UIL COMPUTER SCIENCE THE NEXT 25

Introduction

This guide is written for students preparing to take the Texas University Interscholastic League Computer Science written exam. The first fifteen questions of each UIL Computer Science written exam are purposely more basic (easier) than the remaining twenty-five questions. The Hexco guide titled ***"UIL Computer Science – The First 15"*** covers the topics that make up the first fifteen questions on the test. ***This guide, however, is intended to show you how to correctly answer questions 16 through 40.*** UIL provides a list of topics to be covered by the written exam at http://www.uiltexas.org/files/academics/UILCS-JavaTopicList1516.pdf. Within that document can be found a list of topics used to create the written exam. Those topics will serve as the content for the chapters of this guide.

A few words about what this guide is NOT are appropriate at this time. This is a GUIDE, not a textbook. This guide is not a complete and exhaustive discussion of the topics presented. Reading and understanding this guide is NOT intended as a course in computer science. Using this guide along with enrollment in a good Computer Science course will enhance your success without a doubt. The more that you seek out and use alternative resources that are available; the more successful you will become. A great list of resources is available at http://www.uiltexas.org/academics/computer-science/resources.

Success in the UIL Computer Science competition requires a great deal of hard work and dedication! To be successful, you will have to practice A LOT! To be successful, you and your teammates will have to memorize A LOT! But, if you will read and study this guide, memorize what you are asked to memorize, practice often, and make use of all of the resources available to you, you can learn a lot of computer science and maybe win a few medals along the way.

Let's get started!

Kirby Rankin

Contents

Classes, Objects, Inheritance and Polymorphism

User Defined Classes and Objects

A user defined class contains the code that is necessary to describe the state and actions of a real world object. A class describes what all objects that belong to the class must look like and how they must behave. The implementation of a class requires a set of variables that describe the state of each object derived from the class. These are called ***instance variables***. A class requires a set of ***methods*** that describe the actions of each object derived from the class. A class also requires one or more ***constructors*** that allow a calling program (client code) to create an object that belongs to the class.

A simple example is in order. Suppose we wish to create a class from which we can derive integer objects. (Yes, Java already has one of those but it makes for a good example to learn from.) A class that creates whole number objects needs an instance field (variable) to store the number, methods to set the number, get the number and four more to perform the most simple arithmetic operations. Finally, we need a constructor (which is just a specialized method) to enable the client code to create objects derived from this class.

Here is the code for what we have described:

```
public class MyInteger {

   public int i;

   public MyInteger(int someInt){
      i=someInt;
   }
   public int get(){
      return i;
   }
   public void set(int someInt){
      i=someInt;
   }
   public int add(int someInt){
      return i+someInt;
   }
   public int subtract(int someInt){
      return i-someInt;
   }
   public int multiply(int someInt){
      return i*someInt;
   }
   public int divide(int someInt){
      return i/someInt;
   }
}
```

Notice that this class does not contain a main method. You could not execute this code all by itself. To see what we can do with our `MyInteger` class we have to write a different class that will construct some `MyInteger` objects and then call the methods within the class to make some things happen. This is called ***client code***.

Here is how that class could look:

```
import static java.lang.System.out;
public class MyIntegerTestClass {

   public static void main(String[] args) {
      MyInteger num1 = new MyInteger(5);
      MyInteger num2 = new MyInteger(10);
      MyInteger num3 = new MyInteger(15);
      out.println(num1.get());
      out.println(num2.get());
      out.println(num3.get());
      out.println(num1.add(2));
   }
}
```

Running this program gets us this output:

```
5
10
15
7
```

This line `MyInteger num1 = new MyInteger(5);` creates (***instantiates***) a `MyInteger` object named num1. The reserved word `new` calls the class' constructor method. Once `num1` has been created it contains its own copy of all of the fields and all of the methods described in the implementation of the `MyInteger` class. To access those fields and methods the programmer must reference the object name followed by the selection operator (the dot) and the name of the field or method.

Now let's discuss each part of the `MyInteger` class and see how UIL might ask a question about it.