Meme Magic
Sprint 1 - Design

The year is 2006 and Google has just bought YouTube. As a budding tech entrepreneur, you've had a great idea for a killer new app that allows you to take a photo, apply a funny caption and share it with your friends. You call it a “meme.”

To get started with your app, you're going to need to design some classes. You sit down with your design team and get to work. You decide to model your app with Users and Memes. Each Meme will have a BackgroundImage and the associated caption. Users can assign Ratings to Memes and share Memes on a Feed so that others can view and laugh at their creations.

Learning Goals

In this assignment, we will practice:

1. Creating UML (Unified Modeling Language) diagrams
2. Implementing class design consisting of method stubs
3. Utilizing style guidelines for increased readability and consistency
4. Including appropriate comments for clarity and readability

Designing the System

To begin with, you will use **Unified Modeling Language** to design the classes. You may create these using text editors, an image editor, or standard office program with drawing tools. Export the finished UML diagram(s) as **PDF** before submitting. **Only PDF files of your UML will be graded.**

Create a UML document that reflects the following classes with indicated state and behavior. You do not need to illustrate inheritance and composition for these documents as we have not fully covered that material. We have also not yet covered the `ArrayList<>` types seen below, but we include them for completeness.

All instance variables should be private. You should include getters and setters for every instance variable (i.e., field). These are often left out of UML documents, but it is good practice for you! For each class, include a simple constructor with no parameters. All methods should be public. All methods have a return type of void unless otherwise specified. Classes do not need a main method. No methods should be static.

User

Create a class `User`. The `User` class requires three fields (i.e., instance variables). Their type is indicated in italics:

- userName *String*
- memesCreated *ArrayList<Meme>*
- memesViewed *ArrayList<Meme>*

The `User` class requires the following eight methods in addition to a getter and setter method for each instance variable mentioned above:

- rateMeme (accepts a *Meme* and an *int* (rating))
- createMeme (accepts a *BackgroundImage* and a *String* (caption), returns a *Meme*)
- deleteMeme (accepts a *Meme*, returns a *boolean*)
- shareMeme (accepts a *Meme* and a *Feed*)
- rateNextMemeFromFeed (accepts a *Feed* and an *int* (ratingScore))
- calculateReputation (returns a *double*, by default use 0.0)
- toString (returns a *String*)
- equals (accepts an *Object*, returns a *boolean*)

BackgroundImage

Create a class `BackgroundImage`. The `BackgroundImage` class requires three instance variables:

- imageFileName *String*
- title *String*
- description *String*

`BackgroundImage` requires two methods, plus a getter and setter for each instance variable:

- toString (returns a *String*)
- equals (accepts an *Object*, returns a *boolean*)

Meme

Create a class `Meme`. The `Meme` class requires six instance variables:

- creator *User*
- backgroundImage *BackgroundImage*
- ratings *Rating[]*
- caption *String*
- captionVerticalAlign *String*
- shared *boolean*

The `Meme` class requires the following methods, plus a getter and setter for each instance variable:

- addRating (accepts a *Rating*, returns a *boolean*)
- calculateOverallRating (returns a *double*)
- toString (returns a *String*)
- equals (accepts an *Object*, returns a *boolean*)

Rating

Create a class `Rating`. The `Rating` class requires two instance variables:

- score *int*
- user *User*

The `Rating` class requires the following methods, plus a getter and setter for each instance variable:

- toString (returns a *String*)
- equals (accepts an *Object*, returns a *boolean*)

Feed

Create a class `Feed`. The `Feed` class requires one instance variable:

- memes *ArrayList<Meme>*

The `Feed` class requires the following methods plus a getter and setter for the lone field.

- getNewMeme (accepts a *User*, returns a *Meme*)
- toString (returns a *String*)

Creating Starter Code

Once you have created the UML description document(s), create a new Java Project in Eclipse for `MemeMagic1` and create the classes you just designed as `User.java`, `BackgroundImage.java`, `Rating.java`, `Feed.java`, and `Meme.java`.

