07.03 Virtual Lecture Notes (Part 1)

Arrays, as well as multiple parameters, can be passed to methods, as long as there is a direct one-to-one correspondence between arguments and parameters. However, the new structural organization of the source code requires some mental gymnastics to keep track of all the jumps back and forth between the **`main()`** method and the methods. You will be able to visualize the flow of control more easily if you print out this document before proceeding.

The purpose of this demo program is to grade a True/False test. Applying procedural abstraction, the task can be divided into smaller functional units, which will eventually become static methods.

1. Read a set of True/False test answers.
2. Grade the test with a key.
3. Calculate the percent correct.
4. Print the results.

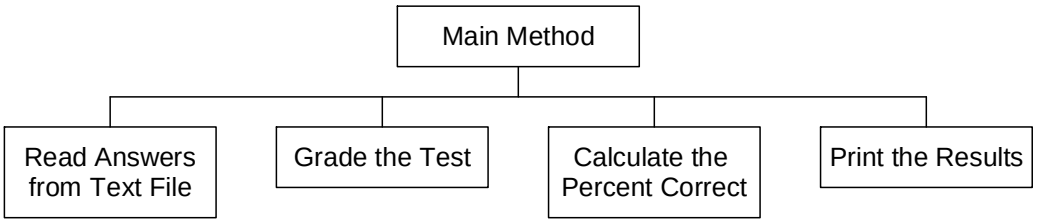
The **`main()`** method will call each static method in turn, and pass along any arguments required. One possible design approach is illustrated by the following pseudocode.

```
Create an array for the student answers.
Initialize an array for the answer key with the correct answers.

Read the test answers from a text file and assign them to an array.
Grade the test and determine the number of correct answers.
Calculate the percent correct.
Print the results.
```

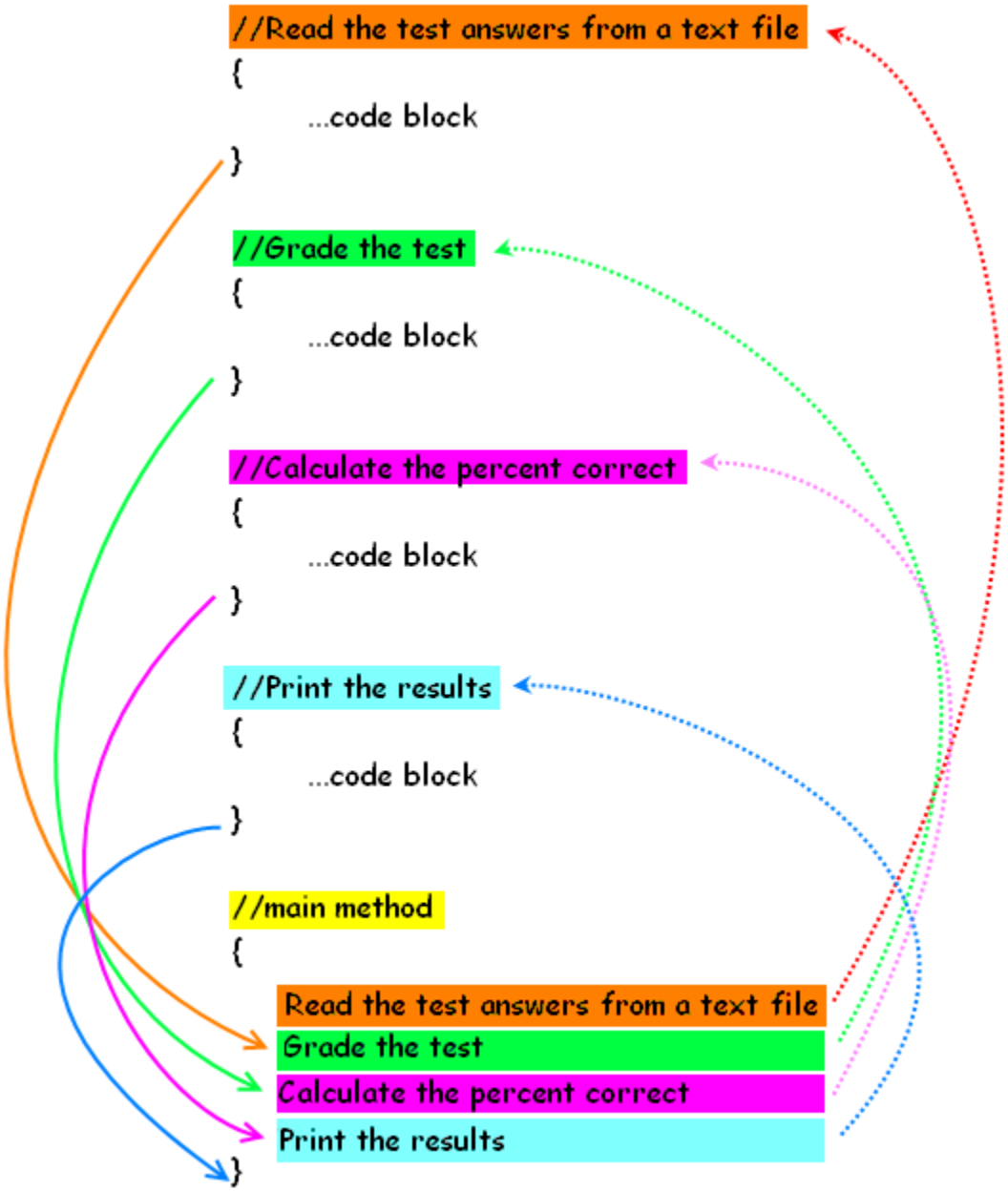
Based on this design, the **`main()`** method would call four methods to complete the task. Although it may be more efficient to combine grading and calculating the percent correct, they are separated here to demonstrate top-down design and procedural abstraction.

