CIS 210

Introduction to Computer Science I

David Atkins
Winter 2010

Course Info

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

www.cs.uoregon.edu/classes/10W/cis210

Check the announcements page frequently

Computer Science Department home page:

www.cs.uoregon.edu
Text Book

- Required Text Book
  - Java Programming: From the Ground Up
  - Authors: Ralph Bravaco and Shai Simonson
- 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
  - Programming a computer is a tool we use to solve problems
  - Programming is mostly about the logical analysis and solution of a problem
  - Being a programmer does not make you a Computer Scientist, but most Computer Scientists know how to do some programming
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 three 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

- **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

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

- **class definition and block**
- **class name**
- **method definition and block**
- **method type**
- **call to another method**
- **modifier**
- **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)

A Closer Look at Java

- Primitive numerical data types
- Operations on data values
- Variables
- Assignment statements
- Other primitive data types and Strings
- Data conversions
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;
        int mpg;
        double cents;
        price = 3.699;
        mpg = 27;
        cents = (100 * price) / mpg;
        System.out.println("Cost is "+ cents + " cents per mile");
    }
}
```

<table>
<thead>
<tr>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>price</td>
</tr>
<tr>
<td>mpg</td>
</tr>
<tr>
<td>cents</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
  - Class names, e.g., `ComputeCost`
  - Method names, e.g., `main`
  - Variable names, e.g., `price`
- 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)
  - \(14 \div 4\) is 3
- Division of floating point and another floating point or integer results in a floating point value
  - \(14.0 \div 4\) 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 `\n` newline, `\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