CMPSCI 187: Programming With Data Structures

Lecture 6: The StringLog ADT
David Mix Barrington
17 September 2012

The StringLog ADT

- Data Abstraction
- Three Views of Data
- Java Interfaces
- Defining the StringLog ADT
- StringLog's methods: Constructors, Transformers, and Observers
- Code for the StringLog interface

Data Abstraction

- What *is* a piece of data, really? Computer scientists are practical people. For us, the essence of a piece of data is *what you can do with it*.

- Different actors should be authorized to do different things with a given piece of data. We want to **hide information** about the data from actors that don't need it. This isn't necessarily a matter of security -- in order for an object to do its job in the best way, we want a description of that job that is as simple as possible. Then we can make use of any flexibility that is available.

- An important general principle of design here is **modularity**. Different elements of our software should have clearly defined responsibilities, and the *interactions* among those elements need to be clearly specified.

- The start of a class definition is the definition of an **abstract data type** or **ADT**, which is a list and specifation of the desired operations for the objects.

Three Views of Data

- DJW talk about three separate levels at which we view an ADT (and thus view any class created to implement the ADT):

- At the **Application Level**, we view the ADT as a **client** what services does it offer us, and how to we access those services. Every predefined class in Java is described in the **application programming interface** or **API**, which is your primary source of information about it as a client.

- At the **Abstract Level**, we describe what data the object holds, and what can be done with it, less specifically than in an API, but still independently of any implementation.

- Finally, at the **Implementation Level** we consider how lower-level programming constructs are actually used to make the object do what it does.

Java Interfaces

- We often want to assume that an object can be used in a certain way, without having to know exactly what class that object is from.

- We've seen how we might, for example, put a `Terrier` object into a `Dog` variable, and use it as a `Dog`. Here `Dog` is a **superclass** of `Terrier`. An **interface** in Java is in effect a superclass that is not an actual class for which you may create objects.

- DJW's example on pages 69-72 is of a `FigureGeometry` interface for geometrical figures. It defines operations that we might want to carry out for circles *or* squares *or* triangles. (They give a `Circle` class that implements this interface.) If a variable's type is `FigureGeometry`, an object of any implementing type may go in that variable. But there is no such thing as a `FigureGeometry` *object*.

Defining the StringLog ADT

- So at the abstract level, what *is* a StringLog? We want it to be a collection of strings, with a single string that is the name of the collection. We want to be able to add new strings to the collection, determine whether a particular string is already there, and print out the entire collection of strings. We'll use this class later in implementing a trivia game.

- There are a few questions that are not answered by this simple description. Does a StringLog have a maximum capacity? (Any data structure on any fixed computer has a maximum capacity because the computer's memory is finite, but are there to be *practical* limits on the size?) Some concrete data structures can resize themselves as needed, and linked structures have no fixed size.

- This description tells us what *methods* we will need. Note that it does *not* tell us what data fields we need, only generally what we need to store. For example, we might or might not have an instance variable for the size.

Constructors, Transformers, and Observers

- In general, we can group the instance methods of a class into three categories. We need one or more **constructors** to create new objects, **transformers** to alter the data fields of an object, and **observers** to read, or otherwise get information about, the data fields. The simplest transformers are **setters** and the simplest observers are **getters**.

- StringLog has two constructors, one with a size field and one without.

- We have two transformers, one to `insert` a new string and one to `clear` the collection of all its strings while retaining the StringLog's name.

- We have five observers: three simple (get the name, ask the size, or find out whether we are full) and two more complicated (is a given string in the log, return a formatted list of all the strings in the log).

Code for the StringLog Interface

- Here is the code, without comments, to declare an interface defining the StringLog ADT. Later classes will `implement` this interface.

- Note that there are no constructors and no code for the methods -- they are **abstract methods** that will be overridden later by the real classes.

- The comments would say what each method does and give the precondition for `insert` (that the StringLog is not currently full).

```
public interface StringLogInterface {
   void insert(String element);
   boolean isFull( );
   int size( );
   boolean contains(String element);
   void clear( );
   String getName( );
   String toString( );}
```