Variables & Assignment
Java Primitive Data Types

Lecture 4

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

- For this week, read
  - Edition 2 and 3: Ch 2.1-2.5, Ch 4.1-4.2
Recap: Comments & White Space

- White space
  - Blanks between identifiers and other symbols
  - Tabs and newline characters are included
- Comments
  - Single line: from // to end of line
  - Multi-line: everything between /* and */
- White space and comments do not affect how programs run
- Use white space and comments to write programs so they are easier for people to understand.
  (Appendix A is a good start.)
Recap: Sample Java Program

```java
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
Recap: Sample Java 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
Recap: Sample Java 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
Recap: Program With More Than One Statement

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!!!!");
    }
}
Recap: Identifiers

- Words we use when writing programs are called **identifiers**
- except those inside the quotes
Recap: Identifiers

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

- Kurt made up identifier Oreo
Recap: Identifiers

- Kurt made up the identifier **Oreo**
- Other programmers chose identifiers **System**, **out**, and **println**
- Special identifiers in Java called **reserved words**
  - don’t use them in other ways

```java
public class Oreo {
    public static void main (String[] args) {
        System.out.println ("Feed me more Oreos!");
    }
}
```
Recap: 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
cchar       final       long        strictfp    volatile
class       finally     native      super       while
cconst      float       new         switch
continue    for         null        synchronized
default     goto        package     this
Recap: 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
Today’s Objectives

- Understand variables and their data types
- Be able to declare variables and use assignment statements
- Learn the Java primitive types
- Learn the most common Java operators for the primitive types
- Combine operators into simple expressions
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

Data values are stored in memory locations – more than one location may be used if the data is large.
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…
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...
Programming With Variables

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

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...
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);
    }
}
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

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;     // what’s in x?
```

- Old value will be overwritten if variable was assigned before

```
x = 2 + 1;     // what’s in x now?
```
Note that = is not a test for equality!

- Assignment is a new concept in CS versus math.
  - Fetch-decode-execute
  - Dynamic. Something happens.
    \[ x = x + 1; \]

Read = as “gets” (short for “gets assigned the value”).
Assignment Statements

Here’s an occasional point of confusion:

```
a = 7;     // what’s in a?
b = a;     // what’s in b?
// what’s in a now???
```
Assignment Statements

Here’s an occasional point of confusion:

```java
a = 7;       // what’s in a?
b = a;       // what’s in b?
             // what’s in a now???
System.out.println("a is " + a + " b is " + b);
```

Find out! Experiments are easy to do in CS
Assignment Statements

Here’s an occasional point of confusion:

```java
a = 7; // what’s in a?
b = a; // what’s in b?
// what’s in a now???
System.out.println("a is " + a + " b is " + b);
```

- Variable values on left of = are clobbered
- Variable values on right of = are unchanged
  - copy of value assigned to a also assigned to b
  - but that doesn’t change value assigned to a
Assignment Statements

Here’s an occasional point of confusion:

```java
a = 7;          // what’s in a?
b = a;          // what’s in b?
               // what’s in a now???
System.out.println("a is " + a + " b is " + b);
a = 8;
System.out.println("a is " + a + " b is " + b);
```

Memory locations `a` and `b` are distinct
- value of `a` at that point assigned to `b`
- changing `a` later does not affect previous copy
Primitive Data Types: Non-numeric

- **Character Type**
  - named `char`
  - Java uses the Unicode character set so each char occupies 2 bytes of memory.

- **Boolean Type**
  - named `boolean`
  - Variables of type boolean have only two valid values
    - true and false
  - Often represents whether particular condition is true

- Ignore these for now. We’ll see them again later.
Data Types

- 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 - today, let's stay with the primitives
Primitive Data Types: Non-numeric

- Character Type
  - named char
  - Java uses the Unicode character set so each char occupies 2 bytes of memory.

