GNU Java Training Wheels

for version 2.0, 25 July 2016

Davin Pearson (davin.pearson@gmail.com)

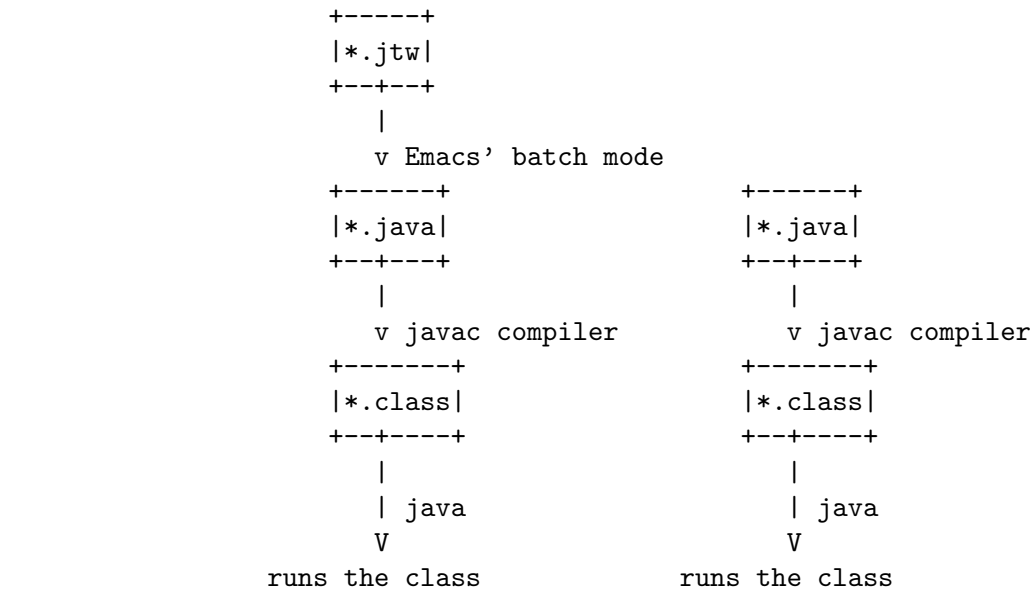
This manual is for GNU Java Training Wheels (version 2.0, 25 July 2016), which is a system for making it easier for novices to learn to program in the Java language.

Table of Contents

1 About GNU Java Training Wheels

This document documents a new programming language written by me, Davin Pearson (email: davin dot pearson at gmail dot com) called J.T.W., short for Java Training Wheels for the sole purpose of making it easier to learn to program in Java. The J.T.W. language has a similar syntax to Delphi, Pascal, BASIC and JavaScript and therefore learning J.T.W. before or while learning Java provides a less steep learning curve than learning Java from scratch. For many reasons you might even prefer to program in J.T.W. rather than Java. Here is why you should learn J.T.W. before or while learning Java:

- The J.T.W. language is supported by a parser that troubleshoots problematic J.T.W. code with clear error messages.
- The J.T.W. language compiles to Java in a natural and straightforward way so it is easy to learn Java once you know J.T.W.. See the following diagram for a comparison of the J.T.W. and Java build processes.



- Pascal-style begin ... end constructs are supported instead of C-style { ... } constructs which is more sensible especially for novices.
- A simple syntax for the main function: beginMain ... endMain rather than the rather cumbersome: public static void main(String[] args) { ... }.
- Class variables, properties, functions, methods and constructors are declared as such much like Delphi which makes your code look clearer. Specifically there are new keywords classVar, constructor, function, method and property.
- The Delphi/Pascal/JavaScript keyword var for clearer local variables.
- The Pascal/BASIC keyword then for clearer if statements.
- The BASIC and C++ style keywords and and or rather than Java's rather cumbersome: && and ||.
- The P.H.P. keyword elseif is supported instead of Java's cumbersome else if.

- As proof of concept, a superfor macro is presented for enhanced BASIC-style for loops. See Chapter 2 [J.T.W. Proof of concept #1 A superfor macro], page 3.
- As proof of concept, file inclusion is supported so that you can spread a class across several files. Natural divisions are methods. Different methods can be placed in different source files for those situations where methods become large and unwieldy. See Chapter 3 [J.T.W. Proof of concept #2 file inclusion], page 10.
- NEW! As of J.T.W version 2.0, packages are now supported. See Chapter 5 [Packages in J.T.W. and Java], page 70.

2 J.T.W. Proof of concept #1 A superfor macro

A proof of concept for the J.T.W. preprocessor is the superfor macro, which is an enhanced BASIC-style for loop. Here is how to invoke the superfor macro in your *.jtw file:

```
beginMain
    superfor (var int i = 0 to 10)
    begin
        System.out.println("i=" + i);
    end
endMain
```

The above code results in the following printout:

```
i=0
i=1
i=2
i=3
i=4
i=5
i=6
i=7
i=8
i=9
i=10
```

The step size argument is optional, here is an example with an explicit step size announced:

```
beginMain
    superfor (var int i = 0 to 10 step 2)
    begin
        System.out.println("i=" + i);
    end
endMain
```

The above code results in the following printout:

```
i=0
i=2
i=4
i=6
i=8
i=10
```

If the downto keyword is given instead of the to keyword then the loop will count downwards from the first given number to the second, even if a postive step size is given. Here is an example with a negative step size:

```
beginMain
    superfor (var int i = 10 downto 0 step 2)
    begin
        System.out.println("i=" + i);
    end
```

```
endMain
```

The above code results in the following printout:

```
i=10
i=8
i=6
i=4
i=2
i=0
```

Note that the specification of the superfor macro doesn't need constants for the values of start, stop and step-size. They can be any variable or more generally any Java expression, and those expressions will be evaluated only once, should your code have side effects, i.e. changes the value of a variable in your code. See the following example. The expression ++y has the side effect of incrementing the value of y before returning the value of y:

```
beginMain
    var int x = 20;
    var int y = 15;
    superfor (var int i = x to (2 * ++y))
    begin
        System.out.println("i=" + i);
    end
endMain
```

The above code results in the following printout:

```
i=20
i=21
i=22
i=23
i=24
i=25
i=26
i=27
i=28
i=29
i=30
i=31
i=32
```

2.1 Elisp source code for the superfor macro

The following code belongs in the file ~/jtw/jtw-build-jtw.el which in itself is too large for inclusion in this manual. You can find this code by visiting [J.T.W. tarball], page 12. Alternatively, you can study this fragment of the file ~/jtw/jtw-build-jtw.el which deals with the superfor macro. In the listing that follows, *pp-namespace* stores a string containing a long arbitrary name to prevent accidental aliasing of the include directives with rest of the comments code.

```
(let (p1 p2 str form type variable T var start stop step-size step-size-
2
```

```
       this_start this_stop this_step this_step_size file line p-prior
       beg0 end0)
;;(checkpoint "2")
(save-excursion
  ;;(setq parse-sexp-ignore-comments t)
  (goto-char (point-min))
  (setq *superfor* 0)
  (while (re-search-forward "\\<superfor\\>" nil t)
    (setq beg0 (match-beginning 0))
    (setq end0 (match-end 0))
    ;;(checkpoint "sitting for 1 seconds...")
    (font-lock-fontify-buffer)
    ;;(sit-for 1)
    (when (save-excursion
            (save-match-data
              (re-search-forward "(" (point-at-eol) t)
              (forward-char -1)
              (re-search-forward "\\<var\\>" nil t)
              (not (warn--inside-comment-or-string))))
      ;;(error "Smelly cat")
      (setq *current-buffer* (current-buffer))
      ;;(switch-to-buffer *current-buffer*)
      (setq p1 beg0)
      (assert (save-match-data
                (looking-at " \t\r\n*(")))
      (setq p2 (save-excursion
                 (forward-sexp 1)
                 (point)))
      (setq str (buffer-substring-no-properties end0 p2))
      ;;(checkpoint "str=%s" str)
      (setq form (read-str str))
      ;;(checkpoint "form=%s" form)
      ;;(debug "form")
      (assert (consp form))
      ;;(error "Rolling Stones plays Cuba")
      (delete-region p1 p2)
      (incf *superfor*)
      (setq this (format "superfor_%d_" *superfor*))
      (when (not (eq (nth 0 form) 'var))
        (warn--log-message "Error 35: Keyword var missing")
        (when (and (not (eq (nth 0 form) 'char))
                   (not (eq (nth 0 form) 'short))
                   (not (eq (nth 0 form) 'int))
                   (not (eq (nth 0 form) 'long))
                   (not (eq (nth 0 form) 'float))
                   (not (eq (nth 0 form) 'double)))
          (warn--log-message (concat
```

```
                                    "Error 37:#1 argument type to superfor macro must be"
                                    " one of char/short/int/long/float/double"))))
          (when (eq (nth 0 form) 'var)
            (if (and (not (eq (nth 1 form) 'char))
                     (not (eq (nth 1 form) 'short))
                     (not (eq (nth 1 form) 'int))
                     (not (eq (nth 1 form) 'long))
                     (not (eq (nth 1 form) 'float))
                     (not (eq (nth 1 form) 'double)))
                (warn--log-message (concat
                                    "Error 37:#2 argument type to superfor macro must be
                                    " one of char/short/int/long/float/double")))
            (progn
              (setq type (nth 1 form))
              (setq T (prin1-to-string type))
              (setq variable (nth 2 form))
              (setq variable-equals (prin1-to-string (nth 2 form)))
              (if (string-match "=" variable-equals)
                  (progn
                    (setq variable (read-str (substring variable-equals 0 (match-
beginning 0))))
                    (setq start    (read-str (substring variable-equals (match-
end 0))))
                    (if (eq (nth 3 form) 'to)
                        (setq to 'to)
                      (if (eq (nth 3 form) 'downto)
                          (setq to 'downto)
                        (assert "The Bug")))
                    (setq stop     (nth 4 form))
                    (if (eq (nth 5 form) 'step)
                        (setq step-size (nth 6 form))
                      ;;(setq step-size 1)
                      ))
                (assert (or (eq (nth 3 form) '=)))
                ;;(debug "Joaquin Rodrigo")
                (setq start (nth 4 form))
                (if (eq (nth 5 form) 'to)
                    (setq to 'to)
                  (if (eq (nth 5 form) 'downto)
                      (setq to 'downto)
                    (assert "Planet of New Orleans")))
                (setq stop (nth 6 form))
                (if (eq (nth 7 form) 'step)
                    (progn
                      (setq step-size (nth 8 form))
                      ;;(assert (numberp (nth 8 form)))
                      )
```

```
                ;;(setq step-size 1)
                )
              ))
            (d-quote
             (debug "Rocket Queen")
             (setq type "")
             (setq T "double")
             (setq variable (nth 0 form))
             (assert (eq (nth 1 form) '=))
             (setq start (nth 2 form))
             (if (eq (nth 3 form) 'to)
                 (setq to 'to)
               (if (eq (nth 3 form) 'downto)
                   (setq to 'downto)
                 (debug "My Michelle")))
             (setq stop (nth 4 form))
             (if (eq (nth 5 form) 'step)
                 (setq step-size (nth 6 form))))
            (setq var         (prin1-to-string variable))
            (setq start       (prin1-to-string start))
            (setq stop        (prin1-to-string stop))
            (setq step-size-2 (prin1-to-string step-size))
            (setq start       (warn--splat-quest start))
            (setq stop        (warn--splat-quest stop))
            (setq step-size-2 (warn--splat-quest step-size-2))
             ----------------------------------------------------
            (setq this_start     (concat this "start"))
            (setq this_stop      (concat this "stop"))
            (setq this_step      (concat this "step"))
            (setq this_step_size (concat this "step_size"))
            ;;(debug "My Parties")
            (insert (concat (concat "var " T " " this_start " = " start ";")
                            (concat "var " T " " this_stop  " = " stop  ";")
                            (if step-size
                                (concat "var " T " " this_step " = " step-
size-2 ";"
                                        "var " T " " this_step_size " = "
                                        (if (eq to 'to) (concat "Math.abs(" this_step ")
cat "-Math.abs(" this_step ")") (debug "Heavy Fuel"))) ";\n")
                              (concat "var " T " " this_step_size " = " (if (eq to 'to)
1" (debug "Ticket to Heaven"))))) ";\n"))
            (setq line 0)
            (setq p-prior
                  (save-excursion
                    (beginning-of-line)
                    (setq str (concat "^ \t*//+ " *pp-namespace* " #location0-
9"
```

```
                                        " (\\(" *drive-spec* "-a-zA-Z0-9_./+\\):\\(0-
9+\\))"))
                  (if (or (looking-at str) (re-search-backward str nil t))
                      (progn
                        ;;(debug "Antonio Vivaldi")
                        (setq file (buffer-substring-no-properties          (match-
beginning 1)
                                                                           (match-
end 1)))
                        (assert (stringp file))
                        (setq line (read-str (buffer-substring-no-properties (match-
beginning 3)
                                                                           (match-
end 3))))
                        (assert (integerp line))
                        (point)
                        )
                    (setq file (concat default-directory *stump* ".jtw"))
                    (setq line 1)
                    (goto-char (point-min))
                    (forward-line 2)
                    (point)
                    )))
          (setq line (+ line (count-lines p-prior (point))))
          (decf line)
          (decf line)
          (insert (format "// %s (setq file-stack '%s)\n" *list-namespace* (prin1-
to-string file-stack)))
          (insert (format "// %s #location3 (%s:%d)\n" *pp-namespace* file line))
          (insert (concat "for (var " T " " var " = " this_start ";"
                          " ((" this_step_size " > 0) ? " var " <= "
                          this_stop " : " var " >= " this_stop "); "
                          var " += " this_step_size ")"))
          ;;(debug "Yehudi Menuhin")
          )))))
)
```

2.2 A bug in J.T.W. superfor

The question mark operator a ? b : c which expands to

```
Type result;
if (a) then
begin
    result = b;
end
else
begin
```

```
        result = c;
    end
```

where type can be any Java type is not directly supported by the arguments to the superfor macro in J.T.W. Elsewhere the question mark is supported. Instead in the superfor macro you have to write the following code to get a question mark operator online:

```
class SuperFor
begin
    beginMain
        foo(1,2);
    endMain
    function void foo(int x, int y)
    begin
        superfor (var int i=0 to (x < y QUEST 10 : 20))
        begin
            System.out.print(" " + i);
        end
        System.out.println();
    end
end
```

where the symbol QUEST compiles into a question mark: ?. When built, the program prints out the following:

```
i=0
i=1
i=2
i=3
i=4
i=5
i=6
i=7
i=8
i=9
i=10
```

