CIS 210

Computer Science I

David Atkins
Winter 2011

Course Info

All important course information will be available on the class web page:

www.cs.uoregon.edu/classes/11W/cis210

Check the announcements page frequently

Computer Science Department home page:

www.cs.uoregon.edu
Text Book

- Required Text Book
  - Building Java Programs
  - Authors: Stuart Reges and Marty Stepp
- Use the book!
  - Read the assigned sections before lecture to get you thinking and asking questions
  - Try out the examples from the book and class – experiment!
- Other resources – see course web pages
  - Another book: Head First Java – also available online through UO library web site

Introduction to Computer Science

- Computer Science is not just programming
  - Computer Science is about problem solving – devising algorithms to solve problems
  - Programming is mostly about the logical analysis and solution of a problem – precisely expressing an algorithm in program code
  - Being a programmer does not make you a Computer Scientist, but most Computer Scientists know how to do some programming
  - All Computer Scientists know how to analyze and solve problems using the same thinking and reasoning processes required to write programs
What we’ll cover this term

- Learn Computer Science concepts
- Learn problem analysis and problem solving
- Learn general programming skills
- Learn basics of software development
- Learn specifics of Java programming
- Learn about Object Oriented design and programming

Pair Programming

- First four programming assignments may be done using the Pair Programming technique
  - Pair programming is done with two people working together at one computer: one driver and one observer
  - Trade roles often
- Pairs chosen at first lab
  - Let GTF know who you are pairing with
  - One result turned in for pair
  - Log must be kept of pair meetings
  - Use "open" lab times in addition to the scheduled lab
  - You can also work together on your own PC
  - Work on your own when you can't schedule time together
  - Lab attendance mandatory (counts toward grade)
A first look at Java

- Java is a **compiled** language
  - Create a textual **source** file, e.g., `Hello.java`
  - Compile source file with the Java compiler
  - Run the compiled program with the JVM
  - May be done from TextPad or a command line, or another Java IDE
- Graphical approach using Java graphics method to create a message dialog:
  - `WindowHello.java`

Basics of Java

- A program is a set of instructions to direct the operation of a computer
- There are many computer programming languages, Java is just one language
- Programming languages are much more rigid than natural (human) languages
  - Everything must be just right (in the correct **syntax**)
  - And everything is taken literally (no interpretation like "oh, you probably meant...")
Anatomy of a Java Program

- Our simple example program Hello.java shows most of the basics

- Comments
- Keywords
- Modifiers
- Statements
- Blocks
- Classes
- Methods
- main method

Components of a Java Program

- **Comments** do not affect the way a program works
  - But comments are essential to document the program and describe how it works
  - Comments should be meaningful and clarifying, but not just state the obvious
- Java has three forms of commenting
  - "One liners": Everything (to end of line) after // is a comment
  - Block comments: Everything between */ and */ is a comment, even on separate lines
  - Javadoc comments: Begin with /** and end with */
    - This is a special form for automatically generating documentation
Components of a Java Program

- **Keywords** are special words in the Java language
  - Have a particular meaning and must be used in specific ways
  - For example: `class public static void import`
  - We'll discuss these and more as we go along

```plaintext
abstract continue float native strictfp void
assert default for new super volatile
boolean do goto null switch while
break double if package synchronized
byte else implements private this
case enum import protected throw
catch extends instanceof public throws
char false int return transient
class final interface short true
const finally long static try
```

- **Modifiers** are keywords that behave like adjectives in English to enhance or change the meaning of various Java constructs
  - `public` and `static` are used as modifiers of a method in our example

Components of a Java Program

- **Statements** are like the basic sentences of Java
  - They describe an action or actions
  - The fundamental building block of a program
  - Statements are terminated with a semicolon

- **Blocks** are groups of statements that belong together (like a paragraph)
  - Blocks are delimited by `{` and `}`

- **Classes** define the most significant grouping in a Java program
  - Classes are the core of Object Oriented Design
  - Classes are like categories and group together the things that belong together
  - A Java program is a collection of classes
Components of a Java Program

