Today’s Topics

• Introduction
• Homework assignment
• Review the syllabus
• Introductory comments on programming languages
• An example of a simple Java program
• Fill out a questionnaire
• Log in and get started with Eclipse (an integrated development environment for Java)
• More Java topics
Who is your instructor?

• I'm Mike Eckmann, an Associate Professor in the Mathematics and Computer Science Dept., Skidmore College. I have taught at Skidmore for 11 years. Before coming to Skidmore I was at Lehigh University in PA.

• I studied Mathematics and Computer Engineering and Computer Science all at Lehigh University.

• I was employed as a programmer (systems analyst) for eight years.
1st Homework

• I'll send everyone an email later today. Please reply to the email so I know you got it. In your reply, let me know if you want any additional email addresses for yourself added to the class list and what they are.
Syllabus

• Office hours
  – Stop by / email
• Text book
• Class rooms
• Assignments
  – Programs & HW

• Collaboration policy
• Grading scheme
• Attendance
• Workload
• Readings

Note: The most up-to-date syllabus will be found on the course web page.

Michael Eckmann - Skidmore
College - CS 106 - Summer 2015
This semester we will ...

• Be introduced to computer science.
• Learn programming (in Java)!
• Solve problems and learn to think like programmers.
• Hopefully have a fun learning experience.
Computer Science is ...

• more than computer programming.
Programming Languages

• Machine
• Assembly
• High-level
  – in no particular order
  – Pascal, C, C++, Basic, Fortran, Java, Python, Ruby, many more ...
• 4th generation (e.g. visual)
• Procedural vs. Object-oriented
Syntax vs. semantics

• Time flies like an arrow
  – 3 meanings (semantics) for 1 syntax
  – That's why English is not a programming language!

• Errors
  – Run time (crashes)
  – Compile time (syntax errors)
  – Logic (semantic errors)

• Have any of you encountered any of these with software that you use? Which kinds of errors? Examples?
Hello, World! program

/* CS106 first program
   written by Michael Eckmann
   purpose: print the text “Hello, World!”
   to the screen */
public class HelloWorld
{
    public static void main(String args[])
    {
        System.out.println("Hello, World!");
    }  // end of main method

}  // end of HelloWorld class
Discussion of “Hello, World!”

- comments in source code
- basic structure of a Java program
- every Java program must have at least one class
- every Java program must have exactly one `main` method
- `System.out.println`
- the use of braces -- begin and end classes and method bodies. Braces are `{}` – I call them curly braces.
- keywords
- public, static, void, class, etc...
Some common programming mistakes (bugs)

- Case matters --- upper case letters in class names etc. are different than lower case letters --- e.g. String and string are understood differently by the compiler. Additionally the capitalized `String` is necessary in the main method argument list declaration.

- **System** must be capitalized when referring to something like: `System.out` because *System* is the name of the class/object, system with a lower case *s* is not.

- Spaces are not allowed in a class name or any identifier / variable name e.g. `HelloWorld` is legal, `Hello World` is not.
Some common programming mistakes (bugs)  (continued)

• Double quotes must be paired up when giving the value of a String  --- can’t forget the second double quote.

• Curly braces must be paired correctly. An extra one dangling somewhere in the code will cause errors, same if one is forgotten.

• Class name must be the same as the main part of the file name.  e.g. a public class named HelloWorld must reside in a file named HelloWorld.java --- not in something like HW.java or any other file.

• Java source code can only be in plain text files. If a Java source code file is stored as a Word document or any other file type besides plain text, the code will not be able to be compiled.
Some common programming mistakes (bugs)  (continued)

• Multi-line comments starting with /* must end with */ and cannot contain a */ in between. The reason is, once the compiler sees /*, it considers everything a comment until the first */ that it sees, then starts interpreting as code once again.

• Note: Encountering bugs is a good thing because it helps the learning process --- e.g. when you encounter similar bugs in the future, you’re likely to find the solution more quickly.
Comments

• Comments are ignored by the compiler
• Single line comments start with // and the comment continues until a newline
• Multi-line comments start with /* and end with */
• Another type of multi-line comment starts with /*** and ends with */
• What’s the purpose of comments if the compiler ignores them?
println vs. print

• println adds a newline character (carriage return) after the String is printed.
• print does not.
println vs. print
System.out.println(“I love java.”);
System.out.println(“It tastes good.”);

If these two statements were inside the main method's curly braces, then the program would print:

