

CS1073 - Introduction to Computer Programming I (in Java)

Andrew McAllister, Ph.D.

The purpose of this document is to help you prepare to write the final exam in this course.

On the following pages you will find a number of example exam questions. The best use of this resource is to answer the example questions as if you were writing a final exam. Do not consult other resources like your course notes or the textbook. Do not Google for help. Do not attempt to compile and run the programs you write. Remember – on the final exam you get graded for the *first* attempt you make at each question, so do the same when you're practicing with these example questions.

There may be more room provided than is required for some answers.

The second half of the document provides sample answers to the example questions. You will get the most value from this document if you can resist the temptation to look at these answers until you have given your best effort to answer the example questions.

Rest assured that these example questions are a good indication of the types of questions you can expect on the final exam for this course. These questions have been taken from actual exams from previous offerings of the course ... and they've been updated slightly to be consistent with the way material is presented in this open access version of the course.

Good luck with your preparation and with the final exam!

A handwritten signature in blue ink, which appears to read 'Andrew', is located at the bottom left of the page.

Question 1: Testing Tires

Examine the following two classes:

```
public class Tire
{
    private int diameter;

    public Tire (int diameterIn)
    {
        diameter = diameterIn;
    }

    public void inflate (int amt)
    {
        diameter = diameter + amt;
    }

    public int getDiameter ()
    {
        return diameter;
    }
}
```

```
public class TireTest
{
    public static void main (String[] args)
    {
        Tire firestoneTire;
        Tire blizzardTire = new Tire(21);
        Tire uniRoyalTire = new Tire(blizzardTire.getDiameter());

        blizzardTire.inflate(5);
        firestoneTire = blizzardTire;
        int amount = 2;
        firestoneTire.inflate(amount);

        System.out.println(firestoneTire.getDiameter());
        System.out.println(blizzardTire.getDiameter());
        System.out.println(uniRoyalTire.getDiameter());
    } // end main
} // end TireTest
```

What will be the output when the TireTest class is executed?

Question 2: Candidate

Write a complete Java class called **Candidate**.

Include two instance variables:

- The name of the candidate; and
- The number of votes they have received in an election;

The constructor accepts the name of the candidate as a parameter and sets the number of votes to zero.

Include the following methods:

- `getName()` – Standard accessor method
- `getNumberOfVotes()` – Standard accessor method
- `addVote()` – Adds one vote to this candidate's number of votes every time this method is called
- `toString()` – Standard `toString()` with the typical result

Provide your answer on this page and the next page.

This image shows a single sheet of white paper with horizontal blue ruling lines. The lines are evenly spaced and run across the width of the page. There are no margins, text, or other markings on the paper.

Question 3: Election

Write a complete Java class called **Election**.

The constructor has a single int parameter, which sets the maximum number of candidates that can be involved in this election. This number is used to create an array capable of holding that number of Candidate objects. (That array is an instance variable.) Initially this array includes zero Candidate objects.

The only other instance variable is numberOfCandidates, which the constructor sets to zero.

The Election class includes three methods:

- The **addCandidate()** method accepts a candidate name as its parameter. If the array of candidate names still has empty elements, then this name is added to the array of candidate names. Otherwise, the array is not changed.
- The **addVote()** method accepts a candidate number as its parameter (that is, the index in the array where the candidate can be found). If this index is valid (based on the current number of candidates), then a vote is added for this Candidate object.
- The **getWinner()** method determines which candidate received the largest number of votes and returns an appropriate message. For example, if a candidate with the name “Jean” has 1000 votes, and the only other candidate (named “Paul”) has 500 votes, then the result returned should be a String like:

Jean won with 1000 votes

Do not attempt to figure out if several candidates win with the same number of votes. If this happens, your program can return the name of any one of them as the winner.

If there are no candidates at the time this method is called, your method should return: No candidates

Provide your answer on the following pages.

This image shows a single sheet of white paper with horizontal ruling lines. The lines are evenly spaced and run across the width of the page. There are no margins, text, or other markings on the paper.

This image shows a single sheet of white paper with horizontal blue ruling lines. The lines are evenly spaced and run across the width of the page. There are no margins, text, or other markings on the paper.

Question 4: Favourite Song

Write a complete Java class called **FavouriteSong**, which has a main() method. Create an Election instance called songElection that can have a maximum of 10 candidates.