- **Methods** are a set of statements that describe an action or behavior at a higher level
  - Methods are executable – they can be "called"
  - Classes contain methods (and data definitions)
  - Methods are often called functions or procedures in other languages

- The **main** method is the starting point for a Java program
  - A Java program must have a main method to be able to be run
  - The main method must be named main, and must have certain modifiers (public and static) and must have a type of void

---

Anatomy of a Java Program

```java
// Java hello, world program
class Hello {
    public static void main(String[] args) {
        System.out.println("Hello, world");
    }
}
```

- Modifier
- Class name
- Class definition and block
- Method definition and block
- Call to another method
- End of main method block
- End of class block
Writing and running Java programs

- Create a Java source file with an editor (e.g., TextPad)
  - File must have the suffix `.java`
  - File name must match the class definition
  - The class must have a definition of `main`

- Compile the program with the Java compiler (`javac.exe`)
  - Compiler will report errors
  - If successful, a byte code file will be created with the suffix `.class`

- Run the program with the Java virtual machine (`java.exe`)
  - Class name must be given (no suffix)

![Compilation and Running Diagram]

A Closer Look at Java

- Primitive numerical data types
- Operations on data values
- Variables
- Assignment statements
- Other primitive data types and Strings
- Data conversions
- Expressions and Operator Precedence
Another Java Example

- Problem: find the cost in cents to drive a mile, given
  - The price in dollars of a gallon of gas
  - The fuel consumption of the car as a ratio (miles per gallon)
- Steps to solve the problem
  - Set the gas price
  - Set the gas mileage rate
  - Compute rate per mile by dividing gas mileage into gallon price
  - Compute the rate in cents by multiplying by 100
  - Display the answer
- Java solution: ComputeCost.java

Tracing the program

```java
public class ComputeCost {
    public static void main(String[] args) {
        double price;  // Allocate space for data values
        int mpg;       // no value
        double cents; // no value
        price = 3.699; // set price value
        mpg = 27;     // set mpg value
        cents = (100 * price) / mpg; // calculate and set
        System.out.println("Cost is "+cents + " cents per mile"); // output
    }
}
```

Memory

<table>
<thead>
<tr>
<th></th>
<th>price</th>
<th>mpg</th>
<th>cents</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3.699</td>
<td>27</td>
<td>13.7</td>
</tr>
</tbody>
</table>
Literals and Symbols

- Literals are constant "hard coded" values in Java programs
  - Strings: enclosed in double quote marks
  - Numbers: digits, possibly with decimal point and/or minus sign (scientific notation also recognized)
  - Single characters: enclosed in single quote marks

- Symbols are usually used for operations
  - E.g., +, *, (, <, =, etc.
  - Symbols may use multiple characters: <=, ++

- Some symbols are used for "punctuation"
  - E.g., ;, {},

Identifiers

- Identifiers are the names we use for components in our Java programs
  - Variable names, e.g., price
  - Class names, e.g., ComputeCost
  - Method names, e.g., main

- Identifiers can use letters, digits, the underscore character (_), or dollar sign ($) but they can not begin with a digit
- Case sensitive – upper and lower case are different

- Keywords, literals, symbols, and identifiers are the vocabulary of a Java program
  - The syntax of the Java language defines the legal ways to combine these entities
First Look at Variables

- **Variables** are used for data items in Java programs
  - A variable name is an identifier
  - Provides symbolic access to a memory location whose value may be changed (i.e., it may vary)
  - Variables must be **declared** before they can be used
  - Each variable must be declared to have a particular **data type**
  - Declaration syntax is: the type name followed by the variable name terminated with semicolon
    - Keyword specifies primitive type, e.g., `int` or `double`
    - For example: `int mpg;`
    - When compiler sees a declaration, it arranges for the use of memory
    - Initial contents of memory are zeroed out
  - Variables may be initialized with a value
    - For example: `int mpg = 27;`

