Logistics

Reading
Eck Ch 1
- Available online: http://math.hws.edu/javanotes/
- Reading ahead is encouraged

Goals
- Basic Model of Computation
- First Java Programs
- Course Mechanics
Most computers have 3 basic, physical components\textsuperscript{1}

1. A CPU which can execute instructions
2. MEMORY where data is stored
3. Some sort of Input/Output device like a SCREEN

The CPU is given a set of instructions, a PROGRAM, that change MEMORY and the SCREEN when executed

Example of a Running Computer Program

\begin{verbatim}
CPU: at instruction 10:
> 10: set box #1024 to 800
  11: set box #1028 to 303
  12: sum #1024,#1028 into #1032
  13: print #1024, "plus", #1028
  14: print "is", #1032

<table>
<thead>
<tr>
<th>Box</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>---</td>
<td>------</td>
</tr>
<tr>
<td>#1024</td>
<td>19</td>
</tr>
<tr>
<td>#1028</td>
<td>12</td>
</tr>
<tr>
<td>#1032</td>
<td>-137</td>
</tr>
</tbody>
</table>
\end{verbatim}

\textsuperscript{1}Of course it’s a \textit{little} more complex than this but the addage, "All models are wrong but some are useful." applies here.
Sample Run Part 1

CPU: at instruction 10:  
  > 10: set box #1024 to 800  
  11: set box #1028 to 303  
  12: sum #1024,#1028 into #1032  
  13: print #1024, "plus", #1028  
  14: print "is", #1032  

MEMORY:  
<table>
<thead>
<tr>
<th>Box</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1024</td>
<td>19</td>
</tr>
<tr>
<td>#1028</td>
<td>12</td>
</tr>
<tr>
<td>#1032</td>
<td>-137</td>
</tr>
</tbody>
</table>

SCREEN:  

CPU: at instruction 11:  
  10: set box #1024 to 800  
  > 11: set box #1028 to 303  
  12: sum #1024,#1028 into #1032  
  13: print #1024, "plus", #1028  
  14: print "is", #1032  

MEMORY:  
<table>
<thead>
<tr>
<th>Box</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1024</td>
<td>800</td>
</tr>
<tr>
<td>#1028</td>
<td>12</td>
</tr>
<tr>
<td>#1032</td>
<td>-137</td>
</tr>
</tbody>
</table>

SCREEN:  

CPU: at instruction 12:  
  10: set box #1024 to 800  
  11: set box #1028 to 303  
  > 12: sum #1024,#1028 into #1032  
  13: print #1024, "plus", #1028  
  14: print "is", #1032  

MEMORY:  
<table>
<thead>
<tr>
<th>Box</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1024</td>
<td>800</td>
</tr>
<tr>
<td>#1028</td>
<td>303</td>
</tr>
<tr>
<td>#1032</td>
<td>-137</td>
</tr>
</tbody>
</table>
Sample Run Part 2

<table>
<thead>
<tr>
<th>CPU: at instruction 13:</th>
<th>MEMORY:</th>
<th>SCREEN:</th>
</tr>
</thead>
<tbody>
<tr>
<td>10: set box #1024 to 800</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11: set box #1028 to 303</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12: sum #1024,#1028 into #1032</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&gt; 13: print #1024, &quot;plus&quot;, #1028</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14: print &quot;is&quot;, #1032</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CPU: at instruction 14:</th>
<th>MEMORY:</th>
<th>SCREEN:</th>
</tr>
</thead>
<tbody>
<tr>
<td>10: set box #1024 to 800</td>
<td></td>
<td>800 plus 303</td>
</tr>
<tr>
<td>11: set box #1028 to 303</td>
<td></td>
<td>------</td>
</tr>
<tr>
<td>12: sum #1024,#1028 into #1032</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13: print #1024, &quot;plus&quot;, #1028</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&gt; 14: print &quot;is&quot;, #1032</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CPU: at instruction 15:</th>
<th>MEMORY:</th>
<th>SCREEN:</th>
</tr>
</thead>
<tbody>
<tr>
<td>10: set box #1024 to 800</td>
<td></td>
<td>800 plus 303</td>
</tr>
<tr>
<td>11: set box #1028 to 303</td>
<td></td>
<td>------</td>
</tr>
<tr>
<td>12: sum #1024,#1028 into #1032</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13: print #1024, &quot;plus&quot;, #1028</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14: print &quot;is&quot;, #1032</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&gt; 15: ....</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

> CPU: at instruction 16: |

<table>
<thead>
<tr>
<th>MEMORY:</th>
<th>SCREEN:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

