Seventh Lesson: The End of Java!

Hope you all had a great first two weeks of school! Today will be our last lesson covering the basics of Java, although we might return to it sometime later on to cover other material.

Characters

A `char` is another data type that we haven't used much. They can represent printable characters like `'d'`, `'!'`, and `'8'`. While strings are wrapped with double quotes (`""`), chars are generally wrapped with single quotes (`''`). Chars are ints that get printed as their corresponding ASCII values.

For example, the exclamation point '!' is the integer 33:

```
char ex = '!';
int exclamation = '!';

System.out.println(ex);  // Prints !
System.out.println(exclamation); // Prints 33

char exAscii = 33;
System.out.println(exAscii); // Prints !
```

Comparing Chars

Chars are just ints, so comparing them is relatively simple, as we only have to use the == operator rather than the `equals()` method for strings.

```
char c = 'a';
char q = 'a';
```

```
if (c == q){
  System.out.println("Equal");
}

else{
  System.out.println("Not equal");
}
```

Important String Methods

Here are some of the most common methods that you can use in your code!

- `charAt(int index)`
  - This identifies whatever character is at the specific index
- `contains(CharSequence s)`
  - Usually used with `booleans (returns True/False)`
  - Essentially checks if a sequence is present in the string
- `equals(String s)`
  - Checks if two strings are equal
- `length()`
  - Returns the length of the string
- `indexOf(int ch)`
  - Returns the index that a character is found
- `substring(int beginIndex)`
  - Will return the "cut" string starting from given index
- `substring(int beginIndex, int endIndex)`
  - inclusive of startIndex, exclusive of endIndex

Here's an example of some code that utilizes a few of the methods mentioned before.

```
public class ExampleString {
    public static void main(String[] args) {
          String str = "Summertime";
          int len = str.length();

          System.out.println(str + " is " + len + " characters
          long.");

          //Print each character of the string on a new line.
          for (int i = 0; i < len; i++){
               System.out.println(str.charAt(i));
          }

          String season = str.substring(0, 6);
```

```
            //Doesn't include the char at index 6

            String time = str.substring(6);
            //Start at index 6 and go to the end

            System.out.println(season);

            System.out.println(time);

            int eIndex = str.indexOf('e');
            //Gets the index of the first occurrence

            int rIndex = str.indexOf('r');

            System.out.println(eIndex + " " + rIndex);
      }
}
```

Basic UI Building

Some of you expressed a desire to build games. One of the vital components of a game is the visual part of it, the UI aspect, so here's a really basic guide to some of the jcomponents (swing components).

A JFrame is your main window where all of your graphics will be displayed. To create a JFrame, you would do something like this:

```
JFrame frame = new JFrame();
```

If you wanted to title your frame, you would put the title of your choice within the parenthesis as a string, like so:

```
JFrame frame = new JFrame("Title");
```

You won't be putting any graphics stuff on your JFrame though; this will all go onto a JPanel, which you would then add onto the JFrame so that it would be displayed onto the open window. A JPanel is a "general purpose container" that holds your graphics components. You can have multiple JPanels on a JFrame, and even JPanels on JPanels. Generally, you would have one main JPanel on which the others are laid on top of.

To create a JPanel, the syntax is very similar to when you're creating a JFrame.

```
JPanel panel = new JPanel();
```

To add JPanels onto JFrames or other JPanels, you would use the `.add()` method. Therefore, it would probably look like this:

```
frame.add(panel);
panel.add(panel1);
```

The same thing applies for if you want to add any GUI element onto your panel. If you wanted to add a button (the class is JButton), you would need:

```
JButton button = new JButton("button");
panel.add(button);
```

Now that you have the frame and the panel done, you can pretty much do whatever you want. Add circles, add squares, change the background color, add text, etc. This is a link to a visual guide of general swing components, including JOptionPanes, JButtons, and JLabels.

Once you've finished adding everything to your main panel and have added that panel to your frame, you must do `frame.pack()` in order for everything to show up.

Note: There are a couple links linked below. Please do check them before moving forward. They include much more information than what is covered here, and it will be necessary to help you understand some of what comes next.

To make the swing components "do something"--say, if you wanted a message to pop up when a button on your frame is clicked--you would either use an action listener or a mouse listener. Action Listeners are a bit more versatile since you don't necessarily need a GUI component to do anything (you could use a timer--more information can be found here regarding the topic). For example, if you wanted to create a button that prints "Hello World" every time it's clicked, you would need an action listener on the button.

Before we try coding the "Hello World" button, there's something else that needs to be covered: interfaces. Both ActionListener and MouseListener are interfaces, which are a bit like classes in that they're something like pre-written methods that are inherited by the class that extends it (`extends` being the keyword required to access these interfaces, whose files you will also have to import). A more detailed explanation of interfaces can be found here.

Here's a sample code for that:

```
public class HelloWorld() extends ActionListener {
     JFrame f; // create an object of JFrame, JPanel, JButton
     JPanel p;
     JButton b;

     public void displayText(){
```

```
            f = new JFrame(); // instantiate said objects
            f.setVisible(true); // makes the frame visible on
                                screen
            p = new JPanel();
            b = new JButton("Click me");
            b.addActionListener(this); // adds ActionListener

            p.add(b);
            f.add(f);
            f.pack();
      }

@Override // this is the imported method that comes from
                the ActionListener
      public void actionPerformed(ActionEvent arg0) {
            // arg0 represents the "action" that ActionListener looks/
                listens for
            if (arg0.getSource() == b) JOptionPane.showMessageDialogue
                                              (null, "Hello World");
            // arg0.getSource() tells you where the action originated
                from. For our purposes we want them to click the
                button "b," so that's what we check for.
      }

      public static void main(String[] args){
            new HelloWorld().run(); // creates an object of the
                            HelloWorld class and calls run method
      }
}
```

If you try running this in your editor, you should see a window pop up containing the button titled "Click me," and clicking the button should have a JOptionPane pop up saying "Hello World." If not, please email us so we can see where we messed up (we're human too!). Or, you could fix it on your own, and then email us with what went wrong.

This concludes the final Java lesson. We hoped you learned lots from these lessons, and that you're enjoying them as well. The next language is going to be HTML (probably with some CSS and Javascript as well because web development is usually one huge chunk of all three), so get excited!
Until next time, Emma and Nicole