Add three of your favourite song titles as candidates in this election.

Using any of the approaches mentioned in this course for generating random numbers, write a loop that calls the election's `addVote()` method 100 times, each time randomly voting for one of the three candidate songs. Finally, display the winner.

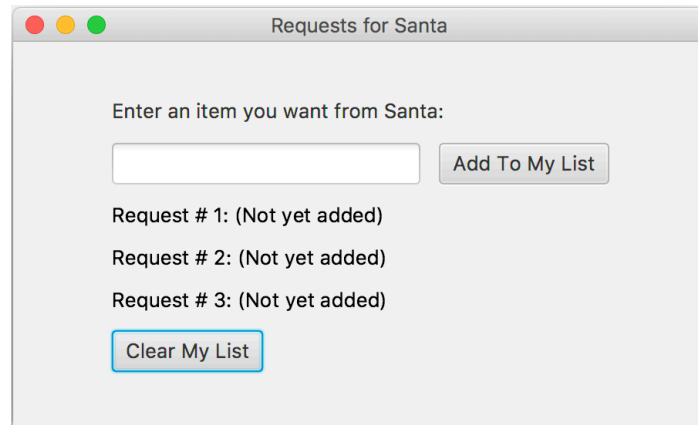
Provide your answer on this page and the next page.

[illegible]

[illegible]

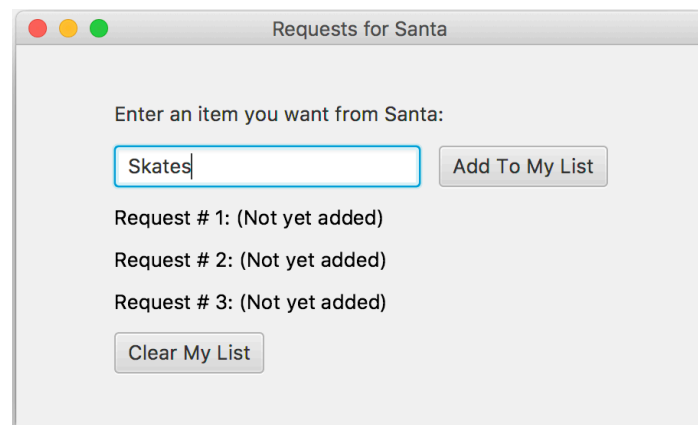
Question 5: Requests For Santa

An incomplete JavaFX program appears on pages 12-14. Fill in the blanks to complete the program so it performs as follows. The initial appearance of the application is:



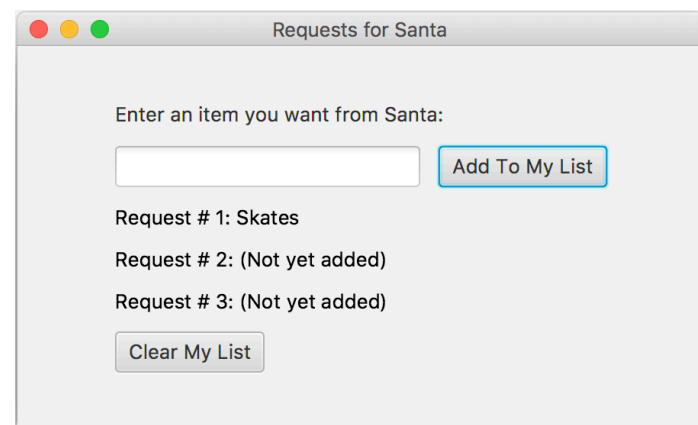
The application window titled "Requests for Santa" has a light gray background. At the top, there are three colored window control buttons (red, yellow, green). Below them is the text "Enter an item you want from Santa:" followed by a text input field and an "Add To My List" button. Below the input field, there are three lines of text: "Request # 1: (Not yet added)", "Request # 2: (Not yet added)", and "Request # 3: (Not yet added)". At the bottom, there is a "Clear My List" button.

The user can type an entry in the text field...