Do not write an implementation for the methods (this will be done in future sprints), simply "stub" the methods out. More specifically, write the correct method header and in the body of the method, provide a *valid* return value (if applicable). If the method returns an object, you may return a default value of `null`.

A few notes about creating your starter code in Eclipse for this assignment:

- Depending on your version of Eclipse, if you use the auto-generators for getters, it may generate `isShared( )` as a getter method for `Meme`'s `shared` instance variable. For our

implementations, we will expect the standard naming of `getShared/setShared` instead.
- 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

Additional Resources

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

- Short guide on writing UML in text documents
- Draw.io is a web-based drawing and diagramming tool
    - To use the tool, create a blank diagram, then open the UML toolbox on the left-hand side. Class diagram templates should be in the top row under UML.
- Short guide on method stubs
- Video on submitting to Gradescope

Submission

You must submit both portions of your assignment to Gradescope.

UML Diagrams: Upload the PDF of your UML diagrams to the "Meme Magic 1 UML" assignment on Gradescope. **Only PDFs will be accepted**.

Java Code: Upload your Eclipse project (the `.java` files) to the "Meme Magic 1 Java" assignment on Gradescope. This submission utilizes an autograder to check your class definitions and method names. Submissions for the Java portion **will open 3 days after the UML submission**.

For this assignment, you may upload your code and/or diagrams an unlimited number of times.

Grading Rubric

The assignment will be worth a total of 100 points:
10 points each - UML Diagrams for User, BackgroundImage, Meme, Rating, and Feed
10 points each - Java files for User.java, BackgroundImage.java, Meme.java, Rating.java, Feed.java

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.

Meme Magic
Sprint 3 - Build 2

You are proud of your team, they are making great progress. We will continue the building of the system in sprint 2. This sprint will include the `User` and `Feed` classes, along with some integration tasks.

First, set up your project:

- Create a new Java project on Eclipse, and call it `MemeMagic3`.
- Copy all of your `MemeMagic2` files into the `MemeMagic3` project.
- Maintain a copy of `MemeMagic2` files (i.e., do not overwrite them.)

Note: you may alternatively copy your project in Eclipse by selecting the `MemeMagic2` 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. Utilizing `ArrayList` to store and organize objects
2. Integrating newly implemented classes into an existing codebase
3. Incorporating main-method testing to check correctness (i.e., behavior matches specifications)

Implementation

Implement methods for the 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.

User

- create a constructor that accepts a `String` for userName
- `toString()` - returns "*username* has rated (*number of memes viewed*) memes, (*reputation*)"
    - **Note**: the reputation should be rounded to 1 decimal place.
    - ex:
      derrickstone has rated (10) memes, (9.0)
- `equals(Object)` - returns true if the parameter is a `User` object and this `User`'s `userName` is equal to that of the parameter.

- `rateMeme(Meme, int)` - this method accepts a `Meme` argument and an `int` for rating score. It will record that `Meme` as having been seen by this user (`memesViewed`) and give it a `Rating` of this `score`.
- `rateNextMemeFromFeed(Feed, int)` - this method accepts a `feed` argument and an `int` for rating score and returns a `boolean`. The method will get a `Meme` from the `Feed` (supplied as an argument) using the `getNewMeme(User)` method of the `Feed` class. It will record that `Meme` as having been viewed by the user, give it the rating score, and return true. If there are no `Memes` left to view, the method should return false (and should not throw an error).
  Note: the return type has been updated from Sprint 1 to return a `boolean`.
- `createMeme(BackgroundImage, String)` - creates a new `Meme` object using the supplied arguments (`String` is the `caption`) and with the current user set as the creator. This method will add the resulting `Meme` to the list of `createdMemes` for the current user.
- `deleteMeme(Meme)` - deletes this `Meme` if found in the `memesCreated` field for the current user, only if the shared field is false. *(Because anything shared on the Internet lives forever.)* If the deletion was successful, return true. Otherwise, return false.
- `shareMeme(Meme, Feed)` - marks that `Meme` as shared (sets the shared field to true) and copies it to the `ArrayList<Meme>` data structure on the supplied `Feed`.
- `calculateReputation()` - returns a value calculated as the average of all overall ratings (`caclulateOverallRating()`) for Memes created by this User. If the user has not created any `Memes` or had any `Memes` rated, `0.0` should be returned.

