Second Iteration Plan: mycheapfriend.com

COMS 4156: Advanced Software Engineering

Team: CheapSkates

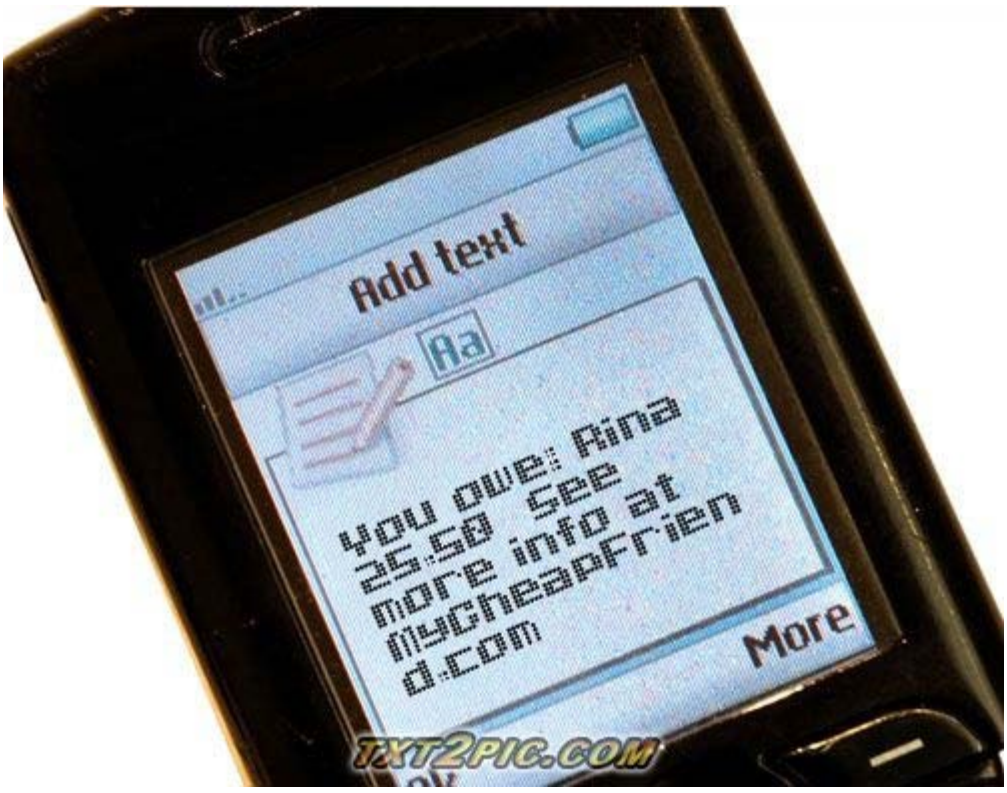
Team Members

Double Trouble: Waseem Ilahi (wki2001@columbia.edu)

Shaoqing Niu (sn2385@columbia.edu)

Dynamic Duo: Michael Glass (mgg2102@columbia.edu)

Huning "David" Dai (hd2210@columbia.edu)

