A Simple Java Program
Compiling
Variables

Lecture 3

Borrowing from slides by Alan Hu, Kurt Eiselt, Paul Carter, and Tamara Munzner
News

- Labs and tutorials start today
  - labs are mandatory (your solutions will be marked on the spot)
  - tutorials are optional but highly recommended
- You should already have finished Lab 0
  - Can be found on WebCT
  - Get your CS account before lab!
Reading Assignments

- For this week, read
  - Edition 2 and 3: Ch 2.1-2.5, Ch 4.1-4.2
Recap: What is a Computer?

How is a computer different from a video game console? Or a DVD player? Or a telephone? Or a bank machine?

The computer is **general**. It can be all of the other devices.

Making the computer do what we want is called **programming** the computer.
Recap: High-Level Language

- A High-Level Language is a computer language designed to be easier for humans:
  - a=b+c;
- Must be translated into machine language so the computer can understand it.
  - You could translate it as you go (**interpreter**).
  - You could translate it in advance (**compiler**).
- Java is the high-level language we’ll use.
  - Modern, widely used, portable, safe.
Recap: Interpreters and Compilers

- An interpreter translates the high-level language into machine language on-the-fly, executing the instructions as it goes.
- A compiler translates the high-level language program all at once in advance.
- Both compilers and interpreters are themselves computer programs.
- Which is better? (Remember George and Stephen in France?)
Recap: Java Does Both!

Your Program.java (Java)

Your Program.class (Java Bytecodes)

javac Compiler

java JVM on MacOS

java JVM on Windows

Windows PC

Macintosh

JVM on Unix

SPARC Server
Today’s Objectives

- Understand the first simple Java program and programming constructs
- Understand variables and their data types
- Be able to declare variables
- A first glimpse on assignment
A Simple Java Program

// Our first Java program.
/* Traditionally, one’s first program in a new language prints out “Hello, World!” */
*/

class HelloTester {
   public static void main(String[] args) {
      System.out.println("Hello, World!");
   }
}
Recap: Sample Java Application Program

//****************************************************
// Oreo.java        Author:  Kurt Eiselt //
// Demonstrating simple Java programming concepts while
// revealing one of Kurt's many weaknesses
//****************************************************

public class Oreo
{
    //****************************************************
    // demand Oreos
    //****************************************************
    public static void main (String[] args)
    {
        System.out.println ("Feed me more Oreos!");
    }
}
Sample Java Application Program

- Comments ignored by Java compiler

```java
//******************************************************
// Oreo.java        Author:  Kurt Eiselt //  // Demonstrating simple Java programming concepts while
// revealing one of Kurt's many weaknesses
//******************************************************

public class Oreo
{
    //****************************************************
    // demand Oreos   //****************************************************
    public static void main (String[] args)
    {
        System.out.println ("Feed me more Oreos!");
    }
}
```
Sample Java Application Program

- Comments could also look like this

```java
/*
 * Oreos.java        Author:  Kurt Eiselt

 Demonstrating simple Java programming concepts while revealing one of Kurt's many weaknesses
 */

public class Oreo
{
    /* demand Oreos */
    public static void main (String[] args)
    {
        System.out.println ("Feed me more Oreos!");
    }
}
```
Sample Java Application Program

- Comments and whitespace are important to people
  - But not to the compiler
- Compiler only cares about

```java
public class Oreo {
    public static void main (String[] args) {
        System.out.println("Feed me more Oreos!");
    }
}
```
public class Oreo {
    public static void main (String[] args) {
        System.out.println ("Feed me more Oreos!");
    }
}
White Space

//*******************************************************
// Oreo1.java       Author:  Kurt Eiselt
//
// Demonstrating mediocre use of white space
//*******************************************************

public class Oreo1 {
    public static void main (String[] args) {
        System.out.println("Feed me more Oreos!");
    }
}
White Space

//***************************************************************
// Oreo2.java       Author:  Kurt Eiselt
//
// Demonstrating bad use of white space
//***************************************************************

public class Oreo2 { public static void main (String[] args) { System.out.println ("Feed me more Oreos!"); } }
public class Oreo3
{
    public static void main (String[] args)
    {
        System.out.println  ("Feed me more Oreos!");
    }
}
public class Oreo4
{
    public static void main(String[] args)
    {
        System.out.println("Feed me more Oreos!")
    }
}
White Space

- White space
  - Blanks between identifiers and other symbols
  - Tabs and newline characters are included

- White space does not affect how programs run

- Use white space to format programs we create so they’re easier for people to understand.
  (Appendix A is a good start.)
Write Readable Programs!

