CSci 1902: Structure of Computer Programming II
Fall 2011
Quiz 1

Name: ______________________
Student ID: __________________

For staff use:

<table>
<thead>
<tr>
<th>Problem</th>
<th>Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>_______ / 36</td>
</tr>
<tr>
<td>2</td>
<td>_______ / 25</td>
</tr>
<tr>
<td>3</td>
<td>_______ / 20</td>
</tr>
<tr>
<td>4</td>
<td>_______ / 19</td>
</tr>
<tr>
<td>TOTAL</td>
<td>_______ / 100</td>
</tr>
</tbody>
</table>
Problem 1.  

Match the following Java concepts to their definitions; write the number of the appropriate definition next to each concept. One definition will not be used. Each correct answer is worth 6 points.

<table>
<thead>
<tr>
<th>Concept</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>____ static</td>
<td>1   A primitive data type that can have the values true or false.</td>
</tr>
<tr>
<td>____ public</td>
<td>2   A loop that can be used to process each element in an array.</td>
</tr>
<tr>
<td>____ for-each loop</td>
<td>3  A loop that repeats its action again and again until a specified boolean expression becomes false.</td>
</tr>
<tr>
<td>____ boolean</td>
<td>4   A special variable that can be used within a method which refers to the object on which the method was invoked.</td>
</tr>
<tr>
<td>____ while loop</td>
<td>5   An instance variable that does not belong to a single object, or a method that is not invoked on a specific object.</td>
</tr>
<tr>
<td>____ this</td>
<td>6   A Java keyword used when you invoke the constructor for a class.</td>
</tr>
<tr>
<td></td>
<td>7   A variable or method that can be accessed in any Java class, not just the one where it is defined.</td>
</tr>
</tbody>
</table>
Problem 2

We have provided you with the code for a class called User, which you will refer to in this problem and the next. This class is designed to represent basic information about a user in a social networking system: their name and their friends.

We defined the instance variables and a few simple methods for the User class. We show the entire implementation except for the definition of the addFriend method. The key relevant detail to notice is that friends are stored in a fixed length array whose length is specified by the static variable maxNumFriends.

Your job in this problem is to complete the definition of addFriend. As noted in the comment before the method, the “friends” relationship is symmetrical (as in Facebook): if A is a friend of B, then B is a friend of A.

Specifically, please fill in the blanks in the addFriend method as instructed by the comments in the code. Each correctly filled-in blank is worth 5 points.

```java
public class User {
    private String name;
    // the other users who are a user's friends are stored in the
    // array variable friends
    private User[] friends;
    private int numFriends;
    private static final int maxNumFriends = 150;

    // make a new user. Initially the user has no friends, but has
    // room for the maximum allowable number of friends.
    public User (String n) {
        name = n;
        numFriends = 0;
        friends = new User[maxNumFriends];
    }

    public String getName() {
        return name;
    }

    public void printFriends () {
        if ( numFriends > 0 )
            for (int i = 0; i < numFriends; i++)
                System.out.format("%s\n", friends[i].getName());
        else
            System.out.println (" No friends.");
    }

    public String toString() {
        return String.format("%s \t %d", name, numFriends);
    }

    // Attempt to make two users be friends. "friend" is a symmetrical
```
// relationship (like in Facebook): if A is a friend of B, then B is
// a friend of A.
// f is the user to try to add as a friend of this user.
// addFriend returns true if the user f could be added as a friend of
// this user and false if not. The only reason a friend cannot be added
// is if one of the two users already has the maximum number of friends.

public boolean addFriend(User f) {

    // Write a condition to check that both the user on which
    // addFriend was invoked and the other user (f) have room for
    // more friends.
    if (______________________________)
    {
        // Write a statement to add f to this user’s friends

        ______________________________

        // Write a statement to add this user to f’s friends

        ______________________________

        // Update this user’s number of friends
        numFriends++;

        // Update the other user’s number of friends
        f.numFriends++;

        // Write a statement to return the appropriate value

        ______________________________
    }

    else {
        // Write a statement to return the appropriate value

        ______________________________
    }
}
Problem 3 / 20 points
This problem will test your knowledge of how information is passed into Java methods. Your job is to show the output of executing the main method below. Refer back to the definition of the User class from the previous problem as necessary. Even if you were not able to write the addFriends method completely or correctly, you still should be able to answer this question. There are 5 lines of output: each correct line is worth 4 points.

To give you an example of the format of the output, suppose we have a user u whose name is “Bob” and u is friends with users named “Tom” and “George. The output of the following code snippet

```java
System.out.println(u);
u.printFriends();
```

would be:

```
Bob  2
    Tom
    George
```

```java
public class SampleOutputProblem {
    public static void tryAddingFriend (User u, User f) {
        f = new User("Larry");
        u.addFriend(f);
        System.out.println(f);
    }

    public static void main(String[] args) {
        User u = new User("Mark");
        System.out.println(u);

        User newFriend = new User("A new friend");
        tryAddingFriend(u, newFriend);

        System.out.println(u);
        u.printFriends();
        System.out.println(newFriend);
    }
}
```
Problem 4

Your job in this problem is to complete a program that does the following:
- Reads in data from a file where each line consists of a student ID followed by the quiz scores of that student.
- For each student, computes the average quiz score ignoring the lowest score. So for example, if a student got a 90, 80, 70, and 60, her average would be 80. Note that you drop only one quiz score, even if two are tied for the lowest.
- Prints the average score for each student.

Most of the code is provided for you: as in Problem 2, you have to fill in the blanks. However, in one case, the blank is bigger: you’ll have to write an entire loop.

The code you’ll fill in uses methods from the GradeReader class. You don’t need to see the implementation details of this class; we simply describe the key methods. Also note that the GradeReader class is very similar to the TweetReader class from Homework 1 and the TweetReader class from Lab 3.

```java
public class GradeReader {
    // Highlights of the class GradeReader
    // Some instance variables were declared here

    /**
     * Construct a new GradeReader to read grades from a file.
     * @param filename The file to read grades from.
     */
    public GradeReader(String filename) {
        ...
    }

    /**
     * Advances the grade reader to the next student.
     * @return {code true} if there is another student to access, false otherwise.
     */
    public boolean advance() {
        ...
    }

    /**
     * Get the current student's ID.
     * @return The ID of the current student
     */
    public int getStudentID() {
        ...
    }

    /**
     * Returns the scores for all the quizzes of a student
     * @return an array of doubles: all the quiz scores for the student
     */
    public double[] getQuizScores() {
        ...
    }
}
```
public static void main(String[] args) {
    // Instantiate a new GradeReader to read data from the file
    // named “students.dat”. The variable gr will refer to the new object.

    GradeReader gr = _________________________________

    while (gr.advance()) {
        int studentId = gr.getStudentID();

        // Supply the proper type for the variable scores
        ______________ scores = gr.getQuizScores();

        double sumOfScores = 0;
        // The variable minScore will hold the minimum quiz score seen
        // so far for each student. At the end of the loop you’ll write,
        // minScore will be the overall lowest score for the student.
        // Supply an appropriate initial value for minScore.
        double minScore = _______________________________

        // Write a for loop to sum the quiz scores. Make sure your loop
        // does not include the minimum quiz score in the sum! Note that
        // you only need a single loop.

        ____________ avg = sum / (scores.length - 1);
        System.out.format("%d\t%.2f", studentId, avg);
    }
}