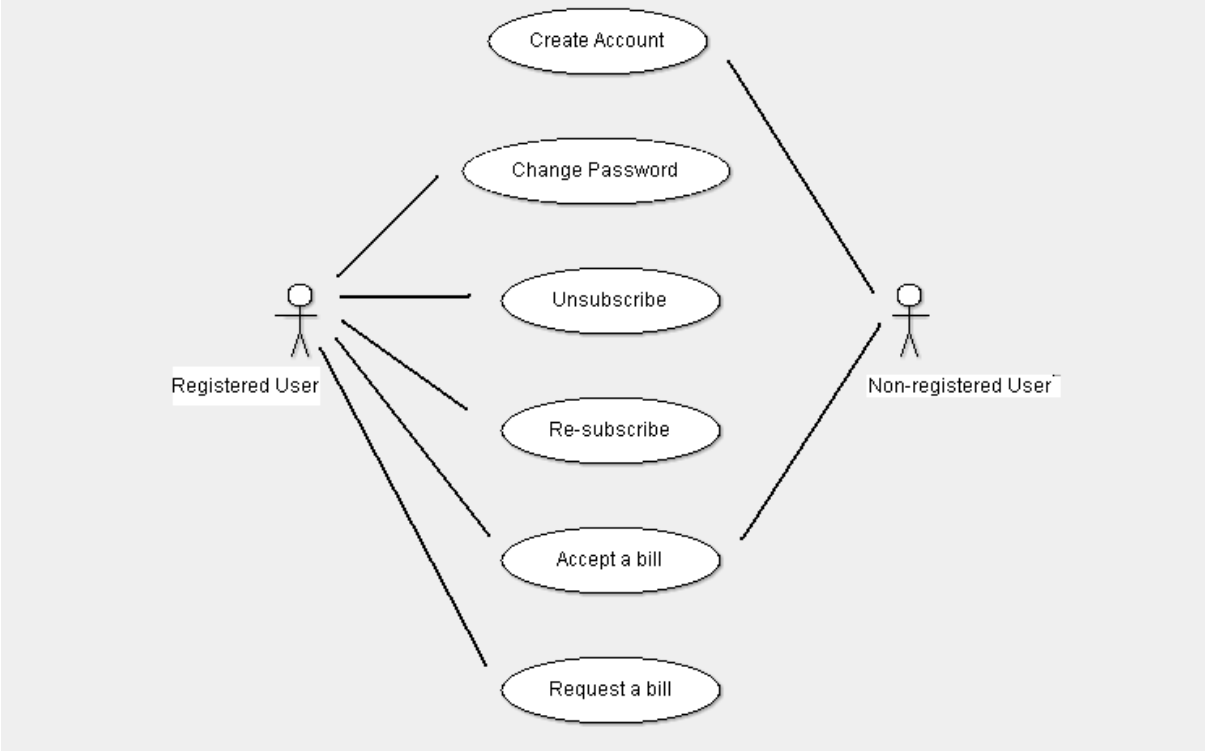


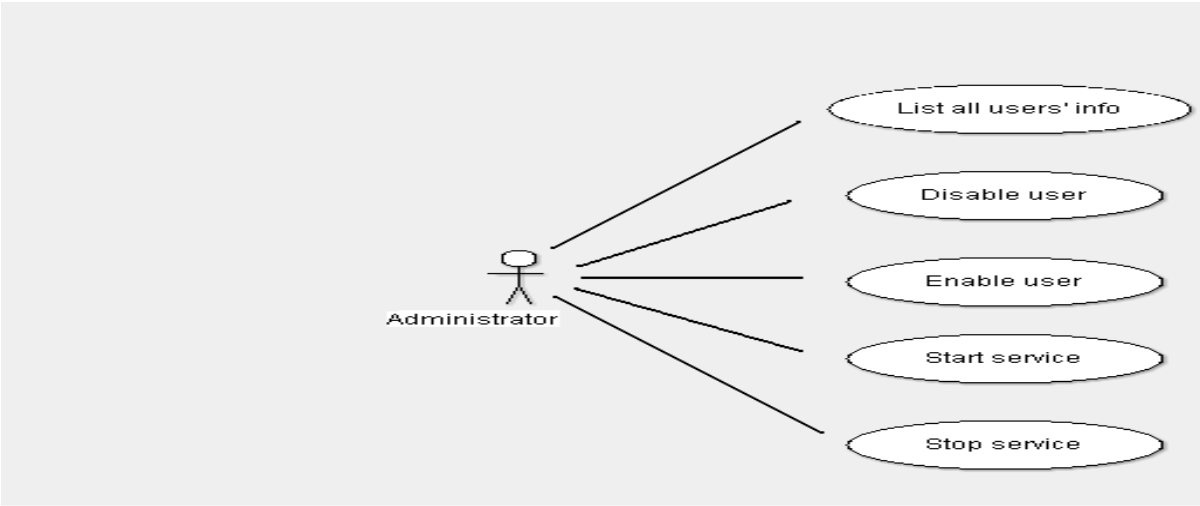
The **Cheapskates** from **MyCheapFriend.com** will lend you our **Sir** if you give us back feedback soon.

2. Requirements (UML):

1. Behavioral diagrams

We use two use case diagrams, one for the registered/non-registered users and one for the administrators.





