Strange Syntax

- In Java:
  ```java
  int i=2;
i = (i=3) + i;
  ```
  - Is valid! What does it do?
  - Why would somebody do this?

This week

- Some Strings
- More work with references and basic types
- More work with methods
- More about types and casting
- Introduction to writing classes

Boolean Operators

<table>
<thead>
<tr>
<th>Description</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic operators</td>
<td>-</td>
</tr>
<tr>
<td>Assignment</td>
<td>=</td>
</tr>
<tr>
<td>Conditional</td>
<td>?:</td>
</tr>
<tr>
<td>Arithmetic</td>
<td>+, -, *, /, %</td>
</tr>
<tr>
<td>Shift</td>
<td>&lt;&lt;, &gt;&gt;, &gt;&gt;&gt;</td>
</tr>
<tr>
<td>Boolean And</td>
<td>&amp;&amp;</td>
</tr>
<tr>
<td>Boolean Or</td>
<td></td>
</tr>
<tr>
<td>Relational</td>
<td>&lt;, &gt;, &lt;=, &gt;=</td>
</tr>
<tr>
<td>Logical And</td>
<td>&amp;</td>
</tr>
<tr>
<td>Logical Or</td>
<td></td>
</tr>
<tr>
<td>Logical Not</td>
<td>!</td>
</tr>
<tr>
<td>Unary Prefix</td>
<td>++, --</td>
</tr>
<tr>
<td>Unary Postfix</td>
<td>+, -</td>
</tr>
</tbody>
</table>

Examples

- System.out.println((true && false));
- System.out.println((false || false));
- System.out.println((true || ("hi" == "bye")));
The String Class

- Where do we find information on the String class?

Strings and Char’s

- What makes a String different from a string of characters in Java?

How do we compare Strings?

- Strings are objects (reference types)
- They are not necessarily equal with the == operator!
- Thus, the methods compareTo and equals exist to compare strings

String Concatanation

- The ‘+’ operator may be used as a shortcut in concatenating strings
- Examples:

```java
String concatString = "concat" + "String";
```

String Comparison

```java
class TestString {
    public static void main(String[] args) {
        String h = new String("Hello");
        String h2 = h;
        String h3 = new String("Hello");
        String lo = "lo";
        System.out.print(h == h2 + " ");
        System.out.print(h == h2 + " ");
        System.out.print(Other.h == h + " ");
        System.out.print(Other.h == h2 + " ");
        System.out.print(Other.h == h3 + " ");
        System.out.print(h.compareTo(lo) + " ");
        System.out.print(h.compareTo(lo) + " ");
        System.out.print(h.equals(h3) + " ");
        System.out.println(h.equals(Other.h));
    }
}
class Other { static String h = new String("Hello"); }
```

First Program

```java
import javabook.*;
public class MyFirstApplication {
    public static void main(String[] args) {
        MainWindow mainWindow;
        mainWindow = new MainWindow();
        // setVisible makes the window visible
        mainWindow.setVisible(true);
    }
}
```
Views of a Class

- A class can be viewed as a sort of contract that specifies what instances of the class can, and cannot do
- It is possible to distinguish between the outside and inside view of a class
- The interface of a class provides its outside view and emphasizes the abstraction
- The implementation of a class is its inside view

Why not public instance variables?

- See Wu book p. 168-169
- You don’t want to be like Doofie the programmer

Class/Object State

- Class values: class cockatoo can keep track of the total number of cockatoo objects – these variables are declared “static”
- Instance values: Things that differ from object to object, for instance, not all cockatoos are pink, but Snoopy is

Access

- Most classes provide three levels of access to their members (state and behavior):
  - Public: the part of the class of the class that is visible to all clients of the class
  - Protected: the part of the class that is only visible to subclasses of the class
  - Private: a part of the class that is not visible to any other classes

When to keep things private and why

- Declare class and instance variables private
- Declare class and instance methods private if they are used only by other methods in the same class
- Declare class and instance methods public if they are used by outside methods

Class/Object Behavior

- Class methods: Methods which only operate on class values and are “static” methods
- Instance methods: Methods that operate on instance values or/and class values.
Static means One

If you want a method or value to be a class method/value, then it must be declared **static**.

- **static** means that there will be exactly one incarnation of the value/method, no matter how many instances are created from a class containing the static value/method.

Final means Constant

- Sometimes these methods/values are also declared **final**, in which case their value or function is constant.

```java
final double PI = 3.14;
```