# 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