Eric Roberts
CS 106A

Handout #54
May 23, 2012

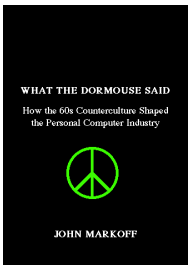
Data-Driven Programs

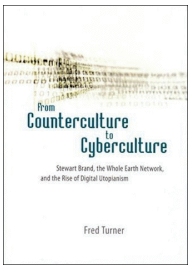
Data-Driven Programs

Eric Roberts
CS 106A
May 23, 2012

Computing and the Counterculture

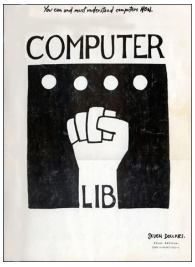
Two recent books argue that the personal computing revolution owes as much to the counterculture of the 1960s as it does to the technological strength and entrepreneurial spirit of Silicon Valley.

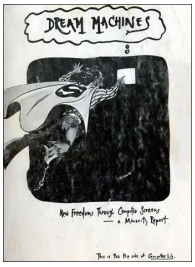




Ted Nelson's Cyberspace Dreams

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.





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

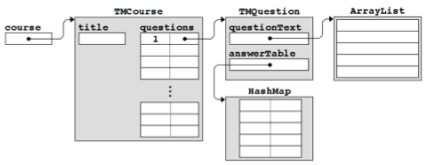
In our teaching machine application, the course designer—who is an expert in the domain of instruction and not necessarily a programmer—creates a data file that serves as the driver. The general format of the whole file is shown on the left, and a specific example of a question and its answers appears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

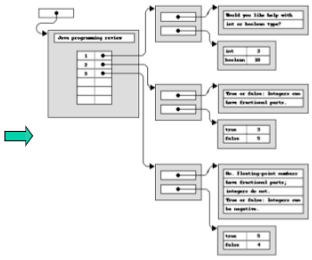


Converting External to Internal Form

```
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
```



Static Factory Methods

- One of the important decisions that you need to consider in designing classes for representing large structures is how you create new instances of those classes.
- Although the traditional strategy of defining a constructor often makes sense, another common strategy is to define a static method that returns a new instance of the class. Such methods are called ***factory methods***.
- Factory methods are especially useful in the following cases:
  - When you are reading data from a file, it often makes sense to return **`null`** as a sentinel to indicate the end of a data file. Constructors cannot return **`null`**, but factory methods can.
  - When you want to return an instance of a subclass of the factory class. Once again, factory methods make that strategy possible.

Code for the `TMQuestion` Class

```
/*
 * File: TMQuestion.java
 * ---------------------
 * This file defines a class to represent a single question.
 */

import acm.util.*;
import java.io.*;
import java.util.*;

/**
 * This class models a single question in the course data base.
 */

class TMQuestion {

/**
 * Creates a new question by reading its data from the specified reader.
 * If no data is left in the reader, this method returns <code>null</code>
 * instead of an <code>TMQuestion</code> value.  Note that this is a
 * static method, which means that you need to call
 *
 *<pre><code>
 *      TMQuestion.readQuestion(rd)
 *</code></pre>
 *
 * @param rd The reader from which the question data is read
 */
```


Code for the `TMQuestion` Class

```
   public static TMQuestion readQuestion(BufferedReader rd) {
      try {
         String line = rd.readLine();
         if (line == null) return null;
         TMQuestion question = new TMQuestion();
         question.questionNumber = Integer.parseInt(line);
         question.questionText = new ArrayList<String>();
         while (true) {
            line = rd.readLine();
            if (line.equals(MARKER)) break;
            question.questionText.add(line);
         }
         question.answerTable = new HashMap<String,Integer>();
         while (true) {
            line = rd.readLine();
            if (line == null || line.length() == 0) break;
            parseAnswerLine(question, line);
         }
         return question;
      } catch (IOException ex) {
         throw new ErrorException(ex);
      } catch (NumberFormatException ex) {
         throw new ErrorException("Illegal question number");
      }
   }
```