> CPU: at instruction 17: |

<table>
<thead>
<tr>
<th>MEMORY:</th>
<th>SCREEN:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

> CPU: at instruction 18: |

<table>
<thead>
<tr>
<th>MEMORY:</th>
<th>SCREEN:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

> CPU: at instruction 19: |

<table>
<thead>
<tr>
<th>MEMORY:</th>
<th>SCREEN:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

> CPU: at instruction 20: |

<table>
<thead>
<tr>
<th>MEMORY:</th>
<th>SCREEN:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

> CPU: at instruction 21: |

<table>
<thead>
<tr>
<th>MEMORY:</th>
<th>SCREEN:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

> CPU: at instruction 22: |

<table>
<thead>
<tr>
<th>MEMORY:</th>
<th>SCREEN:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

> CPU: at instruction 23: |

<table>
<thead>
<tr>
<th>MEMORY:</th>
<th>SCREEN:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

> CPU: at instruction 24: |

<table>
<thead>
<tr>
<th>MEMORY:</th>
<th>SCREEN:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

> CPU: at instruction 25: |

<table>
<thead>
<tr>
<th>MEMORY:</th>
<th>SCREEN:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

> CPU: at instruction 26: |

<table>
<thead>
<tr>
<th>MEMORY:</th>
<th>SCREEN:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

> CPU: at instruction 27: |

<table>
<thead>
<tr>
<th>MEMORY:</th>
<th>SCREEN:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Observations: CPU and Program Instructions

- Program instructions are usually small, simple operations:
  - Put something in a box
  - Copy the contents of one box to another
  - Do arithmetic (add, subtract, multiply, divide) with numbers in boxes and specified constants like 5
  - Print stuff to the screen
- The CPU keeps track of which instruction to execute next
- In many cases after executing it moves ahead by one instruction but we’ll allow jumping around soon
- This program is in pseudocode, not Java
- Pseudocode can have almost anything in it so long as a human reader understands the meaning
- Java has a lot more rules and restrictions to it so that a real computer can actually understand it
Observations: Screen and Memory

Screen versus Memory

- Nothing is on the screen until it is explicitly printed by the program
- Normally you don’t get to see memory while the program runs
- Good programmers can quickly form a mental picture of what memory looks like and draw it when needed
- You will draw memory diagrams in this class

Boxes are Memory Addresses

- The box numbers (#1024 etc.) are somewhat arbitrary
- Box numbers represent memory addresses
- Random Access Memory (RAM): the value in any box can be retrieved FAST
- My laptop has 16GB of memory = 134,217,728 integer boxes (!)
- Box #’s never change
- Box Values/Contents frequently change
Exercise: Swapping Values Badly

The following code attempts to swap the values stored in boxes \#1024 and \#1028. Show what it actually does.

CPU: at instruction 50:
> 50: copy box \#1024 to \#1028
51: copy box \#1028 to \#1024
52: print "first",\#1024
53: print "second",\#1028

MEMORY:  
<table>
<thead>
<tr>
<th>Box</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>------</td>
<td>-------</td>
</tr>
<tr>
<td>#1024</td>
<td>19</td>
</tr>
<tr>
<td>#1028</td>
<td>31</td>
</tr>
</tbody>
</table>

SCREEN:
| #1032| -1    |
Answer/Exercise: Swapping Values Badly

CPU: at instruction 51:

<table>
<thead>
<tr>
<th>Box</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>50: copy box #1024 to #1028</td>
<td></td>
</tr>
<tr>
<td>&gt; 51: copy box #1028 to #1024</td>
<td></td>
</tr>
<tr>
<td>52: print &quot;first&quot;,#1024</td>
<td>#1024</td>
</tr>
<tr>
<td>53: print &quot;second&quot;,#1028</td>
<td>#1028</td>
</tr>
<tr>
<td>54: ...</td>
<td>#1032</td>
</tr>
</tbody>
</table>

MEMORY:

SCREEN:

CPU: at instruction 52:

<table>
<thead>
<tr>
<th>Box</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>50: copy box #1024 to #1028</td>
<td></td>
</tr>
<tr>
<td>51: copy box #1028 to #1024</td>
<td></td>
</tr>
<tr>
<td>&gt; 52: print &quot;first&quot;,#1024</td>
<td>#1024</td>
</tr>
<tr>
<td>53: print &quot;second&quot;,#1028</td>
<td>#1028</td>
</tr>
<tr>
<td>54: ...</td>
<td>#1032</td>
</tr>
</tbody>
</table>

