CSCI 1103: Introduction

Chris Kauffman

Last Updated:
Wed Sep 13 10:43:47 CDT 2017

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

Podunk Model: CPU, Memory, Screen, Program

Most computers have 3 basic, physical components[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

```
CPU: at instruction 10:             MEMORY:             SCREEN:
> 10: set box #1024 to 800          | Box   | Value |
  11: set box #1028 to 303          |-------+-------|
  12: sum #1024,#1028 into #1032    | #1024 |    19 |
  13: print #1024, "plus", #1028    | #1028 |    12 |
  14: print "is", #1032             | #1032 |  -137 |
```

[1] Of course it's a *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:             MEMORY:             SCREEN:
> 10: set box #1024 to 800          | Box   | Value |
  11: set box #1028 to 303          |-------+-------|
  12: sum #1024,#1028 into #1032    | #1024 |    19 |
  13: print #1024, "plus", #1028    | #1028 |    12 |
  14: print "is", #1032             | #1032 |  -137 |

CPU: at instruction 11:             MEMORY:             SCREEN:
  10: set box #1024 to 800          | Box   | Value |
> 11: set box #1028 to 303          |-------+-------|
  12: sum #1024,#1028 into #1032    | #1024 |   800 |
  13: print #1024, "plus", #1028    | #1028 |    12 |
  14: print "is", #1032             | #1032 |  -137 |

CPU: at instruction 12:             MEMORY:             SCREEN:
  10: set box #1024 to 800          | Box   | Value |
  11: set box #1028 to 303          |-------+-------|
> 12: sum #1024,#1028 into #1032    | #1024 |   800 |
  13: print #1024, "plus", #1028    | #1028 |   303 |
  14: print "is", #1032             | #1032 |  -137 |
```

Sample Run Part 2

```
CPU: at instruction 13:             MEMORY:             SCREEN:
  10: set box #1024 to 800          | Box   | Value |
  11: set box #1028 to 303          |-------+-------|
  12: sum #1024,#1028 into #1032    | #1024 |   800 |
> 13: print #1024, "plus", #1028    | #1028 |   303 |
  14: print "is", #1032             | #1032 |  1103 |
```

```
CPU: at instruction 14:             MEMORY:             SCREEN:
  10: set box #1024 to 800          | Box   | Value |   800 plus 303
  11: set box #1028 to 303          |-------+-------|
  12: sum #1024,#1028 into #1032    | #1024 |   800 |
  13: print #1024, "plus", #1028    | #1028 |   303 |
> 14: print "is", #1032             | #1032 |  1103 |
```

```
CPU: at instruction 15:             MEMORY:             SCREEN:
  10: set box #1024 to 800          | Box   | Value |   800 plus 303
  11: set box #1028 to 303          |-------+-------|   is 1103
  12: sum #1024,#1028 into #1032    | #1024 |   800 |
  13: print #1024, "plus", #1028    | #1028 |   303 |
  14: print "is", #1032             | #1032 |  1103 |
> 15: ....
```

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 `print`-ed 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:             MEMORY:             SCREEN:
> 50: copy box #1024 to #1028       | Box   | Value |
  51: copy box #1028 to #1024       |-------+-------|
  52: print "first",#1024           | #1024 |    19 |
  53: print "second",#1028          | #1028 |    31 |
                                    | #1032 |    -1 |
```

Answer/Exercise: Swapping Values Badly

```
CPU: at instruction 51:             MEMORY:             SCREEN:
  50: copy box #1024 to #1028       | Box   | Value |
> 51: copy box #1028 to #1024       |-------+-------|
  52: print "first",#1024           | #1024 |    19 |
  53: print "second",#1028          | #1028 |    19 |
  54: ...                           | #1032 |    -1 |

CPU: at instruction 52:             MEMORY:             SCREEN:
  50: copy box #1024 to #1028       | Box   | Value |
  51: copy box #1028 to #1024       |-------+-------|
> 52: print "first",#1024           | #1024 |    19 |
  53: print "second",#1028          | #1028 |    19 |
  54: ...                           | #1032 |    -1 |

CPU: at instruction 54:             MEMORY:             SCREEN:
  50: copy box #1024 to #1028       | Box   | Value |   first 19
  51: copy box #1028 to #1024       |-------+-------|   second 19
  52: print "first",#1024           | #1024 |    19 |
  53: print "second",#1028          | #1028 |    19 |
> 54: ...                           | #1032 |    -1 |
```

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:             MEMORY:             SCREEN:
  50: copy box #1024 to #1032       | Box   | Value |
> 51: copy box #1028 to #1024       |-------+-------|
  52: copy box #1032 to #1028       | #1024 |    19 |
  53: print "first",#1024           | #1028 |    31 |
  54: print "second",#1028          | #1032 |    19 |
  55: ...
```

