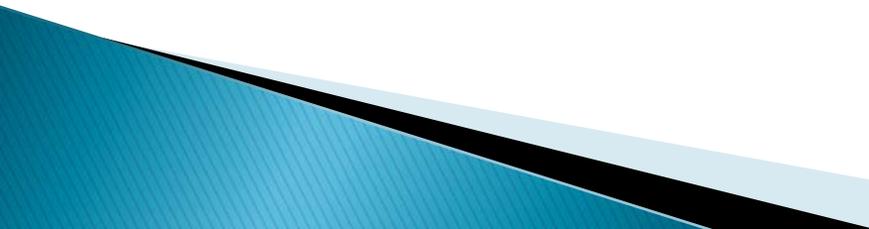


Session 2

- ▶ What you already know about Java
 - From your background in C and Python
- ▶ Reading and using an **API**
 - **Javadoc**
 - The String class
- ▶ Implementing a class
 - Implementing an **interface**
 - Using **documented stubs** before coding
 - Writing **JUnit tests** before coding

Questions on HelloWorldAgain?
On anything?

CSSE 220 in a Nutshell

- ▶ Object-oriented
 - ▶ Concepts: Inheritance, Interfaces, ...
 - ▶ Design: CRC cards, UML class diagrams, ...
 - ▶ Extreme Programming Processes
 - ▶ Documented stubs, Test-first, Pair programming, ...
 - ▶ Data Structures
 - Introduce algorithm efficiency analysis (big O)
 - ▶ Reading and using APIs
 - ▶ Graphical User Interfaces (GUIs) and lots more!
- 

What you already know about Java

(from your background in C and Python)

- ▶ Variables
 - Declaring, assigning
 - Primitive types
 - Printing, reading from the console
- ▶ Objects
 - Dot notation
 - Constructing with *new*
- ▶ Control structures
 - for, while, if, ...
- ▶ Methods
 - Defining
 - Parameters/arguments
- ▶ Classes
 - Fields
 - Methods

Details on each of the above in the next set of slides.

Variables – similar to C

▶ Declaring, assigning

```
int xPosition;  
xPosition = 0;  
int yPosition = 40;
```

Type / Name pattern

```
double r, s;  
r = s;
```

Java compiler flags this mistake (C doesn't!)

▶ Primitive types:

- int byte short long
- double float
- char
- **boolean**
 - true false

Sizes are specified in Java
(C is generally platform-specific).
Details on p. 135 of Big Java

Printing values on the console

Reading values from the console

```
System.out.println(x);
```

String *concatenation*.
Very handy!



```
System.out.println("The value is " + x);
```

```
System.out.printf("The value is %d", x);
```

```
Scanner input = new Scanner(System.in);
```

```
System.out.print("Enter an integer: ");
```

```
int quantity = input.nextInt();
```

Using the Scanner class requires that you *import* it. Eclipse offers a Quick Fix for imports that is almost always right, so I will say no more about imports.

Using Objects and Methods

- ▶ Works just like Python:
 - *object.method(argument, ...)*

Implicit
argument



Explicit
arguments



- ▶ Java Example:

```
String name = "Bob Forapples";  
PrintStream printer = System.out;
```

```
int nameLen = name.length();  
printer.printf("'%s' has %d characters", name, nameLen);
```

Constructing Objects

Note keyword *new*

left, top, width, height

▶ Example:

```
Rectangle box = new Rectangle(5, 10, 20, 30)
```

▶ Several steps are happening here:

1. Java reserves space for a `Rectangle` object
2. `Rectangle`'s *constructor* runs, filling in slots in object
3. Java reserves a variable named `box`
4. `box` is set to refer to the object

What Do Variables Really Store?

- ▶ Variables of number type store *values*
- ▶ Variables of class type store *references*
 - A reference is like a pointer in C, except
 - Java keeps us from screwing up
 - No `&` and `*` to worry about (and the people say, “Amen”)
- ▶ Consider:
 1. `int x = 10;`
 2. `int y = 20;`
 3. `Rectangle box = new Rectangle(x, y, 5, 5);`

Assignment Copies Values

- ▶ Actual value for number types
- ▶ **Reference** value for object types
 - The actual **object is not copied**
 - The **reference value** (“the pointer”) **is copied**
- ▶ Consider:
 1. `int x = 10;`
 2. `int y = x;`
 3. `y = 20;`

 4. `Rectangle box = new Rectangle(5,6,7,8);`
 5. `Rectangle box2 = box;`
 6. `box2.translate(4,4);`

Control structures – similar to C

```
for (int k = 0; k < 100; ++k) {  
    ...  
}  
  
if (x == y) {  
    ...  
} else {  
    ...  
}  
  
while (true) {  
    ...  
    if (...) {  
        break;  
    }  
}
```

```
++k;  
k++;  
k = k + 1;
```

Three ways to do the same thing, in this context.