I love java.
It tastes good.
println vs. print

System.out.print(“I love java.”);
System.out.println(“It tastes good.”);

Will print:

I love java. It tastes good.
println vs. print

System.out.print(“I love java.”);
System.out.print(“It tastes good.”);

How will this print?
Phases of Java Programs

• create / compile the program
  – Create / Edit the source code of the program
  – Compile / Build the source code into bytecodes
  – After the .java file is compiled we get a .class file
    • E.g. if we compile a program called HelloWorld.java the result is HelloWorld.class

• When we execute the program (via the java interpreter), the interpreter does the following:
  – Loads the Classes
  – Verifies the Bytecodes
  – Interprets the Bytecodes

• All of these “execution” steps are done on the .class file(s)
Phases of Java Programs

• terminology you need to understand
  – source code
  – compile, build
  – bytecode
  – execute, run
Phases of Java Programs

• terminology you need to understand
  – source code (= the .java files)
  – compile, build (the source code into bytecode)
  – bytecode (= the .class files)
  – execute, run (the .class file(s) using the Java bytecode interpreter)
Eclipse

Eclipse is the integrated development environment that we will use for this course.

You are free to use others but I will give instructions for Eclipse and show examples during labs, etc. in Eclipse.

Let's try it now.

- Log in, change password
- Start Eclipse, follow instructions.
Escape sequences

• A String's *value* is specified between double quotes.
• Within the double quotes the backslash character \ is special.
• The backslash character followed by another character is called an escape sequence.
• Escape sequences are handled differently than regular characters.
• \n is a common escape sequence to embed a newline
• \t is a way to add a tab to a string
• E.g. `System.out.println("Hello\nWorld");` prints:
  Hello
  World
Escape sequences

• \b is the backspace character
• There are only a total of eight valid escape sequences.
• Let’s say you wanted to print a sarcastic message using double quotes.
• If we do
  System.out.println("Hamilton has a "great" lacrosse team");
• there will be a syntax error at compile time.
Escape sequences

System.out.println("Hamilton has a "great" lacrosse team");
• The string will be
  – Hamilton has a
• and then the compiler will see a g and will generate an error because the second double quote (the one before great) causes the string to end.
• What do we do?
Escape sequences

• There is an escape sequence to represent a ". It is \\
  System.out.println("Hamilton has a "great" lacrosse team");

• Now, the string literal contains the double quotes
  around great. The first double quote and last double
  quote mark the beginning and ending of the String
  literal.

• The above line of code will do what we want now.
Escape sequences

• The \ character is special inside a String. Java will interpret the \ and the next character as an escape sequence (even if it is not one of the valid ones.)

• How might we add a backslash character to a String?

• Let’s say we want a String like:
  – C:\My Documents

• Will this work?
  – System.out.println("C:\My Documents");
Let’s try this problem

• Write an application that displays the numbers 1 to 4 on the same line, with each pair of adjacent numbers separated by one space. Write the program using the following methods:
  – A) Using one System.out statement
  – B) Using four System.out statements

• Now let's do the same thing but on 4 different lines.
  – A) Using one System.out statement
  – B) Using four System.out statements
String concatenation

• To concatenate two Strings together, Java provides the operator +

  e.g.
  System.out.println("Hey " + "now.");

  prints
  Hey now.
String concatenation

- Technically, Strings can only be concatenated to other Strings, but numbers are converted to Strings by Java *automatically* if they are being concatenated to a String.

e.g.
System.out.println("I am " + 21 + " years old.");
Will work the same as:
System.out.println("I am " + "21" + " years old.");
Will work the same as:
System.out.println("I am 21 years old.");
Variables and Type

• A program often needs to be able to store and change data as it is running. For example, a program might need to store a user's name, or the user's age, or the air temperature, or whatever.

• Variables allow names to be given to data that can be stored and changed.
Variables and Type

• A variable is a location in memory (RAM) with the following properties.

• name --- given by the programmer --- try to be descriptive of the data the variable will hold

• type --- decided by the programmer from all available types, depending on what kind of values the programmer wants to store

• size --- amount of memory used, this is determined by the type

• value --- self-explanatory
  – value can be stored / changed / read by a program.
Primitive Types for Variables

• **byte, short, int, long** --- these are integer types --- they hold *whole* numbers like –31, 4256, 56, 2632, 755, -901 (they hold different ranges)

• **char** --- this is a type that holds *one* character like a letter, digit, *, &, or an escape sequence …

• **boolean** --- holds one of *true* or *false*

• **float, double** --- numeric types that hold numbers with decimal values like -5.5, 98.6, 1002.99995 etc. (they hold different ranges and to different precisions)

• We can look up the ranges/precision of the numeric types.
Variables and Type

• To be able to specify to our program that we want to use a variable, before we can use it we must declare the variable. That is we state the desired type as well as a descriptive name that we make up (must start with a letter and may contain letters, digits, and _).