3 J.T.W. Proof of concept #2 file inclusion

When your classes become large and unwieldy, it becomes useful to split a source file into several compilation units. the most natural division into compilation units is at the level of methods. with each method in a separate file you can manage methods that are excessively large. here is how to use file inclusion in the J.T.W. language. first comes the Foo.jtw file with all bodies of methods harvested from them:

```
class Foo
begin
    include "Foo-apple.method";
    include "Foo-banana.method";
    include "Foo-carrot.method";
end
```

Here are the files that get included. the first file is Foo-apple.method:

```
property int prop1; /* declares a property for use with the apple method. */

method void apple(/* parameters */)
begin
    prop1 = prop1 + 1;
    /* rest of body of apple method */
end
```

the second file is Foo-banana.method:

```
method void banana(/* parameters */)
begin
    /* body of banana method */
end
```

the third file is Foo-carrot.method:

```
method void carrot(/* parameters */)
begin
    /* body of carrot method */
end
```

when all of the file inclusions have been carried out by the J.T.W. to java compiler, the code that javac sees will be something like this:

```
/* Automatically generated file. Do not edit! */
// #foomatic #location (Foo.jtw:1)
class Foo
{
    // #foomatic #location (apple.method:1)
    int prop1; /* declares a property for use with the apple method. */

    void apple(/* parameters */)
    {
        prop1 = prop1 + 1;
        /* rest of body of apple method */
    }
```

```
    // #foomatic #location (banana.method:1)
    void banana(/* parameters */)
    {
        /* body of banana method */
    }
    // #foomatic #location (carrot.method:1)
    void carrot(/* parameters */)
    {
        /* body of carrot method */
    }
    // #foomatic #location (Foo.jtw:6)
}
```

Note the use of the value #foomatic of the string *pp-namespace* (where pp stands for pre-processor) that is a long arbitrarily defined string to prevent accidental aliasing with the rest of the commented code that the user of the system might write. The #location directives are used to keep track of the original line number in the source file. Using Emacs batch mode executing the Elisp code: jtw-build-jtw.el, error messages in Foo.java now point back to the original Foo.jtw file, or one of the files that get #included like so: apple.method, banana.method or carrot.method. If you are particularly clever, you can reuse the same method in different classes.

Version 1.0 of J.T.W. used the C Pre-Processor (cpp) to manage the #location directives but unfortunately cpp destroys comments in the target file, and Java uses /** ... */ comments to document the program's behaviour so cpp cannot be used.

4 J.T.W. Tutorials

The following tutorials including full model answers can be found on my Website at the following location:

http://davin.50webs.com/J.T.W

The advantage of using my online tutorials is that the tutorials have superior syntax highlighting of the code examples. See the following link Appendix B [Passwords for the answers to the tutorials], page 83, for the passwords on my Website.

4.1 Tutorial 1 Your first program

Question 4.1.1: Some code to get you started. First, please visit the following Website http://davin.50webs.com/J.T.W/download-links.html for the programs that you need to have installed before you can do any coding in J.T.W. You should then download a tarball (also known as a compressed archive file): http://davinpearson.com/binaries/java-training-wheels-1.1.tar.gz containing the code you need to get started. Then unzip the tarball and change directory to java-training-wheels and issue the following command on GNU/Linux systems: ./configure or on MS Windows systems: bash configure. If you are using M.S. Windows and your HOME variable is unset, then you will need to set it to a sensible value. Examples of sensible values for your HOME variable include, c:\ or c:\home or d:\home if your d drive is a hard drive. To set the HOME variable in windows, press Windows E and right click on My Computer (Windows XP) or This Computer (Windows 10) and click on Properties, then click on Advanced system settings, then click on Advanced, then click on New environment variable to set the HOME variable.

When you run the configure script you will be prompted for the location of prefix directory and the location of the place to keep your *.jtw files. You will also be asked if you want to install just Davin's jtw-mode or Davin's full version of Emacs.

NOTE: If are reading this file on your local filesystem then you would have already completed this question.

Question 4.1.2: Your first J.T.W. program. Traditionally in computer science the first program that you write in any programming language is a program that does nothing else but prints out "Hello, World". The following code does just that. In the following code, note the use of the class construct. In Java and J.T.W., every piece of program code that does some real computational work resides in a class of some description.

```
class MyFirstProgram
begin
    beginMain
        System.out.println("Hello, World!");
    endMain
end
```

The code for any class X in these tutorials should reside in a file called X.jtw. Therefore the above code should be put into a file called MyFirstProgram.jtw. If two classes X and Y use each other and X contains the main function then it is convenient to place them both in a file called X.jtw. To build and run some code, you first need to be in the ~/jtw-tutorials folder and secondly you need to issue the following shell command: make X.run where X is the name of the class that you want to run, so it is

make MyFirstProgram.run

in this case. For all the questions that follow this one, it will be assumed that you know how to do this. See Section 5.5 [How to build a collection of class files or an entire package], page 74, for more information about building classes that use other classes in different files or building entire packages.

Question 4.1.3: Multiple calls to System.out.println(). Change the above code from printing the string "Hello, World!" to printing out the following messages. Please note that it will be easiest to use multiple calls to System.out.println() which sends text to the screen for the purpose of viewing.

```
Hello, Anne! How are you doing?
Hello, Brian! How are you doing?
Hello, Clare! How are you doing?
```

Question 4.1.4: Functions,parameters and arguments. A function is a piece of code that does some computational work and optionally returns a value. Notice how the hello function below takes a value of whose name to say hello to. This value who is called a parameter. The values passed to the parameter by the call to the function is called an argument. For the purposes of this question, add two more calls to the hello function in the main function to get the same result as the code for the previous question. The keyword void indicates that this function does not return a value. See the next question for a function that does return a value.

```
class MySecondProgram
begin
    function void hello(String who)
    begin
        System.out.println("Hello " + who + ", how are you doing?");
    end
    beginMain
        hello("Anne");
    endMain
end
```

Question 4.1.5: Return values. Notice how the following hello function returns a string rather than printing out the string. Add two more calls to the hello function below to get the same result as for Question 4.1.4.

```
class MyThirdProgram
begin
    function String hello(String who)
    begin
        return "Hello " + who + ", how are you doing?";
    end
    beginMain
        System.out.println(hello("Anne"));
    endMain
end
```

Question 4.1.6: Ignoring return values. In J.T.W. and Java, it is not necessary to use a value that is returned by a function. Sometimes this wastes computational resources since

the value that is computed by the function is not used but other times when the function whose value is to be ignored does some additional work by setting the value(s) of some variable(s) to different values then the function call is not a waste of resources. To ignore the value returned by the hello function, simply call the function without using the value like so: hello("Ignored"); For the purposes of this question, try calling the hello function without using the return value by adding a line of code to the main function.

Question 4.1.7: Comments. Study the following code. Note the use of comments. Comments are used to disable code for debugging purposes and also to help explain how a program works. The most useful comment in J.T.W. and Java is /** until the first */. This type of comment is harvested by Javadoc to produce documentation on how a class works. The second and third most useful comments are (respectively) // until the end of the line and /* until the first */. The third type of comment is not very useful because in Java you are not allowed to have one comment inside another, so if you use this type of comment you will constantly need to search for and remove */ closing comments. In the tutorials that follow you will see many comments, although mainly the first and second types of comments.

```
/** This comment is harvested by Javadoc
        to document the MyFourthProgram class */
class MyFourthProgram
begin // I am a single line comment
    /* I am
            a multi-line
                    comment */
    /** This comment is harvested by Javadoc
            to document the hello function */
    function String hello(String who)
    begin
        return "Hello " + who + ", how are you doing?";
    end )
    /** This comment is harvested by Javadoc
            to document the main function */
    beginMain
        System.out.println(hello("Anne"));
    endMain
end
```

4.2 Tutorial 2 Introduction to programming in Java

Question 4.2.1: The following code returns whether or not the current parameter ch is a vowel. The parameter ch is of type char which is used to hold the components of a string. That is to say, strings are built out of sequences of chars. Also note the use of the Character.toUpperCase function to convert chars into uppercase chars so that the code works equally well for isVowel('a') and isVowel('A'). Study, compile and run the following code. Does it print what you expected it to? If not, then fix the bug.

```
class Scrabble
begin
    function boolean isVowel(char ch)
```

```
    begin
        ch = Character.toUpperCase(ch);
        if ((ch == 'A') or (ch == 'E') or (ch == 'I') or (ch == 'O') or (ch == 'U'))
        then return true;
        else return false;
    end
    beginMain
        System.out.println(isVowel('a'));
    endMain
end
```

In the above code, note the difference between a = b example: ch = Character.toUpperCase(ch) and a == b example: ch == 'A'. The first is an assignment that sets a to be whatever the value of b is, while the second is a question that says whether or not the two arguments a and b are equal.

Note that later on in this tutorial you will learn that this is not the way to compare two strings. Also note the use of the boolean return type. This means that the return value is either true or false.

Question 4.2.2: By copying the pattern established by the above code, write a function isConsonant which returns whether or not the given argument is not a vowel. The easiest way to do this is to write isVowel(ch) == false which means: "ch is not a vowel". You will also need to ensure that the parameter ch is greater than or equal to 'A' and less than or equal to 'Z'. Then test your code by calling isConsonant from the main function.

Question 4.2.3: By copying the pattern established in the following code:

```
function int countVowels(String word)
begin
    var int result = 0;
    superfor (var int i=0 to i<word.length()-1)
    begin
        var char ch = word.charAt(i);
        if (isVowel(ch)) then result = result + 1;
    end
    return result;
end
```

write a function that counts the number of consonants in a word. Note the use of the var keyword for defining variables that are local to functions. Local variables are very much like parameters that were introduced in the previous tutorial. In the above code, note the use of word.charAt(i) and word.length(). The first of these results the character at location in the string word given by the value of i and the second of these returns the length of the string word. In Tutorial 11 you will learn that these are called methods which are different from functions that currently know how to write. Until we get to this tutorial and we are ready to teach you how to write your own methods, you will only call existing methods such as the above methods of the String class. Then test your code by calling it from the main function.

Question 4.2.4: Write a function simpleScoreWord that calls countVowels and countConsonants to give a Simple Score of a word. The Simple Score of a word is the number of

vowels in the word plus the number of consonants in the word times ten. Then test your code by calling it from the main function.

Question 4.2.5: Write a function advancedScoreLetter that returns the Advanced Score of a letter. Here is a breakdown of the distribution of letters for the purpose of the calculation of the Advanced Scores.

```
2 blank tiles (scoring 0 points)
1 point: E 12 tiles, A 9 tiles, I 9 tiles, O 8 tiles, N 6 tiles, R 6 tiles, T 6 tiles,
2 points: D 4 tiles, G 3 tiles
3 points: B 2 tiles, C 2 tiles, M 2 tiles, P 2 tiles
4 points: F 2 tiles, H 2 tiles, V 2 tiles, W 2 tiles, Y 2 tiles
5 points: K 1 tiles
8 points: J 1 tiles, X 1 tiles
10 points: Q 1 tiles, Z 1 tiles
```

Then test your code by calling it from the main function.

Question 4.2.6: Write a function advancedScoreWord that returns the Advanced Score of a word. The Advanced Score of a word is the sum of the Advanced Scores of each letter in the word. If the word is eight letters long then you should add an extra, say, 50 points to the score. Then test your code by calling it from the main function.

Question 4.2.7: Comparing strings. Amend the advancedScoreWord function so that swear words get a score of zero. For the purposes of this question you only need to think of three swear-words to add to the code. In the interests of not offending anyone, please keep your choice of swear words very tame. When comparing strings it is a mistake to use == which you already know is how you compare the following types that you know of so far: booleans, chars and ints. Using == on strings compiles and runs but gives you the incorrect result. The correct method to compare strings is to use the equals method of the string class like so: word.equals("bugger") which returns true or false, depending on whether or not the string word currently holds the value "bugger".

Question 4.2.8: Change the advancedScoreWord function so it works equally well with uppercase words and lowercase words. You will need write to call either word.toUpperCase() or word.toLowerCase() and store the result in word.

4.3 Tutorial 3 superfor loops and for loops

Question 4.3.1a: For loops that count up in steps of one. Study the following code and verify that it prints out "2 3 4 5 6 7 8 9 10" by compiling and running it. Notice that the System.out.print() function call doesn't print a carriage return after printing the argument value. That is why the System.out.println() function call is needed at the end of the for loops superfor and for, to print a carriage return at the end of the line. Also note the use of the plus sign to concatenate a string and the number to produce string result.

```
class ForTest
begin
    beginMain )
        /* Here is the superfor loop: */
        superfor (var int i=2 to 10) System.out.print(" " + i);
        System.out.println();
        /* Here is the ordinary for loop: */
```

```
        for (var int i=2; i<=10; i=i+1) System.out.print(" " + i);
        System.out.println();
    endMain
end
```

Question 4.3.1b: Change the superfor loop and the ordinary for loop to print out the following numbers: "5 6 7 8 9 10".

Question 4.3.1c: Change the superfor loop and the ordinary for loop to print out the following numbers: "234 235 236 237 238".

Question 4.3.1d: Change the superfor loop and the ordinary for loop to print out the following numbers: "48 49 50 ... 75 76".

Question 4.3.1e: Change the superfor loop and the ordinary for loop to print out the following numbers: "-5 -4 -3 -2 -1 0 1 2 3".

Question 4.3.2a: For loops that count up in steps greater than one. Study the following code and verify that it prints out "10 15 20 25 30 35 40" by compiling and running it.

```
class ForTest
begin
    beginMain )
        /* Here is the superfor loop: */
        superfor (var int i=10 to 40 step 5) System.out.print(" " + i);
        System.out.println();
        /* Here is the ordinary for loop: */
        for (var int i=10; i<=40; i=i+5) System.out.print(" " + i);
        System.out.println();
    endMain
end
```

Question 4.3.2b: Change the superfor loop and the ordinary for loop to print out the following numbers: "20 25 30 35 40".