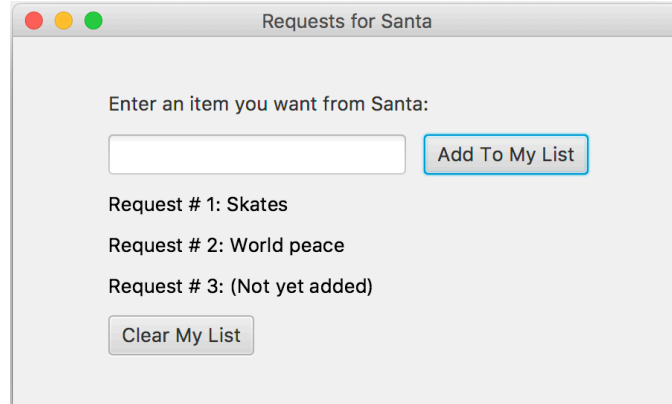
The application window is the same as before, but the text input field now contains the word "Skates". The "Add To My List" button and the list of requests remain unchanged.

...and press the "Add To My List" button so the list is altered as follows. Note the text field is reset to blank:



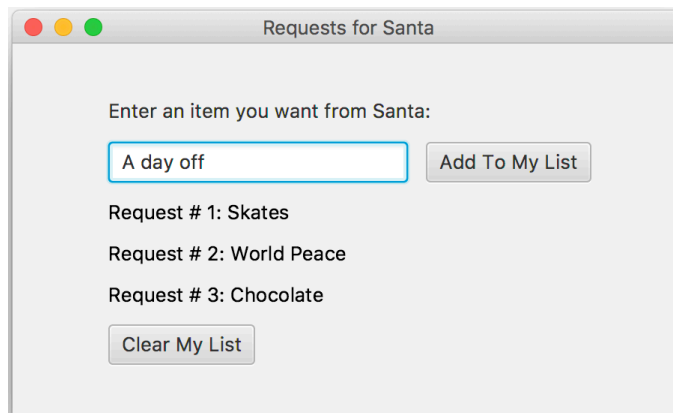
The application window shows the result of pressing the "Add To My List" button. The text input field is now blank. The first item in the list, "Request # 1: (Not yet added)", has been replaced with "Request # 1: Skates". The other two items, "Request # 2: (Not yet added)" and "Request # 3: (Not yet added)", remain the same. The "Clear My List" button is still at the bottom.

After a second addition, the appearance is:



The screenshot shows a window titled "Requests for Santa". It contains a text input field with the placeholder text "Enter an item you want from Santa:". To the right of the input field is a button labeled "Add To My List". Below the input field, there is a list of requests: "Request # 1: Skates", "Request # 2: World peace", and "Request # 3: (Not yet added)". At the bottom of the window is a button labeled "Clear My List".

Only three items can be added to the list in this way. Once all three requests have been added, any further attempts (such as shown below) are ignored (but after pressing "Add To My List" the text field goes blank).



The screenshot shows the same window titled "Requests for Santa". The text input field now contains the text "A day off". The "Add To My List" button is now disabled (grayed out). The list of requests has been updated: "Request # 1: Skates", "Request # 2: World Peace", and "Request # 3: Chocolate". The "Clear My List" button remains at the bottom.

Pressing "Clear My List" resets the application to the original appearance and allows the user to begin adding requests to the list once more.

(Continued on the next page...)

Fill in the blanks to complete the following program so it performs as described on the preceding pages:

```
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.scene.layout.*;
import javafx.scene.text.*;
import javafx.geometry.*;
import javafx.event.ActionEvent;

public class Santa extends Application
{
    //***** Instance variables *****

    private Label      enterLabel;
    private TextField  entryField;
    private Text[]     requestArray; // Note the array!
    private Button     addButton;
    private Button     clearButton;
    private final int  MAXREQUESTS = 3;
    private int        numRequests;

    //***** start method *****
    public void start(Stage primaryStage)
    {
        _____("Requests for Santa");
        numRequests = 0;

        // Create the GUI components
        enterLabel = new _____
            ("Enter an item you want from Santa:");

        entryField = new TextField();
        entryField.setPrefWidth(200);

        // Create the array of Text objects
        requestArray = new _____;
        for (int i=0; i<_____; i++)
        {
            requestArray[i] = new _____("Request # "
                + (_____)
                + ": (Not yet added)");
        }
    }
}
```

```

addButton = new Button("Add To My List");
addButton.setOnAction(this::eventHandler);

clearButton = new Button("Clear My List");
clearButton.setOnAction(this::eventHandler);

GridPane mainPane = new _____;
mainPane.setAlignment(Pos.CENTER);

mainPane.add(_____, 0,0,2,1);
mainPane.add(_____, 0,1,1,1);
mainPane.add(_____, 1,1,1,1);

for (int i=0; i<requestArray.length; i++)
    mainPane.add(_____, 0,2+i,2,1);

mainPane.add(_____, 0,5,1,1);

mainPane.setHgap(12);
mainPane.setVgap(12);

Scene scene = new Scene(mainPane, 450, 250);
primaryStage.setScene(scene);
primaryStage.show();
} // end start()

