What abUTA You?

Due Tuesday, February 11 at 8 a.m.

CSE 1325 - Sprint #2 of 4 - Homework #4- Rev 1

Assignment Overview

NOTE: Revision 1 adds a data validation check to the `Group` constructor, requirements for the subclasses and JavaDoc, and a suggested build.xml file that supports generating the JavaDoc documentation.

Everyone seems to be unhappy with their social media apps these days - Facebook, YouTube, WhatsApp, Instagram, the almost-late (in the US) TikTok, and X (née Twitter). What should any self-respecting geek do? Write our own, of course!

This is sprint 2 of a 4-sprint, 4-week project (with the first exam in the middle - sorry) that you can add to your growing resume, with emphasis on writing a user interface, file I/O, inheritance, and polymorphism (all coming soon to a lecture near you). We'll have a final, stand-alone Java assignment to practice threading before Exam #2. Implementation guidance will be provided each week to help you with your implementation, and you may have a few checkpoints where you may switch to the professor's suggested solution if you lose your footing.

For the second sprint (P04), we'll divide our code into `account` and `message` packages. We'll add two types of messages - `Post` (visible to all within a Group) and `DirectMessage` (visible to the recipient) - as subclasses to our Message class, along with basic group management. For this, we'll write a `Group` class and of course a `TestGroup` regression test for it. And we'll add JavaDoc documentation to `Account` (and others for bonus credit).

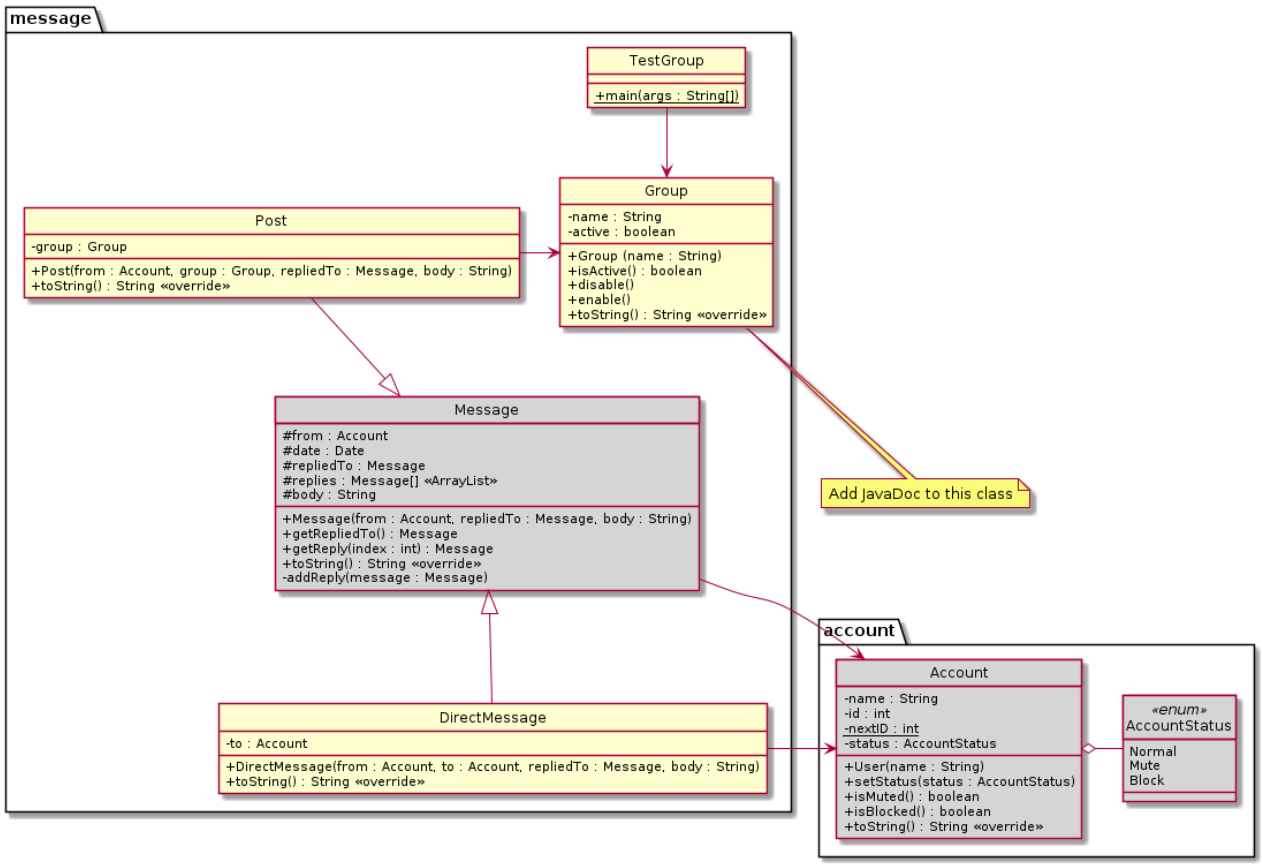
If your first sprint left something to be desired, contact the professor and ask about baselining the suggested solution. While we *prefer* that you make this project entirely your own, we want to ensure that you're able to complete it. Let's talk!

Sprint #2

This is a partial requirements document. Expect an update soon with the rest of the requirements.

Consider this class diagram on the next page. The more darkly shaded classes are *mostly* unchanged from the last sprint, while the more lightly colored classes are new this sprint.