Question 4.3.2c: Change the superfor loop and the ordinary for loop to print out the following numbers: "100 105 110 115 120 125".

Question 4.3.2d: Change the superfor loop and the ordinary for loop to print out the following numbers: "2 4 6 8 10 12 14".

Question 4.3.2e: Change the superfor loop and the ordinary for loop to print out the following numbers: "10 13 16 19 22 25".

Question 4.3.3a: For loops that count down in steps of one. Study the following code and verify that it prints out "10 9 8 7 6 5 4 3 2 1" by compiling and running it.

```
class ForTest
begin
    beginMain
        /* Here is the superfor loop: */
        superfor (var int i=10 downto 1) System.out.print(" " + i);
        System.out.println();
        /* Here is the ordinary for loop: */
        for (var int i=10; i>=1; i=i-1) System.out.print(" " + i);
        System.out.println();
```

```
    endMain
end
```

Question 4.3.3b: Change the superfor loop and the ordinary for loop to print out the following numbers: "10 9 8 7 6 5 4".

Question 4.3.3c: Change the superfor loop and the ordinary for loop to print out the following numbers: "20 19 18 17 16 15 14 13 12".

Question 4.3.3d: Change the superfor loop and the ordinary for loop to print out the following numbers: "66 65 64 ... 47".

Question 4.3.3e: Change the superfor loop and the ordinary for loop to print out the following numbers: "3 2 1 -1 -2 -3 -4 -5 -6 -7".

Question 4.3.4a: For loops that count down in steps greater than one. Study the following code and verify that it prints out "100 90 80 70 60 50 40 30 20" by compiling and running it.

```
class ForTest
begin
    beginMain
        /* Here is the superfor loop: */
        superfor (var int i=100 downto 20 step -10) System.out.print(" " + i);
        System.out.println();
        /* Here is the ordinary for loop: */
        for (var int=100; i>=20; i=i-10) System.out.print(" " + i);
        System.out.println();
    endMain
end
```

Question 4.3.4b: Change the superfor loop and the ordinary for loop to print out the following numbers: "80 70 60 50 40 30 20".

Question 4.3.4c: Change the superfor loop and the ordinary for loop to print out the following numbers: "500 490 480 470 460".

Question 4.3.4d: Change the superfor loop and the ordinary for loop to print out the following numbers: "10 8 6 4 2 0".

Question 4.3.4e: Change the superfor loop and the ordinary for loop to print out the following numbers: "33 28 23 18 13 8 3".

Question 4.3.5a: For loops that use floating point numbers to count. Study the following code and verify that it prints out "1.1 2.2 3.3 4.4" by compiling and running it. The type name double is short superfor double precision floating point. It is natural to ask: why not use single precision floating point? The answer to this question is that double precision floating point gives fewer compilation errors than single precision floating point does.

```
class ForTest
begin
    beginMain )
        /* Here is the superfor loop: */
        superfor (var double i=1.1 to 4.4 step 1.1) System.out.print(" " + i);
        System.out.println();
        /* Here is the ordinary for loop: */
```

```
        for (var double i=1.1; i<=4.4; i=i+1.1) System.out.print(" " + i);
    endMain
end
```

Question 4.3.5b: Change the superfor loop and the ordinary for loop to print out the following numbers: "0 2.2 4.4 6.6". Note that rounding errors may prevent you from getting this exact answer. Also note that the answer to this question is not what your would naively expect without running the code.

Question 4.3.5c: Change the superfor loop and the ordinary for loop to print out the following numbers: "-30 -19.9 -9.8 0.3 10.4 20.5".

Question 4.3.5d: Change the superfor loop and the ordinary for loop to print out the following numbers: "100.0 96.7 93.4 90.1 86.8 83.5 80.2 76.9".

Question 4.3.5e: Change the superfor loop and the ordinary for loop to print out the following numbers: "-100.0 -105.5 -111.0 -116.5".

Question 4.3.6a: For loops that use chars to count. Study the following code and verify that it prints out "a b c d e f g h i j k l m n o p q r s t u v w x y z" by and running it.

```
class ForTest
begin
    beginMain
        /* Here is the superfor loop: */
        superfor (var char i='a' to 'z') System.out.print(" " + i);
        System.out.println();
        /* Here is the ordinary for loop: */
        for (var char i='a'; i<='z'; i=i+1) System.out.print(" " + i);
        System.out.println();
    endMain
end
```

Question 4.3.6b: Change the superfor loop and the ordinary for loop to print out the following numbers: "a b c d e f".

Question 4.3.6c: Change the superfor loop and the ordinary for loop to print out the following numbers: "z y x w v u t s r q p o n m l k j i h g f e d c b a".

Question 4.3.6d: Change the superfor loop and the ordinary for loop to print out the following numbers: "p o n m l k j i h".

Question 4.3.6e: Change the superfor loop and the ordinary for loop to print out the following numbers: "A B C D E F G H I J K L M N O P Q R S T U V W X Y Z".

4.4 Tutorial 4 Four looping constructs

Question 4.4.1: Study, compile and run the following code:

```
class LoopTest
begin
    function int powerOf2A(int n)
    begin
        var int counter = n;
        var int result  = 1;
        while (counter > 0)
```

```
    begin
        result = 2 * result;
        counter = counter - 1;
    end
    return result;
end

function int powerOf2B(int n)
begin
    var int counter = n;
    var int result  = 1;
    do
    begin
        result = 2 * result;
        counter = counter - 1;
    end while (counter > 0);
    return result;
end

function int powerOf2C(int n)
begin
    var int result = 1;
    for (var int counter = n; counter > 0; counter = counter - 1)
    begin
        result = 2 * result;
    end
    return result;
end

function int powerOf2D(int n)
begin
    var int result = 1;
    superfor (var int counter=n downto 1)
    begin
        result = 2 * result;
    end
    return result;
end

 /**
  * Prints a row of stars of a given length.
  */
function void printLineC(int length)
begin
    for (var int i = 0; i<length; i=i+1)
    begin
        System.out.print("#");
```

```
        end
        System.out.println();
    end

    beginMain
        // For Question 4.4.1 add some code here...
    endMain
end
```

Question 4.4.2: To the main function add some code to call the functions powerOf2A, powerOf2B, powerOf2C and powerOf2D to verify that they all return the same result. To inspect the result you will need to apply the System.out.println() statement to the values returned by those functions.

Question 4.4.3: There is a bug in the powerOf2B function because it does not behave correctly in the case when n is zero or less. Put an if (...) then ... statement at the top of this function to make it handle the case of zero properly. Also make it return 1 in the case that n is less than zero.

Question 4.4.4: There is a bug in the powerOf2D function because it does not behave correctly in the case when n is zero or negative. Make it return 1 if n <= 0. Put an if (...) then ... statement at the top of this function to make it handle these cases properly. Since this function returns an int, make it return 1 in these cases.

Question 4.4.5: By copying the pattern of powerOf2A, powerOf2B, powerOf2C and powerOf2D, write functions printLineA, printLineB and printLineD that work identically to the function printLineC, except that they use while loops, do loops and superfor loops, respectively. Add some code to the main function to test them out.

Question 4.4.6: Based on the previous three questions, is there a best looping construct? Or does it depend on what the looping construct is going to be used for?

4.5 Tutorial 5 A beer drinking song

Question 4.5.1: Study the following code and then compile and run it to verify that it prints out the lyrics to a popular song:

```
class BeerSong
begin
    beginMain
        System.out.println("Five bottles of beer on the wall.");
        System.out.println("Five bottles of beer on the wall.");
        System.out.println("If one bottle of beer should accidentally fall,");
        System.out.println("there'd be four bottles of beer on the wall.");
        System.out.println();
        System.out.println("Four bottles of beer on the wall.");
        System.out.println("Four bottles of beer on the wall.");
        System.out.println("If one bottle of beer should accidentally fall,");
        System.out.println("there'd be three bottles of beer on the wall.");
        System.out.println();
        System.out.println("Three bottles of beer on the wall.");
        System.out.println("Three bottles of beer on the wall.");
```

```
            System.out.println("If one bottle of beer should accidentally fall,");
            System.out.println("there'd be two bottles of beer on the wall.");
            System.out.println();
            System.out.println("Two bottles of beer on the wall.");
            System.out.println("Two bottles of beer on the wall.");
            System.out.println("If one bottle of beer should accidentally fall,");
            System.out.println("There'd be one bottle of beer on the wall.");
            System.out.println();
            System.out.println("One bottle of beer on the wall.");
            System.out.println("One bottle of beer on the wall.");
            System.out.println("If one bottle of beer should accidentally fall,");
            System.out.println("there'd be no bottles of beer on the wall.");
            System.out.println();
        endMain
    end
```

Question 4.5.2: The following is the first attempt to make the code smaller but to keep the same output: If you compile and run the following code you will notice that it counts up from one rather than down from n. Change the for loop so that it runs down rather than up. For information about how to write the for loop, please consult Tutorial 4.2.

```
    class BeerSong
    begin
        function song(int n)
        begin
            superfor (var int i=1 to n)
            begin
                System.out.println(i + " bottles of beer on the wall");
                System.out.println(i + " bottles of beer on the wall");
                System.out.println("If one bottle of beer should accidentally fall,");
                System.out.println("there'd be " + (i-1) + " bottles of beer on the wall")
                System.out.println();
            end
        end

        beginMain
            song(5);
        endMain
    end
```

Question 4.5.3: Finish the number2string function below and add a new function call to this function in the song function so that it print textual numbers rather than digits.

```
    function String number2string(int n)
    begin
        assert n>=0 : n;
        assert n<=10: n;
        if (n == 0) then return "no";
        if (n == 1) then return "one";
        if (n == 2) then return "two";
```

```
    /* rest of code goes here */
    if (n == 9) then return "nine";
    if (n == 10) then return "ten";
    assert false;
end
```

Question 4.5.4: Add a new function String capitalize(String s) that capitalizes the first word in a String and call this function from the song function so that the first words in each sentence are capitalized. You should find the function Character.toUpperCase and the methods charAt and substring in the package java.lang helpful for writing this function. See the class String in the package java.lang at http://docs.oracle.com/javase/1.5.0/docs/api for more details.

Question 4.5.5: Add new function call String plural(int n) that returns the string "s" if n is not equal to 1 and the empty string "" otherwise. Then call this function from the song function so that the phrase "bottle" is pluralized when it should be.

Question 4.5.6: Write a function called number2string2 that can handle values up to but not including 100. Note that you will need multiple if ... then statements to achieve this. Note that if n is a number then the following expressions are useful:

```
var int temp1 = n / 10 % 10 results in temp1 holding the tens digit of n and is z
var int temp2 = n % 10 results in temp2 holding the ones digit of n.
```

Also make it print out "one hundred or more" in the case that n>=100

Question 4.5.7: Change the song function so that the following function call: song(5,"rum"); in the main function results in the following printout:

```
Five bottles of rum on the wall.

...

there'd be no bottles of rum on the wall.
```

Question 4.5.8: Once all the code is working, add the following line to the main function: song(100,"gin"); so that it prints out the following:

```
One hundred bottles of gin on the wall.

...

there'd be zero bottles of gin on the wall.
```

Question 4.5.9: Write a new function number2string3 that works like number2string2 and number2string except that it handles numbers up to 999. Internally number2string3 should call number2string2. You might find the following function useful:

```
function String textand(String a, String b)
begin
    if (a.equals("") or b.equals("")) then return a + b;
    else return a + " and " + b;
end
```

Question 4.5.10: Tricky! Write a new function number2string4 that works like number2string3 execpt that it handles numbers up to nine hundred and ninety-nine million nine

hundred and ninety-nine thousand nine hundred and ninety-nine, i.e. 999,999,999. The function number2string4 should internally call number2string3 like so:

```
var String ones = number2string3(n % 1000);
var String thousands = number2string3(n / 1000 % 1000);
var String millions = number2string3(n / 1000 / 1000 % 1000);
```

Note that the variables above will have values from 0 to 999 inclusive.

4.6 Tutorial 6 Class variables

Question 4.6.1: Study, compile and run the following code. Note the use of the class variable myMoney. A class variable is different from a variable that is local to a function because the lifetime of the class variable is for the duration that the program is run, whereas the lifetime of a local variable is for the duration of the function call. In the code that follows, the variable myMoney is used to store a numerical value, for how much money you have.

```
class Money
begin
    /** Property myMoney stores money value in dollars */
    classVar int myMoney;

    function void spend(String item, int value)
    begin
        myMoney = myMoney - value;
        System.out.println("*** spent $" +
                                value +
                                " on " + item +
                                ", leaving you with $" + myMoney);
        end
    end
    beginMain
        myMoney = 100;
        spend("aquarium",50);
        spend("shoes",100);
        spend("lipstick",20);
    endMain
end
```

Question 4.6.2: Change the myMoney class variable so that it is a double (short for double-precision floating point) rather than an int. You will need to add a new function money2string that converts double values into strings. For example the floating point number 1.2345 should be printed out as $1.23. If x is a double then the following expression converts x from a double into a number of dollars (int)x and the following expression converts x into a number of cents (int)(money * 100) - 100 * dollars. Note that you will need to make it so that $1.03 prints out as this value.

Question 4.6.3: Add an if statement to the spend function so that it uses System.out.println() to print out an error message if the person does not have enough funds in their bank account to pay for the item parameter.

Question 4.6.4: Add a new class variable double governmentsMoney and make it so that 12.5% of the cost of each item goes to the government in the form of G.S.T., which stands for Goods and Services Tax, a value-added tax.

Question 4.6.5: Add a new class variable numBattleships that records how many batteships are owned by the government. Write a function buyBattleShips that causes the government to buy as many battleships as it can afford. Make it so that the buyBattleShips function prints out how many battleships were purchased. Let the cost of each battleship be one million dollars and store this value in a variable called costOfShip. Please note that if the government's money is less the one million dollars then no battleships will be purchased.

Question 4.6.6: Set the initial value for governmentsMoney to be two millions dollars, then call the buyBattleShips function and verify that two battleships were purchased.

