COS 126 **General Computer Science** **Spring 2013**

Programming Exam 2

This programming exam has 2 parts, weighted as indicated. The exam is semi-open: you may use the course website, the booksite, any direct links from those two, the textbook, any printed or written notes, and your past coursework on your computer. You may not use other websites. No communication is permitted, except with course staff members.

Upload your code to the dropbox. As with the COS 126 assignments, you can submit your code multiple times for testing, but only the last version will be graded.

Your code will be graded primarily on correctness. You will lose a substantial number of points if your program does not compile. However, efficiency, clarity, and code style are also factors in your grade. Remember to include headers and comments in your code. Headers MUST include your name, netID and precept number.

Print your name, netID, and precept number on this page (now), and write out and sign the Honor Code pledge before turning in this paper. It is a violation of the Honor Code to discuss this exam until everyone in the class has taken the exam. You have 90 minutes to complete the test. **Do not remove this exam from the exam room.**

Write out and sign the Honor Code pledge before turning in the test:
"I pledge my honor that I have not violated the Honor Code during this examination."

Pledge: ______________________________

____

Signature: ______________________________

P01	12:30 TTh	Dave Pritchard
P01A	12:30 TTh	Donna Gabai
P01B	12:30 TTh	Pawel Przytycki
P02	1:30 TTh	Tom Funkhouser
P02A	1:30 TTh	Allison Chaney
P02B	1:30 TTh	Pawel Przytycki
P02C	1:30 TTh	Vivek Pai
P02D	1:30 TTh	Siddhartha Chaudhuri
P03	2:30 TTh	Tom Funkhouser
P03A	2:30 TTh	Allison Chaney
P04	3:30 TTh	Vivek Pai
P04B	3:30 TTh	Shilpa Nadimpalli
P05	7:30 TTh	Shilpa Nadimpalli
P06	10am WF	Lennart Beringer
P07	1:30 WF	Dave Pritchard
P07A	1:30 WF	Kevin Lee
P07B	1:30 WF	Siyu Liu
P08	12:30 WF	Donna Gabai
P08A	12:30 WF	Judi Israel
P09	11am WF	Judi Israel

Name: ______________________________

NetID: ______________________________

Precept: ______________________________

Problem	**Score**
`Person`	/22
`TigerBook`	/8
Total	/30

Your task is to create two `.java` files:

- `Person.java`, which models a person, his/her friendships, and messages they can read.
- `TigerBook.java`, which models a collection of `Person` objects.

Several code fragments and test files that we refer to below are located at

`http://www.cs.princeton.edu/courses/archive/spring13/cos126/docs/data/TigerBook`

To submit your code click on *Submit* for *Exam 2* on the **Assignments Page**, or directly visit

`https://dropbox.cs.princeton.edu/COS126_S2013/Exam2`

Part 1: Person

In the first part of this exam, you will create a simple model `Person` for a person, their friends, and a list of messages they can read. It should implement the following API:

```
public class Person
---------------------------------------------------------------------------------------------
        Person(String name)       // Create a new Person with this name.

   void meet(Person otherPerson)  // Make these two people become friends with each other.
                                  // If otherPerson is the same as this person, throw a
                                  // RuntimeException (you can't be your own friend).

boolean knows(Person otherPerson) // Are these two people friends?
                                  // If otherPerson is the same as this person, throw a
                                  // RuntimeException (since you can't be your own friend).

   void post(String message)      // Post a message. It should be added to the message
                                  // list of this Person, and the message lists of
                                  // all current friends of this Person.

   void listMessages()            // Print a header (see format on next page), then all
                                  // messages ever posted to this Person, most recent first.
---------------------------------------------------------------------------------------------
(everything listed in this API must be public)
```

Here are some additional suggestions to help interpret the API requirements.

- The constructor should initialize the instance variables. Think carefully about what instance variables you need in order to store the name, list of friends and list of messages for each `Person`. The **last page of this exam** contains APIs for some useful Abstract Data Types that you may use. Uploading `.java` files for ADTs other than these will not be permitted by the submission webserver.
- The `meet()` method should add `otherPerson` to the friend list of this `Person`. *Friendship is reciprocated*, so also add this `Person` to the friend list of `otherPerson`. The methods `knows()` and `post()` will use this list.

- In `meet(otherPerson)` you may assume that `otherPerson` is not yet a friend (we will never test this case).
- Remember to throw a `RuntimeException` in `knows()` or `meet()` if this person is the same as the other person. The exception message can be anything you choose.

Here is a small example. In the comments we describe the expected behaviour.

```
Person first = new Person("Kim");
Person second = new Person("Pat");

StdOut.println(first.knows(second));    // should print "false"

first.meet(second);

StdOut.println(first.knows(second));    // should print "true"
StdOut.println(second.knows(first));    // should print "true"

first.knows(first);                     // should throw a RuntimeException
```

(This example is `Person.example1.txt` in the online testing files.)

The `post()` method should add a message to the message list of this `Person`, and to all friends of this `Person`. These messages will be read using `listMessages()`.