Assignment Statements

- An assignment changes a value stored in memory
  - Uses the assignment operator `=` (a single equals sign)
  - Entity to be changed (e.g., a variable) is to the left of `=`
  - The new value for the variable is to right of `=`
  - The type of the value being assigned must match the declared type of the variable
    - Compiler will complain if types are not compatible
  - Examples
    - `mpg = 27;`
    - `price = 3.69;`
    - `mpg2 = mpg;`
Operations on Numerical Types

- Usual arithmetic operations + − * /
- Division of two integers results in an integer (fractional part is truncated)
  - 14 / 4 is 3
- Division of floating point and another floating point or integer results in a floating point value
  - 14.0 / 4 is 3.5
- Remainder operator (integers only) gives remainder from division
  - 14 % 4 is 2
- Remainder of division by 2 is zero or one
- Easy way to see if a number is even or odd
- Comparison operations
  - Compare for equality == (two equals signs, no space)
  - Compare for inequality !=
  - Less than < , less than or equal to <=
  - Greater than > , greater than or equal to =>
  - All comparisons result in a value of true or false

Numerical Data Types

- Java numerical data types

<table>
<thead>
<tr>
<th>Name</th>
<th>Range</th>
<th>Storage Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte</td>
<td>-128 to 127</td>
<td>8 bits signed</td>
</tr>
<tr>
<td>short</td>
<td>-32768 to 32767</td>
<td>16 bits signed</td>
</tr>
<tr>
<td>int</td>
<td>-2147483648 to 2147483647</td>
<td>32 bits signed</td>
</tr>
<tr>
<td>long</td>
<td>-9223372036854775808 to 9223372036854775807</td>
<td>64 bits signed</td>
</tr>
<tr>
<td>float</td>
<td>± 1.4E-45 to 3.4028235E+38</td>
<td>32 bit IEEE 754</td>
</tr>
<tr>
<td>double</td>
<td>± 4.9E-324 to 1.7976931348623157E+308</td>
<td>64 bit IEEE 754</td>
</tr>
</tbody>
</table>
Data Conversions

- Numerical data types are not necessarily interchangeable
  - For example, a `double` can not be assigned to an `int` since this would lose information
  - However, an `int` can be assigned to a `double` since Java will implicitly convert and no accuracy will be lost
  - Generally, widening conversions from a smaller data type to a larger happen automatically
- Explicit conversions may be performed with a `cast`
  - `double fraction = 7.25;`  
  - `int whole = (int) fraction;`
  - Information is lost (the fractional part is truncated)
  - This is a narrowing conversion
  - Casting may be dangerous since we are telling the compiler to combine incompatible types
- Character conversions
  - A `char` may be cast to an `int` : the integer is the numeric ASCII code for the character
  - An `int` may be cast to a `char` : the integer corresponds to the character with that ASCII code

```
Convert.java
```

Other Primitive Data Types

- **Boolean**
  - Keyword `boolean`
  - Possible values: `true` or `false` (these are literal constants)
  - Example: `boolean isEmpty = true;`
  - Comparison expressions result in a `boolean` value
    ```
    boolean isSame = (x == y);
    ```
- **Character**
  - Keyword `char`
  - Possible values: single characters (7 bit ASCII or 16 bit Unicode)
  - Example: `char initial = 'X';`
  - Example: `char zero = '0';`
  - Escape sequences for special characters for '
  - Escape sequences for special characters for 't' for tab, for single quote, etc.
Strings

- **String** is a data type in Java
  - Have already seen String literals: "Hello, world"
  - May also define String variables
  - `String message = "Hello, world";`

Addition of Strings

- "Sum" of two Strings is a String formed by putting the two Strings together one after the other
- `String h = "Hello", w = "world";
  String message = h + ", " + w;`
- If we "add" numbers to a String, the numerical value is converted to a String,
  e.g., "Cost is " + cents

- **String** is an object type, not a primitive
  - More about objects later

Assignment Revisited

- The right side of an assignment can be an expression involving one or more operators
  - May even use variable being assigned
  - `int i = 7;
    i = 6 * i; // Now i is 42`

- Compound assignment operators are shorthand for an operation and assignment
  - `price *= .75; // Same as price = price * .75`
  - `hours += 12; // Same as hours = hours + 12`
  - `a += b * c; // Same as a = a + (b*c)`