Feed

- `getNewMeme(User)` - return a `Meme` from the current `Feed` that the `User` has not seen (does not exist in that `User`'s `memesViewed` list) and that the `User` did not create themselves. If there is no `Meme` to return, return `null`.
- `toString()` - returns all the memes in the feed, each `Meme` on a new line. Note that the `toString()` methods for both `Meme` and `Rating` need to be updated to integrate with the `User` and `Feed` implementation.
  - 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] - created by derrickstone
    Too sad <Image of tearful kitten> 'When you laugh at insistence on comments in school and then get a job programming where nobody comments' -2.0 [+1: 4, -1: 6] - created by derrickstone
    Robots Eating <Robots seated at table with server> 'When you go out for a byte' 1.0 [+1: 3, -1: 2] - created by user01001011

Integration

Now that you have implemented the User class, we must integrate it into the previously-built portions of the system. Using the following specifications, update these methods of the `Meme` and `Rating` classes.

Meme

- `toString()` - returns "backgroundImage ‘caption' *overallRating* [+1: the number of +1 ratings, -1: the number of -1 ratings] - created by *userName*"
    - 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] - created by derrickstone
    - **Note**: overallRating is the value returned by `calculateOverallRating()`.
    - **Note**: userName is the `userName` of the `creator User`.
    - **Hint**: there is a lot going on in this string. Consider how additional *private* helper methods might make this easier to read.

Rating

- `toString()` - returns "*userName* rated as *type_of_rating*"
    - For example, if the user object has userName *derrickstone:*
        - if the score is +1, then it will return: *derrickstone rated as an upvote*
        - if the score is -1, then it will return: *derrickstone rated as a downvote*
        - if the score is 0, then it will return: *derrickstone rated as a pass*

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 new constructor we’ve asked for in this sprint (User)
- User, Feed, Meme, and Rating’s `toString()` methods
- User’s `equals()` method
- Feed’s `getNewMeme()`
- User’s `rateMeme()`
- User’s `rateNextMemeFromFeed()`
- User’s `createMeme()`
- User’s `deleteMeme()`
- User’s `shareMeme()`
- User’s `calculateReputation()`

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 deleting a shared meme in `deleteMeme()` or getting the next meme from an empty feed in `rateNextMemeFromFeed()`.

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
- The following code will print the string `5.0`

```
System.out.println(String.format("%.1f", 4.999999));
```

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 3" assignment on Gradescope. You should submit `User.java`, `Feed.java`, `Rating.java`, `BackgroundImage.java`, and `Meme.java`. 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
15 points - Main method testing
5 points - Code readability (organized, well-indented, readable by others)

Academic Integrity and Moving into Sprint 4

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.

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.

Meme Magic
Sprint 5 - Testing

Your development build is coming along nicely. Before you release it to your users, however, you want to ensure the software is of good quality. To protect the reputation of your startup, you decide to develop a set of automated tests using JUnit.

First, set up your project:

- Create a new Java project on Eclipse, and call it `MemeMagic5`.
- Copy and import all of your `MemeMagic4` files into the `MemeMagic5` project.
- Maintain a copy of `MemeMagic4` files (i.e., do not overwrite them.)

Note: you may alternatively copy your project in Eclipse by selecting the `MemeMagic4` 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.

Next, set up your testing environment (this is important!):

- Right-click your new project in the Package Explorer and choose “New” and “Source Folder”. For “Folder name”, type “test” and click Finish. This will provide a separate source folder for all your JUnit tests.
- Right-click the new “test” folder in Package Explorer and choose “New” and “JUnit Test Case”. (If it’s not in the menu there, choose “Other...” and then search for “JUnit Test Case”.) This will open a window to create a new JUnit test file, and will also add JUnit to our build path.
    - Choose “New JUnit 4 test” at the top of the window, and type the Name of the test class, such as “FeedTest” to test the Feed class. Then click Finish.
      Note: you must use JUnit 4 for Gradescope to correctly run your tests.
    - You should get a warning that “JUnit 4 is not on the build path. Do you want to add it?” with the action “Add JUnit 4 library to the build path” pre-selected. Click OK to proceed.
    - That should open the new file with one test that defaults to fail. Now you’re ready to implement your own tests!

