COMPUTER SCIENCE
SECTION II

Time—1 hour and 45 minutes
Number of questions—4
Percent of total grade—50

Directions: SHOW ALL YOUR WORK. REMEMBER THAT PROGRAM SEGMENTS ARE TO BE WRITTEN IN Java.

Write your answers in pencil only in the booklet provided.

Notes:

- Assume that the classes in the Quick Reference have been imported where needed.
- Unless otherwise stated, assume that parameters in method calls are not `null` and that methods are called only when their preconditions are satisfied.
- In writing solutions for each question, you may use any of the accessible methods that are listed in classes defined in that question. Writing significant amounts of code that can be replaced by a call to one of these methods may not receive full credit.

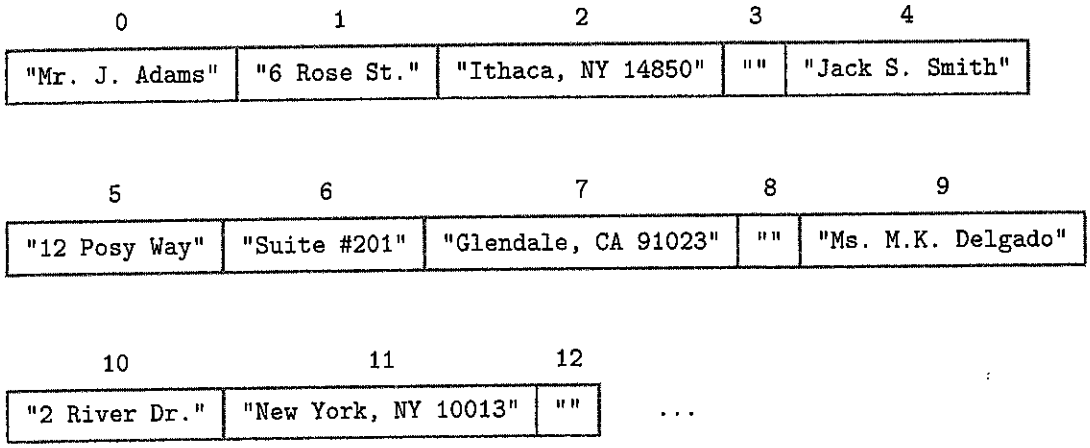
1. Consider a system for processing names and addresses from a mailing list. A `Recipients` class will be used as part of this system. The lines in the mailing list are stored in an `ArrayList<String>`, a private instance variable in the `Recipients` class. The blank line that separates recipients in the mailing list is stored as the empty string in this list, and the final element in the list is an empty string.
A portion of the mailing list is shown below, with the corresponding part of the `ArrayList`.

```
Mr. J. Adams
6 Rose St.
Ithaca, NY 14850

Jack S. Smith
12 Posy Way
Suite 201
Glendale, CA 91203

Ms. M.K. Delgado
2 River Dr.
New York, NY 10013

...
```



The Recipients class that processes this data is shown below.

```
public class Recipients
{
    /** The list of lines in the mailing list */
    private List<String> lines;

    /** Constructor. Fill lines with mailing list data.
     *  Postcondition:
     *    - Each element in lines is one line of the mailing list.
     *    - Lines appear in the list in the same order
     *      that they appear in the mailing list.
     *    - Blank line separators in the mailing list are stored
     *      as empty strings.
     */
    public Recipients()
    { /* implementation not shown */ }

    /** Postcondition: Returns the city contained in the cityZip
     *                 string of an address.
     *  @param  cityZip contains the city, state, and zipcode
     *  line of an address
     *  @return the city substring contained in cityZip
     */
    public String extractCity(String cityZip)
    { /* to be implemented in part (a) */ }

    /** Precondition:  The recipient name is the first line of each
     *                 label on the mailing list.
     *  Postcondition: Prints a list of recipient names to console,
     *                 one per line.
     */
    public void printNames()
    { /* to be implemented in part (b) */ }
```

```
    /** Postcondition: Returns the address of the recipient with
     *                 the specified name.
     *  @param name a name in the lines ArrayList
     *  @return the address of the recipient with the given name
     */
    public String getAddress(String name)
    {/* to be implemented in part (c) */}

    //Other methods are not shown.
}
```

(a) Write the `extractCity` method of the `Recipients` class. In the `cityZip` parameter the city is followed by a comma, then one blank space, then two capital letters for a state abbreviation, then a space and 5-digit zip code. For example, if `cityZip` is `"Ithaca, NY 14850"`, the method call `extractCity(cityZip)` should return `"Ithaca"`.

Information repeated from the beginning of the question

```
public class Recipients

private List<String> lines
public Recipients()
public String extractCity(String cityZip)
public void printNames()
public String getAddress(String name)
```

Complete method `extractCity` below.

```
/** Postcondition: Returns the city contained in the cityZip
 *                 string of an address.
 *  @param  cityZip contains the city, state, and zipcode
 *  line of an address
 *  @return the city substring contained in cityZip
 */
public String extractCity(String cityZip)
```

(b) Write the `printNames` method of the `Recipients` class. Method `printNames` prints the names of all recipients to the console, one per line. For the sample part of the mailing list shown at the beginning of the question, the output for `printNames` would be:

```
Mr. J. Adams
Jack S. Smith
Ms. M.K. Delgado
```

Complete method printNames below.

```
/** Precondition: The recipient name is the first line of each
 *                label on the mailing list.
 *  Postcondition: Prints a list of recipient names to console,
 *                one per line.
 */
public void printNames()
```

(c) Write the getAddress method of the Recipients class. This method should return a string that contains only the address of the corresponding name parameter. For example, if name is "Jack S. Smith", a string containing the three subsequent lines of his address should be returned. This string should contain line breaks in appropriate places, including after the last line of the address. This ensures that the address will have the proper address format when printed by a client class.

