Meme Magic
Sprint 4 - Feature Add 1

Your users have asked for their first additional feature. They love the Meme Magic system so far, but they would really like the ability to sort Memes in their Feeds.

First, set up your project:

- Create a new Java project on Eclipse, and call it `MemeMagic4`.
- Copy and import all of your `MemeMagic3` files into the `MemeMagic4` project.
- Maintain a copy of `MemeMagic3` files (i.e., do not overwrite them.)

Note: you may alternatively copy your project in Eclipse by selecting the `MemeMagic3` project in the Package Explorer, copy with Ctrl+c (or Command+c on mac), and paste with Ctrl+p (or Command+p on mac). On pasting, Eclipse will ask you to name the newly-pasted project.

Learning Goals

In this assignment, we will practice:

1. Implementing the `Comparable` interface to produce a natural ordering of objects
2. Implementing the `Comparator` interface to produce additional orderings of objects
3. Sorting lists of objects utilizing the `Comparable` and `Comparator` interfaces
4. Using `TreeSet` to store unique objects defined by `hashCode()` and `equals()`
5. Accessing and referencing Java API for objects

Integration

First, we'll need to provide a natural ordering for Memes, Users, and BackgroundImages. We will also make some updates to the classes with our new knowledge of the Java Collections Framework.

BackgroundImage

- `compareTo(BackgroundImage)` - implement the Comparable interface and provide the compareTo method that orders BackgroundImages as follows:
    - First, by imageFileName (ascending)
    - Then, if imageFileNames are identical, by title (ascending)
    - Lastly, if imageFileNames and titles are identical, by description (ascending)

Meme

- `compareTo(Meme)` - implement the Comparable interface and provide the compareTo method that orders Memes as follows:
    - First, by caption (ascending)
    - Then, if captions are identical, by BackgroundImage (using its natural ordering)
    - Then, if BackgroundImages are identical, by overall rating (descending)
        - **Note:** "overall rating" is defined as the result of `calculateOverallRating()`.
    - Lastly, if overallRatings are identical, shared memes should come first
- **Note:** Remember that compareTo returns an int; when comparing doubles or booleans, we'll need to return an appropriate int value.

User

- `compareTo(User)` - implement the Comparable interface and provide the compareTo method that orders Users as follows:
    - First, by userName (ascending)
    - Then, if userNames are identical, by the number of memes created (descending)
- Encapsulation: Since we want the list of Memes viewed by the user to be unique, replace the field (instance variable) `ArrayList<Meme> memesViewed` with a `TreeSet<Meme>`. However, we do not want to change the interface we're providing to other classes (encapsulation), so update the getters and setters for memesViewed to correctly return an `ArrayList` instead of a `TreeSet`.
  Notes: Please remember to update the Constructors where you instantiate `memesViewed`. Also, see the Java API for the `Collection` interface for helpful methods.

Implementation

We will also need some Comparators to allow us to sort Meme objects in different ways. Create and implement the following classes that implement the Comparator interface:

CompareMemeByRating

- `compare(Meme, Meme)` - compares two Memes and provides the following ordering:
    - First, by overall rating (descending)
    - Then, if overall ratings are identical, by caption (ascending)
    - Then, if captions are identical, by BackgroundImage (using its natural ordering)
    - Lastly, if BackgroundImages are identical, order by creator (using natural order)

CompareMemeByCreator

- `compare(Meme, Meme)` - compares two Memes and provides the following ordering:
    - First, by creator (using User's natural ordering)
    - Then, if creators are identical, by overall rating (descending)
    - Then, if overall ratings are identical, by caption (ascending)
    - Then, if captions are identical, by BackgroundImage (using its natural ordering)
    - Lastly, if BackgroundImages are identical, shared memes should come first

Create a new class, `OrderableFeed`, that extends the Feed class and provides methods for the sorting of Memes in the feed.

OrderableFeed

Include each of the following public sort functions in the OrderableFeed class. They should take **no** parameters and return `void`. They will re-order the instance's list of memes.

- `sortByCaption()` - reorders the feed by caption, using Meme's natural ordering
- `sortByRating()` - reorders the feed by rating, using the CompareMemeByRating comparator
- `sortByCreator()` - reorders the feed by creator, using the CompareMemeByCreator comparator

Main Method Testing

In this sprint, we expect you to do your own main method testing. Although the amount of main method testing is not limited, please provide **at least two tests each** for the following:

- `BackgroundImage`, `Meme`, and `User`'s `compareTo` methods
- `CompareMemeByRating` and `CompareMemeByCreator`'s `compare` methods
- `OrderableFeed`'s three sort methods
- `OrderableFeed`'s inherited `getNewMeme` method to check for ordering

Note: Your main method should have enough testing to provide sufficient evidence to determine that the behavior implemented matches the behavior described above. It is *highly recommended* to write the tests before implementation as it will help you to understand the exact behavior and what is the expected output of different inputs.

Our goal in this assignment is to encourage main method testing so that you know your code works well before submitting to Gradescope. Therefore, you will only be able to submit to Gradescope a total of 10 times for this assignment.

Additional Resources

The following resources may be helpful when completing and submitting this sprint.

- JavaDoc style documentation for each of the classes and methods described above
- One example of main method testing for `Feed`'s `getNewMeme` method in `OrderableFeed` after sorting
- Java API for Collection interface, Comparable interface, Comparator interface
- Video on submitting to Gradescope

Submission Information

Method and Class Naming: You must match method names, instance variable names, and data types exactly. You must use correctly formatted Java code. Declare fields in the class definition, and create a default constructor for each class that initializes every instance variable. For methods that you are overriding (i.e., `compare( )` and `compareTo( )`), use the `@Override` annotation before the method header.

Coding Style: In real-world software development, it is paramount to create readable and easily maintainable code. That is typically achieved through the use of style and commenting guidelines. Since you will be updating this code over the next few weeks, we have provided a style guide and formatting guide that we strongly encourage you to follow:

- Coding Style Guide (includes installation instructions for Eclipse)
- Eclipse Style File

Submitting: Upload your Eclipse project (the `.java` files) to the "Meme Magic 4" assignment on Gradescope. You should submit `User.java`, `Feed.java`, `Rating.java`, `BackgroundImage.java`, `Meme.java`, `OrderableFeed.java`, `CompareMemeByRating.java`, and `CompareMemeByCreator.java`. This submission utilizes an autograder to check that your code follows these specifications. If it spots an issue, it will alert you, but you should **NOT** use the submission system as your testing. We encourage you to test your code during the implementation phase. Therefore, you may upload your code a **maximum of ten (10) times**.

Note: After the 10th submission, Gradescope will still allow submissions, but they will NOT be graded by the system.

Grading Rubric

The assignment will be worth a total of 100 points:

80 points - Method and implementation correctness, auto-graded using Gradescope
15 points - Main method testing
5 points - Code readability (organized, well-indented, readable by others)

Academic Integrity and Moving into Sprint 5

After submitting this sprint (*and after the late submission deadline*), you are encouraged to collaborate with your group members on what you missed and you *are* allowed to share code for those parts. However, everyone who collaborates **must have already submitted their code and *may not resubmit* to this sprint**.

You *must* understand any code you incorporate, as you'll be using it to complete the next (and subsequent) sprint. We will *not* be directly testing the code from this sprint again. You **must** list your collaborators in the comment at the top of each file that has any amount of shared code.