Quick Reference and Troubleshooting Tips for Java

Bowdoin College Computer Science Department – CS 107
Stephen Majercik

Expressions

Operators according to precedence; each group of operators is at the same level of precedence. In an expression, operators at the same precedence level are evaluated left to right.

parentheses	()
function calls, e.g.	sqrt(number)
logical NOT	!
unary minus	-
increment 1	++
decrement 1	--
multiplication	*
division	/
mod (remainder)	%
addition	+
subtraction	-
less-than	<
less-than-or-equal-to	<=
greater-than	>
greater-than-or-equal-to	>=
equals	==
not-equals	!=
logical AND	&&
logical OR	||
assignment	=

Variable and Constant Declarations and Assignment Statements

```
final double TAX_RATE = 0.05;
char response;
int numBloodCellCounts = 0;
double totalCost = baseCost * (1.0 + TAX_RATE);
double root1 = (-b + Math.sqrt( Math.pow(b, 2) - 4.0 * a * c )) / (2.0 * a);
```

`if` Statements

In all of the following, I have put curly braces around even single statements, even though that's not necessary.

The "simple" `if`:

```
if (numWins == TOTAL_GAMES_IN_SEASON) {
    System.out.println("Congratulations on your undefeated season!");
}
```

The `if-else`:

```
if (divisor != 0) {
    double quotient = dividend / divisor;
    System.out.println(dividend + " divided by " + divisor +
                       " = " + quotient);
}
else {
    System.out.println("divison by zero is not defined!!");
}
```

The `if-else-if`:

```
if (guess < targetNumber) {
    System.out.println("Your guess is too low.");
}
else if (<boolean-expression>) {
    System.out.println("Your guess is too high.");
}
else {
    System.out.println("You guessed it!");
}
```

Loops

```
int possibleFactor = 2;
while (possibleFactor <= numberToFactor && numberToFactor > 1) {
    // do all the stuff to count the number of times
    // possibleFactor is a factor of numberToFactor
    ++possibleFactor;
}
```

```
for (int i = 0 ; i < costs.length ; ++i) {
    System.out.println("Cost " + (i+1) + " is " + costs[i]);
}
```

Arrays

```
final int NUM_COSTS = 5;
double[] costs = new double[NUM_COSTS];

// It's also possible to use an integer variable to
// specify the array size:
System.out.print("Enter the number of .....: ");
int numThingies = r.readInt();  // where r is a ReadStream
r.readLine();
double[] costs = new double[numThingies];
```

You can have an array reference anywhere you can have a reference to the kind of item in the array, e.g. on the left-hand side of an assignment statement, as part of an expression on the right-hand side of an assignment statement, in a Boolean expression, in an input or output statement. Also, you can use an integer variable (or even an expression that evaluates to an integer value) to reference a particular item in the array, i.e. `costs[i]` would refer to the item in the `costs` array at the index specified by the value of `i`. Frequently, you access each item in turn using a loop – see the `for`-loop example above, where the fact that an array knows its own length (`costs.length`) comes in handy.

Terminal Window Commands

Click on the Terminal icon in the dock to get a Terminal window. Once it's open, you can leave it open and use it for the entire session. Click on the yellow circle in the upper left corner to temporarily minimize the window and put its icon in the dock; click on the icon to get it back. Click on the red circle in the upper left corner or type exit in the Terminal window to get rid of it.

Compiling and Executing Java Programs

`=> javac <filename>.java`	Compiles this particular Java file, producing a <filename>.class byte-code file
`=> javac *.java`	Compiles all Java files in the directory you are in, producing .class byte-code files
`=> java <filename>`	Runs this java file (be sure to leave out the file extension (.java or .class)

Note that many of the following Directory Navigation and File Management functions can be done by manipulating icons in the Mac GUI (graphical user interface), but here's how they can be done in the Terminal window.