2. Structural diagrams

We use a class diagram in Figure 0 to show the structure of our system. This is the entire internal structure for MyCheapFriend.

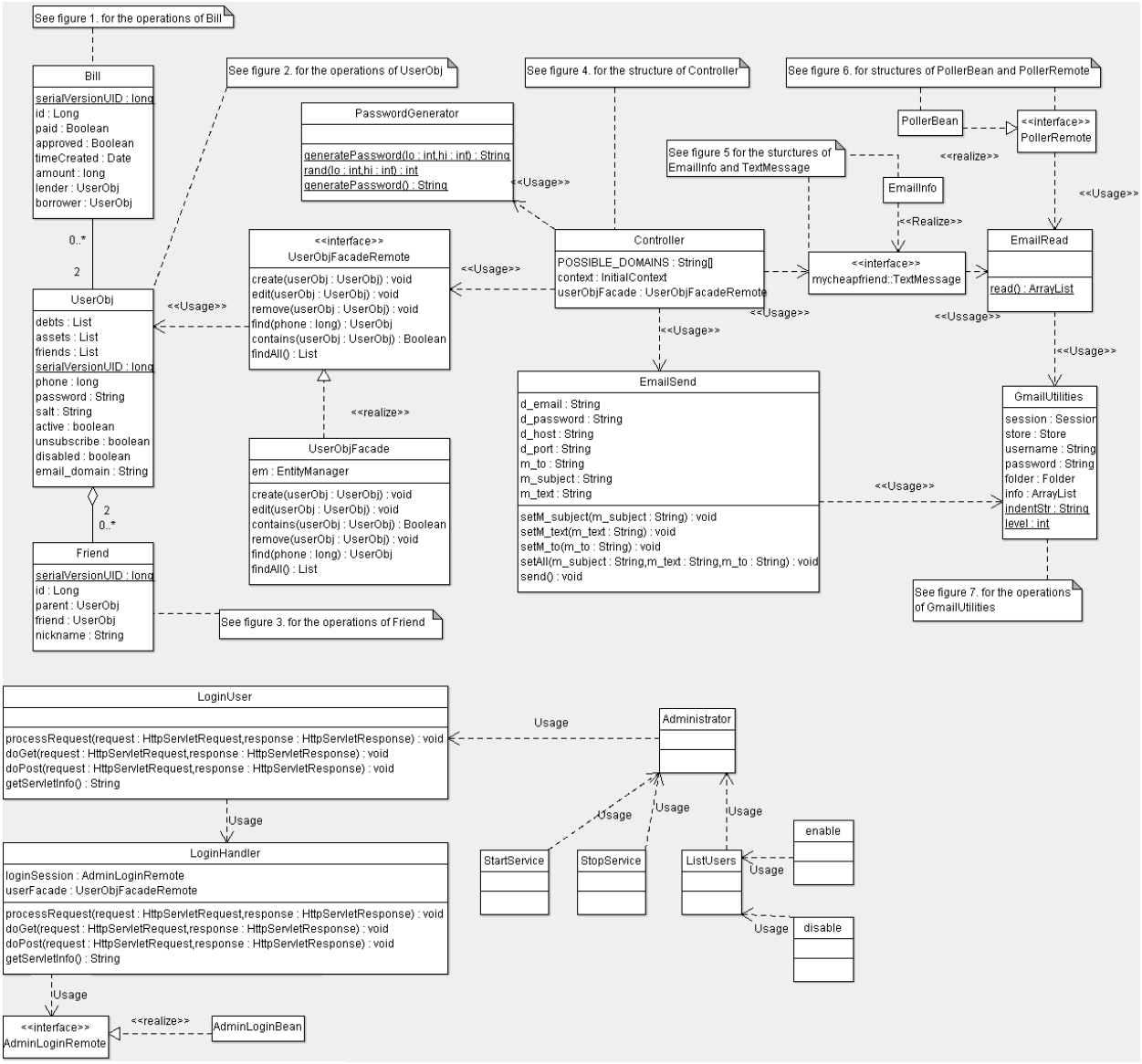


Figure 0

Figures 1,2 & 3 are the class diagrams for the data model (persistent entities).