MEMORY:

SCREEN:

CPU: at instruction 54:

<table>
<thead>
<tr>
<th>Box</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>50: copy box #1024 to #1028</td>
<td></td>
</tr>
<tr>
<td>51: copy box #1028 to #1024</td>
<td></td>
</tr>
<tr>
<td>52: print &quot;first&quot;,#1024</td>
<td>#1024</td>
</tr>
<tr>
<td>53: print &quot;second&quot;,#1028</td>
<td>#1028</td>
</tr>
<tr>
<td>&gt; 54: ...</td>
<td>#1032</td>
</tr>
</tbody>
</table>

MEMORY:

SCREEN:

Fix this: Adjust the program so that it swaps correctly.

Hint: You might need to use a third box.
Answer: Swapping Values Better

CPU: at instruction 51:

50: copy box #1024 to #1032
> 51: copy box #1028 to #1024
52: copy box #1032 to #1028
53: print "first", #1024
54: print "second", #1028
55: ...

MEMORY:  
<table>
<thead>
<tr>
<th>Box</th>
<th>Value</th>
</tr>
</thead>
</table>

SCREEN:

CPU: at instruction 52:

50: copy box #1024 to #1032
51: copy box #1028 to #1024
> 52: copy box #1032 to #1028
53: print "first", #1024
54: print "second", #1028
55: ...

MEMORY:  
<table>
<thead>
<tr>
<th>Box</th>
<th>Value</th>
</tr>
</thead>
</table>

SCREEN:

CPU: at instruction 52:

50: copy box #1024 to #1032
51: copy box #1028 to #1024
52: copy box #1032 to #1028
> 53: print "first", #1024
54: print "second", #1028
55: ...

MEMORY:  
<table>
<thead>
<tr>
<th>Box</th>
<th>Value</th>
</tr>
</thead>
</table>

SCREEN:

Victory: First program done
Variables: Named Boxes

- Dealing with box numbers is tedious
- Any programming language worth its salt will have variables: names associated with a box
- You pick variable names; automatically gets translated to an appropriate box#

**SWAP PROGRAM BOX# ONLY**

<table>
<thead>
<tr>
<th>CPU: at instruction 50:</th>
<th>MEMORY:</th>
</tr>
</thead>
<tbody>
<tr>
<td>50: copy box #1024 to #1032</td>
<td>Box</td>
</tr>
<tr>
<td>&gt; 51: copy box #1028 to #1024</td>
<td>-------+-------</td>
</tr>
<tr>
<td>52: copy box #1032 to #1028</td>
<td>#1024</td>
</tr>
<tr>
<td>53: print &quot;first&quot;,#1024</td>
<td>#1028</td>
</tr>
<tr>
<td>54: print &quot;second&quot;,#1028</td>
<td>#1032</td>
</tr>
</tbody>
</table>

**SWAP PROGRAM WITH NAMED BOXES**

<table>
<thead>
<tr>
<th>CPU: at instruction 50:</th>
<th>MEMORY:</th>
</tr>
</thead>
<tbody>
<tr>
<td>50: copy x to temp</td>
<td>Box</td>
</tr>
<tr>
<td>&gt; 51: copy y to x</td>
<td>-------+----------</td>
</tr>
<tr>
<td>52: copy temp to y</td>
<td>#1024</td>
</tr>
<tr>
<td>53: print &quot;first&quot;,x</td>
<td>#1028</td>
</tr>
<tr>
<td>54: print &quot;second&quot;,y</td>
<td>#1032</td>
</tr>
</tbody>
</table>
Correspondence of Java Programs to Memory

- Java programs require box names to be declared with the type of thing they will hold.
- The equal sign (=) means "store the result on the right in the box named on the left"
- Creating a box and giving it a value can be combined

```java
int a;               // give me a box named a that will hold an integer
a = 800;            // put 800 in box a
int b = 303;         // give me a box named b and put 303 in it right away
int c = a + b;       // third box named c, fill with sum of a and b
```

Notice each of these lines ends with a semicolon (;)

Other Rules

- Java looks ahead and figures out how many boxes will be needed based on variable declarations like `int a;` and `int c=20;`
- All boxes are filled with zeroey things initially which is the number 0 for integers
- Lines that only declares a variable do nothing except indicate a box is needed
Sample Run of First Java Program (1)

CPU: at instruction 10:
10: int a;
11: a = 800;
12: int b = 303;
13: int c = a + b;