- Boolean Type
  - named boolean
  - Variables of type boolean have only two valid values
    - true and false
  - Often represents whether particular condition is true
  - Ignore these for now. We’ll see them again later.
### Primitive Data Types: Numbers

<table>
<thead>
<tr>
<th>Type</th>
<th>Size</th>
<th>Min</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte</td>
<td>1 byte</td>
<td>-128</td>
<td>127</td>
</tr>
<tr>
<td>short</td>
<td>2 bytes</td>
<td>-32,768</td>
<td>32,767</td>
</tr>
<tr>
<td>int</td>
<td>4 bytes</td>
<td>-2,147,483,648</td>
<td>2,147,483,647</td>
</tr>
<tr>
<td>float</td>
<td>4 bytes</td>
<td>approx -3.4E38 (7 sig.digits)</td>
<td>approx 3.4E38 (7 sig.digits)</td>
</tr>
<tr>
<td>double</td>
<td>8 bytes</td>
<td>approx -1.7E308 (15 sig. digits)</td>
<td>approx 1.7E308 (15 sig. digits)</td>
</tr>
</tbody>
</table>

- Six primitives for numbers
  - fixed size, so finite capacity
  - integer vs. floating point
# Data Type Sizes

<table>
<thead>
<tr>
<th>Type</th>
<th>Size</th>
<th>Min</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>4 bytes</td>
<td>-2,147,483,648</td>
<td>2,147,483,647</td>
</tr>
<tr>
<td>double</td>
<td>8 bytes</td>
<td>approx -1.7E308</td>
<td>approx 1.7E308</td>
</tr>
</tbody>
</table>

- fixed size, so finite capacity

---

![Diagram of memory address and integer representation]

-One integer – 4 bytes.
Floating Point Numbers

- significant digits
  - 42
  - 4.2
  - 42000000
  - .000042
Floating Point Numbers

- significant digits
  - 42  = 4.2 x 10  = 4.2 x 10^1
  - 4.2  = 4.2 x 1  = 4.2 x 10^0
  - 42000000 = 4.2 x 10000000  = 4.2 x 10^7
  - .000042  = 4.2 x .00001  = 4.2 x 10^{-5}
Floating Point Numbers

- significant digits
  - $42 = 4.2 \times 10 = 4.2 \times 10^1$
  - $4.2 = 4.2 \times 1 = 4.2 \times 10^0$
  - $42000000 = 4.2 \times 10000000 = 4.2 \times 10^7$
  - $0.000042 = 4.2 \times 0.00001 = 4.2 \times 10^{-5}$

- only need to remember
  - nonzero digits
  - where to put the decimal point
    - floats around when multiply/divide by 10
    - (all in binary inside computer)
  - enormous range, but limited precision
## Primitive Data Types: Numbers

<table>
<thead>
<tr>
<th>Type</th>
<th>Size</th>
<th>Min</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte</td>
<td>1 byte</td>
<td>-128</td>
<td>127</td>
</tr>
<tr>
<td>short</td>
<td>2 bytes</td>
<td>-32,768</td>
<td>32,767</td>
</tr>
<tr>
<td>int</td>
<td>4 bytes</td>
<td>-2,147,483,648</td>
<td>2,147,483,647</td>
</tr>
<tr>
<td>float</td>
<td>4 bytes</td>
<td>approx -3.4E38 (7 sig.digits)</td>
<td>approx 3.4E38 (7 sig.digits)</td>
</tr>
<tr>
<td>double</td>
<td>8 bytes</td>
<td>approx -1.7E308 (15 sig. digits)</td>
<td>approx 1.7E308 (15 sig. digits)</td>
</tr>
</tbody>
</table>

- Primary primitives are **int** and **double**
- Just worry about those for now
- Don’t need to memorize exact limits, but know roughly what the limits are.
Data Types: Int and Double

- int
  - integer
  - 4 bytes, about -2 billion to 2 billion
- double
  - real number
  - (double-precision floating point)
  - 8 bytes, 15 sig figs, humongous range
- (Number systems briefly explained in Appendix L)
Variable Declaration Examples

- person’s age in years
- height of mountain to nearest meter
- length of bacterium in centimeters
- number of pets at home