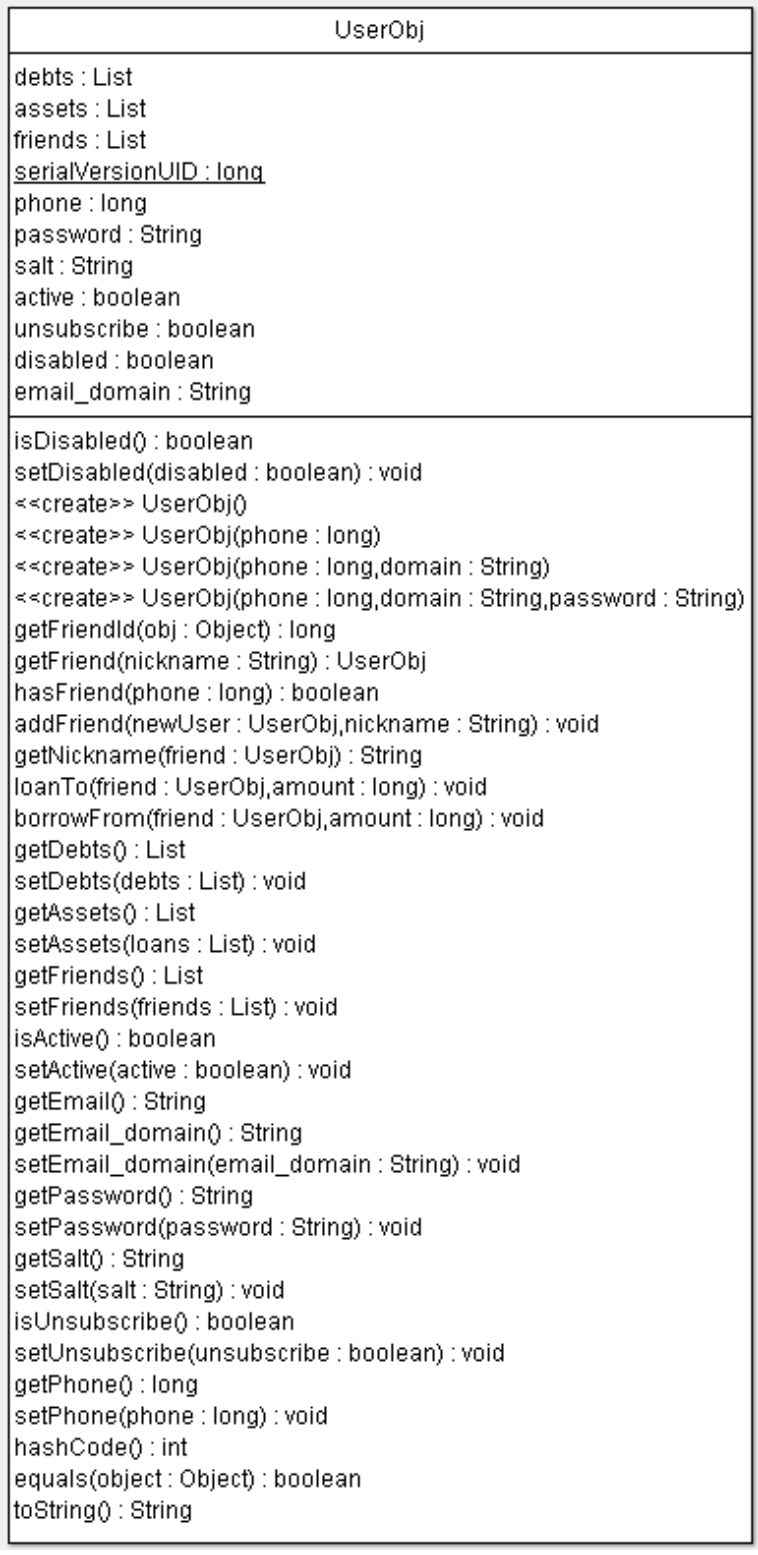


Figure 1

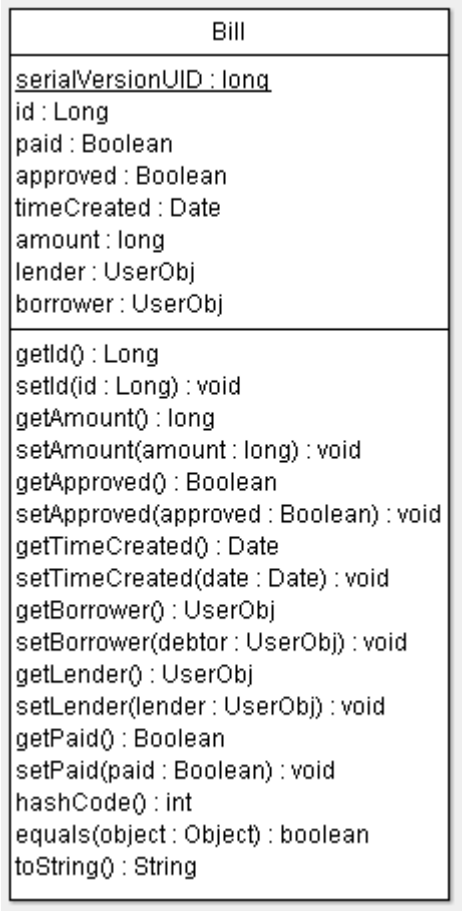


Figure 2

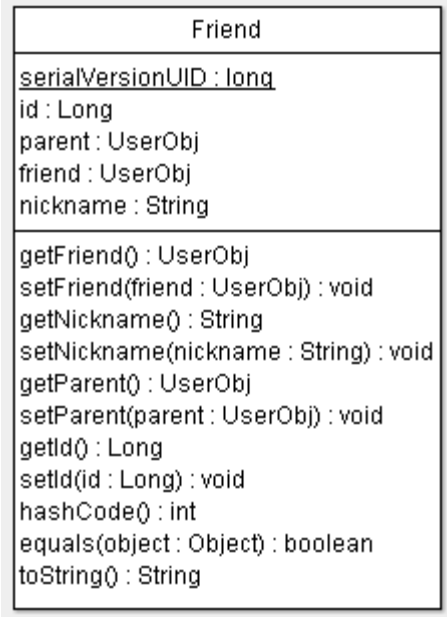


Figure 3

The controller class implements the business logic for the system.