- Use comments, formatting, and other ways to make your program easy to understand.
  - TAs will be grateful.
  - Other programmers will be grateful.
  - You will be grateful.
- A program is both an expression of an idea, and a way to make a computer do a task.
  (Interesting legal/philosophical issues…)
Sample Java Application Program

```java
public class Oreo {
    public static void main (String[] args) {
        System.out.println("Feed me more Oreos!");
    }
}
```

- Whole thing is the definition of a **class**
- Package of instructions that specify
  - what kinds of data will be operated on
  - what kinds of operations there will be
- Java programs will have one or more classes
  - For now, just worry about one class at a time
Sample Java Application Program

```java
public class Oreo {
    public static void main (String[] args) {
        System.out.println("Feed me more Oreos!");
    }
}
```

- Instructions inside class definition grouped into one or more procedures called **methods**
  - group of Java statements (instructions) that has name, performs some task
- All Java programs you create will have **main** method where program execution begins
Sample Java Application Program

```java
public class Oreo {
    public static void main (String[] args) {
        System.out.println ("Feed me more Oreos!");
    }
}
```

- These class and method definitions are incomplete at best
  - good enough for now
  - expand on these definitions as course continues
public class ManyOreos {
    public static void main (String[] args) {
        System.out.println("Feed me more Oreos!");
        // I'm being sloppy. You'll learn better ways to do this later.
        javax.swing.JOptionPane.showMessageDialog(null,"Give me a cookie!");
        javax.swing.JOptionPane.showMessageDialog(null,"I want cookie!");
        javax.swing.JOptionPane.showMessageDialog(null,"Cookie!!!!");
    }
}
Sample Java Application Program

```java
public class Oreo {
    public static void main (String[] args) {
        System.out.println ("Feed me more Oreos!");
    }
}
```

- Words we use when writing programs are called **identifiers**
- except those inside the quotes
public class Oreo
{
    public static void main (String[] args)
    {
        System.out.println ("Feed me more Oreos!");
    }
}

- Kurt made up identifier Oreo
Sample Java Application Program

```java
public class Oreo {
    public static void main (String[] args) {
        System.out.println("Feed me more Oreos!");
    }
}
```

- Other programmers chose identifiers `System`, `out`, and `println`
  - They wrote printing program.
  - Part of huge library of useful programs that comes with Java
Sample Java Application Program

```java
public class Oreo {
    public static void main (String[] args) {
        System.out.println("Feed me more Oreos!");
    }
}
```

- Special identifiers in Java called **reserved words**
  - don’t use them in other ways
Reserved Words

- Get familiar with these (Appendix G)
- But you don’t need to memorize all 52 for exam

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

- Identifier must
  - Start with a letter and be followed by
  - Zero or more letters and/or digits
    - Digits are 0 through 9.
  - Letters are
    - the 26 lowercase letters in English alphabet
    - the 26 uppercase letters in English alphabet
    - plus the $ and _
    - also alphabetic characters from other languages
Identifiers

- Identifier must
  - Start with a letter and be followed by
  - Zero or more letters and/or digits
    - Digits are 0 through 9.
    - Letters are the 26 characters in English alphabet
      - both uppercase and lowercase
      - plus the $ and _
      - also alphabetic characters from other languages

- Which of the following are not valid identifiers?
  - userName
  - user_name
  - $cash
  - 2ndName
  - firstName
  - user.age
  - _note_
  - note2
Identifiers

- Identifier must
  - Start with a letter and be followed by
  - Zero or more letters and/or digits
    - Digits are 0 through 9.
    - Letters are the 26 characters in English alphabet
      - both uppercase and lowercase
      - plus the $ and _
      - also alphabetic characters from other languages
  - Which of the following are not valid identifiers?
    - userName  user_name  $cash  2ndName
    - first name  user.age  _note_  note2
Identifiers

- Java is case sensitive
- Oreo  oreo  OREO  0reo
  - are all different identifiers, so be careful
  - common source of errors in programming

- are these all valid identifiers?
Identifiers

- Creating identifiers in your Java programs
  - Remember other people read what you create
  - Make identifiers meaningful and descriptive for both you and them
- No limit to how many characters you can put in your identifiers
  - but don’t get carried away

```java
public class ReallyLongNamesWillDriveYouCrazyIfYouGoOverboard {
    public static void main (String[] args) {
        System.out.println ("Enough already!");
    }
}
```
Memory and Identifiers

