CSCI 1933 Lab 4

Problem Solving with Arrays

Lab Rules

You may work individually or with *one* partner in lab. We suggest that you have a TA evaluate your progress on each milestone before you move onto the next. The labs are designed to be completable by the end of lab. If you are unable to complete all of the milestones by the end of lab, you have **until the last office hours on Monday** to get those checked off by **any** of the course TAs. We suggest you get your milestones checked off as soon as you complete them since Monday office hours tend to become crowded. If you worked with a partner, both of you must be present when getting checked off to receive credit. You will only receive credit for the milestones you have checked off by a TA. There is nothing to submit to Moodle for this lab.

Attendance

Per the syllabus, students with more than 3 unexcused lab absences over the course of the semester will automatically fail the course. The TAs will take attendance during lab; it is expected that students will be actively working on the lab material. *Your physical presence in lab is not sufficient to be marked as present for the purposes of attendance.*

The equals method – We kind of lied

Since the beginning of the course, we've been telling you that you should be using the `equals()` method to check if two objects are equal. This is because using `==` on two objects checks if the two objects share the same memory location. Well, the `equals` method does the same thing **if you don't write it yourself**. Remember the `BankAccount` class? Before reading any further, try using `equals` on two seemingly equal `BankAccounts`, what happens?

Hopefully you tried it out before reading the following. The reason you are seeing that behavior is because if you do not write the `equals` method in your class, e.g., `BankAccount`, the behavior of the `equals` method is the same as using `==`. Therefore, to get the desired behavior, you have to define what it means for two objects, e.g., `BankAccounts`, to be equal. That is, do the `passwords`, `balance`, and `name` have to match, or just a subset of them?

1 RateMyBooks

Books are cool, but picking one out of many can sometimes be difficult because some have amazing stories while others, not so much. To make picking books easier, many websites that sell/rent out books have some sort of rating system in place. For this milestone, we are going to implement a simple 5-star rating system. Before we can do so, we need to model our books. Create a `Book` class with the following member variables: `title, author, genre`, and `numPages`. The `Book` class should also implement the following methods:

- `public void addRating(int rating)` – This will add the given rating to the book. You may assume that `rating` is between 1 and 5 (inclusive).

  > **Constraint:** You are only allowed to create one additional member variable to keep track of the individual ratings.

- `public double getAverageRating()` – This will return the average of all of the ratings this book has accumulated. Return 0 if there are no ratings.
- `public int getNumRatings()` – This returns the number of ratings this book has
- `public boolean equals(Book other)` – This will return true if the book's title, author, and average rating are equal to other's title, author, and average rating.
- `public String getRatingSummary()` – This will return a String in the following format.

```
Avg Rating : 3.69
1 | ***
2 |
3 | **
4 | *
5 | *******
```

  The actual average and number of stars will depend on the book's ratings. You do not have to worry about rounding the average to two decimal places. However, if you are curious, look up how `String.format()` works.

> **Milestone 1:** Create three or more `Books` (you'll need them for the next milestone), each with a different `title, author, genre, numPages,` and set of ratings. Show them to a TA and be prepared to explain how you approached `addRating` and `getRatingSummary`

2 A Smart Bookshelf

Ok, so we have a rating system to easily distinguish good books from bad books. But what about picking books that cater to our preferences? For example, if you are only interested in non-fiction books, why should you care about a sci-fi book that has five stars? Now we need to implement some sort of filtering feature – we are going to do this by creating a virtual bookshelf. Before we can start, we first need to model our preferences by creating a `ReaderProfile` class with the following member variables:

- `String favoriteAuthor`
- `String favoriteGenre`
- `double pagesPerMinute` – How many pages this reader can read per minute
- `int desiredTime` – The maximum number of minutes the reader would like to finish the book by, e.g., Alice only wants to read books that take ≤ 360 minutes to finish.

After creating your `ReaderProfile` class, create a `Bookshelf` class with the following methods:

- `public Bookshelf(Book[] books)` – This will set up your `Bookshelf` to hold the books that are passed in.
- `public Book[] getBooksByAuthor(String author)` – This will return all of the books that are written by `author`. If there are no books written by `author`, return an empty array. The length of the array must be exactly the number of books authored by `author`
- `public Book[] getBooksFor(ReaderProfile reader)` – This will return all of the books that satisfies all of the `reader`'s preferences. If there are no books that satisfies the `reader`, return an empty array. The length of the array must be exactly the number of books that satisfy the `reader`'s preferences.
- `public Book getMostPopularBook()` – This will return the `Book` with the most number of ratings. Break ties however you want to.

Milestone 2: Create a `ReaderProfile` for yourself. Be prepared to show off your `Bookshelf` using your `ReaderProfile` to a TA. Also be prepared to answer the following questions:

1. What happens if we pass in an array of books where some of the slots are empty? For example, we pass in an array of length 10, but we only have it hold 5 books.
2. What does the length of an array tell you? What does it not tell you?