```

// Continued on the next page...

```

// Note: There may be more lines provided than you need

public void _____(ActionEvent event)
{    // Handle a press of addButton
    if ( _____ )
        _____
        _____
        _____
        _____
        _____
        _____
        _____
        _____
        _____
        _____

    else // Handle a press of clearButton
        _____
        _____
        _____
        _____
        _____
        _____
        _____
        _____
        _____
        _____

    // Blank out the entryField regardless of which
    // button was pressed
    entryField.setText( _____ );

    } // end method
} // end class

```

Question 6: 2D Array

What does the following program print?

```
public class Q6
{
    public static void main(String[] args)
    { int[][] stuff = new int[3][2];

        for (int a=0; a < stuff[0].length; a++)
        { for (int b=0; b < stuff.length; b++)
            { stuff[b][a] = (a * 5) + (b * -3);
            }
        }

        for (int a=stuff.length-1; a >=0 ; a--)
        { for (int b=0; b < stuff[0].length; b++)
            { System.out.print(stuff[a][b] + " ");
            }
            System.out.println();
        }
    } // end main method
} // end class
```

Provide your answer in the space provided below.

Sample Solutions

The following pages provide sample solutions for the example questions.

If you haven't yet attempted the questions, it is highly recommended that you do so before checking out the sample solutions.

Sample solution for **Question 1: Testing Tires**

The output from executing TireTest is:

28
28
21

The TireTest class has three variables that are capable of pointing to a Tire object.

This variable:

```
Tire firestoneTire;
```

initially points to no object – it contains the value null

Then we create two variables, each of which point to separate Tire objects that both contain the number 21:

```
Tire blizzakTire = new Tire(21);  
Tire uniRoyalTire = new Tire(blizzakTire.getDiameter());
```

Then we change the number in one of the objects to be 26:

```
blizzakTire.inflate(5);
```

Now here's a key to answering this question – this next line does NOT create a third Tire object. Instead, we end up with firestoneTire and blizzakTire both pointing to the same Tire object. (*This is similar to the discussion at the beginning of the instructional video for Module 10.*) That object, you'll recall, currently has the number 26:

```
firestoneTire = blizzakTire;
```

That object pointed to by two variables – the value inside that object now becomes 28:

```
int amount = 2;  
firestoneTire.inflate(amount);
```

So that's why 28 is printed twice (because two variables both point to the same object) and the uniRoyalTire value is unchanged – still 21.

Sample solution for **Question 2: Candidate**

This question tests whether you can write a very basic class that creates objects. If you struggled with this question, you should do some additional review before attempting the final exam.

```
public class Candidate
{
    private String name;
    private int    numberOfVotes;

    public Candidate(String nameIn)
    {    name = nameIn;
        numberOfVotes = 0;
    }

    public String getName()
    {    return name;
    }

    public int getNumberOfVotes()
    {    return numberOfVotes;
    }

    public void addVote()
    {    numberOfVotes++;
    }

    public String toString()
    {    return "Candidate[name="
        + name
        + ", numberOfVotes="
        + numberOfVotes + " ]";
    }
}
```