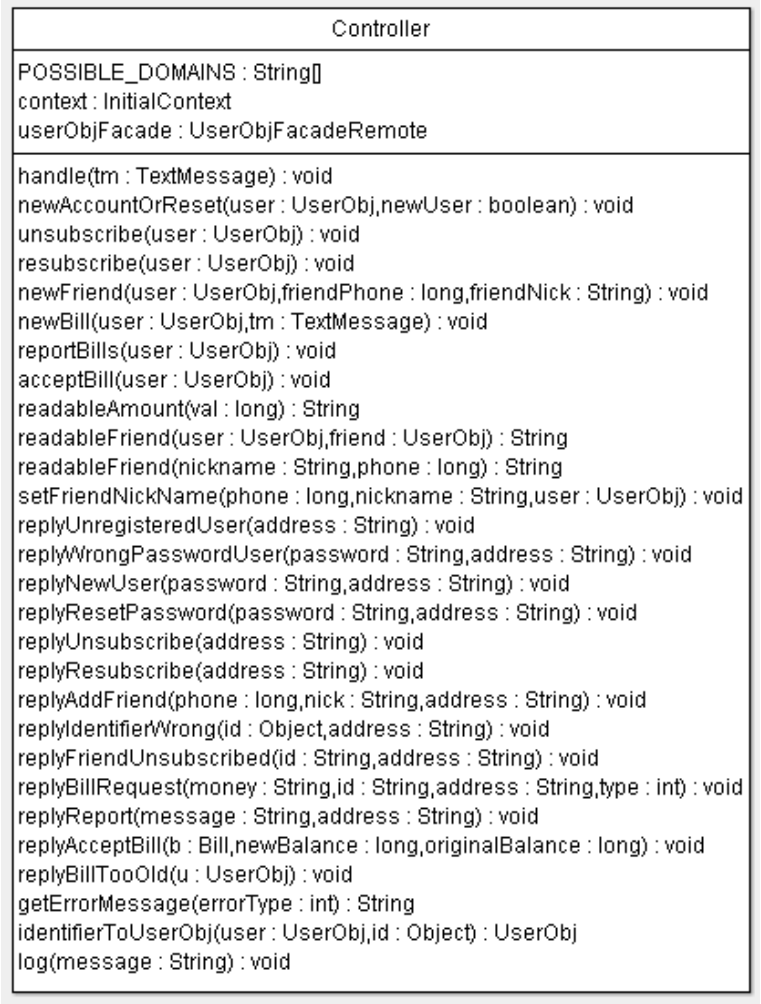


Figure 4

These are the Parser and the Poller Bean classes. The poller's timeout method calls the parser, and hands over the parsed texts to the controller to do the required functions.