3 Heat Grids

Heat grids are typically used to show the effects of a source on its surrounding areas. Take a campfire for example. The closer you are to the campfire, the hotter is becomes. Conversely, the further you are, the colder it becomes. A more complex example that uses intensity in-place of heat are weather radars to show how widespread/severe a thunderstorm can be. For this milestone, we are going to use a 2-D array to make a simple heat grid. To do so, we are going to have several heating and cooling sources which are shown in Table 1.

	Icon	Heat	Range	Decay
light-bulb	`"l"`	+1	0	0.00
campfire	`"c"`	+4	2	0.50
furnace	`"f"`	+10	4	0.35
ice-cube	`"i"`	-2	1	0.50
refrigerator	`"r"`	-8	3	0.20
glacier	`"g"`	-20	5	0.15

Table 1

	0	1	2	3	4	5	6	7	8
0		1	1	1	1	1			
1		1	2	2	2	1			
2		1	2	4	2	1			
3		1	2	2	2	1			
4		1	1	1	1	1			
5									
6									
7									

Figure 1

As an example, suppose we have a campfire at (x, y) = (3, 2) on a 9×8 grid. Using its properties from Table 1, we would have a grid that looks like Figure 1.

Notice how (3, 2) has a heat of +4 since that is where the campfire is located. Since a campfire has a range of 2 and a decay of 0.50, every square that is a distance of one away from the source has a heat of 2. Likewise, every square that is a distance of two away has a heat of 1.

Since the goal is to build a heat grid and not to figure out the math behind computing the decayed heat, we are giving you the code to compute the heat for the squares surrounding the source:*

```
// In Figure 1, we can pass in 0.50, 4, and 2 to compute the heat for (4, 0)
private int decayHeat(double decay, int heat, int distance) {
    return (int) (heat * Math.exp(-1 * decay * distance));
}
```

Hint: The distance is nothing more than just the "current range" you are working on, e.g., 1, 2, . . . , `range`.

*This is an exponential decay if you were curious

When multiple sources are added to the grid, the effects from each source may overlap. For example, suppose we add an ice-cube to (6, 2), then the grid would look like Figure 2. Notice how some of the rows in column 5 have turned to 0. This is because (5, 1), (5, 2), and (5, 3) are within the range of the campfire and the ice-cube, thus their heats are added together.

	0	1	2	3	4	5	6	7	8
0		1	1	1	1	1			
1		1	2	2	2	0	-1	-1	
2		1	2	4	2	0	-2	-1	
3		1	2	2	2	0	-1	-1	
4		1	1	1	1	1			
5									
6									
7									

Figure 2

	0	1	2	3	4	5	6	7	8
0		1	1	1	1	1			
1		1	2	2	2	0	-1	-1	
2		1	2	5	2	0	-2	-1	
3		1	2	2	2	0	-1	-1	
4		1	1	1	1	-3	-4	-4	-4
5						-4	-5	-5	-5
6						-4	-5	-6	-6
7						-4	-5	-6	-8

Figure 3

We also need to consider when sources are placed along the edges or have part of their range extend beyond the grid. Suppose we add a refrigerator to (8, 7), then the grid will look like Figure 3.

To implement the heat grid, create a `HeatGrid` class and implement the following methods:

- `public HeatGrid(int width, int height)` – This will construct your 2-D array to represent your heat grid; each square should start off with a heat of 0.
- `public boolean placeSource(String src, int x, int y)` – If the coordinates are out-of-bounds or there is already a source there, then do nothing and return `false`. Otherwise, this will place the `src` with its respective properties given in Table 1 onto your heat grid, update the surround squares, and return `true`. You may assume that `src` will be one of the six icons in Table 1.
- `public int[][] getHeats()` – This will return your grid with the current heat values.
- `public int getHeat(int x, int y)` – This will return the heat of a particular square in your grid. You may assume that the coordinates are within the boundaries of your grid.
- `public int getNetHeat()` – This will return the sum of all the heat values on the grid.

Tip: Unless we say otherwise, you are always free to add more to your code than what is listed above to help you debug or break down the problems.

Continued onto the next page

Milestone 3: Create two `HeatGrid`s – one that is identical to Figure 3 and one that is completely blank. We will ask you to place one to many sources on the blank one as you are getting checked off. Be prepared to answer the following questions:

1. How did you approach `placeSource`? Do not read off your code, give us a high level explanation.
2. Suppose we have a 2-D array called `foo`. Is there a scenario where the following snippets of code behave differently? Why or why not?

```
// Snippet 1
for (int i = 0; i < foo.length; i++) {
    for (int j = 0; j < foo[0].length; j++) {

    }
}

// Snippet 2
for (int i = 0; i < foo.length; i++) {
    for (int j = 0; j < foo[i].length; j++) {

    }
}
```

Reflection: Why would passing in a `String` to `placeSource` not be the best way to go about this? What is a better approach and why?