# CS 273 Laboratory 11: Java Libraries

This laboratory introduces you to Java’s standard libraries and the use of API (Application Programming Interface) documentation to learn how to use a few of the classes associated with graphics user interfaces (GUIs) in Java.

## Java Libraries

Java has many predefined classes. We’ve seen a few of these already, such as String, Color, Graphics and Math. These classes are collected into packages of related classes which you can import into your program.

Your textbook (i.e. zybooks) covers only a few of the many class libraries in Java; however, for learning how to use a new class, Oracle’s Java API web pages can be very helpful: [**http://docs.oracle.com/javase/8/docs/api/**](http://docs.oracle.com/javase/8/docs/api/)**.**

The page is made up of three frames: two small ones on the left of the page and a larger one on the right. The two frames on the left allow you to browse a list of Java packages (the top left frame) or browse a list of all individual classes in alphabetical order (the bottom left frame). When a package is selected, the lower left frame will show only the classes belonging to the selected package; to get back the list of all classes in all packages, select the "All Classes" link at the top of upper left frame. All parts of the page are searchable, so using your browser’s Find feature (Ctrl + f on Windows, Command + f on Mac) to navigate will often be faster than scrolling.

To see detailed information about a class, click it in the lower left frame. The detailed information will be displayed in the large frame on the right. The class information is typically composed of the following sections:

* a short description of the class
* "Field Summary": Describes the public and protected attributes of the class
* "Constructor Summary": A list of all the available constructors for the class
* "Method Summary": Lists the public and protected methods of the class

Clicking on any of the hyperlinks in the summaries will bring you to the part of the page with more specific details about that item.

**Pro Tip:** Often you can find the API page for the class you want quickly by searching for it using Google. Include “java 8” in your search. For example, if you search for “java 8 jframe” the first hit will likely be the documentation for the JFrame class.

## Laboratory

**There is no starter code for this lab. Instead, follow the directions below to get started.**

Run BlueJ and create a new project in your lab11 folder. Create a new class: JavaAPI. Delete the example code and start fresh with a class definition that contains a main method. To confirm you’re ready, add a System.out.println call to your main method and print out your name. Then run the program to verify that it works.

### Part 1: Explore the Java API

Find the documentation for the String class and display its details in the large frame.

Notice that right above the words Class String are the words java.lang in a smaller font. java.lang is the package in which the String class exists. You already know that often you need to add an import statement to your program to make certain classes available. This is how you find out what import statement you need. As it happens, java.lang is Java’s default package. You never need to add an import statement for java.lang because these classes are always available. If a different package name appears there, you need to add an import statement to your program to use that class. ***You will need to use this fact later in this lab.***

Jill has two strings s1 and s2 that each contain different email addresses (e.g., bill@microsoft.com). She wants to know if both addresses are from academic institutions. The easiest way to do this is to see if both addresses end with “.edu”. Follow the steps below to write the code that Jill should use.

1. Read in an email address as String value from the user.
2. Looking at the String class API, find the best method for determining if a String ends in “.edu”. (Hint: There are better methods than substring and contains.)
3. Using this method, write a short program that demonstrates whether the given input ends in “.edu”.
4. Update your program to read in two String values. Using a single if statement and the method from step #3, write a short program that determines whether both email addresses end in “.edu”.

**checkpoint 1 (15 points): Show your lab instructor or assistant that your program can accept two String values and correctly identify whether they both end in .edu. That means you should have at least one example where they both end .edu and one example where both of them don’t.**

### Part 2: Understanding Components

Answer the questions below:

1. Navigate to the API for the Component class. According to the documentation, what are three examples (known subclasses) of components?
2. Which of the following classes is not a Component (i.e. they are not a descendent of the Component class): Choice, JTextArea, FlowLayout, Canvas? At the top of each class in the Java API, right under the class name there is an inheritance tree. Make sure to check the entire inheritance tree as you search for the answer to this question.

**checkpoint 2 (10 points): Show your lab instructor or assistant your answers.**

### Part 3: Using the JFrame Class

In this section, you will add some code to your class that creates an empty window on the screen. Locate the JFrame class in the Java API documentation. Then follow these steps to use that class in your program.

1. Add the required import statement to the top of your JavaAPI class. You should be able to determine what you need to import by looking at the very top of the JFrame API. Following the name of the library, add a dot followed by the class name. For instance, if you had to do it for the String class, it would be import java.lang.String.
2. Comment out any existing code in the main method that you created for previous checkpoints.
3. Create a default constructor for your JavaAPI class.
4. In the constructor, create a variable of type JFrame. Then, assign a new JFrame object to that variable. Use the constructor that will allow you to set the window title to “Fresh Green Beans”. Another way to say this is that you need to instantiate a new JFrame object and pass “Fresh Green Beans” into the constructor.
5. Next, you need to tell the JFrame that it should be visible on the screen.
   1. The method for doing this has been inherited by JFrame from another class. Find this method and use it in your code to make the JFrame visible.
   2. Add a comment above that line of code to indicate which class the method was inherited from.
6. Now we need to set the size (width and height) of the window.
   1. Locate the required method. (It has also been inherited.)
   2. Set the window size to 300x300.
7. Finally, we want the program to exit when the user clicks the close button on the top-right corner of the window frame. JFrame has a particular method for specifying this behavior. Locate and use this method on your window to get the desired behavior. **Hint:** You will need to pass in a particular value that has been declared in the JFrame class. Access the value via that class like so, JFrame.THE\_VALUE.
8. Have the main method create an instance of your JavaAPI class by calling the constructor.
9. Declare a method named layout in your code that accepts a JFrame object. The method should do nothing for now.   
    private void layout(JFrame jframe)
10. Call the layout method in your constructor, passing it the name of your JFrame object.