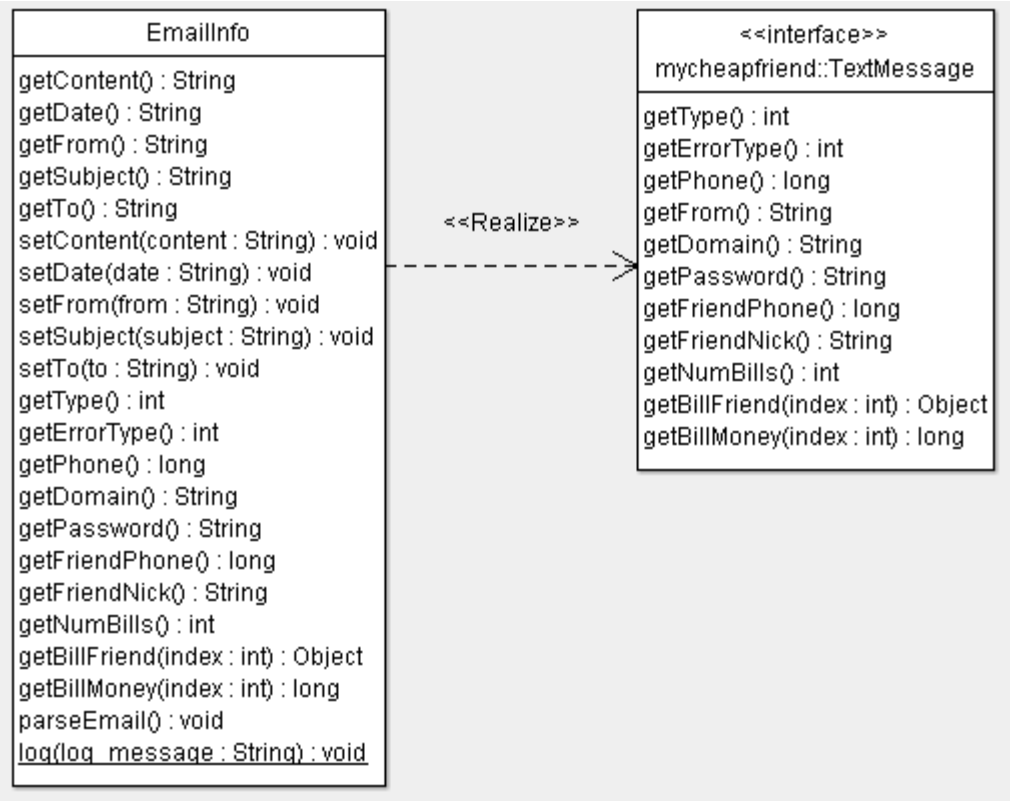


Figure 5

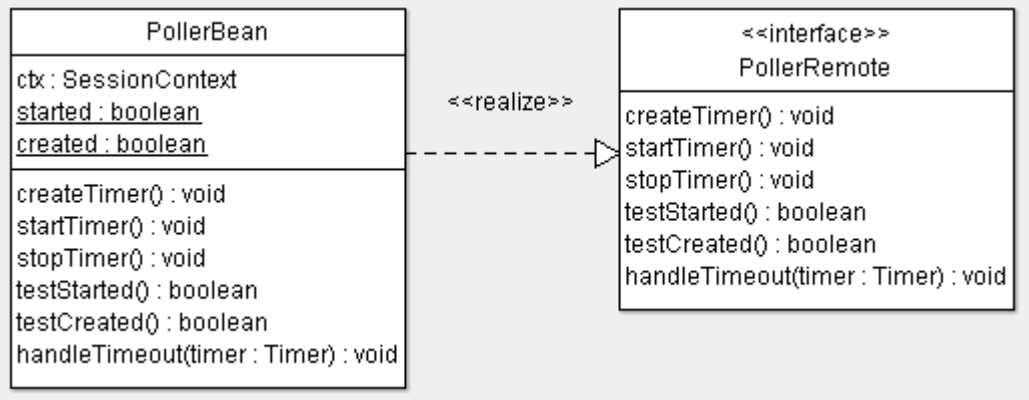


Figure 6

The JavaMail API class. It has been referenced from the outside source. The URL is:

http://forums.sun.com/thread.jspa?threadID=5267916

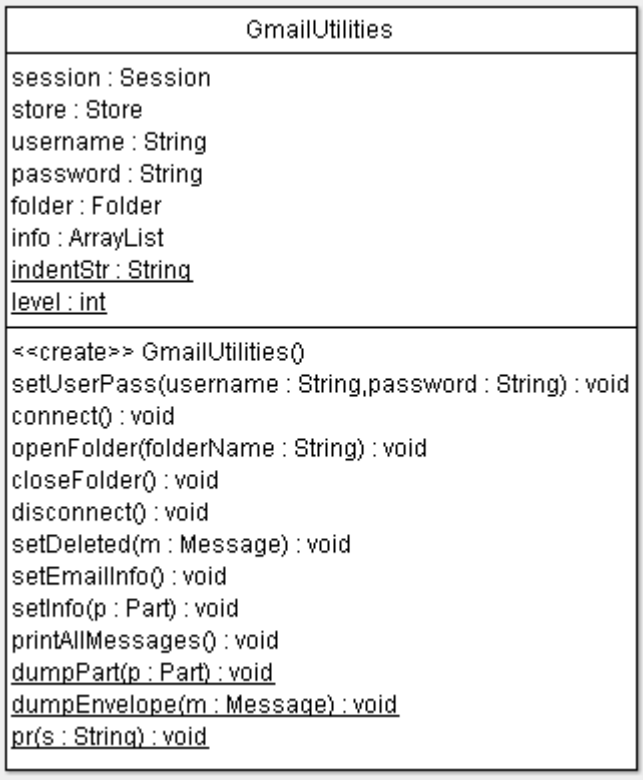


Figure 7

The EmailRead.java and EmailSend.java use the gmail utilities to receive from our Gmail account that accepts all the incoming text messaging. And then send the text message to the users.

3. Unit Testing:

We are testing the parser component (EmailInfo.java) of our system. The parser does lexical and syntax validation of fields without any backend logic checking.

1. The Prefix of User's “from” address, before the “@” sign, must be a 10-digit number.
    a. Valid Prefix: 10-digit number should be accepted.
    b. Invalid Prefix: invalid prefix provided, it should be rejected.
    c. Error cases: Anything not a 10-digit number, e.g., blackbird, etc.
    d. Member: 7182242980
    e. Non-member: joesmith

2. The Prefix of the User's To address must be one of our approved addresses
    a. Valid Prefix: new_account, robot, etc, are valid prefixes and should be accepted
    b. Invalid Prefix: invalid prefix provided by the user, it should be rejected.
    c. Error cases: anything other than new_acccount, reset_pass, unsubscribe, resubscribe, robot, [0-9a-z]{6,6}.
    d. Member: new_account
    e. Non-member: joe_smith

3. The text’s message body for “to prefix”, new_account, reset_pass, unsubscribe, and resubscribe, can be anything.
    a. Valid Message: accept anything user provides.
    b. Invalid Message: nothing is invalid, so everything must be accepted.
    c. Error cases: None
    d. Member: Everything a user can type in the text body.
    e. Non-member: Nothing

