

COMP254 Project 1: Comparing Vectors and Linked Lists

Imagine you are part of a team at a small software company that is developing a new massively multiplayer game. Every player of the game has a *character* (a virtual person) that participates in the game's virtual world. One important feature of the game is that it permits very large numbers of characters to assemble in the same location in the virtual world; a collection of characters in the same location is called a *crowd*. While some members of your team are working on creating state-of-the-art graphics for displaying crowds, and others are working on novel techniques for simulating the behavior of crowds, you have been given the specific task of determining which of two possible data structures will be most suitable for storing data about large crowds within the application. Although no one in the team really knows how popular the new game will be, it is hoped that the game will eventually have hundreds of thousands of characters. Therefore, the chosen data structure needs to work efficiently for crowds of only a few characters up to at least 100,000 characters.

The application is being written in Java. The team has determined that crowds will be stored using either the Vector class from the java.util package, or the LinkedList class, also from the java.util package. It is your job to perform some experiments to investigate whether Vectors or LinkedLists are more suitable, and write a memo to the other members of your team explaining and justifying your results.

The code for the Character class is given in the following listing:

```
class Character {
    // the age of this character in years
    private double age;
    // the height of this character in meters
    private double height;

    /**
     * @param age
     *         the age of this character in years
     * @param height
     *         the height of this character in meters
     */
    public Character(double age, double height) {
        this.age = age;
        this.height = height;
    }

    /**
     * @return the age of this character in years
     */
    public double getAge() {
        return age;
    }

    /**
     * @return the height of this character in meters
     */
    public double getHeight() {
        return height;
    }

    /**
     * This method is intentionally empty at present. The
     * development team will fill it in later with some
     * useful computations that are used to "process" the
```

```

        * current character in order to simulate crowd
        * behavior, but for our purposes, it is fine to assume
        * that the "process" method does nothing.
        */
    public void process() {
    }
}

```

Note the “process()” method of the Character class: some other members of the team are working on this method, which will compute some interesting crowd behavior based on the character's fields. For your experiments, it is fine to assume that the process() method does nothing. However, the team has already determined that the characters in a crowd will need to be processed in *FIFO order* (i.e. first in, first out). In particular, the data structure you recommend for storing a crowd will need to support the following sequence of operations:

1. Create a new crowd, and add a fixed number of characters to it in a given order.
2. Remove the characters one at a time from the crowd, in the same order in which they were added, calling the process() method on each character after it is removed from the crowd.

In other words, the Java code for processing a crowd will closely resemble the following method:

```

// create a Vector with the given number of elements,
// and then remove all elements from the Vector one at a
// time, removing from the *head* of the Vector, and
// "processing" each element in turn.
public static void doVectorExperiment(int numElements) {
    // STEP 1. create the list and add elements to it
    Vector<Character> vectorCrowd = new Vector<Character>();
    for (int i = 0; i < numElements; i++) {
        // the values 22.5 and 1.6 are chosen
        // arbitrarily and have no particular
        // significance for this experiment.
        vectorCrowd.add(new Character(22.5, 1.6));
    }

    // STEP 2. remove elements from the head of the
    // list, one at a time, processing each one in
    // sequence
    for (int i = 0; i < numElements; i++) {
        Character character = vectorCrowd.remove(0);
        character.process();
    }
}

```

Note that the above method uses the Vector class; your experiments will need to employ an additional, very similar, method that uses the LinkedList class.

Your memo should be 3–5 pages in length. Any Java code you use for experiments should be included in the memo as an appendix, but this appendix does not count as part of the suggested length of 3–5 pages. The memo should start with a very clear and concise summary of the task you addressed, and your conclusions. (You need to describe the task, because not all members of the team will be aware of the job you were given. You need to summarize your conclusions at the start, because not all of your colleagues will have the time or inclination to read your entire memo.) Following this summary, you should give a complete description of your experiments and the reasoning that led to the stated

conclusion. You must give sufficient details that anyone reading the memo could replicate the experiments. Any data that you use should be presented in a suitable form, such as a table or a graph. At the end of the memo, briefly summarize your conclusions again.