which returns an array containing the subset of library resources that have no copies available. It should have this signature:
  ```
  public Resource[] getUnavailable ()
  ```

- **getAllUserHasCheckedOut**, which takes a String (the name of the borrower), and returns an array of *Resource* objects that the user has borrowed. It should have this signature:
  ```
  public Resource[] getAllUserHasCheckedOut (String user)
  ```

- **getAllBooks**, which returns an array containing the subset of *Resource* objects that are books. You can check what type an object is by using the instanceof operator. For example if res is a variable whose type is Resource, you can say:

```
if (res instanceof Book)
```

This will return true if res is a Book, and false if res is not a Book. It should have this signature:

```
public Resource[] getAllBooks ()
```

- **getAllCDs**, which returns an array containing the subset of *Resource* objects that are CDs. It should have this signature:

  ```
  public Resource[] getAllCDs ()
  ```

Each function should return an array that contains only the Resources that it needs to; it should not have null values. You will need to first figure out how many Resources are needed to create your new, smaller array, before going through and adding to that array!

Main

We provide a Main class with a main method in it. The main method should be used to test that your code works. Feel free to edit the main class to do more or different testing. Initially, you may want to comment out the code that we provide so you can test individual methods. The Main class contains a method called printResources that will display whatever is in an array of resources that is passed to it.

When you submit your code on gradescope, more tests will be done than is main does, so adding more tests of your own would be a good idea.

Here is what main has in the starter:

```
public class Main {
   private static void printResources(Resource[] res) {
      for (int i = 0; i < res.length; i++) {
          System.out.println (res[i]);
      }
   }

   public static void main(String[] args) {
      Library l = new Library (6);
      Book b1 = new Book ("The Fifth Season", "Orbit", 2015, false, 1,
                  "N.K. Jemisin", 378);
      CD cd1 = new CD ("Wrong Places", "RCA Records", 2020, 1, "H.E.R.", 38);
      Book b2 = new Book("On Earth We’re Briefly Gorgeous", "Penguin Press",
          2019, false, 3, "Ocean Vuong", 256);
      Book b3 = new Book("Small Gods", "Harper Collins", 1992, true, 5,
                  "Terry Pratchett", 400);
      CD cd2 = new CD("Harry's House", "Real World", 2022, 2, "Harry Styles",
           42);
      CD cd3 = new CD("Renaissance", "Parkwood Entertainment", 2022, 4,
           "Beyonce", 62);
```

```
      l.addResource(b1);
      l.addResource(cd1);
      l.addResource(b2);
      l.addResource(b3);
      l.addResource(cd2);
      l.addResource(cd3);

      System.out.println ("*** All resources ***");
      printResources(l.getAllResources());

      b1.checkOut("Barbara");
      b3.checkOut("Barbara");
      cd3.checkOut("Dovan");

      System.out.println ("\n*** All unavailable ***");
      printResources(l.getUnavailable());

      System.out.println ("\n*** All checked out by Barbara ***");
      printResources(l.getAllUserHasCheckedOut("Barbara"));

      System.out.println ("\n*** All books ***");
      printResources(l.getAllBooks());

      System.out.println ("\n*** All CDs ***");
      printResources(l.getAllCDs());

   }
}
```

And here is the output that is produced:

```
*** All resources ***
N.K. Jemisin, The Fifth Season, 2015, 378
H.E.R., Wrong Places, 2020, 38
Ocean Vuong, On Earth We’re Briefly Gorgeous, 2019, 256
Terry Pratchett, Small Gods, 1992, 400
Harry Styles, Harry's House, 2022, 42
Beyonce, Renaissance, 2022, 62

*** All unavailable ***
N.K. Jemisin, The Fifth Season, 2015, 378

*** All checked out by Barbara ***
N.K. Jemisin, The Fifth Season, 2015, 378
Terry Pratchett, Small Gods, 1992, 400

*** All books ***
N.K. Jemisin, The Fifth Season, 2015, 378
Ocean Vuong, On Earth We’re Briefly Gorgeous, 2019, 256
Terry Pratchett, Small Gods, 1992, 400
```

```
*** All CDs ***
H.E.R., Wrong Places, 2020, 38
Harry Styles, Harry's House, 2022, 42
Beyonce, Renaissance, 2022, 62
```

Documentation and Organization

Comments are required for this and all future assignments. Make sure every method has a doc comment, and that your code is well-organized. Refer back to Lab 3 for more.

Hints and Advice

It’s a bad idea to try writing every class at once, and only then testing. A suggested order of operations:

1. Write Resource.java. Start with the variables, the constructor, and toString. Make this class non-abstract for now.
2. Comment out the code currently in the main method and write some test code that creates simple Resource objects and prints them out.
3. Write the addResource and getAllResources methods in Library.
4. Change the main method to add the resources you create to the library, call getAllResources, then call printResources provided in the Main class.
5. Keep adding and testing methods in Resource until you have them all.
6. Write one of the child classes, such as Book.java. Start with the variables, the constructor, and toString. Change Resource to be abstract.
7. In the main method, replace the Resources with Books. Make sure everything still works as expected. The array can still be of type Resource, since all Books are Resources
8. Write the second child class, CD.
9. In the main method of LibraryTester, add some CDs to the array. Again, make sure things still work.   Your array should be of Resources, since both CDs and Books are Resources, so both can be added to arrays.
10. Then, move on to writing the additional methods in the library class. Write one method at a time, testing each in main after you write it.

Null Pointer Exceptions

You can’t call a method on an object that’s null, or you’ll get a NullPointerException. A good check to include before calling a method is testing if a variable is null. The check can be written like so:

```
if (resources[i] != null && !resources[i].isAvailable()) {
        …
}
```

Evaluation:

This section contains a summary of how your program will be evaluated.

Your Java program will be evaluated on:

- Correctness - Do the constructor and methods behave consistently with the description above?
- Array usage - Are arrays used and updated correctly?
- Loops - Are loops used appropriately?
- Inheritance - Is inheritance used appropriately?
- Documentation: Are there comments present, describing what the code does?
- Style: Are variables named appropriately? Is the indentation consistent?

Reflection on Impacts

You should submit a PDF containing 2-3 short paragraphs of reflection or separate responses to each question.

In your reflection, answer each of the following prompts:

- As you saw through the theming of this assignment, libraries are asked to keep track of a lot! Thinking about your experience working on this assignment, what challenges would there be if the library also had a collection of magazines with many years of issues? How well would the existing design work for that?

- It seems necessary for a library to remember who currently has an item borrowed. Suppose a library kept a long-term record of which items a library user has borrowed, even after they have been returned. What might be a benefit to the library user of having this feature? What might be a risk of this feature?

To get credit for your reflection, we are looking for both **evidence** (what are you basing your responses on or responding to?) **& interpretation** (your explanation or analysis of the evidence) in your answers. In other words, to receive full credit, **it is not enough to say that something is good, bad, or in between – you must provide an explanation of *why*.** What impacts or consequences do you see? Who (or what) is harmed?