See the *Getting Started with Sprint 5* video **(not provided for review)**.

Learning Goals

In this assignment, we will practice:

1. Writing unit tests for Java using JUnit
2. Calculating code coverage of unit testing

Unit Testing Meme Magic

Write JUnit tests that provide at least **90%** code coverage for your project. 90% code coverage means that the tests confirm that 9 out of every 10 lines of written code is measured by the test. *(You are using a whitebox approach to testing!)*

For each Class completed through Sprint 4, create a JUnit 4 Test named ***`Class`***`Test`. For example, to test the `Meme` class, create the `MemeTest` class. (The .java file for the new `MemeTest` class should reside in the `test/` folder.)

For each method in your classes, create a ***`methodName`***`Test`. For example, to test the `deleteMeme()` method, create the `deleteMemeTest()` method.

For getter and setter methods, it is sufficient to create one method, `setAndGet`***`Variable`***`Test()`, where *Variable* is the name of the field (instance variable). For example, to test the `userName` field, you should write the `setAndGetUserNameTest()` method, which calls both the getter and setter for `userName` and asserts that the value was set correctly. For all other methods, use a separate test method to keep your tests organized.

Your code **must** pass your Unit tests. If it does not pass, code coverage will not be calculated on Gradescope.

Important Information

You can confirm that your tests are evaluating all of your code using the Coverage feature in Eclipse. **Note:** If you do not remove or comment out your main method tests, these may count as lines that need to be covered.

To access the Coverage feature, with your project open, select *Run*, then *Coverage*. The menu may vary by Eclipse version. If you don't see the *Coverage* option, search for coverage in help. We encourage you to set up a *Run Configuration* to run all your Unit test files at once. See the *Getting Started with Sprint 5* video **(not provided for review)**.

Code coverage is counted as a percentage of the instructions in your classes covered by your tests. Therefore, it is important to separate the tests from the codebase. As seen in Figure 1, we currently have only 2.8% code coverage of our classes (even though it appears we have 100% coverage of our test code and 5.3% overall coverage in the project itself).

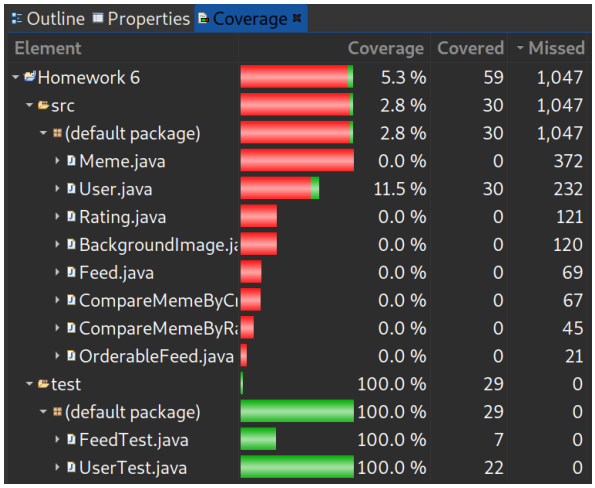

Figure 1. Screenshot of Eclipse Code Coverage display. Shows 2.8% code coverage of our classes, including 11.5% code coverage of the User class.

Additional Resources

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

- JavaDoc style documentation for each of the classes and methods from Sprint 4
- Video on Getting Started with Sprint 5
- 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. **Classes to perform tests MUST have names ending with "`Test`".** That is, if you want a JUnit test to run on Gradescope, it must be in a class named *Something*`Test`.

