Pause and reflect

Think about the following code.

```
1  private void initializeFromSource(String source) {
2    adjectiveList=readIt(source+"/adjective.txt");
3    nounList     =readIt(source+"/noun.txt");
4    colorList    =readIt(source+"/color.txt");
5    countryList  =readIt(source+"/country.txt");
6    nameList     =readIt(source+"/name.txt");
7    animalList   =readIt(source+"/animal.txt");
8    timeList     =readIt(source+"/time.txt")
9  }
```

How would you write a HashMap version to be more flexible?

Continue

Have a question? Discuss this lecture in the week forums.

Interactive Transcript

Search Transcript

English

0:03
In this lesson, we'll look at how we can use the HashMap class to make our GladLib class easier to extend, have fewer lines of code, and be a good example of how to become a more skilled and experienced software designer.

0:17
As a reminder, extending the class to use a new label like angle bracket verb angle bracket, requires modifying the code in three places. You'll need to create an array list instance variable, initialize it properly, and use it as the source for your random replacements.

0:35
You should also follow a convention of using field names, like verbList, for the label verb. This makes it difficult to use text files or URLs for the source of word replacements unless all such sources follow the same conventions, such as using a file name noun.txt for the label noun, or the field, nounList, or color.txt for colorlist in the field color.

1:02
Let's take a look at the concepts behind these requirements for extending the GladLib class, to look at a new way of structuring data in classes. Each label is associated with an ArrayList instance variable. You see the label <noun> associated with nounList, the label <color> with colorList, and so on.

1:22
These named instance variables make for a poor design.

1:26
Adding a new label, like verb, requires defining an instance variable by name, initializing it by name, and using it by name. That's three places in which the program must be modified. Instead, we'll use a HashMap to help create a better and more flexible design. The HashMap will allow us to label or align the label to an ArrayList without ever having to name the ArrayList, itself.

1:53
Given a label, the code will look up or find the associated ArrayList in the HashMap structure. As you've seen, this is similar to how indexOf works for finding a value in an ArrayList or a character in a string. Getting the value associated with a label will return an ArrayList.

2:13
Let's take a closer look at how using a HashMap creates a more flexible design.

2:18
One HashMap will replace seven or more instance variables. The HashMap will reference as many Arraylists as needed, rather than us having to define separate instance variables, and following a naming convention, as you see here.

2:32
The code will use a single instance variable, a HashMap named myMap. This will associate an ArrayList with each label. So the keys in the map are strings, the label in the GladLib program. The value associated with each key is the ArrayList of replacement words for that label.

2:53
This means that to add a new label and a new ArrayList, we don't have to add a new instance variable. We simply need to store new values in the single HashMap instance named, myMap.

3:07
Let's look at how the method, getSubstitute, works with a HashMap. In the original program, a sequence of if statements was used to identify the instance variable associated with the particular label. The naming convention of using countryList for country allows a programmer to extend the code. But there will always be as many if statements in the getSubstitute method as there are labels and instance variables.

3:32
The last if statement is different. You can see that the label, angle bracket, number, angle bracket, generates a random number instead of finding one in a list of numbers.

3:44
When using a Hashmap, the getSubstitute method is much more simple. The Hashmap associates a label with the ArrayList of replacements.

3:52
The ArrayList for a label is accessed using the HashMap.get method to get the ArrayList associated with a string label like country or noun or color. Adding a new label doesn't require modifying this method at all. And that's an example of the open/closed principle we talked about in a previous video. Using a HasMap makes for a more flexible class that's easier to extend, but there's room for even more improvement using HashMaps again.

4:22
The original program reads a file or URL to store information in each named instance variable, the ArrayList of replacement values for that label.

4:32
That was done in a sequence of statements that called the helper method, readIt.

4:38
The HashMap version still associates a label with a file name, and that file name must be specified in the program. But the code is different because we use a loop to associate each label with a file name. This wasn't possible in the original program.

4:54
Note the private helper method, readIt, is still called.

4:58
What changes if we want to add a new label like verb? The program will still associate the name of the file of replacement values in verb.txt with the new label.

5:11
We could store a new string, like quote verb quote, in the local string array of variable labels. We could add that just after the string, timeframe, for example. Unfortunately, we still have the limitation in that the code uses a naming convention for files like verb.text for the label verb.

5:31
We could use a HashMap in a different way to associate file names with labels, without modifying the program. The program could be designed to read a file of information that specifies where to find the words to replace the labels. Rather than requiring the code to modified, compiled, tested, and run to simply find nouns in a different file or website.

5:54
This kind of file is often called a .properties or property file. As shown here, it simply associates a label with a source of replacements for that label.

6:05
The convention of using a colon to separate values in a .properties file is common, but equals could also separate the values. The ,properties file can be read using a file resource object, for example, and the information in the properties file stored in a HashMap.

6:22
Suppose a HashMap instance variable named myLabelSouce is used to associate labels like noun with the source of words that are now replacements, like GladLibs.com/nounsfunny.txt. The method initialized from source would simply loop over the values stored as keys in the HashMap. Recall that keys are accessed via the method, keySet.

6:50
The string that specifies the source for each label is retrieved from the map using the .get method. The source is used to read values into the ArrayList for the label.

7:03
You could add such a feature to the GladLib program to make it even more extensible.

Downloads

- **Lecture Video** mp4
- **Subtitles (English)** WebVTT
- **Transcript (English)** txt

Would you like to help us translate the transcript and subtitles into additional languages?