MEMORY:
<table>
<thead>
<tr>
<th>Box</th>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>a</td>
<td>800</td>
</tr>
<tr>
<td></td>
<td>b</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>c</td>
<td>0</td>
</tr>
</tbody>
</table>

SCREEN:

CPU: at instruction 11:
10: int a;
11: a = 800;
12: int b = 303;
13: int c = a + b;

MEMORY:
<table>
<thead>
<tr>
<th>Box</th>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>a</td>
<td>800</td>
</tr>
<tr>
<td></td>
<td>b</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>c</td>
<td>0</td>
</tr>
</tbody>
</table>

SCREEN:

CPU: at instruction 12:
10: int a;
11: a = 800;
12: int b = 303;
13: int c = a + b;

MEMORY:
<table>
<thead>
<tr>
<th>Box</th>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>a</td>
<td>800</td>
</tr>
<tr>
<td></td>
<td>b</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>c</td>
<td>0</td>
</tr>
</tbody>
</table>
Sample Run of First Java Program (2)

CPU: at instruction 12:  MEMORY:  SCREEN:
  10: int a;          | Box | Name | Value |
  11: a = 800;    |-------+------|-------|
> 12: int b = 303; | #1024 | a   | 800   |
  13: int c = a + b; | #1028 | b   | 0     |
  | #1032 | c   | 0     |

CPU: at instruction 13:  MEMORY:  SCREEN:
  10: int a;          | Box | Name | Value |
  11: a = 800;    |-------+------|-------|
  12: int b = 303; | #1024 | a   | 800   |
> 13: int c = a + b; | #1028 | b   | 0     |
  | #1032 | c   | 0     |

CPU: at instruction 14:  MEMORY:  SCREEN:
  10: int a;          | Box | Name | Value |
  11: a = 800;    |-------+------|-------|
  12: int b = 303; | #1024 | a   | 800   |
  13: int c = a + b; | #1028 | b   | 303   |
> 14: ...            | #1032 | c   | 1103  |
Exercise: Quick Review

Recall this information from last time:

1. What are three **physical** components to a computer (in our podunk model)?
2. Do Box numbers like #1024 ever change? What does change about boxes?
3. What do programming languages usually call "boxes" with names?
4. What is Java:
   - A tasty, caffeinated beverage?
   - An island part of the country Indonesia
   - A high-ish level programming language for computers
5. How does one ask for a named box in Java?
Output In Java

Java output to the screen is a bit tedious. Typical way is to use `System.out.println()` method which is a mouthful.

Examples of `System.out.println()`

- `System.out.println("Hello world");`  
  Prints Hello World to the screen

- `System.out.println(a);`  
  Prints the contents of variable `a`

- `System.out.println(a + " plus " + b)`  
  With `a=800; b=303`; prints 800 plus 303

Output in a Java Program

```java
CPU: at instruction 15:
10: int a;
11: a = 800;
12: int b = 303;
13: int c = a + b;
14: System.out.println(a + " plus " + b);
> 15: System.out.println("is " + c);

CPU: at instruction 16:
10: int a;
11: a = 800;
12: int b = 303;
13: int c = a + b;
14: System.out.println(a + " plus " + b);
15: System.out.println("is " + c);
> 16: ...
```
Exercise: Swap in Java

Original Code

SWAP PROGRAM WITH NAMED BOXES

CPU: at instruction 50:
> 50: copy x to temp
  51: copy y to x
  52: copy temp to y
  53: print "first",x
  54: print "second",y

MEMORY:
<table>
<thead>
<tr>
<th>Box</th>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>------</td>
<td>------</td>
<td>-------</td>
</tr>
<tr>
<td>------</td>
<td>------</td>
<td>-------</td>
</tr>
<tr>
<td>------</td>
<td>------</td>
<td>-------</td>
</tr>
<tr>
<td>------</td>
<td>------</td>
<td>-------</td>
</tr>
<tr>
<td>------</td>
<td>------</td>
<td>-------</td>
</tr>
<tr>
<td>------</td>
<td>------</td>
<td>-------</td>
</tr>
</tbody>
</table>

SCREEN:

Translate this to Java

- Use variable names given above: x, y, temp
- Declare the boxes with type int as they hold integers
- Give them the initial values shown: 19, 31, -1
- Assign using the = operator
- Print using System.out.println()
int x = 19;
int y = 31;
int temp = -1;
temp = x;
x = y;
y = temp;
System.out.println("first " + x);
System.out.println("second " + y);