4.7 Tutorial 7 Non-Object arrays

This tutorial teaches you how to create single-dimensional and multi-dimensional arrays of non-Objects. The non-Object types in Java are those which aren't declared inside a class, so it includes the following types: boolean, char, int, float and double. A helpful convention in Java is that the non-object types start with a lowercase letter, while object types start with an uppercase letter, such as for example the String class as an example of an Object type. In addition to this, two different array initialization syntaxes are presented.

4.7.1 Single-dimensional non-Object arrays

Question 4.7.1: Here is an example of a convenient one dimensional array initialization syntax. Study, compile and run the following code. The code int[] should be read out loud as int array indicating the variable a is an int array, also known as an array of ints. Note that the first value of the for loop below is zero. This is because in J.T.W. and Java, the first index of an array is zero not one. This convention harks back to the old days of the C Programming Language and is used because it is more efficient in the low level of machine language than counting arrays from one.

```
beginMain
    var int[] a = { 1,2,3 };
    superfor (var int i=0 to a.length-1)
    begin
        System.out.println("a[" + i + "]=" + a[i]);
    end
endMain
```

Due to a design oversight by the creators of Java you cannot use this syntax to re-initialize an array like so:

```
a = { 4,5,6 }; // Compilation error
```

Luckily there is a way array around this oversight and that is to use a design pattern where you introduce a temporary variable like so:

```
var int[] temp = { 4,5,6 };
a = temp; // Array "a" now holds 4 5 6
```

Later you will learn why this design pattern is useful for re-initializing multi-dimensional arrays.

Question 4.7.2: Write a function print that takes an int array argument and prints out the array. You will need to use the length property of the array parameter so your function works with arbitrary sized arrays. Change the main function to what follows so that it contains a call to the print function.

```
var int[] a = { 1,2,3 };
print(a);
```

Question 4.7.3: Write a function with same name as the previous print function, except that this one should take an argument that is a double[], also known as a double array. Two functions with the same name in the same class is allowed in Java and the practice of using has a special name that is: function name overloading. Overloading is only allowed when the two functions with the same name have different parameters. When you call an overloaded function J.T.W. and Java looks at the number and types of the arguments a determines from this which of the overloaded functions to call. Change the main function to what follows so that it initializes an array of double-precision floating point variables and then calls the second print function.

```
var double[] b = { 1.1,2.2,3.3 };
print(b);
```

Here is an example of a second initialization syntax. For this particular example it is better to use the simpler, earlier initialization syntax, but when the size of the array to be created is to be determined at run-time, then the second syntax should used. The next question will show you an example of this.

```
beginMain
    var int[] a = new int[3]; )
    // at this point the array is all zeroes
    for (var int i=0; i<3; i=i+1)
    begin
        a[i] = i;
    end
    print(a);
endMain
```

Question 4.7.4: Write a function create takes one int argument, the size of the array to create and returns an int array of that size. Make it so the ith element of the array is initialized to i. Call this function from the main function like so:

```
beginMain
    var int[] a = create(3);
    print(a);
endMain
```

Question 4.7.5: Write a function create2 takes one int argument, the size of the array to create and returns a double array of that size. Make it so the ith element of the array is initialized to i.i. Why is it not possible to overload that create function? Try it and see what the compiler says. Call create2 from the main function like so:

```
beginMain
    var double[] a = create2(3);
    print(a);
endMain
```

Question 4.7.6: Write a function doubler that takes an int array x and returns a new int array result that is twice as big as x. Copy x into result before you return it. The extra elements in the result should all be zero.

Question 4.7.7: Change the doubler function so that every zero in the array result is set to the value 13.

4.7.2 Two dimensional non-Object arrays

Question 4.7.8: Here is an example of a convenient two dimensional array initialization syntax. Study, compile and run the following code. The code int[][] should be read out loud as int array array indicating that the variable a is an int array array, also known as a two-dimensional array of ints.

```
beginMain
    var int[][] a = { { 1,2,3 } { 4,5 } { 6 } }

    for (var int y=0; y<a.length; y=y+1)
    begin
        for (var int x=0; x<a[y].length; x=x+1)
        begin
            System.out.print(" " + a[y][x]);
        end
        System.out.println();
    end
endMain
```

Question 4.7.9: By copying the pattern of the code above, do some more overloading of the print function by writing two new print functions, one taking a two dimensional array of ints, the other taken a two dimensional array of doubles. The call both of these functions from the main function.

Note that if x is a two dimensional array of ints, then x[i] is a one dimensional array of ints for each in the range 0 ... x.length-1. Note that in the above code, a[0] is an array of three ints, a[1] is an array of two ints and a[2] is an array of one int. The reason these sub-arrays are all of different sizes is to save your computer's precious memory. For example you can have one sub-array much longer than all of the others without needing to allocate a whole bunch of memory that will go unused. Since a[0] is an int array, you would naively expect it to be able to be re-initialized like so:

```
a[0] = { 4,5,6,7};)
```

so that after this code a[0] holds the four element long array 4,5,6 and 7. But as mentioned above in [Single-dimensional non-Object arrays], page 25, this doesn't work because of a design oversight by the creators of Java. Luckily as mentioned above there is a way around this oversight and that is to use a temporary variable like so:

```
var int[] temp = { 4,5,6,7};
a[0] = temp;) // Array "a[0]" now holds 4 5 6 7
```

Like with one dimensional arrays, there is a second initialization syntax for two dimensional arrays and here it is. Unlike the above code the sub-arrays a[0], a[1] and a[2] are all of equal size, namely three.

```
var int[][] a = new int[3][3];
```

```
a[0][0] = 1; a[1][0] = 2; a[2][0] = 3;
a[0][1] = 4; a[1][1] = 5;
a[0][2] = 6;
```

Question 4.7.10: Write a function create3 and create4 that takes on int argument size and returns a two dimensional array of ints or doubles, respectively. Make is so that if a is the name of the returned array, then a[y][x] is set to the value of x+y.

4.7.3 Three-dimensional non-Object arrays

Question 4.7.11: Using the knowledge you have gained so far about arrays, create, initialize and print a three-dimensional array of ints.

4.8 Tutorial 8 Accessing functions and class variables from another class

Question 4.8.1: Study, compile and run the following code which resides in a file called Box.jtw. Notice the use of System.out.print() to print without a trailing newline and System.out.println() to print with a trailing newline. The ln part tells you this.

```
class Box
begin
    function void square(int n)
    begin
        superfor (var int y=0 to n-1)
        begin
            superfor (var int x=0 to n-1)
            begin
                if ((x == 0) or (x == n-1) or (y == 0) or (y == n-1))
                then System.out.print("#");
                else System.out.print(" ");
            end
            System.out.println();
        end
    end
    beginMain
        square(5);
    endMain
end
```

Notice that here is the output of the above code for different values of the n parameter:

```
n=1                 #

n=2                 ##
                    ##
```

```
n=3     ###
        # #
        ###

n=4     ####
        #  #
        #  #
        ####

n=5     #####
        #   #
        #   #
        #   #
        #####
```

Question 4.8.2: By copying the pattern established in the above code, write a now function square2 that generates the following output. Note that you will need to remove some of the or clauses in the square method above to get the following output:

```
n=1     #

n=2     ##
        ##

n=3     ###

        ###

n=4     ####

        ####

n=5     #####

        #####
```

Question 4.8.3: By copying the pattern established in the above code, write a now function square3 that generates the following output: n=1

```
n=1                 #

n=2                 ##
                    ##

n=3                 # #
                    # #
                    # #

n=4                 #  #
                    #  #
                    #  #
                    #  #

n=5                 #   #
                    #   #
                    #   #
                    #   #
                    #   #
```

Question 4.8.4: Study, compile and run the following code which resides in a file called Box.java:

```
class Box
begin
    function void x(int n)
    begin
        superfor (var int y=0 t0 n-1)
        begin
            superfor (var int x=0 to n-1)
            begin
                if ((x == y) or (x == n-1-y)) then System.out.print("#");
                else System.out.print(" ");
            end
            System.out.println();
        end
    end

    beginMain
        x(5);
```

```
        endMain
    end
```

Notice that here is the output of the above code for different values of the n parameter:

```
n=1                 #


n=2                 ##
                    ##


n=3                 # #
                     #
                    # #


n=4                 #  #
                     ##
                     ##
                    #  #


n=5                 #   #
                     # #
                      #
                     # #
                    #   #
```

Question 4.8.5: By copying the pattern established in the above code, write a now function x2 that generates the following output. Note that you will need to remove one of the or clauses in the x function above to get the following output:

```
n=1                 #


n=2                 #
                     #


n=3                 #
                     #
                      #
```

```
n=4                 #
                     #
                      #
                       #

n=5                 #
                     #
                      #
                       #
                        #
```

Question 4.8.6: By copying the pattern established in the above code, write a now function x3 that generates the following output. Note that you will need to remove one of the or clauses in the x method above to get the following output:

```
n=1                 #

n=2                  #
                    #

n=3                   #
                     #
                    #

n=4                    #
                      #
                     #
                    #

n=5                     #
                       #
                      #
                     #
                    #
```

Question 4.8.7: Study, compile and run the following code which resides in a file called Box.java:

```
class Box
begin
    function void triangle(int n)
```

```
        begin
            superfor (var int y=0 to n-1)
            begin
                superfor (var int x=0 to n-1)
                begin
                    if (x < y)
                    then System.out.print("#");
                    else System.out.print(" ");
                end
                System.out.println();
            end
        end
        beginMain
            triangle(5);
        endMain
    end
```

Notice that here is the output of the above code for different values of the n parameter:

n=1

```
#
```

n=2

```
#
##
```

n=3

```
#
##
###
```

n=4

```
#
##
###
####
```

n=5

```
#
##
###
####
#####
```

Question 4.8.8: By copying the pattern established in the above code, write a new function triangle2 that generates the following output. Note that you will need to change the if clause in the triangle method above to get the following output: n=1

```
n=1     #

n=2     ##
        #

n=3     ###
        ##
        #

n=4     ####
        ###
        ##
        #

n=5     #####
        ####
        ###
        ##
        #
```

Question 4.8.9: Write a new function called box that generates the following output. Note that you will need to modify the triangle method above to get the following output:

```
n=1     #

n=2     ##
        ##

n=3     ###
        ###
        ###

n=4     ####
        ####
        ####
        ####
```

n=5

```
#####
#####
#####
#####
#####
```

Question 4.8.10: Add the following code to Box.java:

```
class Grid
begin
    /* NOTE: the use of "final" below to denote a value whose value cannot be changed.
    final classVar int SIZE = 20;

    /* NOTE: the array below is a two-dimensional array */
    classVar boolean[][] array = new boolean[SIZE][SIZE];

    function void set(int x, int y, boolean v)
    begin
        if ((x>=0) and (x<SIZE) and (y>=0) and (y<SIZE)) then
        begin
            array[x][y] = v;
        end
    end

    function void print(int size)
    begin
        superfor (var int y=0 to size-1)
        begin
            superfor (var int x=0 to size-1)
            begin
                if (array[x][y])
                then System.out.print("#");
                else System.out.print(" ");
            end
            System.out.println();
        end
        System.out.println();) // prints an empty line between shapes
    end
end
```

Question 4.8.11: The following question will guide you through the process of making the drawing algorithm more powerful. Instead of printing the shapes directly to the screen, they will be stored in an array to be printed out only when the array has been completely set. You don't need to know a great deal about arrays to answer the remaining questions of this section as the array code has been written for you in the Grid class above. For every call to System.out.println() in Box.java, replace it with a call to the set method of the Grid class. Note that the third parameter in the set method is of type boolean, that is to say

it must be either true or false. To call a function of another class you need to prefix the name of the class like so: Grid.set(/* argument values */). Finally at the end of all of the functions in the Box class except for the main function you will need to call the Grid.print method of the Grid class to actually print out the array.

Question 4.8.12: Re-initialize the boolean array array named array from the main function of the Box class. HINT: to access a class variable from another class, you need to prefix it with the name of its class name, in this case it is Grid. Re-initialize the array variable to a two-dimensional array of dimensions 100 x 100. Also set the size variable to 100 so that the functions of the Grid class still work.

4.9 Tutorial 9 Mapping class variables to instance variables (also known as properties) and functions to methods

4.9.1 Elementary classes: using a single class for everything

For the purpose of the text that follows, O.O.P. stands for Object Oriented Programming.

Question 4.9.1: Study, compile and run the following code:

```
class PersonDriver1
begin
    classVar String homersName = "Homer Simpson";
    classVar int    homersAge  = 40; // Homer's age in years

    classVar String fredsName  = "Fred Flintstone";
    classVar int    fredsAge   = 45; // Fred's age in years

    classVar String darthsName = "Darth Vader";
    classVar int    darthsAge  = 55; // Darth's age in years

    function void growHomer()
    begin
        homersAge = homersAge + 1;
    end
    function void growFred()
    begin
        fredsAge = fredsAge + 1;
    end
    function void growDarth()
    begin
        darthsAge = darthsAge + 1;
    end

    function void knightHomer()
    begin
        homersName = "Sir " + homersName;
    end
    function void knightFred()
    begin
```

```
        fredsName = "Sir " + fredsName;
    end
    function void knightDarth()
    begin
        darthsName = "Sir " + darthsName;
    end

    function void printHomer()
    begin
        System.out.println("I am " + homersName + ", my age is " + homersAge);
    end
    function void printFred()
    begin
        System.out.println("I am " + fredsName + ", my age is " + fredsAge);
    end
    function void printDarth()
    begin
        System.out.println("I am " + darthsName + ", my age is " + darthsAge);
    end

    beginMain
        growHomer();
        knightHomer();
        printHomer();
        printFred();
        printDarth();
    endMain
end
```

Question 4.9.2: By copying the pattern established in the existing code write a some new class variables to represent a new person called Barak Obama. Note that he was born August 4, 1961 so at the time of writing this manual he is 54 years old.

Question 4.9.3: Then write some functions to work with this new person.

Question 4.9.4: Finally call those functions from the main function.

4.9.2 Improved classes: one object per class

As your program gets large (say over 1000 lines) then it becomes no longer practical to put all of your code in the same class. So it is natural to put each piece of related code in its own class. The J.T.W. programming language supports splitting a class into its constituent methods and having one file for each method. Simply use the include directive and J.T.W. will include the file for you like so:

include "a.method";

will include a method named a.