Initially, the hardest thing to get a handle on when a program is written with methods is the organizational structure flow of control within the program. Sometimes a picture is worth several hundred lines of code, so programmers often start by laying out a modular design as shown below.



This diagram illustrates the organizational hierarchy of the static methods in relation to the **`main()`** method and implies that execution will proceed from left to right; however, what you need at this stage is a way to superimpose on the organizational structure the flow of control within the program. Although the following diagram may appear a little jumbled, the arrows reveal how the **`main()`** method will call each of the static methods and where control returns after executing each code block. Method calls are shown with the dotted arrows on the right and returns are shown with

solid arrows on the left. The top-down nature of the algorithm is depicted as the main method calls each method in turn from top to bottom. Start with the first statement in the **`main()`** method and follow the arrows to gain a sense of how this program executes.



For a while, most of your programs will follow this top-down style with only one call to each method; however, eventually that, too, will change.

Analyzing structure and flow of control is good programming practice and is essential to the design of effective and efficient code. In fact, notice that no code has been discussed at this point. Planning always saves time!

The program (color coded) to analyze the True/False test is shown below. Please study it carefully and run the program previously downloaded. Notice how the [0] index position used in the program.

```
import java.util.Scanner;
import java.io.File;
import java.io.IOException;
public class TrueFalseTest
{
   //read in the test data from a text file
   public static String [ ] readTestData()throws IOException
   {
      int index = 0;
      String [ ] answers = new String[11];

      File fileName = new File("TestData1.txt");
      Scanner inFile = new Scanner(fileName);
      while (inFile.hasNext())
      {
         answers[index] = inFile.next( );
         index++;
      }
      inFile.close();
      return answers;
   }
   //grade the test and count the number correct
   public static int gradeTest(String[ ] test, String[ ] key)
   {
      int correct = 0;

      for(int n = 1; n < test.length; n++)
      {
         if(test[n].equals(key[n]))
            correct++;
      }
      return correct;
   }
   //calculate the percent correct
   public static double calcPercent(int n)
   {
      return (n/10.0) * 100;
   }
   //print the results
   public static void printResults(String studentId, double pct)
   {
      System.out.printf("Student Number: %s   Grade: %4.1f%n", studentId, pct);
   }
   //the main method
   public static void main(String[ ] args) throws IOException
   {
      String[ ] testAnswers = new String[11];
      String[ ] answerKey = {"name", "F", "F", "T", "T", "T", "T", "F", "F", "T", "T"};

      testAnswers = readTestData();
      int numCorrect = gradeTest(testAnswers, answerKey);
      double percent = calcPercent(numCorrect);
      printResults(testAnswers[0], percent);
   }
}
```