You may ONLY add the public members shown on the diagram to your classes. No additional public members are permitted. Don't duplicate any class responsibilities in your other methods, including main!



Organize abUTA into Packages

Create your GitHub-managed working directory **cse1325/P04**.

- Create an **account** subdirectory, and copy your account-related classes from cse1325/P03/full_credit or (if you implemented it) cse1325/P03/bonus into it. This will include AccountStatus.java, Account.java, and perhaps TestAccount.java and DemoAccount.java. Add `package account;` to the top of each of these files.
- Create a **message** subdirectory, and copy your message-related classes from cse1325/P03/full_credit or (if you implemented it) cse1325/P03/bonus into it. Add `package message;` to the top of each of these files. In addition, when `Account` or `AccountStatus` is used in a file, also add `import account.Account;` or `import account.AccountStatus;`, since those classes are now in different packages.
- **Keep build.xml in the cse1325/P04 directory.** Do ALL builds and run ALL programs from this directory. To build a single file such as Account.java, from P04 run `javac account/Account.java` (adjusting as needed for your environment). To build all files, from P04 run `ant`. To execute a program such as TestAccount, from P04 run `java account.TestAccount`. See how this works?

Verify that all of your regression tests still work.

class Group

In package message, write class Group as shown in the class diagram.

- The constructor sets field `name` to the parameter and `active` to true. If `name` is null or an empty string, throw an `IllegalArgumentException`.
- Method `isActive` returns field `active`.
- Method `disable` sets field `active` false.
- Method `enable` sets field `active` true.
- Method `toString` returns the name and, if inactive, " [inactive]".

Regression Test TestGroup

Write a regression test for class Group similar to the examples provided in TestAccount and TestMessage. Cover the following test vectors.

- Test the constructor and toString together.
- Verify that the newly constructed group is active by default.
- Verify that the `disable` method makes the Group inactive.
- Verify that an inactive group reports " [inactive]" as part of its `toString` method.
- Verify that method `enable` makes an inactive Group active again.

Message Subclasses

We have two subclasses of class `Message` for this project, `Post` (a public message in a specific `Group`) and `DirectMessage` (a private message sent to a specific individual). These are all in package `message` as shown in the UML diagram.

The two subclasses are quite similar - each has a single additional field, a constructor that must chain to the superclass constructor, and a `toString()` method that puts the group (for `Post`) or recipient (for `DirectMessage`) in front of the superclass' `toString()` results.

After writing these classes, your `TestAccount`, `TestMessage`, and `TestGroup` regression tests should still work because the tests ignore the new subclasses. We'll fix this in the bonus.

JavaDoc

Write JavaDoc for class Group. For the class, include (at a minimum) a description and the author, version, and since. For each public member, include a description and list of parameters (if any), throws (for the constructor's IllegalArgumentException), return (for `isActive`), and since. Non-public members do NOT need any JavaDoc at all.

You MUST provide a working build.xml file so that the `ant` tool works. A working build.xml is provided at `cse1325-prof/P01/full_credit/build.xml` or in any of the lecture code.

You MUST include at least 3 substantial commits of your evolving work. A "one and done" commit, even with trivial additional commits just to get to 3, will incur *substantial* grade penalties.

Practice good version control!

New build.xml

This new build.xml file includes a new target `javadoc` which builds the doc/api directory with documentation for your `account` and `message` packages. The `clean` target has also be updated to remove the JavaDoc documentation.

The one change you'll likely want to make is the copyright statement in line 20. Replace my information with whatever information you'd like to include about yourself.

To build your JavaDoc documentation, just type `ant javadoc`. The open **P04/doc/api/index.html** and navigate around your documentation. Be sure to look at the `message.Group` class once you've added the JavaDoc comments to ensure they look as you expected.

```
<?xml version="1.0"?>
<project name="CSE1325" default="build">

 <target name="build" description="Compile source tree java files">
   <javac includeantruntime="false" debug="true" failonerror="true">
     <src path="."/>
   </javac>
 </target>

 <target name="javadoc" description="Generate JavaDoc">
     <javadoc packagenames="account.*,message.*"
          sourcepath="."
          defaultexcludes="yes"
          destdir="docs/api"
          author="true"
          version="true"
          use="true"
          windowtitle="abUTA API">
          <doctitle><![CDATA[<h1>Mavs abUTA Social Network</h1>]]></doctitle>
         <bottom><![CDATA[<i>Copyright &#169; 2025 Professor George F. Rice. Licensed as CC BY-SA International 4.0</i>]]></bottom>
         <tag name="license.agreement" scope="all" description="Licensed under:"/>
     </javadoc>
 </target>

 <target name="clean" description="Clean output files">
   <delete dir="docs/api"/>
   <delete dir="docs"/>
   <delete>
     <fileset dir=".">
       <include name="**/*.class"/>
     </fileset>
   </delete>
 </target>
</project>
```

Bonus

You have two options for bonus points.

1. Also add JavaDoc documentation to the `AccountStatus`, `Account`, `Message`, `Post`, and `DirectMessage` classes, receiving 2 points per class.
2. Create simple `TestPost` and `TestDirectMessage` regression tests for 5 points each. These should simply verify the `toString()` of each class.

Doing both of these would be considered the Extreme Bonus, except that if you lose any points on one, the work on the other could potentially still get you to 10 points.