Meme Magic
Sprint 2 - Build 1

You are proud of your team, the design is solid and ready to be built. We will sprint out the build into three phases, each with a scenario test. This first sprint will include the `Meme`, `BackgroundImage`, and `Rating` classes.

First, set up your project:
- Create a new Java project on Eclipse, and call it `MemeMagic2`.
- Copy all of your `MemeMagic1` files into the `MemeMagic2` project.
- Maintain a copy of `MemeMagic1` files (i.e., do not overwrite them.)

Note: you may alternatively copy your project in Eclipse by selecting the `MemeMagic1` 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 default and overloaded constructors
2. Implementing getter and setter (accessor and mutator) methods
3. Utilizing an array to store reference objects
4. Experiencing inheritance from Object: overriding toString() and equals() methods
5. Incorporating main-method testing to check correctness (i.e., behavior matches specifications)

Implementation

Implement methods for the three classes as described below. When creating a constructor, if a class has more fields than specified in the constructor, initialize these fields with default values. Note that some of these classes make use of the User class, which has method stubs but is not fully implemented at this point. We will implement it in the next sprint.

BackgroundImage

- Create a constructor that accepts an `imageFileName`, a `title`, and `description` as arguments

- Implement all getters and setters
- `toString()` - returns "title <description>"
    - ex:
      How bots laugh <Image of Joquain Phoenix in his role as Joker, laughing maniacally>
- `equals(Object)` - return true if the parameter is a BackgroundImage object and this object's `title`, `description`, and `imageFileName` all match those of the parameter.

Meme

- Create a constructor that accepts a `backgroundImage`, `caption`, and `creator` (User) as arguments. It must initialize `ratings` to an array of size 10 and the `captionVerticalAlign` to "bottom".
- Implement all getters and setters
- `toString()` - returns "backgroundImage 'caption' *overallRating* [+1: the number of +1 ratings, -1: the number of -1 ratings]"
    - ex:
      How bots laugh <Image of Joquain Phoenix in his role as Joker, laughing maniacally> 'When your professor calls an in person meeting at 9 AM EST and you live in Cali' 5.0 [+1: 6, -1: 1]
    - **Note:** overallRating is the value returned by calculateOverallRating()
    - **Hint**: there is a lot going on in this string. Consider how additional *private* helper methods might make this easier to read.
- `equals(Object)` - return true if the parameter is a Meme object and both instances match on `creator`, `caption` and `backgroundImage`
    - **Note**: since we have not specified the equality of `User` objects, you may need to update your **`User`** class' `equals()` method to use its parent's method with the super keyword:

      ```
      public boolean equals(Object o) {
            return super.equals(o);
      }
      ```

      By making this change, two `User` objects will be equal if they are the same object in memory, since the parent of `User` is `Object`. *We'll relax that constraint in Sprint 3 when we specify the functionality of the **`User`** class.*
- `calculateOverallRating()` - return a double that is a sum of all rating scores for this Meme. If a Meme has no ratings, 0.0 should be returned.
- `addRating(Rating)` - adds the Rating object to the Meme's array of ratings, returning true if successful and false otherwise. If the array is full, it must shift all ratings one position up and insert the new one at the last position in the array. It will discard the original first element.

- `setCaptionVerticalAlign(String)` - This setter requires specifically-allowed values only. The only allowed values for `captionVerticalAlign` are: “top”, “middle”, and “bottom”, which define the placement of the caption on the background image. This method will therefore return a boolean: true if the captionVerticalAlign could be updated based on the given parameter, and false otherwise.

Rating

- Create a constructor that accepts a `user` and a `score`
    - Note the requirements for the `score` defined below in `setScore()`. If an improper score value is given, then set the score to 0.
- Implement all getters and setters not explicitly specified below
- `toString()` - returns "Rating was *type_of_rating*"
    - Ex: if the score is +1, then it will return: *Rating was an upvote*
    - Ex: if the score is -1, then it will return: *Rating was a downvote*
    - Ex: if the score is 0, then it will return: *Rating was a pass*
    - Remember: We have not implemented the `User` class, so we cannot include it in the String at this point. We’ll have to do some integration tasks later!
- `equals(Object)` - returns true if the parameter is a Rating object and this Rating’s `score` and `user` are equal to those of the parameter.
    - **Note**: since we have not specified the equality of `User` objects, you may need to update your **`User`** class’ `equals()` method to use its parent’s method with the super keyword:

      ```
      public boolean equals(Object o) {
            return super.equals(o);
      }
      ```

      By making this change, two `User` objects will be equal if they are the same object in memory, since the parent of `User` is `Object`. *We’ll relax that constraint in Sprint 3 when we specify the functionality of the **`User`** class.*
- `setScore(int)` - This setter requires specifically-allowed values only. Users can upvote, downvote or pass on a rating, giving a score of +1, -1, or 0, respectively. Ensure this method will *only* accept those values. This method will therefore return a boolean: true if the score could be updated based on the given parameter, and false otherwise.

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:

- The three (3) new constructors we’ve asked for in this sprint
    - One each for BackgroundImage, Meme, and Rating
- BackgroundImage, Meme, and Rating’s `toString()` methods

- BackgroundImage, Meme, and Rating's `equals()` methods
- Meme's `calculateOverallRating()`
- Meme's `addRating()`
- Meme's `setCaptionVerticalAlign()`
- Rating's `setScore()`

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. That will be especially important for corner cases, such as a full or empty array for `addRating()`.

You will only be able to submit to Gradescope a total of 10 times for this assignment; please test your code's functionality with main method testing before submitting.

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
- 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., `equals()` and `toString()`), 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 2" assignment on Gradescope. You should submit `BackgroundImage.java`, `Meme.java`, `Rating.java`, along with `User.java` ("as is" from Sprint 1), and `Feed.java` ("as is" from Sprint 1). This submission utilizes an autograder to check that your code follows these specifications. If it spots a disconnect or bug, it will alert you, but you should **NOT** use the submission system as your testing. Testing should be done during the implementation phase. Therefore, for this assignment, 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
(we will only be checking BackgroundImage, Meme, and Rating classes)
15 points - Main method testing
5 points - Code readability (organized, well-indented, readable by others)

Academic Integrity and Moving into Meme Magic 3

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.