Eric Roberts and Jerry Cain
CS 106J

Handout #42
May 24, 2017

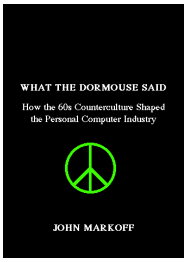
Data-Driven Programs

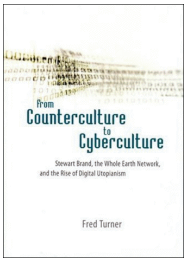
Data-Driven Programs

Eric Roberts and Jerry Cain
CS 106J
May 24, 2017

Computing and the Counterculture

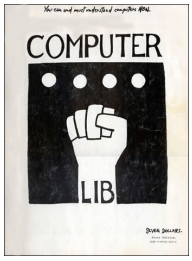
Two recent books argue that the personal computing revolution owes as much to the counterculture of the 1960s as it does to the technological strength and entrepreneurial spirit of Silicon Valley.

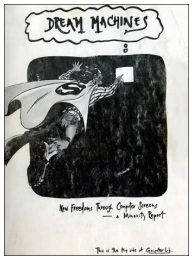




Ted Nelson's Cyberspace Dreams

The countercultural vision comes across particularly clearly in the two-sided book *Computer Lib/Dream Machines* which was written by cyberspace visionary Ted Nelson in 1974.





Designing Data Structures

- When you design a program, one of the first tasks you need to undertake is understanding how the underlying data structures fit together and how each level of the data hierarchy can best be represented.
- This process is similar to that of decomposing a large problem into a set of successively simpler subproblems. In the data domain, the information your program needs to process must be decomposed into successively simpler data structures until everything can be represented using a built-in JavaScript value, such as a number or a string.
- The tools for data decomposition you have seen so far include
  - *Arrays,* which implement sequences of values
  - *Aggregates,* which represent collections of related values
  - *Maps,* which establish a relationship between keys and values

Exercise: Localization

- To be successful in our global economy, modern applications must support a wide variety of languages. The process of tailoring an application to communicate with users in the appropriate language is called ***localization.***
- Suppose you want to be able to specify labels for buttons in English but would like to design a data structure that converts those names to the ***locale,*** represented as a two-character abbreviation, such as `"fr"` for France or `"de"` for Germany.
- Design a data structure that stores localized translations for any number of button labels in any number of localizations. The idea is that your data structure should allow clients to call `localize(name, locale)` with the English version of the button name and have it return the appropriate translation for the specified locale.

Data-Driven Programs

- In most programming languages, data structures are easier to manipulate than code. As a result, it is often useful to design applications so that as much of their behavior as possible is represented as data rather than in the form of methods. Programs that work this way are said to be ***data driven***.
- In a data-driven system, the actual program (which is called a ***driver***) is usually very small. Such driver programs operate in two phases:
  1. Read data from a file into a suitable internal data structure.
  2. Use the data structure to control the flow of the program.
- To illustrate the idea of a data-driven system, we're going to spend most of this lecture building a programmed-instruction "teaching machine" of the sort that Ted Nelson discusses (mostly critically) in *Dream Machines*.

The Course Data File

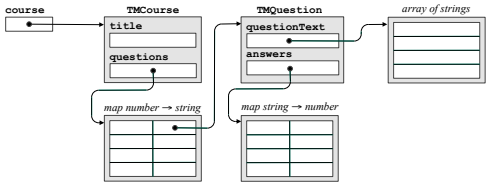
In our teaching machine application, the course designer—who is an expert in the domain of instruction and not necessarily a programmer—creates a data file that serves as the driver. The general format of the whole file is shown on the left, and a specific example of a question and its answers appears on the right.

Course title
First question and its answers
Next question and its answers
⋮
Additional question/answer entries
⋮
Last question and its answers
<end of file>

```
5
What is the value of 17 % 4?
   a.  0
   b.  1
   c.  3
   d.  4
-----
a:       6
0:       6
b:       7
1:       7
c:       6
3:       6
d:       6
4:       6
```

Choosing an Internal Representation

The first step in building the teaching machine is to design a set of classes that can represent the data and relationships in the file. All of the relevant data should be accessible from a single structure that contains all relevant information in a nested series of classes.