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 5" assignment on Gradescope. You should submit all your .java files from both your src/ and test/ directories. Gradescope will calculate code coverage for your classes and use that to calculate your final score. *If your code does not compile, it cannot check for code coverage.* We strongly encourage you to run your tests locally before submitting; you should **not** use only Gradescope to run your tests. Therefore, you may upload your code a **maximum of ten (10) times**.

Note: Gradescope's code coverage values may differ slightly from Eclipse's reported coverage. We have found Gradescope to be slightly higher in most cases.

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

Grading Rubric

Your score will be determined by lines of code coverage. The assignment will be worth a total of 100 points:

90 points - Code coverage, proportional to the amount of code coverage
50% coverage or less = 0 points -- *up to* -- 90% coverage or more = 90 points

10 points - Code readability (organized, well-indented, readable by others)

Meme Magic
Sprint 6 - Feature Add 2

Your team is ready to expand the text-based memes into image-based memes. It's time to create the GUI for the Meme Magic application.

First, set up your project:

- Create a new Java project on Eclipse, and call it `MemeMagic6`.
- Copy and import all of your `MemeMagic5` (or `MemeMagic4`) files into the `MemeMagic6` project. (Remember the Academic Integrity policy from the end of `MemeMagic4` - you are free to share your `MemeMagic4` code with your group members and use `MemeMagic5` code from your group members as long as you understand any code you include.)
- Maintain a copy of `MemeMagic5` (or `MemeMagic4`) files (i.e., do not overwrite them.)

Note: you may alternatively copy your project in Eclipse by selecting the `MemeMagic5` 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.

For this assignment, we've provided a few implementations to start the process:

- Download our `GraphicalMeme.java` file and add it to your project. It extends your Meme class.
- Download our `MemeMagic.java` file and add it to your project. It provides a skeleton of the GUI for Meme Magic. Try it out! (It should look similar to Figure 1.)

Learning Goals

In this assignment, we will practice:

1. Implementing graphical user interfaces (GUIs) using event-driven programming
2. Organizing GUI content using Java Swing containers and components
3. Handling Exceptions with try/catch blocks

Implementing the Graphical User Interface

Let's get started. Open the `MemeMagic.java` file that we provided and run it. If everything is working correctly you should see the following screen (Figure 1).