- Compound assignment operators
  - `+=  -=  *=  /=  %=`
Increment and Decrement

- A common operation is to increase (or decrease) a counter by one
  - \( i = i + 1; \)
  - Or, using compound assignment: \( i += 1; \)
- Java has a special increment operator which is equivalent
  - \( ++i; \quad //\text{ Statement has same effect as } i = i + 1; \)
  - Similarly \( --i; \quad \text{is equivalent to } i = i - 1; \)
- Increment operator is an expression whose value can be used
  - \( \text{int } i = 4, \ a = 3 * ++i; \)
  - \( a \) will have the value 15, and \( i \) will have the value 5
- Increment and decrement may also be used in postfix form
  - In this case, the increment or decrement is still done, but the value used in the expression is the original value
  - \( \text{int } i = 4, \ a = 3 * i++; \)
  - \( a \) will have the value 12, and \( i \) will have the value 5
- \( ++ \) and \( -- \) permit compact notation for complex expressions
  - But used too much, they can make code hard to read

Operator Precedence

- How does an expression like \( x = a + b * c; \) evaluate in Java?
  - Multiplication is done "first", then addition
  - Same rules as you learned in middle school
- Operators in Java have precedence levels
  - Precedence determines grouping, i.e., as if you had used parentheses
- Associativity also affects grouping
  - Associativity is left to right for everything but assignment
    - E.g., addition and subtraction have the same precedence level, but they are left associative, so \( a + b - c \) is the same as \( (a + b) - c \)
    - But \( a += b -= c \) is the same as \( a += (b -= c) \)
### Operator Precedence Table

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>++, --</td>
<td>var++, var-- (Postfix)</td>
</tr>
<tr>
<td>+, -, ++, --</td>
<td>Unary plus, minus, ++var, --var (Prefix)</td>
</tr>
<tr>
<td>(type)</td>
<td>Casting</td>
</tr>
<tr>
<td>!</td>
<td>Not</td>
</tr>
<tr>
<td>*, /, %</td>
<td>Multiplication, division, remainder</td>
</tr>
<tr>
<td>+, -</td>
<td>Binary addition and subtraction</td>
</tr>
<tr>
<td>&lt;, &lt;=, &gt;, &gt;=</td>
<td>Comparison</td>
</tr>
<tr>
<td>==, !=</td>
<td>Equality, Inequality</td>
</tr>
<tr>
<td>&amp;</td>
<td>Unconditional (bitwise) AND</td>
</tr>
<tr>
<td>^</td>
<td>Exclusive OR</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>&amp;&amp;</td>
<td>Conditional AND</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>=, +=, -=, *=, /=, %=</td>
<td>Assignment operators</td>
</tr>
</tbody>
</table>

### Evaluation Order

- **In what order do we do the operations in `a*b + c*d`?**
  - Precedence groups the two products, so `a*b` and `c*d` must be computed.
  - Certainly these products must be computed before we can compute the sum.
  - But which is done first? `a*b` or `c*d`?
  - Who cares – we’ll get the same answer either way.
- **But operators can have side effects (like `++`, `--`, assignment)**
  - Is order important in `a++ + (b=a)`?
  - Yes, suppose we start with 2 for `a`, and 3 for `b`. If we do the `a++` first then `a` becomes 3, and `b` becomes 3. If we do `b=a` first, then `b` becomes 2, and `a` becomes 3.
- **Java guarantees left to right evaluation order**
  - But expressions like the previous example are still not a good idea since they are hard to read and understand.
  - Other similar languages (like C or C++) may not guarantee the order of evaluation.
## Evaluation Order

What is the order of evaluation in the following expressions?

\[
\begin{align*}
    &a + b + c + d + e \\
    &a / (b + c) - d \% e \\
    &a += b = c = d = e \\
    &c = a++ + (b = a)
\end{align*}
\]

\[
\begin{align*}
    &a + b * c - d / e \\
    &a / (b * (c + (d - e)))
\end{align*}
\]