• Example: If we wanted a place to store a person's age (years only e.g. 18, 21, 55 etc.), we might create a variable like this:

• `int age; // declares the variable age of type int`

• `age` is a name we made up, what does int do?
Variables and Type

• `int age;` // declares the variable age of type int

• age is a name we made up, what does int do?

• We chose int because we knew we wanted to only store whole numbers and the int type stores whole numbers. We could have chosen byte, short or long which also hold whole numbers but we typically use int for whole numbers unless we have a good reason to use one of those other three.
Variables and Type

int area = 50;  // This line of code not only
               // declares a variable but also initializes its value.

• name of variable is area
• type of variable is int
• size of variable is 32 bits (because an int has a size of 32
  bits.)
• value of variable is currently 50
• note the same thing can be done with 2 lines of code:
  int area;
  area = 50;  // this is an assignment statement (it assigns
               // the value 50 to the variable area
Program Flow

• Java programs start executing in the main method.
• They execute sequentially, in order (unless something changes the program flow --- more on that later today and throughout the semester) until the end of the main method when the program ends.
public class VariableExample
{
    public static void main(String args[])
    {
        int result;  // declares variable with name result and type int
                    // allocates enough memory for an int to be stored
        int some_number;  // declares variable with name some_number and type
                           // int and allocates enough memory for an int to be stored
        result = 35;  // an assignment statement that stores the value 35 in the
                       // variable result.
        some_number = 24;  // an assignment statement that stores the value 24
                            // in the variable some_number.
        result = some_number + 5;
                        // uses the + operator to add the value in some_number to 5
                        // and stores (assigns) the sum to the variable result, thereby
                        // changing result’s value.
    } // end main method body
} // end class VariableExample body
<table>
<thead>
<tr>
<th>Memory after line 1</th>
<th>result</th>
<th>some_number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory after line 2</td>
<td>result</td>
<td>some_number</td>
</tr>
<tr>
<td>Memory after line 3</td>
<td>result</td>
<td>some_number</td>
</tr>
<tr>
<td>Memory after line 4</td>
<td>result</td>
<td>some_number</td>
</tr>
<tr>
<td>Memory after line 5</td>
<td>result</td>
<td>some_number</td>
</tr>
</tbody>
</table>

Michael Eckmann - Skidmore College - CS 106 - Summer 2015
Types for Variables

• Variables can be of the primitive types (int, char, double, etc.) as well as predefined classes provided by Java or a class that a programmer creates.
• For example a variable can be of type String which is a class provided by Java.
• We’ll see more of variables and types shortly.
An aside about names of characters

• {} are called braces or curly braces
• ( ) are called parentheses
• [ ] are called brackets or square brackets
• Please memorize the names of these as we all need to use the same names for the same characters.
Equality and Relational operators

==  is equal to operator
!=  is not equal to operator
<   is less than operator
>   is greater than operator
<=  is less than or equal to operator
>=  is greater than or equal to operator
if / else structure

- So far all the code we’ve seen was executed sequentially --- every line of code in order from the first to the last within the main method.

```java
if (condition)
{
    statement_to_do_if_condition_is_true;
    another_statement_to_do_if_condition_is_true;
}
else
{
    statement_to_do_if_condition_is_false;
    another_statement_to_do_if_condition_is_false;
    yet_another_statement_to_do_if_condition_is_false;
}
```
if / else structure

• condition is something that will evaluate to true or false (usually using relational or equality operators.)
• if you only want to do one statement (in the if or else portion) curly braces are not required
• if you want to do more than one statement the curly braces are necessary
• also, the else portion of the if / else structure is optional
valid if / else example

if (radius > 0)
{
    area = 3.14159 * radius * radius;
    circumference = 2 * 3.14159 * radius;
}
else
{
    System.out.println("Radius is invalid");
}
a few more valid if / else examples

if (numerical_grade >= 90)
    System.out.println("your grade is an A!");

/*-----------------------------------------------*/

if (account_balance < 200)
    System.out.println("Acct below $200 minimum");
else
    System.out.println("Do not charge a service fee");