**checkpoint 3 (30 points): Show your lab instructor or assistant that your program creates a 300x300 window with the title “Fresh Green Beans” and the code you wrote to make that happen.**

**Note**: In the remainder of this lab, many of the details about how to use a class are not given so that you will gain experience using the API documentation.

### Part 4: Using Components

The layout method you created will be used to layout the content of the window.

First, you must set the layout manager for the window. The layout manager determines how the components will be arranged inside the window. The simplest layout manager is FlowLayout. Add code to your layout method that creates a FlowLayout object and then sets it as the layout manager for your JFrame object. You will likely need to add a new import statement to the top of your file to support this.

Now add code to the layout method that creates the following components. You may need to add new import statements to the top of your file to support this.

1. A JButton labeled “Clear”
2. A JTextArea that has 5 rows and 25 columns. Insert some initial text into this text area that reads: “I love to eat fresh green beans!”
3. A JButton labeled “Append”

Add code to the layout method to add the three components above to your window.

**checkpoint 4 (20 points): Show your lab instructor or assistant that your program now has two buttons and a text area. If you make your window wider, you’ll see the components adjust accordingly.**

### Part 5: Learning about Action Events

When the user presses a button, it creates an event. Right now, your program does nothing on a button press. If you want to have your program respond, your class needs to be able to “listen” for action events from the buttons.

Use the Java API to read about the ActionListener interface.

1. How many methods are declared in the ActionListener interface?

Note, an ActionEvent occurs whenever an action is performed by the user. The ActionEvent object is passed to the object implementing the ActionListener interface.

1. What method on a JButton object would you call to add an action listener to it? Check the JButton API.

**checkpoint 5 (5 points): Show your answers to your lab instructor or assistant.**

### Part 6: Implement the Clear Button

In this part of the lab, you will implement the 'Clear' button so that when a user presses it, all the text in the text area will be removed.

1. In your layout method, you created a JTextArea object. To make the object accessible to other methods in the class, declare an instance variable in the JavaAPI class and have the layout method assign the JTextArea object to this variable. You must decide whether your new instance variable should be public or private.
2. You should import the ActionListener interface and the ActionEvent class.
3. Implement ActionListener with your JavaAPI.This will create a compiler error until you complete step 4.
4. To fulfill its obligation as an implementor of ActionListener, create an actionPerformed() method in the JavaAPI class. This method must have exactly the definition required by the ActionListener interface. When called, the method should clear the content of the JTextArea by setting it to an empty string (which is not the same as a space between quotes). **Hint:** The best method for setting the text in the JTextArea is a method that JTextArea inherited from the JTextComponent class.
5. In your layout method, the JavaAPI object should add itself as the action listener for the “Clear” JButton. **Hint:** What’s the keyword for a class to refer to itself?

**checkpoint 6 (15 points): Show your working Clear button to your lab instructor or assistant.**

### Part 7: Implement the Append Button

Implement the Append button so that when the user presses it, “ Yum!” is appended to the existing text in the text area.

To do this, your actionPerformed method will need to distinguish between events that it received from the Clear button vs. the Append button. You can accomplish this by examining the ActionEvent object that is passed to the actionPerformed method using the getSource method. Don't forget to add a listener to the Append button.

**checkpoint 7 (5 points): Show your lab instructor or assistant that both your Append button and your Clear button behave correctly.**

**EXTRA CREDIT (optional)**

Write another mini-program to show your ability to understand the String API. You can comment out the code for checkpoints 1-7.

Find the documentation for the String class and display its details in the large frame.

Suppose Costco’s membership card is stamped with a unique ID that contains three sections separated by hyphens. These sections are always in the form of four digits, dash, four characters, dash, eight digits. For example, an ID might be   
“1994-NEPR-48573923”.

regionMatches is a good method to use to determine if a portion of two ID strings match. Follow the steps below to demonstrate.

* + - 1. Instead of reading in input from the user, we can create variables and set the values as we want to make for faster testing. So, create two String variables and set the value of each variable to be in the form ####-AAAA-######## (four numbers, four letters, 8 numbers). Set the value of the four letters to be the same in both Strings.
      2. Using the regionMatches method, write a short program to demonstrate that the two String variables contain the same four letters. To learn how to use regionMatches, consult the String API. Note, the Method Summary section of the API gives you a high-level summary about each method. For more details, click on the method name.
      3. Change the value of one of the variables so that the letters are no longer the same. Run your program again (you shouldn’t have to change anything else) to make sure you still get the proper result, which is that they are not the same.

**checkpoint EC1 (5 points): Show your lab instructor or assistant that your program can correctly identify whether two IDs have the same four letters or not. That means you should have one example where they are the same and one example where they are not.**