Code for the `TMQuestion` Class

```
/**
 * Returns the number of this question.
 */
   public int getQuestionNumber() {
      return questionNumber;
   }

/**
 * Returns an ArrayList containing the text for this question.
 */
   public ArrayList<String> getQuestionText() {
      return questionText;
   }

/**
 * Looks up the answer in the table of possible answers for this question.
 * If a match is found, the number of the associated next question is
 * returned.  If not, the method returns -1.
 */
   public int lookupAnswer(String answer) {
      Integer value = answerTable.get(answer.toUpperCase());
      if (value == null) return -1;
      return value;
   }
```


Code for the `TMQuestion` Class

```
/**
 * This method scans the answer line to separate the text of the answer
 * from the number of the next question.  The value of the next question
 * is entered into the HashMap stored as part of this TMQuestion structure.
 */
   private static void parseAnswerLine(TMQuestion question, String line) {
      int colon = line.indexOf(":");
      if (colon == -1) {
         throw new ErrorException("Missing colon in " + line);
      }
      String response = line.substring(0, colon).toUpperCase().trim();
      int nextQuestion = Integer.parseInt(line.substring(colon + 1).trim());
      question.answerTable.put(response, nextQuestion);
   }

/* Private constants */
   private static String MARKER = "-----";

/* Instance variables */
   private int questionNumber;
   private ArrayList<String> questionText;
   private HashMap<String,Integer> answerTable;
}
```

Code for the `TMCourse` Class

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

import acm.util.*;
import java.io.*;
import java.util.*;

public class TMCourse {
```

page 1 of 3

Code for the `TMCourse` Class

```
/**
 * Creates a new course for the teaching machine by reading the
 * data in the specified file.  The file format for the data is
 * defined in Handout #56 ("Data-Driven Programs").
 *
 * @param filename The name of the data file
 */
   public TMCourse(String filename) {
      questions = new TreeMap<Integer,TMQuestion>();
      try {
         BufferedReader rd = new BufferedReader(new FileReader(filename));
         title = rd.readLine();
         while (true) {
            TMQuestion question = TMQuestion.readQuestion(rd);
            if (question == null) break;
            questions.put(question.getQuestionNumber(), question);
         }
         rd.close();
      } catch (IOException ex) {
         throw new ErrorException("Can't open " + filename);
      }
   }
```

page 2 of 3

Code for the `TMCourse` Class

```
/**
 * Returns the title of the course.
 *
 * @return The title of the course
 */
   public String getTitle() {
      return title;
   }

/**
 * Returns the question corresponding to a particular question
 * number, or <code>null</code> if no such question exists.
 *
 * @param number The question number
 * @return The <code>TMQuestion</code> object with that number
 */
   public TMQuestion getQuestion(int number) {
      return questions.get(number);
   }

/* Instance variables */
   private String title;
   private Map<Integer,TMQuestion> questions;
}
```

page 3 of 3

Code for the `TeachingMachine` Class

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

import acm.program.*;
import acm.util.*;

public class TeachingMachine extends ConsoleProgram {

   public void run() {
      TMCourse course = readCourseFile();
      stepThroughCourse(course);
   }
```

page 1 of 3

Code for the `TeachingMachine` Class

```
/**
 * Prompts the user for a course name and then reads in the
 * data for that course from the associated data file.  If the
 * <code>TMCourse</code> constructor signals an error, the
 * user is asked to supply a new course name.
 *
 * @return A <code>TMCourse</code> object for the desired course
 */
   private TMCourse readCourseFile() {
      while (true) {
         try {
            String courseName = readLine("Enter course name: ");
            return new TMCourse(courseName + ".txt");
         } catch (ErrorException ex) {
            println(ex.getMessage());
         }
      }
   }

/**
 * Steps through the questions in the order specified by the course.
 *
 * @param course A <code>TMCourse</code> object for the desired course
 */
```

page 2 of 3

Code for the `TeachingMachine` Class

```
   private void stepThroughCourse(TMCourse course) {
      println(course.getTitle());
      int questionNumber = 1;
      while (questionNumber != 0) {
         TMQuestion question = course.getQuestion(questionNumber);
         if (question == null) {
            throw new ErrorException("Missing question " + questionNumber);
         }
         for (String line : question.getQuestionText()) {
            println(line);
         }
         String answer = readLine();
         int nextQuestion = question.lookupAnswer(answer);
         if (nextQuestion == -1) {
            println("I don't understand that response.");
         } else {
            questionNumber = nextQuestion;
         }
      }
      println("Done");
   }

}
```

page 3 of 3