Question 4.9.5: Study, compile and run the following code: Each of these classes can be put in their own file. For each class X, this class can be put into a file called X.jtw. However for the purposes of this tutorial you will probably find it easier to merge all of the classes into the same file into a file called PersonDriver2.jtw

```
class Homer
begin
    classVar String name = "Homer Simpson";
    classVar int    age  = 40;) // Homer's age in years

    function void grow()
    begin
        age = age + 1;
    end
    function void knight()
    begin
        name = "Sir " + name;
    end
    function void print()
    begin
        System.out.println("I am " + name + ", my age is " + age);
    end
end

class Fred
begin
    classVar String name = "Fred Flintstone";
    classVar int    age  = 45;) // Fred's age in years

    function void grow()
    begin
        age = age + 1;
    end
    function void knight()
    begin
        name = "Sir " + name;
    end
    function void print()
    begin
        System.out.println("I am " + name + ", my age is " + age);
    end
end

class Darth
begin
    classVar String name = "Darth Vader";
    classVar int    age  = 55;) // Darth's age in years

    function void grow()
    begin
        age = age + 1;
    end
```

```
    function void knight()
    begin
        name = "Sir " + name;
    end
    function void print()
    begin
        System.out.println("I am " + name + ", my age is " + age);
    end
end

class PersonDriver2
begin
    beginMain
        Homer.grow();
        Fred.knight();
        Homer.print();
        Fred.print();
        Darth.print();
    endMain
end
```

Question 4.9.6: By copying the pattern established in the existing code write a new class to represent Barak Obama.

Question 4.9.7: Call the functions from the main function of the driver class.

4.9.3 True O.O.P.: more than one object per class

To allow for more than one object per class, most if not all class variables needs to be made into what are called instance variables (or more simply and more commonly known as properties) and most if not all functions need to be made into what are called methods.

Question 4.9.8: Study, compile and run the following code:

```
class Person
begin )
    /*
    * NOTE: the use of the "property" keyword here instead of the
    * "classVar" keyword.
    */
    property String name; // Person's full name
    property int    age;  // Person's age in years

    /*
    * NOTE: the use of the "method" keyword here instead of the
    * "function" keyword.
    */
    method void grow()
    begin
        age = age + 1;
    end
```

```
    method void knight()
    begin
        name = "Sir " + name;
    end

    method void print()
    begin
        System.out.println("I am " + name + ", my age is " + age);
    end

    beginMain

        var Person h = new Person();
        h.name = "Homer Simpson";
        h.age  = 40;

        var Person f = new Person();
        f.name = "Fred Flintstone";
        f.age  = 45;

        var Person d = new Person();
        d.name = "Darth Vader";
        d.age  = 55;

        h.grow();
        h.knight();
        h.print();
        f.print();
        d.print();
    endMain
end
```

In the above code, note the use of three references h, f and d.

Question 4.9.9: By copying the pattern established in the existing code add some code to the main function add some code to create a new person for Barak Obama.

4.9.4 A common design pattern: private properties,public constructor and public getters

A common design pattern in Java and one that I present for you in the following code is to make all of the properties of a class effectively read-only to all client classes by making all of the properties private and providing non-private getter methods for getting the values of the properties. It is possible for the original class to change the values of the properties but other classes (such as PersonTest below) are not capable of doing this, without calling a method of the original class such the grow and knight methods of the Person class. Finally an additional thing known as a constructor is used to ensure that objects are initialized with meaningful values for their properties.

Question 4.9.10: Study, compile and run the following code:

```
class Person
begin

    private property String name;
    private property int    age; // Age in years
)
    //
    // NOTE: Getter methods
    //
    public method String getName()
    begin
        return name;
    end

    public method int getAge()
    begin
        return age;
    end

    public constructor Person(String aName, int anAge)
    begin
        this.name = aName;
        this.age  = anAge;
    end

    public method void grow()
    begin
        age = age + 1;
    end

    public method void knight()
    begin
        name = "Sir " + name;
    end

    public method void print()
    begin
        System.out.println("I am " + name + ", my age is " + age);
    end
end

class PersonDriver3
begin
    beginMain
        //
```

```
        // NOTE: In the following constructor calls the age and name are set by the co
        //
        var Person h = new Person("Homer Simpson",40);
        var Person f = new Person("Fred Flintstone",45);
        var Person d = new Person("Darth Vader",55);

        h.grow();
        h.knight();
        h.print();
        f.print();
        d.print();

        h.name = "Luke Skywalker";)                // ERROR: name is private
        h.age = h.age + 1;                         // ERROR: age is private

        System.out.println("name=" + h.name);      // ERROR: name is private
        System.out.println("age="  + h.age);       // ERROR: age is private

        System.out.println("name=" + h.getName()); // OK: getter is non-private
        System.out.println("age="  + h.getAge());  // OK: getter is non-private
    endMain
end
```

Note that you will have to remove the error lines from the above file for the code to compile.

Question 4.9.11: By copying the pattern established in the existing code add some code to the main function to create a new person called Hillary Clinton. Hillary Clinton was born on October 26, 1947 so at the time of writing this manual she was 68 years old

4.9.5 Comparing strings

Question 4.9.12: Add a method unknight() which removes the "Sir " title if he has one. One trap for young players in J.T.W. or Java is to use the operator == to compare strings like so:

```
function boolean myCompare(String a, String b)
begin
    return a == b;) // Works but not as expected!
end
```

It compiles without error, but doesn't give you the result you were expecting. Instead you need to use the equals method of the String class like so:

```
function boolean myCompare(String a, String b)
begin
    return a.equals(b);
end
```

More generally, if x and y are a references to objects, then x == y returns whether or not x and y are pointing to the same object, whereas x.equals(y) returns whether or not the contents of the objects referred to by x and y are equal. The meaning of the word contents

varies from class to class, but in the case of strings it means that the strings contain the same data.

You will also find the String class' substring and (toUpperCase or toLowerCase) methods useful here too. See the class String in the package java.lang at http://docs.oracle.com/javase/1.5.0/docs/api for more details of these two methods.

4.9.6 The null value for references

As soon as you learn how to use references you need to know that all reference variables could conceivably hold the value null, meaning no value. In particular when properties are themselves references as you will discover in Tutorial 11, then those properties are initialized to null by default. Object arrays that you will learn about in Tutorial 10 using the second of two initialization syntaxes are also initialized to null by default.

4.9.7 Why the toString method is better than any other method or

property for debugging

If x is a reference to a class X (including this for the current class) and if m is a method of X and p is a property of X, and if x is currently null, then the following lines result in a NullPointerException being thrown when executed:

```
x.p;
x.m();
```

whereas if x is null then

```
System.out.println(x); and
System.out.println("x=" + x);
```

prints out, respectively:

```
null, and
x=null.
```

If x is not null, it calls

```
System.out.println(x.toString());
System.out.println("x=" + x.toString());
```

so these expressions are safer to use than any other method or property in situations where x might be null. The syntax of the toString method is as follows:

```
public method String toString()
begin
    // Code goes here...
end
```

Importantly for reasons which will be explained later the toString method must be declared with public visibility. For other properties and methods to be used safely with null references you need to wrap a conditional if construct around the calling of the method or property like so for properties:

```
if (x != null)
then begin
    System.out.println(x.p);
end
```

or like so for methods:

```
if (x != null)
then begin
    System.out.println(x.m());
end
```

Therefore the toString method is more convenient than any other method or property.

Question 4.9.13: Change the print method above from a method that prints out to the screen to a method called toString that returns a String.

Question 4.9.14: Call the toString method instead of the print methods in the main function.

4.10 Tutorial 10 Object arrays

This tutorial teaches you how to create single-dimensional and multi-dimensional arrays of Objects. The Object types are all types execept for boolean, char, int, float and double. A helpful convention in Java is that the Object types start with an uppercase letter, while non-Object types start with a lowercase letter, such as for example the String class as an example of an Object type. In addition to this, two different array initialization syntaxes are presented.

4.10.1 Single-dimensional arrays of Objects

Question 4.10.1: Here is an example of a convenient one-dimensional array initialization syntax. Study, compile and run the following code. The code Person[] should be read out loud as "person array" indicating the variable a is a person array, also known as an "array of persons".

```
class Person
begin
    private property String name;

    constructor Person(String aName)
    begin
        name = aName;
    end

    public String toString()
    begin
        return name;
    end
end

class PersonTest
begin
    beginMain
        var Person[] a = { new Person("Person # 1"), new Person("Person # 2"), new Per
        superfor (var int i=0 to a.length-1)
        begin
```

```
            System.out.println("a[" + i + "]=" + a[i]);
        end
    endMain
end
```

Due to a design oversight by the creators of Java you cannot use this syntax to re-initialize an array like so:

```
// Compilation error
a = { new Person("Person # 4"), new Person("Person # 5"), new Person("Person # 6"), ne
```

Luckily there is a way array around this oversight and that is to use a design pattern where you introduce a temporary variable like so:

```
// No error
var Person[] temp = { new Person("Person # 4"), new Person("Person # 5"), new Person("
a = temp; // Array "a" now holds Person # 4,Person # 5,Person # 6,Person # 7
```

Later you will learn why this design pattern is useful for re-initializing multi-dimensional arrays.

Question 4.10.2: Write a function in the class PersonTest called print that takes a Person array argument and prints out the array. You will need to use the length property of the array parameter so your function works with arbitrary sized arrays. Change the main function to what follows so that it contains a call to the printx function.

```
var Person[] a = { new Person("Person # 1"), new Person("Person # 2"), new Person("Per
print(a);
```

Question 4.10.3: Write your own class called Mine similar to the Person class with a one int parameter constructor, a private int property p and a toString method that converts p to a string. Then write a function in the PersonTest class with same name as the previous print function, except that this one takes a Mine[], also known as a Mine array. You might recall from Tutorial 7 that this practice of having two functions with the same name is called function name overloading. Change the main function to what follows so that it initializes an array of Mine point variables and then calls the second print function.

```
var Mine[] b = { new Mine(1), new Mine(2), new Mine(3) };
print(b);
```

Here is an example of a second initialisation syntax. For this particular example it is better to use the simpler, earlier initialisation syntax, but when the size of the array to be created is to be determined at run-time, then the second syntax should used. The next question will show you an example of this.

```
beginMain
    var Person[] a = new Person[3];
    // at this point the array is all nulls
    superfor (var int i=0 to a.length-1)
    begin
        a[i] = new Person("Person # " + (i+1));
    end
    print(a);
endMain
```

Question 4.10.4: Write a function create takes one int argument, the size of the array to create and returns a Person array of that size. Make it so the ith element of the array is initialised to "Person # " + i. Call this function from the main function like so:

```
beginMain
    var Person[] a = create(3);
    print(a);
endMain
```

Question 4.10.5: Write a function create2 takes one int argument, the size of the array to create and returns a Mine array of that size. Make it so the ith element of the array's toString method prints out "Mine # " + i. Why is it not possible to overload that create function? Try it and see what the compiler says. Call create2 from the main function like so:

```
beginMain
    var Mine[] a = create2(3);
    print(a);
endMain
```

Question 4.10.6: Write a function doubler that takes a Person array x and returns a new Person array called result twice as big as x. Copy x into the result before you return it. The extra elements in result should all be null.

Question 4.10.7: Change the doubler function so that every null in the array result is set to a new Person make it so that every new Person Object has a different name property.

4.10.2 Two-dimensional arrays of Objects

Question 4.10.8: Here is an example of a convenient two dimensional array initialization syntax. Study, compile and run the following code. The code Person[][] should be read out loud as person array array indicating the variable a is a person array array, also known as a two-dimensional array of persons.

```
beginMain
    var Person[][] a = { { new Person("Person # 1"), new Person("Person # 2"), new Per
                           { new Person("Person # 4"), new Person("Person # 5") },
                           { new Person("Person # 6") } };

    superfor (var int y=0 to a.length-1)
    begin
        superfor (var int x=0; to a[y].length-1)
        begin
            System.out.print(" " + a[y][x]);
        end
        System.out.println();
    end
endMain
```

Question 4.10.9: By copying the pattern of the code above, do some more overloading of the print function by writing two new print functions, one taking a two dimensional array of Person, the other taken a two dimensional array of Mine. The call both of these functions from the main function.

Since a[0] is a Person array,you would naively expect it to be able to be re-initialised like so:

```
a[0] = { new Person("Person # 4"),
            new Person("Person # 5"),
            new Person("Person # 6") };
```

so that after this code a0 holds the four element long array Person #4,Person #5 and Person #6, but it does't work owing to a design oversight by the creators of Java. Luckily as mentioned above there is a way around this oversight and that is to use a temporary variable like so:

```
var Person[] temp = { new Person("Person # 4"),
                                 new Person("Person # 5"),
                                 new Person("Person # 6") };
a[0] = temp; // Array "a[0]" now holds Person # 4,Person # 5,Person # 6
```

Like with one-dimensional arrays, there is a second initialization syntax for two-dimensional arrays and here it is. Unlike the above code the sub-arrays a[0],a[1] and a[2] are all of equal size, namely three.

```
beginMain
    var Person[][] a = new Person[3][3];
    a[0][0] = new Person("Person # 1");
    a[0][1] = new Person("Person # 2");
    a[0][2] = new Person("Person # 3");
    a[1][0] = new Person("Person # 4");
    a[1][1] = new Person("Person # 5");
    a[1][2] = new Person("Person # 6");
    a[2][0] = new Person("Person # 7");
    a[2][1] = new Person("Person # 8");
    a[2][2] = new Person("Person # 9");
endMain
```

Question 4.10.10: Write a function create3 and create4 that takes an int argument size and returns a two-dimensional array of Person or Mine, respectively. Make is so that each Person or Mine Object has its own number, using a separate counter variable var int count.

4.10.3 Three-dimensional arrays of Objects

Question 4.10.11: Using the knowledge you have gained so far about arrays, create, initialize and print a three-dimensional array of Persons. Make it so that each Person Object is given its own number using a separate counter variable var int count.

4.11 Tutorial 11 References to another class

The following code presents example involving three classes Flea, Dog and DogOwner to represent the idea that a dog has a flea and a dog-owner has a dog. The class DogTest is the driver class. The key concept of this tutorial is that classes can have references of objects of another class in order to set up a relationship between the two classes.

