Section II

1. (a)
```
public String extractCity(String cityZip)
{
    int commaPos = cityZip.indexOf(",");
    return cityZip.substring(0, commaPos);
}
```

(b)
```
public void printNames()
{
    System.out.println(lines.get(0));
    int index = 1;
    while(index < lines.size() - 1)
    {
        if (lines.get(index).equals(""))
            System.out.println(lines.get(index + 1));
        index++;
    }
}
```

(c)
```
public String getAddress(String name)
{
    int index = 0;
    while(index < lines.size() && !name.equals(lines.get(index)))
        index++;
    index++;
    String s = "";
    while (!(lines.get(index).equals("")))
    {
        s += lines.get(index) + "\n";
        index++;
    }
    return s;
}
```

NOTE

- In part (b), the empty string signals that the next element in the list will be a name. This is why you should be careful that you don't miss the first name in the list, which is at index 0. Notice, too, that you can avoid the empty string at the end of the list by having

  ```
  index < lines.size() - 1
  ```

  as the test in the `while` loop. If you don't do this, the final

  ```
  lines.get(index + 1)
  ```

  will cause an `IndexOutOfBoundsException`.
- Part (c) first finds the name that matches the parameter, and then builds a string out of the next two or three lines that comprise the address. Again, the empty string signals that the end of the address has been reached.
- The escape character string, `"\n"`, inserts a line break into the string.

2. (a)
```
public static int countA(WordSet s)
{
    int count = 0;
    while (count < s.size() &&
           s.findkth(count + 1).substring(0, 1).equals("A"))
        count++;
    return count;
}
```

Alternatively,

```
public static int countA(WordSet s)
{
    boolean done = false;
    int count = 0;
    while (count < s.size() && !done)
    {
        String nextWord = s.findkth(count + 1);
        if (nextWord.substring(0,1).equals("A"))
            count++;
        else
            done = true;
    }
    return count;
}
```

(b)
```
public static void removeA(WordSet s)
{
    int numA = countA(s);
    for (int i = 1; i <= numA; i++)
        s.remove(s.findkth(1));
}
```

Alternatively,

```
public static void removeA(WordSet s)
{
    while (s.size() != 0 &&
           s.findkth(1).substring(0, 1).equals("A"))
        s.remove(s.findkth(1));
}
```

(c)
```
public static WordSet commonElements(WordSet s1, WordSet s2)
{
    WordSet temp = new WordSet();
    for (int i = 1; i <= s1.size(); i++)
    {
        String nextWord = s1.findkth(i);
        if (s2.contains(nextWord))
            temp.insert(nextWord);
    }
    return temp;
}
```

NOTE

- To test whether a word starts with "A", you must compare the first letter of word, that is, word.substring(0,1), with "A".
- In part (a), you must check that your solution works if s is empty. For the given algorithm, count < s.size() will fail and short circuit the test, which is desirable since s.findkth(1) will violate the precondition of findkth(k), namely that k cannot be greater than size().
- The parameter for s.findkth must be greater than 0. Hence the use of s.findkth(count+1) in part (a).
- For the first solution in part (b), you get a subtle intent error if your last step is s.remove(s.findkth(i)). Suppose that s is initially {"FLY", "ASK", "ANT"}. After the method call s.remove(s.findkth(1)), s will be {"FLY", "ASK"}. After the statement s.remove(s.findkth(2)), s will be {"ASK"}!! The point is that s is adjusted after each call to s.remove. The algorithm that works is this: If N is the number of words that start with "A", simply remove the first element in the list N times. Note that the alternative solution avoids the pitfall described by simply repeatedly removing the first element if it starts with 'A." The alternative solution, however, has its own pitfall: The algorithm can fail if a test for s being empty isn't done for each iteration of the while loop.
- Part (c) could also be accomplished by going through each element in s2 and checking if it's included in s1.

3. (a)
```
public void updateLocation(int newRow, int newCol)
{
    loc = new Location(newRow, newCol);
}
```

(b)
```
public void sortAllRows()
{
    for(Contestant[] row: contestants)
        sort(row);
}
```

(c)
```
public String findWinnerName()
{
    sortAllRows();
    int max = contestants[0][0].getScore();
    String winner = contestants[0][0].getName();
    for(int k = 0; k < NUM_ROWS; k++)
    {
        Contestant c = contestants[k][CONTESTANTS_PER_ROW - 1];
        if (c.getScore() > max)
        {
            winner = c.getName();
            max = c.getScore();
        }
    }
    return winner;
}
```