4. The text’s message body for “to prefix”, robot is of the form "^(y[a-z]*)?$"
    a. Valid Message: "^(y[a-z]*)?$" is the valid entry and should be accepted.
    b. Invalid Message: invalid message provided, it should be rejected.
    c. Error cases: the message that doesn’t start with the letter “y” and is non-empty.
    d. Member: y
    e. Non-member: no
5. The text’s message body for “to prefix”, “any valid password”, must be either “report” or a collection of identifiers and amounts (all identifiers should be identifiers; all amounts should be amounts, nothing else besides spaces).
    a. Valid Message: “report”, etc is valid and should be accepted.
    b. Invalid Message: invalid message body, reject the text.

c. Error cases: anything that is not “report” or the collection of identifiers and amounts, where the identifiers are approved identifiers and the amounts are integers. Also “not” something that follows the following rules:
    i. if identifiers includes "me", can only have one amount, only one instance of "me"
    ii. 2 if (# of amounts) > 1, # of amounts must == # of identifiers, must occur in pairs (ie, [amount identifier] [identifier amount] [identifier amount])
    iii. (if # of amount = 1) must be one or more identifiers
d. Member: report
e. Non-member: 74 74

4. Code Inspection Checklist:

We will be inspecting the "Controller" unit of our code.

1. Controller handles all parsing errors (from the parser).
2. Controller can create a new account.
3. Controller can reset a user's password.
4. Controller can unsubscribe a user.
5. Controller prevents all communication with unsubscribed user.
6. Controller can resubscribe a user.
7. Controller can report user’s current debts.
8. Controller can issue a new bill.
    a. Controller can reject a bill with an invalid identifier.
9. Controller can accept a bill.
10. Controller can accept a new nickname.
11. Controller can reset a nickname.
12. Check that code complies with Java coding conventions.
13. Check for appropriate code comments (is code's behavior clear).
14. Check for appropriate code factorization into atomic parts.

5. Security

Our user interface is relatively simple, so testing each aspect of it doesn't need to be too complicated. We will split the attack into attacks on each of the interfaces.

The Web UI:

The web UI is an administrator interface, where admins should have "root" access once they are authenticated. Here is our attack plan for the relatively simple Web UI.

1. Going through the security
   Try the overflowing the fields in the web form.
2. Going around the security
   Try accessing administrator URLS without logging in.
3. Accessibility attack.

If we break into the system, the root user's index page is heavy weight, as it lists all database elements. Repeatedly requesting this page (assuming A) the database is thoroughly populated and B) we have access to the page) would be a way of attacking the system's accessibility. That being said, once a root user has been compromised, they could simply turn off the system, so it's probably not worth exploring this attack.

The Text UI:

The Text UI receives emails from text messages and parses them into logical system objects. Given the:

1. Going through the security
   Try impersonating phone numbers with phone-number like email addresses from other hosts (impersonating SMS). ie 6462294050@gmail.com, or sending an email from our own smtp server impersonating vtext.com or att.com.
2. Accessibility attack:
   Sending emails with large attachments, or very large message bodies might slow down our e-mail fetcher's speed of retrieving new messages.

6. Schedule

The Schedule for the Second Iteration has been attached at the end of this document.

7. Controversies

There are no controversies among the team at this point.

Task	Completed
• **1) Define 2nd Iteration Plan**	100%
• 1.1) System requirements in UML	100%
• 1.2) Unit testing plan	100%
• 1.3) Code inspection plan	100%
• 1.4) Security & Robustness plan	100%
• 1.5) Schedule	100%
• 2) Code Inspection	0%
• **3) Unit Tetsting**	0%
• **3.1) Write tests**	0%
• 3.1.1) Test parser FROM: field	0%
• 3.1.2) Test parser TO: field	0%
• 3.1.3) Test parser BODY field	0%
• 3.2) Write defect log	0%
• 3.3) write 2nd Iteration Progress Report	0%
• **4) Security & Robustness Testing**	0%
• **4.1) Administrator UI**	0%
• 4.1.1) Login Field Overflow attacks	0%
• 4.1.2) URL attack	0%
• **4.2) Email UI**	0%
• 4.2.1) impersonation via GMAIL	0%
• 4.2.2) impersonation via smtp	0%
• 4.2.3) stress test with long messages	0%
• 5) Prepare for Demo	0%
• 6) Write 2nd Iteration Final Report	0%