Question 4.11.1: Study the following code and find the two bugs in it. Fix the bugs and then compile and run it to verify that it prints out "p=I am a flea called Pop".

```
class Flea
```

```
begin
    property String name;

    constructor Flea(String aName)
    begin
        aName = name;
    end

    public method String toString()
    begin
        return "I am a flea called " + name;
    end
end

class Dog
begin
    property String name;
    property int age; // Age in years
    property Flea dogsFlea;

    constructor Turtle(String aName, int anAge, Flea aFlea)
    begin
        name     = aName;
        age      = anAge;
        dogsFlea = aFlea;
    end
end

class DogTest
begin
    beginMain
        Flea p = new Flea("Pop");
        Flea s = new Flea("Squeak");
        Flea z = new Flea("Zip");
        System.out.println("p=" + p);
    endMain
end
```

Question 4.11.2: In the main function of the DogTest class, write code to call the toString method for the fleas referenced by s and z.

Question 4.11.3: In the main function of the DogTest class, write code to construct three dogs called "Fido", "Jimbo" and "Rex". For the purposes of the rest of these questions, let the name of the references for Fido, Jimbo and Rex be f, j and r. Note that the third parameter to the Dog class is of type Flea. Therefore you will need to supply a Flea reference for each dog. Make it so that Fido has a flea called Pop, Jimbo has a flea called Squeak, and Rex has a flea called Zip.

HINT: If the flea called Pop is referenced by the variable name p, then this reference should appear as the third argument in one of the calls to the Dog constructor.

Question 4.11.4: Write a toString method in the Dog class that works like the toString method in the Flea class. Then call this method from the main function to print out the full statistics of the three dogs that you have just created in Question 11.3.

Question 4.11.5: By copying the pattern of the Flea and Dog classes, write a class DogOwner that has three non-private properties: name, salary and ownersDog. Also write a three-parameter constructor for the DogOwner class that sets these properties.

Question 4.11.6: Add some code into the main function to construct three dog owners called Angus, Brian and Charles. Make it so that Angus has a dog called Rex, Brian has a dog called Jimbo, and Charles has a dog called Fido. For the purposes of the rest of these questions, let the name of the references for Angus, Brian and Charles be (respectively) a, b and c. Use the Dog references that you created in Question 11.3 to achieve this. Make it so that Angus, Brian and Charles have initial salaries of 10,000, 20,000 and 30,000.

Question 4.11.7: Without changing the call to the DogOwner constructor, change the value of the salary property of object referenced by a to 1,000,000. Note that since the salary property of the DogOwner class is non-private you should be able to set the value of the salary property from the main function of DogTest.

Question 4.11.8: Write a toString method for the class DogOwner and add some code to the main function to call it for Angus, Brian and Charles.

Question 4.11.9: What is the value of: a.ownersDog.dogsFlea.toString()? Add some code to the main function to find out if it does what you think it should do.

4.12 Tutorial 12 Overloading methods

Question 4.12.1: Write constructors for the classes SportsShoe and Runner below, by looking at the main function to see how many arguments each constructor has.

```
class SportsShoe
begin

    property String model;      // what kind of shoe it is
    property double speedBoost; // the boosting factor of the shoe

    // constructor goes here:

    // Useful method for debugging
    public method String toString()
    begin
        return "I am a shoe called " + model + " and my boosting factor is " + speedBo
    end
end

class Runner
begin
    private property String     name;  // Runner's name.
    private property int        speed; // speed of runner in km/hr.
```

```
    private property SportsShoe shoes; // which shoe they are wearing.

    // constructor goes here:

    // Useful method for debugging
    public method String toString()
    begin
        return "I am a runner and my name is " + name + " and my shoes are " + shoes;
    end
    /**
    *** This private method computeSpeed works out the runners speed,
    *** based on their basic speed and the speed boost due to the
    *** SportsShoe that they are currently wearing.
    */

    // method goes here:


    /**
    ** Prints the result of racing two runners against each other.
    */
    function void race(Runner r1, Runner r2)
    begin
        if (r1.computeSpeed()>r2.computeSpeed())
        then begin
            System.out.println("Runner " + r1.name + " beats " + r2.name);
        end
        else begin
            System.out.println("Runner " + r2.name + " beats " + r1.name);
        end
    end
    /**
    ** Swaps the shoes of two runners.
    */
    function void swapShoes(Runner r1, Runner r2)
    begin
        var SportsShoe tempShoe = r1.shoes;
        r1.shoes = r2.shoes;
        r2.shoes = tempShoe;
    end
end

class RunnerTest
begin
    beginMain
        var SportsShoe nike    = new SportsShoe("Nike NX-71", 2.0);
        var SportsShoe reebock = new SportsShoe("Reebock R20", 2.3);
```

```
            var SportsShoe puma    = new SportsShoe("Puma P200-MMX",4.8);

            var Runner sg = new Runner("Speedy Gonzalez", 55, nike);
            var Runner sw = new Runner("Slick Willy", 49, reebock);
            var Runner fa = new Runner("Fat Albert", 15, puma);

            Runner.race(sg,sw);
            // Runner.race(sg,sw,fa);
            // sg.raceAgainst(sw);
        endMain
    end
```

Question 4.12.2: In the Runner class, write the private method computeSpeed that has no arguments and returns a double-precision floating point value that equals the runner's running speed. Note that the speed of a runner is determined by multiplying their speed property with the speedBoost property of the shoes that they are wearing. For example, Speedy Gonzalez's running speed = 55 * 2.0 = 110.0.

Question 4.12.3: Fix the race method so that it checks for a draw.

Question 4.12.4: By copying the race method, write a three-parameter race method for racing three runners against each other. Two methods in the same class with the same name is called overloading in J.T.W. and Java. Add a call to this method from the main function.

Question 4.12.5: What is the difference between a method and a function? Write a one parameter method raceAgainst that behaves exactly like two-parameter function race. There are two ways of doing this, one is to optionally use the this keyword rather than one of the parameters r1 or r2. The second way is for race to simply call race using this as one of the arguments to the function.

Question 4.12.6: Is it true that any method can be re-worked into a function and vice versa?

Question 4.12.7: The swapShoes method in the Runner class swaps the shoes of two runners. Add some code to the main function to swap the shoes of two runners and verify that the shoes do indeed get swapped.

Question 4.12.8: Write a method called swapNames that swaps the names of two runners. You can put this function into any class but it makes the most sense to put it into the Runner class since it has two Runner parameters.

Question 4.12.9: Write a method swapSpeeds that swaps the speed properties of two runners.

4.13 Tutorial 13 More about references

Question 4.13.1: Study, compile and run the following code:

```
class Car
begin
    /**
    * Car's model name
    */
```

```
    property String model;

    /**
    * Car's value in dollars
    */
    property int value;

    /**
    * Car's serial number
    */
    property int serialNumber;

    /**
    *   Global serial number counter
    */
    private classVar int serialCounter = 1000;

    constructor Car(String aModel, int aValue)
    begin
        model = aModel;
        value = aValue;
        serialNumber = serialCounter;
        serialCounter = serialCounter + 1;
    end

    public method String toString()
    begin
        return "I am a car, model=" + model + ", value=" + value +
                 ", serial number=" + serialNumber;
    end
end

class Owner
begin

    /**
    * Owner's full name
    */
    property String name;

    /**
    * Owner's money in dollars
    */
    property int money;

    /**
    * Owner's car
```

```
        */
        property Car ownersCar;

        constructor Owner(String aName, int aMoney, Car aCar)
        begin
            name = aName;
            money = aMoney;
            ownersCar = aCar;
        end

        public method String toString()
        begin
            return "I am a car owner, name=" + name + ", money=" + money +
                     ", car=" + ownersCar;
        end

        public method void describe()
        begin
            System.out.println(toString());
        end
    end

    class CarTest
    begin
        beginMain
            var Car   ford = new Car("Ford Escort",1000);
            var Car   nissan = new Car("Nissan Nivara",2000);
            var Owner joe = new Owner("Joe Bloggs",500,ford);
           ) // Mary has no car to start with.
            var Owner mary = new Owner("Mary Smith",600,null);
            joe.describe();
        endMain
    end
```

Question 4.13.2: What is the purpose of the class variable serialCounter?

Question 4.13.3: Write a method sellCar that increases the owner's money by half the value of their car and the owner's car reference gets set to null, for no car. If the owner owns no car (null) simply do nothing.

Question 4.13.4: Write a method in the Owner class called purchase so that:

```
Car newCar = new Car("Mini Cooper",1000);
joe.purchase(newCar);
```

results in Joe's money going down by $1000 and Joe's car being set to newCar. Before Joe purchases their new car, call the sellCar method so that Joe sells his current car before

Question 4.13.5: Write a function in the Owner class called netWorth so that:

```
System.out.println("Joe's net worth = " + joe.netWorth());
```

prints out Joes' money plus the value of his car, if he has a car.

Question 4.13.6: Write a method in the Owner class called smashCar so that:

```
mary.smashCar();
```

halves the value of Mary's car.

Question 4.13.7: Write a method in the Owner class called stealCarFrom so that:

```
mary.stealCarFrom(joe);
```

results in Mary selling his current car (if he has one) for its market value and Mary acquiring ownership of Joe's car. Also call Mary's sellCar method so that Mary sells her current car before stealing Joe's car.

Question 4.13.8: Write a function in the Owner class called swapMoney so that:

```
Owner.swapMoney(joe,mary);
```

swaps the money of Joe and Mary.

Question 4.13.9: Write a function in the Owner class called swapCars so that:

```
Owner.swapCars(joe,mary);
```

swaps the cars of Joe and Mary.

Question 4.13.10: Write a function in the Car class called swapSerialNumbers so that:

```
Car.swapSerialNumbers(ford,nissan);
```

swaps the serial numbers of ford and nissan.

Question 4.13.11: Write a function in the Owner class called sellCarTo so that

```
joe.sellCarTo(mary);
```

results in Joe's money going up by the value of his car and Mary's money going down by the value of his car, and the ownership of Mary's car gets transferred to Joe.

4.14 Tutorial 14 Linked lists

Dr Seuss' story "Yertle the Turtle" (https: / / en . wikipedia . org / wiki / Yertle_the_Turtle) describes how a turtle called Yertle sits at the top of a pile of other turtles. In this example, the pile of turtles is represented by a linked list of Turtle objects, with the down property serving to connect one Turtle object to another. If a Turtle object has a non-null down property, then this represents the fact that it is sitting on top of another turtle. The last turtle in the linked list is the turtle that is at the bottom of the pile, which has a null value for its down property. Note that you cannot use the superfor construct for iterating through a linked list. In this case the for construct is the most sensible.

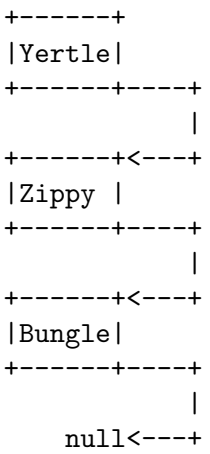
Question 4.14.1: Study, compile and run the following code:

```
class Turtle
begin

    private property String     name;   // Turtle's name
    private property int        age;    // Turtle's age in years
    private property double     weight; // Turtle's weight in kg

    // NOTE: this property allows for linked lists
    property Turtle down;
```

```
    constructor Turtle(String aName, int anAge, double aWeight)
    begin
        name   = aName;
        age    = anAge;
        weight = aWeight;
    end

    /** Getter method for name property */
    method String getName()
    begin
        return name;
    end

    /** Useful method for debugging */
    public method String toString()
    begin
        return name;
    end
end

class TurtleTest
begin
    beginMain

        var Turtle yertle = new Turtle("Yertle", 103, 20);
        var Turtle zippy  = new Turtle("Zippy",  102, 30);
        var Turtle bungle = new Turtle("Bungle", 101, 40);

        // *** see later
        yertle.down = zippy;
        zippy.down = bungle;
        bungle.down  = null; // NOTE: not needed as bungle.down is null by default

        var int totalWeight = 0;
        // NOTE: demonstrates how to iterate through a linked list:
        for (var Turtle current = yertle; current != null; current=current.down)
        begin
            totalWeight = totalWeight + current.getWeight();
        end
        System.out.println("The total weight is " + totalWeight);
    endMain
end
```

The code in the main function after the *** sets down the following relationships between the three Turtle objects (Yertle, Bungle and Zippy). The following diagram shows the relationship between the different turtles. When you traverse the list of turtles you must

always start at the top turtle (known as the head of the linked list). If you give a different value for the top turtle, your code will think that the given turtle is the one at the top of the pile and you will get the wrong result.



Question 4.14.2: Move the code for calculating the total weight of the turtles from the main function to a function called function void printTotalWeight(Turtle bottom) in the Turtle class that prints out the total weight of the turtles. Then call that function from the main function to get the same result as before. Note that that if printTotalWeight was a method then calling that method using null (representing an empty list) like so: null.printTotalWeight() would be an error, whereas Turtle.printTotalWeight(null) wouldn't be and therefore is better. This is one example of how methods and functions differ.

Question 4.14.3: Revision question for getters. By copying the pattern established by the getName method, add two getter methods to the Turtle class: getAge which returns the current turtle's age and getWeight which returns the current turtle's weight. Then call these methods on the Yertle object in the main function. Note that the toString method would be more appropriate as it handles nulls better but you know that the yertle reference is not null so you know it is safe to call the getAge and getWeight methods on the yertle reference.

Question 4.14.4: Write a function Turtle findBottomTurtle(Turtle top) that returns the Turtle object that is at the top of the pile, and returns null if there isn't one.

Question 4.14.5: Then call this function from the main function using System.out.println() and the top turtle Yertle.

Question 4.14.6: Write a function Turtle findOldestTurtle(Turtle top) that returns the oldest turtle or null if there isn't one.

Question 4.14.7:Then call this function from the main function using System.out.println() and the top turtle Yertle.

Question 4.14.8: Write a function Turtle findHeaviestTurtle(Turtle top) returns the heaviest turtle, or null if there isn't one.

Question 4.14.9: Then call this function from the main function using System.out.println() and the top turtle Yertle.