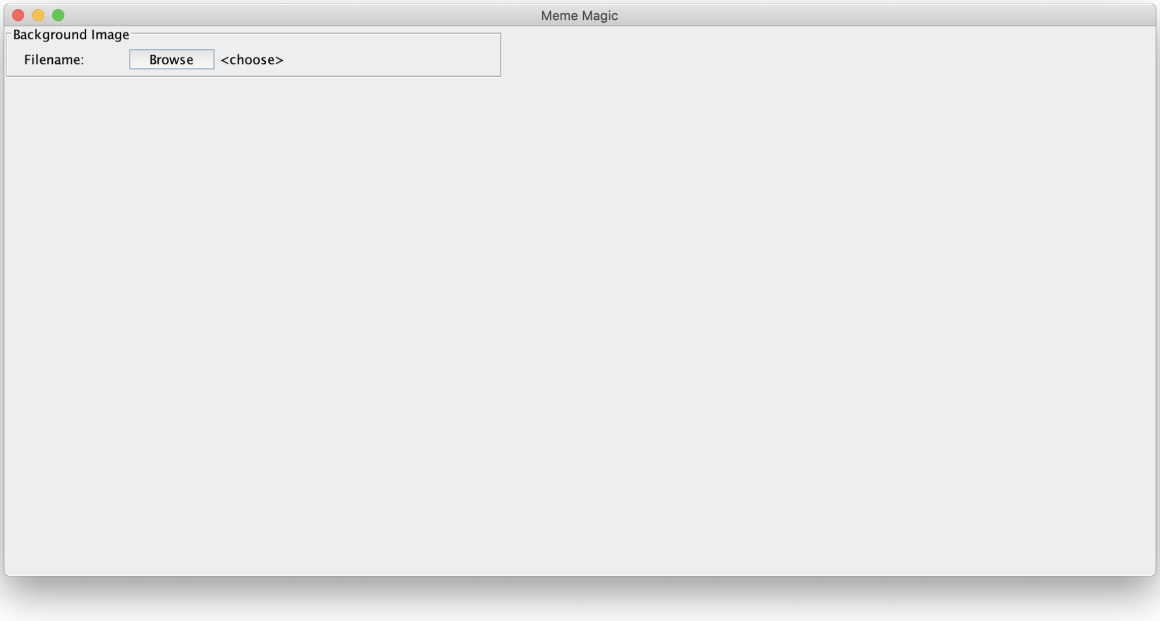

Figure 1: *Screenshot of the initial (scaffolding) interface of Meme Magic (in macOS).*

When we're done your design should look like this (Figure 2).

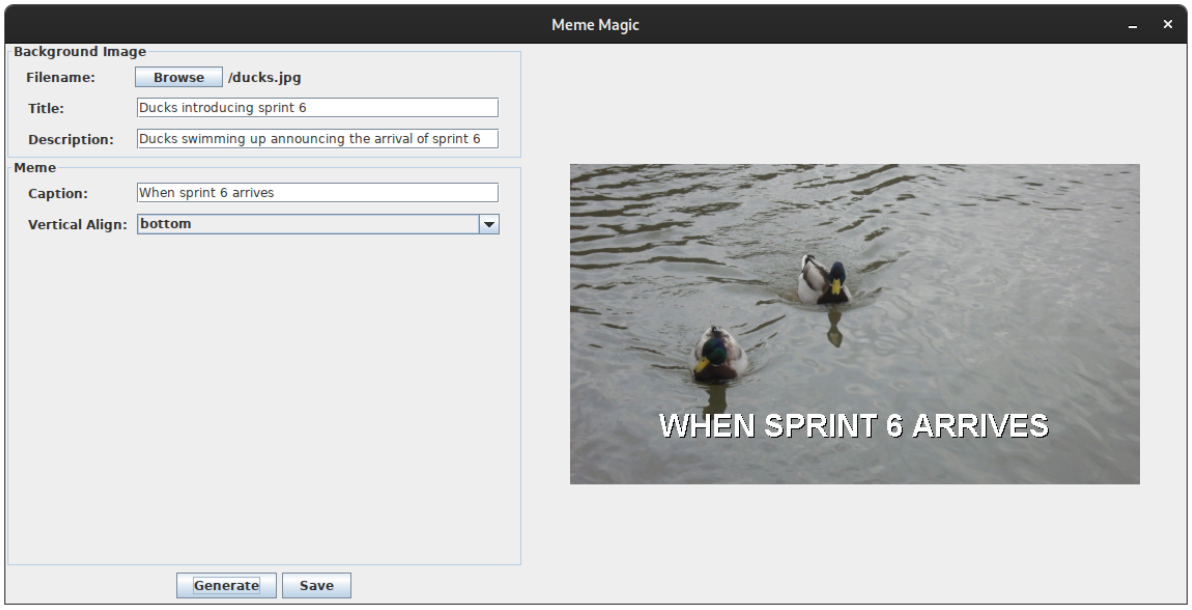

Figure 2: *Screenshot of the Meme Magic application once it is fully implemented (in Linux).*
Photo Credit: John R. Hott

Implementing the BackgroundImage JPanel

We'll begin by adding the Title and Description fields to the `backgroundImagePanel`. Take a Look at the skeleton code in the `MemeMagic.java` file. We've implemented the first row as an example.