- Example of a high-level instruction
  - $A = B + C$

- Tells computer to
  - go to main memory and find value stored in location called B
  - go to main memory and find value stored in location called C
  - add those two values together
  - store result in memory in location called A

- Great! But... in reality, locations in memory are not actually called things like a, b, and c.
Memory Recap

- Memory: series of locations, each having a unique address, used to store programs and data
- When data is stored in a memory location, previously stored data is overwritten and destroyed
- Each memory location stores one byte (8 bits) of data

*For total accuracy, these addresses should be binary numbers, but you get the idea, no?
Memory and Identifiers

- So what’s with the a, b, and c?
  - Machine language uses actual addresses for memory locations
  - High-level languages easier
    - Avoid having to remember actual addresses
    - Invent meaningful identifiers giving names to memory locations where important information is stored
  - pay_rate and hours_worked vs. 5802 and 5806
    - Easier to remember and a whole lot less confusing!
Memory and Identifiers: Variables

- **Variable**: name for location in memory where data is stored
  - A little like variables in algebra class

- `pay_rate, hours_worked, a, b,` and `c` are all variables

- Variable names begin with lower case letters
  - Java convention, not compiler/syntax requirement

- Variable may be name of single byte in memory or may refer to a group of contiguous bytes
  - More about this later…
Let's give it a try...
Programming With Variables

//******************************************************
// Test.java       Author: Kurt
//
// Our first use of variables!
//******************************************************

public class Test
{
    public static void main (String[] args)
    {
        a = b + c;
        System.out.println ("The answer is " + a);
    }
}

- Let’s give it a try...
  - b and c cannot be found!
  - need to assign values
public class Test2
{
    public static void main (String[] args)
    {
        b = 3;
        c = 5;
        a = b + c;
        System.out.println ("The answer is "+ a);
    }
}
Programming With Variables: Take 2

// *************************************************
// Test2.java       Author: Kurt
//
// Our second use of variables!
// *************************************************

public class Test2
{
    public static void main (String[] args)
    {
        b = 3;
        c = 5;
        a = b + c;
        System.out.println ("The answer is " + a);
    }
}

- Now what?
  - such a lazy computer, still can’t find symbols...
Now What?

- Java doesn’t know how to interpret the contents of the memory location
  - are they integers? characters from the keyboard? shades of gray? or....
Data Types

- Java requires that we tell it what kind of data it is working with.

- For every variable, we have to declare a data type.

- Java language provides eight primitive data types.
  - i.e. simple, fundamental

- For more complicated things, can use data types.
  - created by others provided to us through the Java libraries
  - that we invent
    - More soon - for now, let’s stay with the primitives

- We want a, b, and c to be integers. Here’s how we do it...
public class Test3
{
    public static void main (String[] args)
    {
        int a; //these
        int b; //are
        int c; //variable declarations
        b = 3;
        c = 5;
        a = b + c;
        System.out.println ("The answer is " + a);
    }
}

//***************************************** // Test3.java       Author: Kurt // // Our third use of variables! //*****************************************

Programming With Variables: Take 3
Variable Declaration

- variable declaration is instruction to compiler
  - reserve block of main memory large enough to store data type specified in declaration
- variable name is specified by identifier
- syntax:
  - `typeName variableName;`
Variable Declaration and Assignment

- variable declaration is instruction to compiler
  - reserve block of main memory large enough to store data type specified in declaration
- variable name is specified by identifier
- syntax:
  - `typeName variableName;`
  - `typeName variableName = value;`
    - can declare and assign in one step
Assignment

/***************************************************************************/
// Test3.java       Author: Kurt
//
// Our third use of variables!
/***************************************************************************/

public class Test3
{
    public static void main (String[] args)
    {
        int a;
        int b;
        int c;
        b = 3;    // these
        c = 5;    // are
        a = b + c; // assignment statements
        System.out.println ("The answer is " + a);
    }
}

Assignment Statements

- Assignment statement assigns value to variable
  - sometimes say binds value to variable

- Assignment statement is
  - identifier
  - followed by assignment operator (=)
  - followed by expression
  - followed by semicolon (;)

\[
\begin{align*}
  b &= 3; \\
  c &= 8; \\
  a &= b + c; \\
  \text{weekly_pay} &= \text{pay_rate} \times \text{hours_worked};
\end{align*}
\]

- Note that = is not a test for equality!
Assignment Statements

- Java first computes value on right side
- Then assigns value to variable given on left side

\[ x = 4 + 7; \quad // \text{what's in } x? \]

- Old value will be overwritten if variable was assigned before

\[ x = 2 + 1; \quad // \text{what's in } x \text{ now?} \]