Question 4.14.10: Write a function sayPile(Turtle top) that prints the names of the turtles in the pile starting from the top turtle and finishing at the bottom turtle. Then call this function from the main function.

Question 4.14.11: Under what circumstances would it be okay to change the visibility of the down property to private, like the name, age and weight properties?

Question 4.14.12: Add an extra parameter to the constructor which is a reference the to the turtle on below the current one. Then remove all occurrences of the down property from the main function. Note that you will need to reverse the order that the turtles are created so the bottom turtle is constructed first and so on. The advantage of this is that it enables you to change the visibility of the down property to private.

4.15 Tutorial 15 Introducing inheritance

4.15.1 Basic Inheritance

When you see the following code: class X extends Y, it means that class X inherits from the class Y. Class X is called the subclass and the class Y is called the super-class or sometimes the parent class. When the class X extends from Y, it pulls in all of the non-private methods and properties from the super-class Y. Inherited methods can override the behaviour of that same method in the super-class to give behaviour that is specific to the subclass. The concept of methods overriding other methods is called dynamic method binding or more commonly the more impressive-sounding name: polymorphism. The main thing that this tutorial shows is the idea that inheritance is a non-symmetrical relationship. For example: in the code that follows, the Bird class inherits from the Animal class, which corresponds to the idea that every bird is an animal. The reverse, every animal is a bird is plainly not true! Inheritance forces you to recognize this.

Question 4.15.1: Study, compile and run the following code. The following code shows how inheritance works. In the following code, the Bird class inherits from the Animal class. The Bird class pulls in the Animal class's age property and the canFly and talk methods. Importantly the canFly property overrides the behaviour of the canFly method of the parent Animal class, which reflects that fact that generally speaking, birds can fly. In the code that follows, note that int properties are initialized to zero by default and the super method (also known as the constructor of the super-class) is called by default if there is a zero parameter constructor in the super-class, which there is by default, even if you don't write one!

```
class Animal
begin

    property int age;    // Animal's age in years
    property int health; // Animal's health in hit points

    constructor Animal()
    begin
        age    = 0; // NOTE: not needed as set by default
        health = 100;
    end
```

```
    method boolean canFly()
    begin
        return false;
    end

    method void talk()
    begin
        System.out.println("Hello");
    end
end

class Bird extends Animal
begin

    property double flySpeed;) // Bird's speed in km/h

    constructor Bird()
    begin
        super();      // NOTE: not needed as called by default
        flySpeed = 0; // NOTE: not needed as set by default
    end

    method boolean canFly()
    begin
        return true;
    end

    method void peck()
    begin
        System.out.println("peck");
    end
end

class InheriTest
begin
    beginMain
        var Bird eagle = new Bird();
        eagle.talk();
        eagle.peck();
    endMain
end
```

Question 4.15.2: Override the talk method of the Animal class in the Bird class to print out "Tweet Tweet!" rather than "hello" to give more accurate talking of bird objects.

Question 4.15.3: By copying the pattern established in the Bird class, change the eagle from an instance of the Bird class to its own class in its own right and then create an instance of that class in the main function of InheriTest. Your Eagle class should have one

property: int numberOfKills and one method: void attack() that internally increments the value of numberOfKills. In the main function you should call every method of the Eagle class and its super-classes.

Question 4.15.4: What is the advantage of using a new separate class to represent a new object rather than using an instance of an existing class?

Question 4.15.5: Create a new class Kiwi that inherits from the Bird class. Your Kiwi class should override the canFly method to return false, which reflects the fact that generally speaking birds can fly, but the kiwi bird in particular does not fly. Your Kiwi class have a property numberOfWorms. Once you have written the Kiwi class you should create an instance of the Kiwi class in the main function.

Question 4.15.6: Why does the following line of code in the main function print out 100 but there is no setting of that variable to that value in the Kiwi class?

```
System.out.println(k.health);
```

Question 4.15.7: In the classes Animal, Bird, Eagle and Kiwi, remove all of the canFly methods and replace it with a single canFly property of the Animal class. In the constructors you will need to set the value of the canFly property to a value that is appropriate for that class. For example in the Bird class's constructor you should set the canFly property to true, while in the Kiwi class's constructor you should set the canFly property to false.

Question 4.15.8: What is the advantage of having a canFly property over a bunch of canFly methods?

There is an equally valid alternative to having a public property in the Animal class and that is to have in the Animal class a private property canFly and a pair of methods for getting and setting the value of the canFly property like so. These methods in J.T.W. and Java are called getter methods and setter methods since, as their names suggest, getters are used for getting the value of something and setters are used for setting the value of something. Nore that the canFly method of the code above corresponds to getCanFly method in the code below.

```
private property boolean canFly;

method boolean getCanFly()
begin
    return canFly;
end

method void setCanFly(boolean aCanFly)
begin
    canFly = aCanFly;
end
```

You might think that it is simpler to have one thing (a single non-private property) rather than three things (a private property and a non-private getter method and a non-private setter method) and you would be right. However from the point of view of the client code that uses the Animal class, the two approaches are identical. Later on when you learn more you will understand under what circumstances the second getter and setter approach is better.

Question 4.15.9: Change the main function to what follows:

```
beginMain
    var Bird b = new Bird(10);
    var Animal a = b;
    a.talk();
    a.peck();
endMain
```

When you compile this code it gives a compilation error. What line gives the error and what is the reason for the error?

Question 4.15.10: Change the main function to what follows:

```
beginMain
    var Animal a = new Animal();
    var Bird b = a;
    b.talk();
    b.peck();
endMain
```

When you compile this code it gives a compilation error. What line gives the error and what is the reason for the error?

4.15.2 Run-time type inquiry

In J.T.W. and Java there is a keyword called instanceof that does a run-time check on the type of an object. The following function:

```
function void say(Animal a)
begin
    System.out.println(a instanceof Bird);
end
```

uses the instanceof keyword to determine the run-time type of the reference a and prints out whether or not the reference is referring to a Bird object. Some examples should clarify the situation:

- say(new Bird()) prints true, Since the parameter a is pointing to a bird object at run-time,
- say(new Animal()) prints false since not every animal is a bird,
- say(new Eagle()) prints true, since every eagle is a bird, and
- say(new Kiwi()) prints true, since every kiwi is a bird.
- var Animal a = new Animal(); say(a); prints false since at run-time a is not pointing to a bird object
- var Animal a = new Bird(); say(a); prints true since at run-time a is pointing to a bird object.

In Tutorial 17 you will learn why in most cases it is better to use polymorphism instead of the instanceof keyword for run-time type enquiry.

4.15.3 The superclass of all objects

Every class in Java inherits either directly or indirectly from a class called Object. That is to say if x is a reference variable, then the run-time expression x instanceof Object is always

true except for the pathological case where x is null (i.e. is currently pointing to no object). The Object class contains a method called toString that returns a string containing the run-time class name of the object concatenated with the hash code of the memory address of the object in base 16 (also known as hexadecimal) format. Since every class inherits from Object, every object can have toString invoked upon it. Even better, every class X can override toString to provide debugging information that is tailored to X. Therefore the toString method is convenient for debugging. Since the toString method is a public method of the Object class it must be overridden as a public method, since your overridden function cannot have weaker access privileges.

4.16 Tutorial 16 More inheritance

This tutorial shows you a practical example of inheritance. The file StarWars.jtw is comprised of three classes: XWing, TieFighter and StarWars. The first two represent spacecraft from the two sides of the Star Wars films. The class StarWars is the driver class and contains code for executing a battle between the X-Wings and the Tie Fighters.

Question 4.16.1: Study, compile and run the following code:

```
class XWing
begin

    private property int     shields;
    private property int     weapon;
    private property boolean dead;

    constructor XWing()
    begin
        shields = 1000;
        weapon  = 10;
    end

    method int getWeapon()
    begin
        return weapon;
    end
    method boolean isDead()
    begin
        return dead;
    end
    method void hit(int damage)
    begin
        shields = shields - damage;
        if (shields<0)
        then begin
            System.out.println("BOOM!!!");
            dead = true;
        end
    end
```

```
end

class TieFighter
begin

    private property int     shields;
    private property int     weapon;
    private property boolean dead;

    constructor TieFighter()
    begin
        shields = 500;
        weapon  = 20;
    end

    method int getWeapon()
    begin
        return weapon;
    end
    method boolean isDead()
    begin
        return dead;
    end
    method void hit(int damage)
    begin
        shields = shields - damage;
        if (shields<0)
        then begin
            System.out.println("BOOM!!!");
            dead = true;
        end
    end
end

class StarWars
begin

    private function void duel(XWing x, TieFighter t)
    begin

        for (;;)
        begin
            x.hit(t.getWeapon());
            if (x.isDead())
            then begin
                System.out.println("X-Wing is dead");
                break;
```

```
            end
            t.hit(x.getWeapon());
            if (t.isDead())
            then begin
                System.out.println("Tie Fighter is dead");
                break;
            end
        end
    end

    private function void battle(XWing good, TieFighter evil)
    begin
        var int g          = 0;
        var int e          = 0;
        var int goodDeaths = 0;
        var int evilDeaths = 0;

        while (g<good.length and e<evil.length)
        begin
            System.out.println("battling X-Wing #" + g + " versus Tie Fighter #" + e);
            duel(goodg,evile);
            if (goodg.isDead())
            then begin
                g = g + 1;
                goodDeaths = goodDeaths + 1;
            end
            if (evile.isDead())
            then begin
                e = e + 1;
                evilDeaths = evilDeaths + 1;
            end
        end

        var int finalGood = good.length - goodDeaths;
        var int finalEvil = evil.length - evilDeaths;

        System.out.println();
        System.out.println("Battle Report:     X-Wings     Tie Fighters");
        System.out.println("-----------------------------------------");
        System.out.println();
        System.out.println("Initial ships:" + good.length + "  " + evil.length);
        System.out.println();
        System.out.println("Killed ships:"  + goodDeaths  + "  " + evilDeaths);
        System.out.println();
        System.out.println("Final ships:"    + finalGoodPD + "  " + finalEvil);
        System.out.println();
        if (finalGood>finalEvil)
```

```
            then begin
                System.out.println("The rebel alliance is victorious!");
            end
            else begin
                System.out.println("The dark side has conquered!");
            end
            System.out.println();
        end

        beginMain
            // defines the goodies array
            var XWing goodies = new XWing3;
            // initialises the elements of the goodies array
            superfor (var int i=0 to goodies.length-1)
            begin
                goodiesi = new XWing();
            end

            // defines the baddies array
            var TieFighter baddies = new TieFighter3;
            // initialises the elements of the baddies array
            superfor (var int i=0 to baddies.length-1)
            begin
                baddiesi = new TieFighter();
            end

            battle(goodies,baddies);

        endMain
    end
```

Question 4.16.2: Compile and run this file to see the battle between the X-Wings and the Tie Fighters unfold.

Question 4.16.3: If you look at the Java code for the XWing and TieFighter classes you will notice that they are almost identical: They have the same methods and properties, the only difference is that the XWing objects are initialized with a different value for their shields and weapon properties to the TieFighter objects.

The next few questions will guide you through the process of using inheritance to eliminate this unnecessary duplication of code. A new class called SpaceShip will be created and all of the code that is common to XWing and TieFighter will be moved into this class. The XWing and TieFighter classes will then be modified so that they both inherit from SpaceShip.

Question 4.16.4: The first step in this process is to create the outer shell of the SpaceShip class, which you should now type in:

```
class SpaceShip
begin
end
```

Question 4.16.5: Move the properties shields, weapon and dead out of the XWing and TieFighter classes and into the SpaceShip class. You must change the privacy status of the properties from private to protected. The protected modifier was invented as an intermediate level of privacy between public and private. Like private, it allows visibility to the same class in which the method or property was defined, but unlike private it also allows visibility to sub-classes of the class in which the method or property was defined.

Question 4.16.6: Move the three methods getWeapon, isDead and hit out of the XWing and TieFighter classses and into the SpaceShip class. At this point, the XWing and TieFighter classes should contain nothing but a constructor.

Question 4.16.7: Finally, add the extends keyword to the first line of the XWing and TieFighter classes:

```
class XWing extends SpaceShip)
```

and

```
class TieFighter extends SpaceShip)
```

Question 4.16.8: Compile and run your program again, making sure that it produces the same results now that it is using inheritance.

Question 4.16.9: The SpaceShip class is a super-class of both XWing and TieFighter containing everything that X-Wings and Tie Fighters contain in common. Because the role of the SpaceShip class is simply to hold these commonalities, we might choose to label the class with the abstract keyword:

```
abstract class SpaceShip)
```

This prevents us from creating instances of the SpaceShip class. Without the abstract modifier, we could happily create a new SpaceShip(), which would be an object that is not an X-Wing, nor a Tie Fighter, but just a vague "space ship". If we consider this to be a logical mistake then we can use abstract to prevent such calls to the SpaceShip constructor. Change the class SpaceShip to be abstract and observe how the compiler will not accept any lines of the form:

```
var SpaceShip s = new SpaceShip();) // compiler error
```

Remove the abstract keyword and notice how the compiler will then allow this line to compile.

4.17 Tutorial 17 Arrays inheritance and polymorphism

Question 4.17.1: Study, compile and run the following code:

```
class AnimalTest
begin
    private function void chatter(Animal[] a)
    begin
        superfor (var int i=0 to a.length-1)
        begin
            a[i].talk();
        end
    end
    beginMain
```

```
        var Animal[] farm = { new Dog(), new Cow(), new Fish() };
        var Animal[] ark  = { new Dog(), new Dog(), new Cow(), new Cow(), new Fish(),
        var Cow[]    herd = { new Cow(), new Cow(), new Cow() };
        chatter(farm);
        chatter(ark);
        chatter(herd);
    endMain
end

class Animal
begin
    method boolean breathesUnderwater()
    begin
        return false;
    end

    method boolean isPredator()
    begin
        return false;
    end

    method void talk()
    begin
    end
end

class Dog extends Animal
begin
    method boolean isPredator()
    begin
        return true;
    end

    method void talk()
    begin
        System.out.println("Woof woof!");
    end
end
```

Question 4.17.2: Write the following classes that subclass the Animal class above: Cow, Cat, Fish, and Whale.