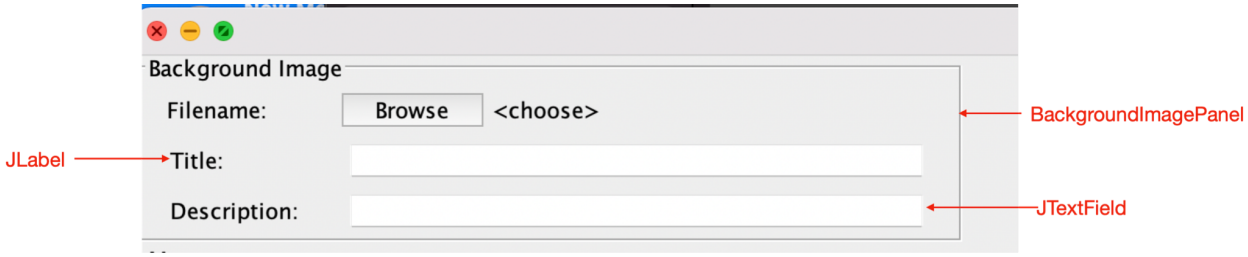

Figure 3: *GUI elements on the BackgroundImage JPanel.*

Adds the following to `backgroundImagePanel`:

1. A JLabel that shows the text value: "Title"
2. A JTextField that allows the user to enter the title they want.
3. A JLabel that shows the text value: "Description"
4. A JTextField that allows the user to enter the description they want.

Hint: Remember to group the JLabel and JTextField in a JPanel so that they all line up on one line.

Connect the OpenButtonListener Listener

Now let's implement the feature that allows the user to click the Browse button and open file explorer dialog. We've implemented that functionally for you, but you will need to connect the OpenButtonListener in the `MemeMagic.java` file to the browse button by adding it as an action listener.

Implementing the Meme JPanel

Figure 4: *Close-up view of the Meme JPanel.*

This panel (Figure 4) includes a component that allows a user to select items from a dropdown list. You can implement this component using a JComboBox. For examples on how to use the JComboBox and other Swing components, visit this link: [Helpful GUI examples in Swing]. The

combo box should have three options: “top”, “middle”, or “bottom”. This element will allow the user to indicate where the caption will be placed on the image.

Add the Meme JPanel to the JPanel called `controlPanel`. See Figure 5 below.

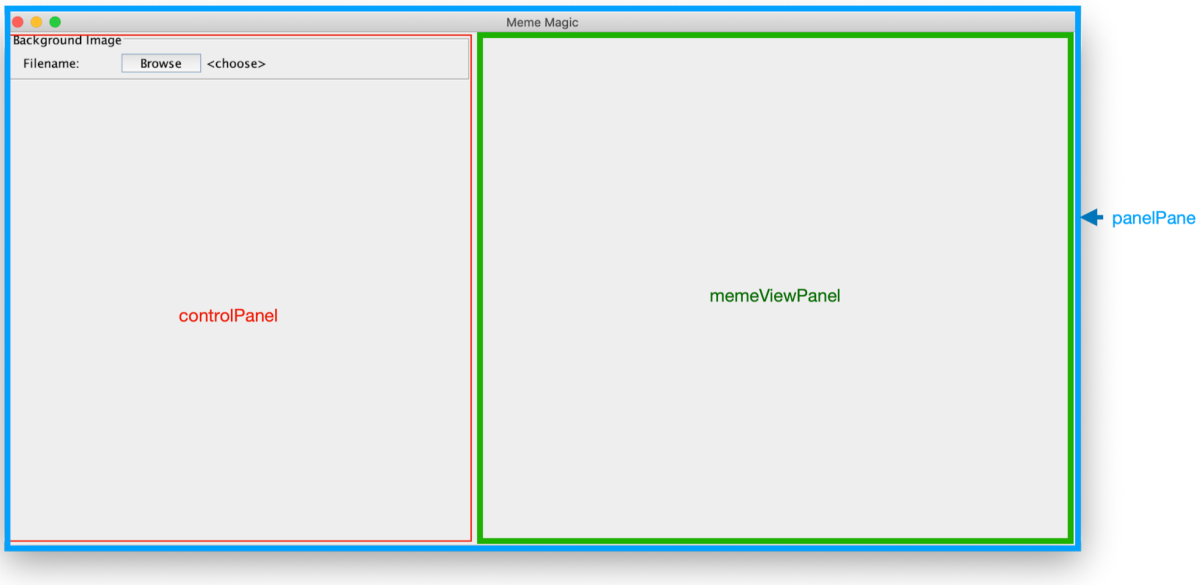

Figure 5: *Nesting JPanels like this allows us to structure the layout of the screen.*

