COMP 110
Introduction to Programming

Fall 2015
Time: TR 9:30 – 10:45
Room: AR 121 (Hanes Art Center)

Jay Aikat
FB 314, aikat@cs.unc.edu

Previous Class

• What did we discuss?
Today

• Announcements
  • Lab1: **due Wed, Sep 9 at 11:55 PM**
  • Midterm is on Thu, Oct 8

• More If-else

Some Terminology follows...
Class Loader

- A Java program typically consists of several pieces called *classes*.
- Each class may have a separate author and each is compiled (translated into byte-code) separately.
- A *class loader* (called a *linker* in other programming languages) automatically connects the classes together.

Programmer, User, Package...

- The person who writes a program is called the *programmer*.
- The person who interacts with the program is called the *user*.
- A *package* is a library of classes that have been defined already.
  
  ```java
  import java.util.Scanner;
  ```
Arguments, Variables...

- The item(s) inside parentheses are called argument(s) and provide the information needed by methods.
- A variable is something that can store data.
- An instruction to the computer is called a statement; it ends with a semicolon.
- The grammar rules for a programming language are called the syntax of the language.

Programming

- Programming is a creative process.
- Programming can be learned by discovering the techniques used by experienced programmers.
- These techniques are applicable to almost every programming language, including Java.
Object-Oriented Programming

- Our world consists of objects (people, trees, cars, cities, dogs, etc.).
- Objects have state and behavior. A car has state (model, color, fuel level, etc), and behavior (start, change gear, brake, etc).
- An object’s behavior can affect its state and the state of other objects.
- Object-oriented programming (OOP) treats a program as a collection of objects, each with behaviors and state.

OOP Terminology

- Behaviors are included as methods.
- State is contained in a set of attributes.
- A class defines the methods and attributes.
- An object is an instance of a class. A program may have several instances of a class. (myCar, yourCar, herCar)
- Each object (instance of a class) has the same set of methods, but its own attribute values. (state)
Scanner Class

Scanner keyboard = new Scanner(System.in);
keyboard.nextLine();

- **Scanner** is a class
- **keyboard** is an instance of the Scanner class
- **nextLine()** is a method (behavior) of the Scanner class.
- The Scanner has an internal attribute (state) which stores the delimiter.

Scanner Class

- The object performs an action when you *invoke* or *call* one of its methods

  `objectName.methodName(argumentsTheMethodNeeds);`
Algorithms

• By designing methods, programmers provide actions for objects to perform.
• An algorithm describes a means of performing an action.
• Once an algorithm is defined, expressing it in Java (or in another programming language) usually is easy.

Algorithms

• An algorithm is a set of instructions for solving a problem.
• An algorithm must be expressed completely and precisely.
• Algorithms usually are expressed in English or in pseudocode.
The Class **String**

- We've used constants of type *String* already.
  "Enter a whole number from 1 to 99."
- A value of type *String* is a
  - Sequence of characters
  - Treated as a single item.

---

**String Methods**

- An object of the *String* class stores data consisting of a sequence of characters.
- Objects have methods as well as data
- The *length()* method returns the number of characters in a particular *String* object.
- Try this:
  ```java
  String greeting = "Hello";
  int n = greeting.length();
  System.out.println("Length of the string is " + n);
  ```
The Method **length()**

- The method `length()` returns an `int`.
- You can use a call to method `length()` anywhere an `int` can be used.

```java
int count = command.length();
System.out.println("Length is " + command.length());
count = command.length() + 3;
```

**String Indices**

<table>
<thead>
<tr>
<th>Indices</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>J</td>
<td>a</td>
<td>v</td>
<td>a</td>
<td>i</td>
<td>s</td>
<td>f</td>
<td>u</td>
<td>n</td>
<td>.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Positions start with 0, not 1.
- The 'J' in "Java is fun." is in position 0
- A position is referred to an an index.
- The 'f' in "Java is fun." is at index 8.
### String Methods

**charAt( Index )**
Returns the character at Index in this string. Index numbers begin at 0.

**compareTo( A_String )**
Compares this string with A_String to see which string comes first in the lexicographic ordering. (Lexicographic ordering is the same as alphabetical ordering when both strings are either all uppercase letters or all lowercase letters.) Returns a negative integer if this string is first, returns zero if the two strings are equal, and returns a positive integer if A_String is first.

**concat( A_String )**
Returns a new string having the same characters as this string concatenated with the characters in A_String. You can use the ↓ operator instead of concat.

**equals( Other_String )**
Returns true if this string and Other_String are equal. Otherwise, returns false.

**Methods and their return types...**

---

### String Methods

**equalsIgnoreCase( Other_String )**
Behaves like the method equals, but considers uppercase and lowercase versions of a letter to be the same.