```
CPU: at instruction 52:             MEMORY:             SCREEN:
  50: copy box #1024 to #1032       | Box   | Value |
  51: copy box #1028 to #1024       |-------+-------|
> 52: copy box #1032 to #1028       | #1024 |    31 |
  53: print "first",#1024           | #1028 |    31 |
  54: print "second",#1028          | #1032 |    19 |
  55: ...
```

```
CPU: at instruction 52:             MEMORY:             SCREEN:
  50: copy box #1024 to #1032       | Box   | Value |
  51: copy box #1028 to #1024       |-------+-------|
  52: copy box #1032 to #1028       | #1024 |    19 |
> 53: print "first",#1024           | #1028 |    31 |
  54: print "second",#1028          | #1032 |    19 |
  55: ...
```

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
CPU: at instruction 51:           MEMORY:
  50: copy box #1024 to #1032     | Box   | Value |
> 51: copy box #1028 to #1024     |-------+-------|
  52: copy box #1032 to #1028     | #1024 |    19 |
  53: print "first",#1024         | #1028 |    31 |
  54: print "second",#1028        | #1032 |    19 |

SWAP PROGRAM WITH NAMED BOXES     MEMORY:
CPU: at instruction 51:           | Box   | Name | Value |
  50: copy x to temp              |-------+------+-------|
> 51: copy y to x                 | #1024 | x    |    19 |
  52: copy temp to y              | #1028 | y    |    31 |
  53: print "first",x             | #1032 | temp |    -1 |
  54: print "second",y
```

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

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

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

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 |
```

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    |   303 |
                          | #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

```
CPU: at instruction 15:                         MEMORY:               SCREEN:
  10: int a;                                    | Name | Value |      800 plus 303
  11: a = 800;                                  +------+-------|
  12: int b = 303;                              | a    |   800 |
  13: int c = a + b;                            | b    |   303 |
  14: System.out.println(a + " plus " + b);     | c    |  1103 |
> 15: System.out.println("is " + c);
```

```
CPU: at instruction 16:                         MEMORY:               SCREEN:
  10: int a;                                    | Name | Value |      800 plus 303
  11: a = 800;                                  +------+-------|      is 1103
  12: int b = 303;                              | a    |   800 |
  13: int c = a + b;                            | b    |   303 |
  14: System.out.println(a + " plus " + b);     | c    |  1103 |
  15: System.out.println("is " + c);
> 16: ...
```

Exercise: Swap in Java

Original Code

```
SWAP PROGRAM WITH NAMED BOXES     MEMORY:                    SCREEN:
CPU: at instruction 50:           | Box   | Name | Value |
> 50: copy x to temp              |-------+------+-------|
  51: copy y to x                 | #1024 | x    |    19 |
  52: copy temp to y              | #1028 | y    |    31 |
  53: print "first",x             | #1032 | temp |    -1 |
  54: print "second",y
```

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()`

Answer: Swap in Java

```java
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

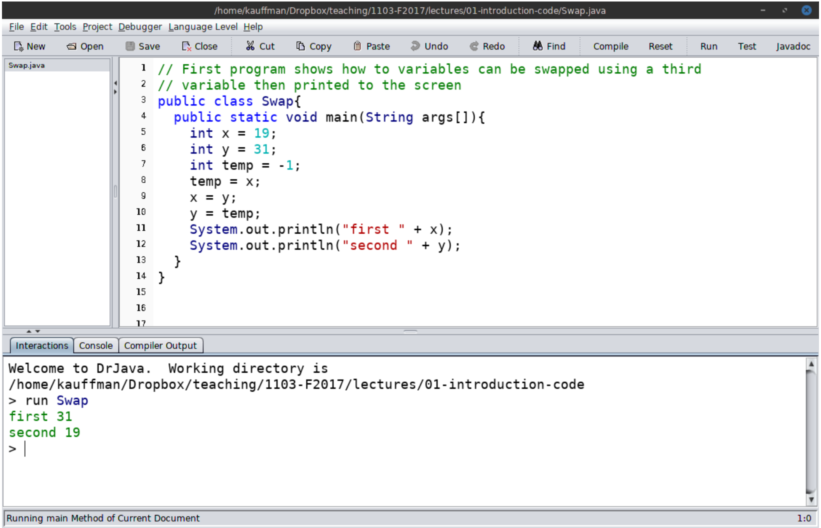
```
public class Swap{
  public static void main(String args[]){
    int x = 19;
    int y = 31;
    int temp = -1;
    temp = x;
    x = y;
    y = temp;
    System.out.println("first " + x);
    System.out.println("second " + y);
  }
}
```

- 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


/home/kauffman/Dropbox/teaching/1103-F2017/lectures/01-introduction-code/Swap.java
File Edit Tools Project Debugger Language Level Help
New
Open
Save
Close
Cut
Copy
Paste
Undo
Redo
Find
Compile
Reset
Run
Test
Javadoc
Swap.java
Interactions
Console
Compiler Output

```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
> 
```

Running main Method of Current Document 1:0

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                     # output of program on these 2 lines
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.

```
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`