Complete method getAddress below.

```
/** Postcondition: Returns the address of the recipient with
 *                the specified name.
 *  @param name a name in the lines ArrayList
 *  @return the address of the recipient with the given name
 */
public String getAddress(String name)
```

2. A `WordSet`, whose partial implementation is shown in the class declaration below, stores a set of `String` objects in no particular order and contains no duplicates. Each word is a sequence of capital letters only.

```
public class WordSet
{
    /** Constructor initializes set to empty. */
    public WordSet()
    { /* implementation not shown */ }

    /** @return the number of words in set */
    public int size()
    { /* implementation not shown */ }

    /** Adds word to set (no duplicates).
     *  @param word the word to be added
     */
    public void insert(String word)
    { /* implementation not shown */ }

    /** Removes word from set if present, else does nothing.
     *  @param word the word to be removed
     */
    public void remove(String word)
    { /* implementation not shown */ }

    /** Returns kth word in alphabetical order, where 1 <= k <= size
     *  @param k position of word to be returned
     *  @return the kth word
     */
    public String findkth(int k)
    { /* implementation not shown */ }

    /** @return true if set contains word, false otherwise */
    public boolean contains(String word)
    { /* implementation not shown */ }

    //Other instance variables, constructors, and methods are not sh
}
```

The `findkth` method returns the *k*th `word` in alphabetical order in the set, even though the implementation of `WordSet` may not be sorted. The number *k* ranges from 1 (corresponding to first in alphabetical order) to *N*, where *N* is the number of words in the set. For example, if `WordSet s` stores the words `{"GRAPE", "PEAR", "FIG", "APPLE"}`, here are the values when `s.findkth(k)` is called.

k	values of `s.findkth(k)`
1	APPLE
2	FIG
3	GRAPE
4	PEAR

(a) Write a client method `countA` that returns the number of words in `WordSet s` that begin with the letter "A." In writing `countA`, you may call any of the methods of the `WordSet` class. Assume that the methods work as specified.

Complete method `countA` below.

```
/** @param s the current WordSet
 *  @return the number of words in s that begin with "A"
 */
public static int countA(WordSet s)
```

(b) Write a client method `removeA` that removes all words that begin with "A." If there are no such words in `s`, then `removeA` does nothing. In writing `removeA`, you may call method `countA` specified in part (a). Assume that `countA` works as specified, regardless of what you wrote in part (a).

Information repeated from the beginning of the question

```
public class WordSet

public WordSet()
public int size()
public void insert(String word)
public void remove(String word)
public String findkth(int k)
public boolean contains(String word)
```

Complete method `removeA` below:

```
/** @param s the current WordSet
 *  Postcondition: WordSet s contains no words that begin with
 *                 "A", but is otherwise unchanged.
 */
public static void removeA(WordSet s)
```

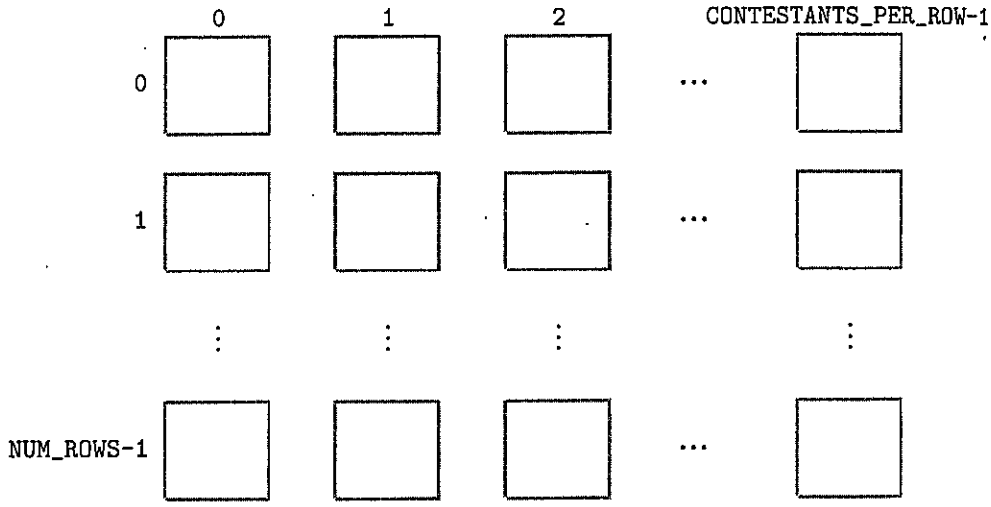
(c) Write a client method `commonElements` that returns the `WordSet` containing just those elements occurring in both of its `WordSet` parameters.

For example, if `s1` is `{"BE", "NOT", "AFRAID"}` and `s2` is `{"TO", "BE", "OR", "NOT"}`, then `commonElements(s1, s2)` should return the `WordSet` `{"BE", "NOT"}`. (If you are familiar with mathematical set theory, `commonElements` returns the intersection of `s1` and `s2`.)

Complete method `commonElements` below.

```
/** @param s1 the first given set
 *  @param s2 the second given set
 *  @return the WordSet containing only the elements that occur
 *          in both s1 and s2
 */
public static WordSet commonElements(WordSet s1, WordSet s2)
```


3. A puzzle-solving competition is held in a large hall with a two-dimensional arrangement of contestants. Each square below represents one contestant.



Since contestants may be moved around during the competition, each contestant keeps track of his or her location, which is the row number and column number. A `Location` object is represented by the class below.