**indexOf( A_String )**
Returns the index of the first occurrence of the substring A_String within this string. Returns -1 if A_String is not found. Index numbers begin at 0.

**lastIndexOf( A_String )**
Returns the index of the last occurrence of the substring A_String within this string. Returns -1 if A_String is not found. Index numbers begin at 0.
String Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>length()</code></td>
<td>Returns the length of this string.</td>
</tr>
<tr>
<td><code>toLowerCase()</code></td>
<td>Returns a new string having the same characters as this string, but with any uppercase letters converted to lowercase.</td>
</tr>
<tr>
<td><code>toUpperCase()</code></td>
<td>Returns a new string having the same characters as this string, but with any lowercase letters converted to uppercase.</td>
</tr>
</tbody>
</table>

String Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>replace(OldChar, NewChar)</code></td>
<td>Returns a new string having the same characters as this string, but with each occurrence of <code>OldChar</code> replaced by <code>NewChar</code>.</td>
</tr>
<tr>
<td><code>substring(Start)</code></td>
<td>Returns a new string having the same characters as the substring that begins at index <code>Start</code> of this string through to the end of the string. Index numbers begin at 0.</td>
</tr>
<tr>
<td><code>substring(Start, End)</code></td>
<td>Returns a new string having the same characters as the substring that begins at index <code>Start</code> of this string through, but not including, index <code>End</code> of the string. Index numbers begin at 0.</td>
</tr>
<tr>
<td><code>trim()</code></td>
<td>Returns a new string having the same characters as this string, but with leading and trailing whitespace removed.</td>
</tr>
</tbody>
</table>
Putting Quotes in a String

- What do you do if you want to output
  - How do I put “quotes” in my string?
- This won’t work!
  - `System.out.println("How do I put "quotes" in my string?");`
- You have to let the computer know that you want the quote marks to be in the String
  - `System.out.println("How do I put "quotes" in my string?");`

Backslashes

- Backslash in a String means: the next character is special
  - `System.out.println("How do I put a \ in my string?");`
- It will print:
  
  How do I put a \ in my string?
Escape Characters

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;</td>
<td>Double quote</td>
</tr>
<tr>
<td>’</td>
<td>Single quote</td>
</tr>
<tr>
<td>\</td>
<td>Backslash</td>
</tr>
<tr>
<td>\n</td>
<td>New line</td>
</tr>
<tr>
<td>\t</td>
<td>Tab</td>
</tr>
</tbody>
</table>

• Each escape sequence is a single character even though it is written with two symbols.

Escape Characters

• How would you print

"Java" refers to a language. ?

• The compiler needs to be told that the quotation marks (" ) do not signal the start or end of a string, but instead are to be printed.

    System.out.println(
    "\"Java\" refers to a language.");
Examples – try these

```java
System.out.println("abc\ndef");
```

```
abc\ndef
```

```java
System.out.println("new\nline");
```

```
new
line
```

```java
char singleQuote = '\'';
System.out.println(singleQuote);
```

```
'
```

If and Else

- You can use only one if statement
  - `if (boolean expression)
    { statements; }`
  - `other statements;`
    - Other statements will always be executed
- You can also use an if-else statement
  - `if (boolean expression)
    { statement 1; }
  else { statement 2; }
  - If the expression is true, run statement 1, otherwise run statement 2
Using If and Else

• Use if-else statement
• Do not use two if statements
• Always pay attention to boundaries
  – Is it “>” or “>=”?  
  – Is it “<” or “<=”?  
  – Do you need a “==”?  

Using If-Else

• Pay attention to the brackets {}
  – You can discard them if there is only one statement

```java
if (inputInt > 0) {
    System.out.println("Positive");
} else {
    System.out.println("Negative or zero");
}
```

```java
if (inputInt > 0)
    System.out.println("Positive");
else
    System.out.println("Negative or zero");
```
Using If-Else

• Pay attention to the brackets {}
  – if there is only one statement in {}, discard them
  – If multiple statements, discarding {} will cause problems
  – if (inputInt > 0)
    System.out.println(“Positive”);
    else
      System.out.println(“Negative or zero”);
      System.out.println(“What’s happening?”);
      // will always be executed

Using If-Else

• Pay attention to the brackets {}
  – As a good habit, don’t discard them, even if you have only one statement in it
    • The only exception: multibranch if-else
Using If-Else

• Never put a semicolon after if or else
  – if (inputInt > 0);
    System.out.println("What’s happening now?");
• Compiler will interpret it as
  – if (inputInt > 0)
    {
      ;
    }
    System.out.println("What’s happening now?");

Next class

• More if-else
  → Reading Assignment: Chapter 2