- Unlike facebook.com, you do not have to spread the message to friends of friends, etc. Only the person themself and the direct friends of the person should get a copy.
- Unlike facebook.com, messages only become visible to people with whom you are friends *right now*. Meeting someone doesn't retroactively give you copies of old messages that s/he posted earlier.

The required behaviour for calling `listMessages()` on a `Person` is that it must:

- First, print a header line to `StdOut` in the format
  `== The wall of Kim ==`
  where "Kim" is replaced by the name of the `Person`.
- Then, print to `StdOut` each of that person's messages on a separate line, with the *newest one first*.

Note that messages are not deleted by `listMessages()`. They remain in the `Person`'s list of messages.

Here is another test, for the `post()` and `listMessages()` methods.

Testing code:

```
Person first = new Person("Kim");
Person second = new Person("Pat");
first.post("Only Kim can read this");

first.meet(second);
second.post("Friends are awesome");
first.post("I agree");

first.listMessages();
second.listMessages();
```

When run, it should print:

```
== The wall of Kim ==
I agree
Friends are awesome
Only Kim can read this
== The wall of Pat ==
I agree
Friends are awesome
```

(This example is `Person.example2.txt` in the online testing files.)

Upload your part 1 code before proceeding, to see if there are any bugs caught by the dropbox tests.
It does not matter if you leave a `main` method in `Person`.
You do *not* need to upload the `.java` files for `Stack`, `Queue` or `ST`.

Part 2: TigerBook

The second class that you will implement is `TigerBook`. Each `TigerBook` object contains a list of registered users (`Person` objects) and their user ids. For the purposes of the `TigerBook` class, you should assume each `Person`'s `id` is distinct. The TigerBook API is:

```
public class TigerBook
----------------------------------------------------------------------------------------------
       TigerBook()                   // Create a new TigerBook instance.

  void register(String id, Person p) // Add a person to the directory of registered users.
                                     // You may assume "id" was never registered before.

Person lookup(String id)             // Return the Person who registered with this id.
                                     // If nobody ever registered with this id, then
                                     // throw a RuntimeException.
----------------------------------------------------------------------------------------------
(everything listed in this API must be public)
```

Here is a small example. In the comments we describe the expected behaviour.

```
TigerBook t = new TigerBook();

Person first = new Person("Tony");
t.register("tony10010", first);

Person personFound = t.lookup("tony10010");
personFound.listMessages();                 // should print:  == The wall of Tony ==

Person nobody = t.lookup("tony-the-tiger");   // should throw a RuntimeException
```

(This example is `TigerBook.example.txt` in the online testing files.) Note that `TigerBook` does not need to deal with messages, friends, or walls, since the `Person` class does this for us.

This completes the description of your tasks. Testing and reference information follows. *Remember to upload your work to the dropbox.* It does not matter if you leave a `main` method in `TigerBook`.

ExampleClient.java

We provide a client `ExampleClient.java` for your programs in the online files, and several test files. It reads in commands from standard input; some of these commands produce output. The `ExampleClient` header contains a detailed description of the input format. Using this testing client is optional.

Example: `friendly.txt` contains

```
Charles registers
Charles posts Difference Engine is meh.
Ada registers
Ada queries Charles
Ada meets Charles
Ada queries Charles
Ada posts Analytical Engine rocks!
Ada reads
Charles reads
```

`java ExampleClient < friendly.txt` outputs:

```
Are Ada and Charles friends? false
Are Ada and Charles friends? true
== The wall of Ada ==
Analytical Engine rocks!
== The wall of Charles ==
Analytical Engine rocks!
Difference Engine is meh.
```

Note: `ExampleClient.java` does not need modification and should not be submitted.

APIs for common ADTs

This section lists some useful parts of the APIs for the `Stack`, `Queue`, and `ST` data types. For convenience, if you have not yet downloaded the files for these data types, you can download them at:

`http://www.cs.princeton.edu/courses/archive/spring13/cos126/docs/data/ADT`

Make sure they are in the exact same directory as your `Person.java` and `TigerBook.java` files.

```
public class Stack<Item> implements Iterable<Item>
-----------------------------------------------------------------------------
        Stack()         // Create an empty stack.
boolean isEmpty()       // Is the stack empty?
   void push(Item item) // Push item onto the top of the stack.
   Item pop()           // Remove and return item at top of stack.

Iteration on Stack uses LIFO order.  Iteration does not alter the Stack.
```

```
public class Queue<Item> implements Iterable<Item>
-----------------------------------------------------------------------------
        Queue()            // Create an empty queue.
boolean isEmpty()          // Is the queue empty?
   void enqueue(Item item) // Add item to the end of the list.
   Item dequeue()          // Remove and return item from beginning of list.

Iteration on Queue uses FIFO order.  Iteration does not alter the Queue.
```

```
public class ST<Key extends Comparable<Key>, Value>
-----------------------------------------------------------------------------
        ST()                   // Create an empty symbol table.
boolean contains(Key key)      // Is there a value paired with key?
   void put(Key key, Value v)  // Put key-value pair into the table.
  Value get(Key key)           // Return value paired with key, or return null
                               // if this key was never put into the table.

Iteration on ST uses the order of the Key.  Iteration does not alter the ST.
```