Directory Navigation

`=> cd  <dirname>`	Change the current working directory to a specific subdirectory called <dirname> in the current directory
`=> cd`	Go to the directory you were in when you started the Terminal window
`=> cd ..`	Move up one directory level
`=> ls`	List the contents of the directory you are in
`=> pwd`	Print out the name of the directory you are in
`=> mkdir <dirname>`	Make a new subdirectory <dirname> in the current directory

```
=> rmdir <dirname>    Remove the subdirectory <dirname> in the current
                      directory (note that the subdirectory <dirname>
                      must be empty)
```

File Management

```
=> less <filename>    Displays the contents of the specified file onto the
                      screen while stopping at each pageful of text:
                          space bar = advance a page,
                          b = go back a page
                          return = advance a line
                          q = quit

=> cp <filename1> <filename2>

                      Makes a copy of <filename1> and calls it <filename2>

=> mv                 [the move command]
                      Moves files; e.g. to move <filename1> in the current
                      directory into the subdirectory <subdir>:

=> mv <filename1> <subdir>/

                      To move <filename1> one directory level up:

=> mv <filename1> ../

                      Also used to change the name of <filename1> to
                      <filename2>:

=> mv <filename1> <filename2>


=> rm <filename>      Removes the specified file.  Be careful -- this does
                      NOT put the file in the Trash, so you can't get it
                      back if you change your mind.
```

Common Problems and Troubleshooting Tips

- Here are some relating to variable and constant declarations:
  - Remember to declare a variable (say what type it is) before you try to use it, but don't declare a variable twice (the compiler will catch this one).
  - A variable is accessible only inside the curly braces in which it was declared (and all curly braces inside those curly braces). You can't declare a variable inside the body of a loop, for example, and then use it when you get out of the loop. Related to this is the fact that a `for`-loop control variable declared in the `for`-loop header is not accessible outside that loop.
  - An especially tricky problem is if you declare a variable at one level in your program and then declare it again inside some curly braces at a lower level (e.g. in a loop). You can do this, but this actually sets up two *different* variables. Inside the loop, you will be referring to the one you declared inside the loop and any changes made will have no effect on the variable with the same name that was declared *outside* the loop.
  - Remember to use named constants. The compiler will not flag this as an error, but *I* will. A good rule of thumb is that you don't need to name any constant that's more or less "generic, " e.g. the zero in a Boolean expression that's checking if a number is positive or negative by comparing to zero, the 2 in a Boolean expression that's checking if a number is even or odd by `mod`ing it by 2.
- You can't assign a `double` value to an `int` variable.
- Remember to use parentheses to force an order of operations in an expression other than what the default precedence prescribes. This is particularly common with regard to logical AND and OR.
- Remember that when you want to do real division either the dividend or divisor (or both) must be `double`s. This can be especially mysterious when both are `int`s and the divisor is greater than the dividend, which gives you 0.
- Remember the update statement in a loop that ensures that the repetition condition eventually becomes false.
- Remember that the first item in an array is at index 0 and, if there are n items in the array, the last item is at index $n - 1$; n is not a legal index.
- Remember to put curly braces around a group of statements that are intended to be treated as a group (all executed or all not executed) in an `if` or an `else` or a `while` or a `for`.
- Remember to create a `ReadStream` object if you need to get input from the user.
- Remember to do an `r.readLine()` statement to consume the rest of the line (e.g. end-of-line character(s)) after you are done reading the data on a line.

- Particular items regarding compiling, running, and debugging:
  - Remember to put a semi-colon at the end of a variable declaration or an assignment statement; often the compiler doesn't realize one is missing until the next line (or even later), so look at the few lines of code before where the error was flagged if it's not obvious what's wrong.
  - When you open the Terminal window, remember to navigate to the folder that contains the files for the program you are working on.
  - When you're not sure where the problem is in your program (e.g. it compiles and runs, but does not do what it's supposed to do), a common debugging technique is to insert `System.out.println` statements at various places in your program to see if it's doing what you think it's doing. (**Be sure to remove these before handing in your program!**)
  - When the error in the Terminal window says that a particular symbol was not able to the be "resolved," it generally is referring to a variable and it means it doesn't know what variable that name is referring to. Usually, this indicates that you have forgotten to declare the variable or that you are referring to it in a place from which it is not accessible if you *did* declare it, e.g. you declared it inside your `while` loop and you're referring to it after you have exited the loop.
  - Pay attention to matching curly braces. Unmatched curly braces are almost always the problem when the compiler says it reached the end of the program unexpectedly or that it's looking for an "interface." (The top-down method of programming I've been advocating lessens the likelihood of this problem substantially.) The `Re-Indent` and `Balance` commands in the `Format` menu of `Xcode` are useful here. You can highlight a section of code (Apple-a to highlight the whole program) and choose `Re-Indent` to have `Xcode` make your indentation look pretty again if it's gotten messed up. If you're not sure what curly braces are matching what other curly braces, you can put the cursor just inside a curly brace and select `Balance`; `Xcode` will highlight all the material between that curly brace and the one it matches.