Java Documentation

- » API Documentation,
Docs in Eclipse,
Writing your own Docs

Recap: Java API Documentation

- ▶ What's an API?
 - Application Programming Interface
- ▶ The Java API on-line and on your computer
 - Google for: **java api documentation 6**
 - Or go to: <http://java.sun.com/javase/6/docs/api/>
 - C:\Program Files\Java\jdk1.6.0_12\docs\api\index.html
- ▶ Find the String class documentation:
 - Click **java.lang** in the top-left pane
 - Then click **String** in the bottom-left pane

Java Documentation in Eclipse

- ▶ Setting up Java API documentation in Eclipse
 - Should be done already, but if the next steps don't work for you, we'll fix that
- ▶ Using the API documentation in Eclipse
 - Hover text
 - Open external documentation (Shift-F2)

Writing Javadocs

- ▶ Written in special comments: `/** ... */`
- ▶ Can come before:
 - Class declarations
 - Field declarations
 - Method declarations
- ▶ Eclipse is your friend!
 - It will generate javadoc comments automatically
 - It will notice when you start typing a javadoc comment

Example Javadoc for a Class

```
/**  
 * This class demonstrates unit testing  
 * and asks you to use the Java API  
 * documentation to find methods to solve  
 * problems using Strings.  
 *  
 * @author Curt Clifton  
 * Created Sep 9, 2008.  
 */
```

Description of
class

@author Tag
followed by author
name and date

```
public class MoreWordGames { ... }
```

Example Javadoc for a Method

```
/**  
 * Converts the original string to a  
 * string representing shouting.  
 *  
 * @param input the original string  
 * @return input in ALL UPPER CASE  
 */  
static String shout(String input) {  
    return input.toUpperCase();  
}
```

Description of method,
usually starts with a verb.

@param tag
followed by
parameter
name and
(optional)
description.
Repeat for each
parameter.

@return tag followed by
description of result. Omit
for void methods.

Javadocs: Key Points

- ▶ Don't try to memorize the Java libraries
 - Nearly 9000 classes and packages!
 - You'll learn them over time
- ▶ Get in the habit of writing the javadocs **before** implementing the methods
 - It will help you **think before doing**, a vital software development skill
 - This is called programming with *documented stubs*
 - I'll try to model this. If I don't, call me on it!

Writing Code to Test Your Code

- »» Test-driven Development,
unit testing and JUnit

Unit Testing

- ▶ Writing code to test other code
- ▶ Focused on testing individual pieces of code (units) in isolation
 - Individual methods
 - Individual objects
- ▶ Why would software engineers do unit testing?
 - Get code right
 - Keep code right as changes are made
 - Confirm our understanding of the method specification before implementing it
 - Provide documentation
 - Confirm pieces in isolation so we don't have to worry about them during integration (when we put code together)

Unit Testing With JUnit

- ▶ JUnit is a unit testing *framework*
 - A framework is a collection of classes to be used in another program
 - Does much of the work for us!
- ▶ JUnit was written by
 - Erich Gamma
 - Kent Beck
- ▶ Open-source software
- ▶ Now used by **millions** of Java developers

Interesting Tests

- ▶ Test “boundary conditions”
 - Intersection points: $-40^{\circ}\text{C} == -40^{\circ}\text{F}$
 - Zero values: $0^{\circ}\text{C} == 32^{\circ}\text{F}$
 - Empty strings
- ▶ Test known values: $100^{\circ}\text{C} == 212^{\circ}\text{F}$
 - But not too many
- ▶ Tests things that might go wrong
 - Unexpected user input: “zero” when 0 is expected
- ▶ Vary things that are “important” to the code
 - String length if method depends on it
 - String case if method manipulates that

Implementing an interface

```
/**  
 * A StringTransformable object can transform one String into another String.  
 *  
 * @author David Mutchler, based on an idea from Lynn Stein  
 *       in her Rethinking CS 101 project.  
 *       Created Mar 12, 2009.  
 */  
public interface StringTransformable {  
    /**  
     * Transform the given String into another String.  
     *  
     * @param stringToTransform The String to transform  
     * @return The transformed String  
     */  
    public String transform(String stringToTransform);  
}
```