Sample solution for **Question 3: Election**

This question includes a “Has A” relationship plus an array

```
public class Election
{
    private Candidate[] candidate;
    private int numberOfCandidates;

    public Election(int maxCandidates)
    {
        numberOfCandidates = 0;
        candidate = new Candidate[maxCandidates];
    }

    public void addCandidate(String name)
    {
        if (numberOfCandidates < candidate.length)
        {
            candidate[numberOfCandidates] = new Candidate(name);
            numberOfCandidates++;
        }
    }

    public void addVote(int index)
    {
        if (index >= 0 && index < numberOfCandidates)
            candidate[index].addVote();
    }

    public String getWinner()
    {
        if (numberOfCandidates == 0)
            return "No candidates";
        int largestIndex = 0;
        for (int i=1; i<numberOfCandidates; i++)
        {
            if(candidate[i].getNumberOfVotes() >
                candidate[largestIndex].getNumberOfVotes())
            {
                largestIndex = i;
            }
        } // end for
        return candidate[largestIndex].getName()
            + " won with "
            + candidate[largestIndex].getNumberOfVotes()
            + " votes";
    } // end getWinner()
}
```

Sample solution for **Question 4: Favourite Song**

```
public class FavouriteSong
{
    public static void main(String[] args)
    {
        Election songElection = new Election(10);

        songElection.addCandidate("Midnight Train to Georgia");
        songElection.addCandidate("Sledgehammer");
        songElection.addCandidate("Peaceful Easy Feeling");

        for (int i=1; i<=100; i++)
            songElection.addVote((int)(Math.random()*3.0));

        System.out.println("\n"+songElection.getWinner()+"\n");
    }
}
```

Sample solution for **Question 5: Requests For Santa**

This question tests your knowledge of building a GUI application with JavaFX.

```
/** Solution to the JavaFX GUI question in the
    Example Exam Questions for the open access
    version of CS1073.

    @author Andrew McAllister
 */
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.scene.layout.*;
import javafx.scene.text.*;
import javafx.geometry.*;
import javafx.event.ActionEvent;

public class Santa extends Application
{
    //***** Instance variables *****

    private Label    enterLabel; // Could also use Text
    private TextField entryField;
    private Text[]   requestArray; // Or 3 separate Text objects
    private Button   addButton;
    private Button   clearButton;
    private final int MAXREQUESTS = 3;
    private int      numRequests;
```

```

//***** start method *****
public void start(Stage primaryStage)
{
    primaryStage.setTitle("Requests for Santa");
    numRequests = 0;

    // Create the GUI components
    enterLabel = new Label
        ("Enter an item you want from Santa:");
    entryField = new TextField();
    entryField.setPrefWidth(200);

    requestArray = new Text[MAXREQUESTS];
    for (int i=0; i<requestArray.length; i++)
    {
        requestArray[i] = new Text("Request # "
            + (i+1)
            + ": (Not yet added)");
    }

    addButton = new Button("Add To My List");
    addButton.setOnAction(this::eventHandler);

    clearButton = new Button("Clear My List");
    clearButton.setOnAction(this::eventHandler);

    GridPane mainPane = new GridPane();
    mainPane.setAlignment(Pos.CENTER);

    mainPane.add(enterLabel, 0,0,2,1);
    mainPane.add(entryField, 0,1,1,1);
    mainPane.add(addButton, 1,1,1,1);

    for (int i=0; i<requestArray.length; i++)
        mainPane.add(requestArray[i], 0,2+i,2,1);

    mainPane.add(clearButton, 0,5,1,1);

    mainPane.setHgap(12); // optional
    mainPane.setVgap(12);

    Scene scene = new Scene(mainPane, 450, 250);
    primaryStage.setScene(scene);
    primaryStage.show();
} // end start()

```

```

//***** eventHandler method *****
public void eventHandler(ActionEvent event)
{
    if (event.getSource() == addButton)
    {
        if (numRequests < MAXREQUESTS)
        {   requestArray[numRequests].setText("Request # "
            + (numRequests+1) + ": "
            + entryField.getText());
            numRequests++;
        }
    }
    else // Then it had to be the Clear button
    {
        for (int i=0; i<requestArray.length; i++)
        {   requestArray[i].setText("Request # "
            + (i+1)
            + ": (Not yet added)");
            numRequests = 0;
        } // end for loop
    } // end else

    // The entry field is cleared regardless of which
    // button was pressed.
    entryField.setText("");

} // end eventHandler()

} // end class

```

Sample solution for **Question 6: 2D Array**

The output produced by the Q6 program is:

```
-6  -1  
-3  2  
0   5
```

If you struggled to figure this out, I recommend you go back and try again. This time, draw a picture of the 2D array elements, then do your best to trace the values of all the variables as the program executes.

You can also add a number of print statements to that program and run it yourself to see what some of the partial results are along the way.