Now to get this to run...
Compile/Run a Basic Java Program in DrJava

The full program requires some incantations to make it runnable. Copy and paste the following into DrJava

```java
public class Swap{
    public static void main(String args[]){
        int x = 19;
        int y = 31;
        int temp = -1;
        temp = x;
        x = y;
        y = temp;
        System.out.println("first " + x);
        System.out.println("second " + y);
    }
}
```

▶ Save the file as Swap.java
▶ Should be able to press the Compile button and then Run it.
Files and Extensions

- Java files usually have the `.java` extension
- Extensions like `.txt`, `.docx`, `.pdf` hint at what type of stuff is in a file so the Operating System knows can select an appropriate program to open it
- `.java` files are NOT executable
- **Compiling** them translates them to a low level representation that the CPU actually understands
- `.class` files result from compiling a Java file
  - Compile `Swap.java` produces `Swap.class`
  - Compile `MyCrazyClass.java` produces `MyCrazyClass.class`
- Operating systems sometimes hide extensions because they are stupid; show them whose boss and tell them to "show extensions"
  - Show File Extensions in Windows 10
  - Show File Extensions in Mac OS X
DrJava Running Swap

```java
// First program shows how to variables can be swapped using a third
// variable then printed to the screen
public class Swap{
    public static void main(String args[]){
        int x = 19;
        int y = 31;
        int temp = -1;
        temp = x;
        x = y;
        y = temp;
        System.out.println("first " + x);
        System.out.println("second " + y);
    }
}
```

Welcome to DrJava. Working directory is
/home/kauffman/Dropbox/teaching/1103-F2017/lectures/01-introduction-code
> run Swap
first 31
second 19
> |
Compile/Run Java Program on the Command Line

- The alternative to an Integrated Development Environment (IDE) like DrJava is to use the command line.
  - Windows: cmd.exe command prompt
  - Mac OS X: Terminal.app command shell
- Command line has more of a learning curve but is powerful
- Must have the Java Development Kit (JDK) installed (for DrJava too)
- May also need to instruct your OS’s command shell where the JDK is installed (Let me google that for you)
- Minimum instructions for command line compile/run are

  ```
  $ cd 01-introduction-code/  # change to folder with java program
  $ javac Swap.java  # compile Swap.java to produce Swap.class
  $ java Swap  # run the main() method of the Swap
  first 31
  second 19
  ```

Most of the time you’ll be fine using DrJava or another IDE in CSCI 1103 but you should know a little command line magic.
Exercise: Birthday Problems

The program below should print out a current age and the age next year but is missing some parts.

```java
public class Birthday{
    public static void main(String args[]){
        System.out.println("I hear you are " + ???);
        System.out.println("Next year you will be "+ ???);
    }
}
```

Solve this by introducing variable(s)

- Do it using one 2 variables
- Do it using 1 variable
- **Constraint:** A variable will need to be initialized to the current age, but ANY age should work
Answer: Birthday Problems

// Using 2 variables
public class Birthday{
    public static void main(String args[]){
        int age = 20;
        System.out.println("I hear you are "+age);
        int next_age = age + 1;
        System.out.println("Next year you will be "+next_age);
    }
}

// Using 1 variable
public class Birthday{
    public static void main(String args[]){
        int age = 20;
        System.out.println("I hear you are "+age);
        age = age + 1;
        System.out.println("Next year you will be "+age);

        // Something extra...
        age = age - 1;
        if(age >= 21){
            System.out.println("Let’s get rickety wrecked! ");
        } else{
            int countDown = 21 - age;
            System.out.println(countDown + " more years...");
        }
    }
}
Comments: Further Human Consumption

- Reading programs is HARD
- Made easier with addition information: comments
- Ignored by the compiler - write in English
- Two styles in Java
  - // comments to the end of line
  - /* starts a comment, ends at */
- DrJava knows how to bulk comment/uncomment regions to turn code on/off

```
// This is a one line comment, goes to end of line.
int x = 1;       // comment about this variable

/* This is a multiline comment which will keep going until the ending symbol is reached which appears at the end of this line. */

// The below code won’t execute as it is commented out
// System.out.println("Hi");
// x = 7;
```
Notes on Naming Things

- Most names in programming
  - start with a letter
  - can have a mixture of letters, numbers, and underscore (_) in the name
- Spaces in names create problems everywhere
  - int my integer = 5; // compile reject
  - public class First Program { // rejected
- Convention in Java is to use camelCase to indicate word boundaries
  - int myInteger = 5; // accept
  - public class FirstProgram { // accept
- Convention in Java is that variables start with lower case, classes start with upper case
  - int MyInteger = 5; // bad style
  - public class first_program { // bad style