Implementing the Generate and Save Buttons

Provide the user with a **Generate** button to create and display the `GraphicalMeme` into the `imageDisplayLabel`. Add a listener to the button so that when the button is clicked, the following actions are taken:

- You can call the `getText()` method to JTextField variable to get the text.
    - For example, `String text = ExampleCaptionTextField.getText();`
- Instantiate a `GraphicalMeme` object (it extends your `Meme` class) with the caption, alignment, and background image information provided by the user in the GUI interface.
- Use `GraphicalMeme`’s `compileMeme()` method to compile the Meme into a `BufferedImage` (image data) and display the graphical version of the meme on the provided `imageDisplayLabel`.
    - Catch any exceptions that might be thrown from the `compileMeme()` method.
    - The Java API for `ImageIcon` and `JLabel` and our **Pineapple Pizza Example** on Collab provide resources for how to display an image on a label.

    - **Note**: You will need to `repaint()` the `memeViewPanel` for the image to display.

Add a **Save** button that opens a save dialog box, calls the `compileMeme()` method, and writes the image to a file. We have included a listener (`SaveButtonListener`) that opens the save dialog box and gets the chosen destination filename.

- Modify the listener to include the commented-out `ImageIO.write()` call and handle any exceptions that might be thrown. If an exception occurs, provide the user with an appropriate (and detailed) message of what went wrong, either by printing a message to `System.err` or displaying the message in a new GUI dialog box.
    - The Java API for `ImageIO` and `File` and our documentation for `GraphicalMeme` provide information about which exceptions may be thrown.

Additional Resources

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

- JavaDoc style documentation for each of the classes and methods from Sprint 4
- Video on submitting to Gradescope
- Java API Documentation for
    - ImageIcon
    - ImageIO
    - File
    - JLabel
- Helpful GUI examples in Swing

Submission Information

Coding Style: In real-world software development, it is important to write readable and maintainable code. That is typically achieved through the use of style and commenting guidelines. 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 Code: Upload your Eclipse project (the `.java` files) to the "Meme Magic 6" assignment on Gradescope. You should submit at least `User.java`, `Rating.java`, `BackgroundImage.java`, `Meme.java`, `GraphicalMeme.java`, and `MemeMagic.java`.

Submitting Demonstration: Create a short 2-5 minute screen recording demoing your implementation. If you didn't get everything working, that's okay, please show us what you have working and let us know what's missing.

- This video should start by briefly (1-2 minutes max) describing your code: walking through what MemeMagic.java does, what graphical elements you used, and any listeners you created.

- Then, you should run your code to open a MemeMagic window, fill out the fields, and generate a Meme. It can be as funny as you'd like, but please keep it clean. CS-related memes are highly appreciated!
- Use the save button to save your Meme, then open up the created image to show us that your meme saved successfully
- Lastly, close and restart the MemeMagic application and show us your error or Exception handling (display a scenario when one of the try-catch blocks would catch an Exception).

Save and name your video with your name, such as **last-first.mp4**. You can use Zoom to perform the recording, but we ask you to rename your video afterwards. **Upload your video** to the following link: [insert here]

Note: Please keep a copy of your video just in case.

Optional: Post your generated Meme to the class' "memes" channel (or post).

Grading Rubric

The assignment will be worth a total of 100 points:

40 points - Graphical interface with all required elements (with video demo)
20 points - Event listeners for generating and saving memes (with video demo)
20 points - Exception handling and appropriate error messages (with video demo)
20 points - Code readability (organized, well-indented, readable by others)

Where to go from here?

Well, the Meme Magic assignments are done, but the creativity isn't! There are many ways to continue expanding your Meme Magic system. You could customize the look and feel of the memes, provide options to your users when generating them, split the text around the image, build-in services to allow posting to social media, and more! The possibilities are (almost) endless!

We only ask that you do not post your source code publicly (at least without significant changes), since we may use these assignments again in the future.