Question 4.17.3: Write the Shark class which extends Fish class. Override all necessary methods. For the sake of this example and the code that follows, suppose that shark's talk method prints out "Chomp Chomp!".

Question 4.17.4: Run the AnimalTest class to make sure that all the methods work correctly.

Question 4.17.5: Rewrite the chatter method so that it never calls the talk methods and instead uses a series of if (...) then ... statements and the instanceof operator to test the run-time type of each object in the a array. Here is some code to get you started:

```
private function void chatter(Animal[] a)
begin
    superfor (var int i=0 to a.length-1)
    begin
        if (a[i] instanceof Cow) then
        begin
            System.out.println("Moo!");
        end
        elseif (a[i] instanceof Cat) then
        begin
            System.out.println("Meow!");
        end
        /* other code goes here */
    end
end
```

Note that the sub-classes must appear before super-classes in the above code, otherwise the wrong message will be printed out for sub-classes.

Question 4.17.6: Why is the code from the last question not as good as calling each animal's talk method? In general polymorphism is preferable to run-time type inquiry.

4.18 Tutorial 18 Advanced J.T.W.

See Section 5.5 [How to build a collection of class files or an entire package], page 74, for more information about compiling an entire package worth of classes.

4.18.1 Mapping J.T.W. to Java

Here is how to map from J.T.W. to Java:

function -> static
var -> nothing
classVar -> static
property -> nothing
method -> nothing
constructor -> nothing
begin -> {
end -> }
beginMain -> public static void main(String[] args) {
endMain -> }
and -> &&
or -> ||
then -> nothing
elseif -> else if

Here is an J.T.W. program:

```
class HelloWorld
```

```
begin
    beginMain
        System.out.println("Hello, World!")
    endMain
end
```

Here is the same J.T.W. program, after conversion to the Java language:

```
class HelloWorld
{
    public static void main(String[] args)
    {
        System.out.println("Hello, World!")
    }
}
```

Note that these J.T.W. keywords on the left hand side of the above diagram should not map to their Java equivalents inside strings and comments. The transformation was originally written to use the m4 language to map J.T.W. onto Java but this approach had the disadvantage that keywords like begin and end inside strings were mapped to their Java equivalents like so:

```
System.out.println("function");
System.out.println("static");
System.out.println("classVar");
System.out.println("static");
System.out.println("property");
System.out.println("");
System.out.println("method");
System.out.println("");
System.out.println("constructor");
System.out.println("");
System.out.println("begin");
System.out.println("{");
System.out.println("end");
System.out.println("}");
System.out.println("beginMain");
System.out.println("public static void main(String[] args)
    {");
System.out.println("endMain");
System.out.println("}");
System.out.println("and");
System.out.println("&&");
System.out.println("or");
System.out.println("||");
System.out.println("not");
System.out.println("");
System.out.println("elsif");
System.out.println("else if");
```

which is of course the wrong behaviour. A hack to get around this limitation is to break apart the J.T.W. keywords like so:

```
System.out.println("be" + "gin");)
```

This problem can be fixed for good either by using Flex to compile J.T.W. into Java or to use Emacs to do the same thing, only a little slower than what Flex can do. In the end I chose GNU Emacs as the host for the preprocessor language J.T.W. because it is free, libre and open source software, is adequate for my programming needs and is more powerful than Flex or m4. To remedy this deficiency Emacs' batch mode is used to do the transformation from J.T.W. to Java. This implies that GNU Emacs must be present on the client's system to do the J.T.W. to Java mapping. Of course, there is no compulsion to use Emacs as an editor, although there are a couple of advantages in doing this. Number one is that J.T.W. keywords and comments have automatic syntax highlighting. And number two is that Emacs can do correct automatic indentation of J.T.W. code.

4.18.2 Piping the output of javac and java

Output from the executables javac and java have their standard output stream and standard error stream piped into Emacs' batch mode so that error messages like Foo.java:123 point back to the correct file Foo.jtw:123 even if file inclusion (Chapter 3 [J.T.W. Proof of concept #2 file inclusion], page 10) has been used. The programs grep and sed are also used as pipes in the transformation process so they must be present on the client's system.

4.18.3 Makefile for building *.jtw into *.java and running *.class files

Here is the GNU Makefile for building *.java files and *.class files and running them.

```
.PRECIOUS:
.PRECIOUS: %.java %.class

JAVAC_FLAGS = -source 1.5 -Xlint:unchecked -Xlint:deprecation
JAVA_FLAGS  = -enableassertions
SHELL       = /bin/bash

%.java: %.jtw
        @echo "* Stage 1 : Debugging $*.jtw and building $*.java file"
        emacs --batch --eval "(setq *stump* \"$*\")" --load jtw-build-jtw.el --funcall

%.class: %.java
        @echo "* Stage 2 : Debugging *.java and building *.class file(s)"
        javac $(JAVAC_FLAGS) $$(find . -name "*.java") |& emacs --batch --eval "(setq

%.run: %.class
        @echo "* Stage 3 : Running $*.class File"
        java $(JAVA_FLAGS) $* |& emacs --batch --load jtw-java.el --funcall doit |& gr

build: clean
```

The first line .PRECIOUS without any arguments clears the list of precious files, the list of files not to delete during the build process.

5 Packages in J.T.W. and Java

The structure of a package mirrors the file system. For example you can have a package named (for argument's sake) pkg which corresponds to a folder pkg in your ~/jtw-tutorials folder. You can have a sub-package called (for argument's sake) inner which will reside in the folder ~/jtw-tutorials/pkg/inner. Even though the second package resides inside of the first package, they are still considered as separate packages.

There is a naming convention that I will not bother to use that helps to give unique names to your packages. If you own a website like davinpearson.com (http://davinpearson.com) you can name your packages like so: com/davinpearson/inner/inner2 where com/davinpearson where com.davinpearson.inner and com.davinpearson.inner.inner2 are separate packages. The fact that I own the domain name davinpearson.com ensures that my package specification com/davinpearson is unique. The com comes first because it is the actual domain name rather than the com extension that is unique. It is therefore non-sensible to place any code directly in the com folder. So in effect we are piggy-backing onto an existing standard i.e. Internet Domain Names. The same feature is exploited by Websites which ask for your email address as your login, as email addresses are unique to individual people.

5.1 Moving a class into a package

Consider a typical class:

```
 class A
begin
    property int data;

    classVar int data2 = 666;

    constructor A(int d)
    begin
        data = d;
    end

    method void meth1()
    begin
        System.out.println("meth1:" + data);
    end

    method void meth2()
    begin
        System.out.println("meth2:" + data);
    end

    function void func()
    begin
        System.out.println("func:" + data2);
    end
```

```
    beginMain
        var A a1 = new A(123);
        a1.meth1(); // prints out "meth1:123"
        var A a2 = new A(456);
        a2.meth2(); // prints out "meth2:456"
        A.func(); // prints out "func:666"
    endMain
end
```

To move this class into a package called (for argument's sake) pkg, you need to set the class's visibility status from none (i.e. package visibility) to public. Also each package visible (i.e. no private or public or protected specification) class variable, function, method and property needs to have its visibility status changed from package to public if you want to be able to access these items from outside of the package. If you have more than one class in the same file, they will have to be separated into separate files as you can only have one public class per file. Also the name of the package must be declared via a package specification like so package pkg; at the top of the file before any actual class or interface definitions. Here is the same source file, ready to be put into a package:

```
package pkg;

public class A
begin
    public property int data;

    public classVar int data2 = 666;

    public constructor A(int d)
    begin
        data = d;
    end

    public method void meth1()
    begin
        System.out.println("meth1:" + data);
    end

    public method void meth2()
    begin
        System.out.println("meth2:" + data);
    end

    public function void func()
    begin
        System.out.println("func:" + data2);
    end
```

```
    beginMain
        var A a1 = new A(123);
        a1.meth1(); // prints out "meth1:123"
        var A a2 = new A(456);
        a2.meth2(); // prints out "meth2:456"
        A.func(); // prints out "func:666"
    endMain
end
```

Also the source file for the class needs to be moved into the folder ~/jtw-tutorials/pkg. To run the class, you will need to invoke the Makefile command:

make clean pkg/A.run

5.2 Moving a class into a sub-package

Suppose you want to move a class A from no package (the folder ~/jtw-tutorials) to a package called for argument's sake pkg.inner, the steps from Section 5.1 [Moving a class into a package], page 70, needs to be followed, the only difference being that the package spec needs to be changed to package pkg.inner; and the file needs to be moved into the folder pkg/inner. To run the class file you need to invoke the following Make command:

make clean pkg/inner/A.run.

Here is the class definition for the file ~/jtw-tutorials/pkg/inner/A.jtw:

```
package pkg.inner;

public class A
begin
    public property int data;

    public classVar int data2 = 666;

    public constructor A(int d)
    begin
        data = d;
    end

    public method void meth1()
    begin
        System.out.println("meth1:" + data);
    end

    public method void meth2()
    begin
        System.out.println("meth2:" + data);
    end

    public function void func()
    begin
        System.out.println("func:" + data2);
```

```
    end

    beginMain
        var A a1 = new A(123);
        a1.meth1(); // prints out "meth1:123"
        var A a2 = new A(456);
        a2.meth2(); // prints out "meth2:456"
        A.func(); // prints out "func:666"
    endMain
end
```

5.3 Importing a package

When referring to a class or interface in a package you need to specify the package name in front of every class name and interface name in the package you want to access, like so, in the main folder ~/jtw-tutorials (outside of any package):

```
class B
begin
    beginMain
        var pkg.A a1 = new pkg.A(123);
        a1.meth1(); // prints out "meth1:123"
        var pkg.A a2 = new pkg.A(456);
        a2.meth2(); // prints out "meth2:456"
        pkg.A.func(); // prints out "func:666"
    endMain
end
```

To avoid having to qualify each class name and interface name with it's package, you need to use the import directive like so before the definition of the class like so:

```
import pkg.*;

class B
begin
    beginMain
        var A a1 = new A(123);
        a1.meth1(); // prints out "meth1:123"
        var A a2 = new A(456);
        a2.meth2(); // prints out "meth2:456"
        A.func(); // prints out "func:666"
    endMain
end
```

5.4 Importing a package from another package

When referring to a class or interface in a package you need to specify the package name in front of every class name or interface name in the package you want to access, like so, in the folder ~/jtw-tutorials/pkg (i.e.\ in the pkg package).

```
package pkg;
```

```
public class C
begin
    beginMain
        var pkg.inner.A a1 = new pkg.inner.A(123);
        a1.meth1(); // prints out "meth1:123"
        var pkg.inner.A a2 = new pkg.inner.A(456);
        a2.meth2(); // prints out "meth2:456"
        pkg.inner.A.func(); // prints out "func:666"
    endMain
end
```

To avoid having to qualify each class name or interface name with it's package, you need to use the import directive like so after the package declaration but before the definition of the class or interface like so:

```
package pkg;

import pkg.inner.*;

public class C
begin
    beginMain
        var A a1 = new A(123);
        a1.meth1(); // prints out "meth1:123"
        var A a2 = new A(456);
        a2.meth2(); // prints out "meth2:456"
        A.func(); // prints out "func:666"
    endMain
end
```

5.5 How to build a collection of class files or an entire package

When your class X uses another class Y in a different file then you need to add to the build target of your Makefile which is initially like so:

build: clean

to what follows:

build: clean Y.java

If your class Y is in another package such as the class ~/jtw-tutorials/path/to/dir/Y.class i.e. in the package path.to.dir then you need to add to the build target of your Makefile like so:

build: clean path/to/dir/Y.java

This process should be repeated for every class that is called, directly or indirectly from your main class X. By applying this process to every file in your package, you can build an entire package, simply by invoking the Makefile command make build. To actually compile and run the X class, let ~/jtw-tutorials/path2/to/dir/X.class be the location of the X class. Then you need to invoke the following Makefile target:

make build path2/to/dir/X.run

The build target calls the "clean" target which deletes all *.java and *.class files directly or indirectly in the folder ~/jtw-tutorials. If you don't do this then java might run an old version of *.class files despite earlier errors in the build process. This is because the use of pipes in building and executing *.class files hides the return values of the programs javac and java.

5.6 How to invoke javadoc on a package

To invoke javadoc, you first need to issue the following command from the folder ~/jtw-tutorials:

make build

See the Section 5.5 [How to build a collection of class files or an entire package], page 74, for more information about setting up the build target. Then you need to issue the following command from the folder ~/jtw-tutorials:

javadoc path3/to/pkg -d /path4/to/dir

where path3.to.pkg is the name of the package that you want to build and /path4/to/dir is the desired location for your documentation files in *.html format.

Appendix A GNU Free Documentation License

GNU Free Documentation License Version 1.3, 3 November 2008

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

The "publisher" means any person or entity that distributes copies of the Document to the public.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or

distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission. B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement. C. State on the Title page the name of the publisher of the Modified Version, as the publisher. D. Preserve all the copyright notices of the Document. E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices. F. Include,

immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below. G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice. H. Include an unaltered copy of this License. I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence. J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission. K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein. L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles. M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version. N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section. O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties–for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified,

and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

11. RELICENSING

"Massive Multiauthor Collaboration Site" (or "MMC Site") means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A "Massive Multiauthor Collaboration" (or "MMC") contained in the site means any set of copyrightable works thus published on the MMC site.

"CC-BY-SA" means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

"Incorporate" means to publish or republish a Document, in whole or in part, as part of another Document.

An MMC is "eligible for relicensing" if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subse-

quently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

Copyright (c) YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with...Texts." line with this:

with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Appendix B Passwords for the answers to the tutorials

Here are the passwords for the tutorials, which are located at the following Website: http://davin.50webs.com/J.T.W. They can be found by clicking on the link in Section 3 Answers to the tutorials of the document.

No.	Password:
1	policefish
2	chessweta
3	tallpencil
4	freshwhale
5	sneakermagic
6	kingpump
7	lakemarmite
8	nutriciouslamps
9	sadbutter
10	skyfresh
11	fivemagpies
12	phonesheds
13	dawnsweet
14	nightroads
15	blackscrews
16	snowfrog
17	tenflower

Index

I

J

L

M

N

O

P

R

S

T

V

W

Y