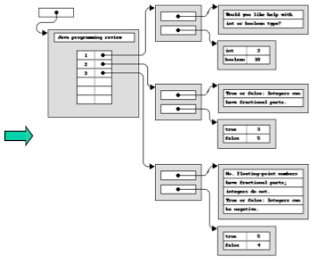


Converting External to Internal Form

```
Java programming review
1
Would you like help with
int or boolean type?
-----
int:      2
boolean: 10

2
True or false: Integers can
have fractional parts.
-----
true:     3
false:    5

3
No. Floating-point numbers
have fractional parts;
integers do not.
True or false: Integers can
be negative.
-----
true:     5
false:    4
```



Code for the `TMQuestion` Class

```
/*
 * Creates a new question containing the text as an array of lines.
 * Clients must add new answer/question pairs by calling addAnswer.
 */

function TMQuestion(text) {
   var answerTable = { };
   return {
      printQuestionText: function() {
                            for (var i = 0; i < text.length; i++) {
                               console.log(text[i]);
                            }
                         },
      addAnswer: function(response, nextQuestion) {
                    answerTable[response] = nextQuestion;
                 },
      lookupAnswer: function(response) {
                       return answerTable[response];
                    }
   };
}
```

Code for the `TMCourse` Class

```
/*
 * File: TMCourse.java
 * -------------------
 * This class defines the data structure for a course for use with
 * the TeachingMachine program.
 */

/* Constants */

const MARKER = "-----";

/*
 * Creates a new course for the teaching machine by reading the
 * data from the specified file, which consists of questions and
 * their accepted answers.
 */

function TMCourse(filename) {
   .
   .
   .
}
```

```
function TMCourse(filename) {
   var lines = File.readLines(filename);
   if (lines === undefined) return null;
   var nLines = lines.length;
   var title = lines.shift();
   var questions = [ ];
   var line = lines.shift();
   while (line !== undefined) {
      var qnum = parseInt(line);
      var text = [ ];
      while (line !== undefined && line !== MARKER) {
         text.push(line);
         line = lines.shift();
      }
      var question = TMQuestion(text);
      line = lines.shift();
      while (line !== undefined && line !== "") {
         var colon = line.indexOf(":");
         var response = line.substring(0, colon).toLowerCase().trim();
         var nextQuestion = parseInt(line.substring(colon + 1).trim());
         question.addAnswer(response, nextQuestion);
         line = lines.shift();
      }
      questions[qnum] = question;
      line = lines.shift();
   }
   return {
      getTitle: function() { return title; },
      getQuestion: function(qnum) { return questions[qnum]; }
   };
}
```

Code for **TeachingMachine.js**

```
/*
 * File: TeachingMachine.js
 * ------------------------
 * This program executes a programmed instruction course.
 */

import "TMCourse.js";
import "TMQuestion.js";
import "file";

function TeachingMachine() {
   var callback = function(filename) {
      var course = TMCourse(filename);
      if (course === null) {
         console.log("Can't read that file.");
         console.requestInput("Enter name of course file: ", callback);
      } else {
         stepThroughCourse(course);
      }
   };
   console.requestInput("Enter name of course file: ", callback);
}
```

```
function stepThroughCourse(course) {
   var qnum = 1;
   var askQuestion = function() {
      if (qnum === 0) {
         console.log("Done");
      } else {
         var question = course.getQuestion(qnum);
         if (question === undefined) {
            console.log("Missing question " + qnum);
         } else {
            question.printQuestionText();
            console.requestInput(checkAnswer);
         }
      }
   };
   var checkAnswer = function(line) {
      var question = course.getQuestion(qnum);
      var nextQuestion = question.lookupAnswer(line);
      if (nextQuestion === undefined) {
         console.log("I don't understand that response.");
      } else {
         qnum = nextQuestion;
      }
      askQuestion();

   };
   console.log(course.getTitle());
   askQuestion();
}
```