Santiago Bermudez

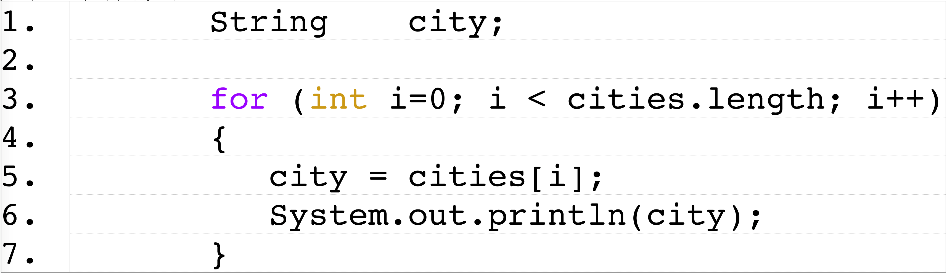
CSC 131 SP’19 In-Class Activity

Design Patterns

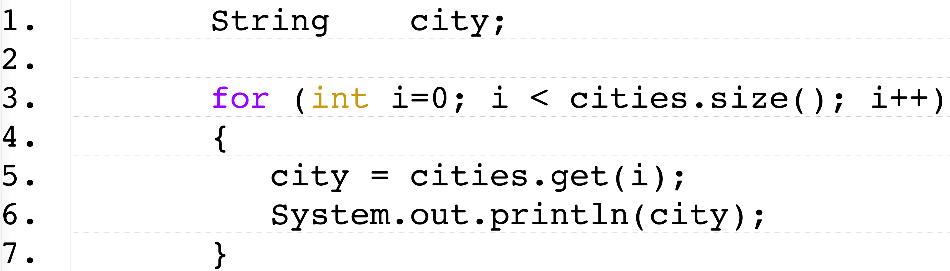
# Model 1 The Iterator Pattern

Study the following three java code snippets and discuss Question 1-3 with your partners.

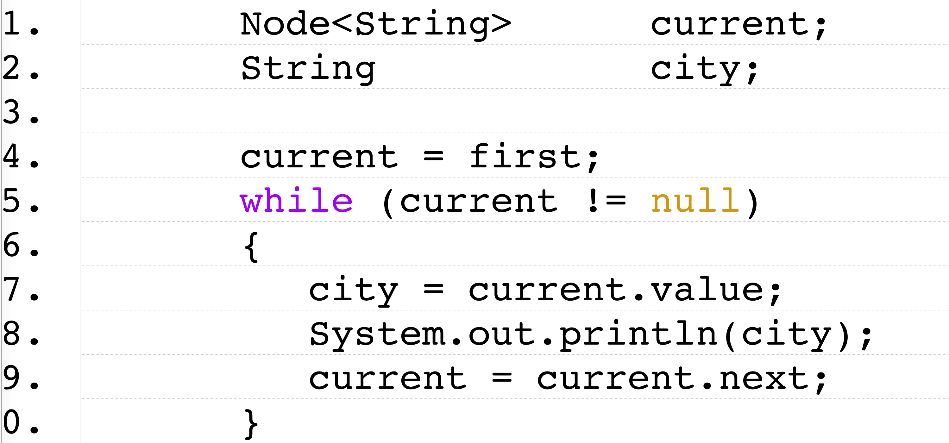
S1: Looping over an array



S2: Looping over an arraylist



S3: Looping over a linked structure



Q1: Fill out the following table:

|  |  |  |
| --- | --- | --- |
| Code | What does it do? | What data structure is used? |
| S1 | This code prints out a list of cities in the order that they were added to the array. It uses a for loop to iterate through the array until it reaches the length of the array. | In this code, it looks like it simply uses an array of strings. |
| S2 | Like the prior loop, it prints out a list of cities in the order that they were added. Although this time it uses an arraylist instead of an array. Again, it uses a for loop to iterate through the arraylist until it reaches the length of the arraylist. | Here, it looks like it uses an arraylist, which is a dynamically resizing array. It could possibly be a dynamic data structure. |
| S3 | With linked nodes, this code first checks that the first node has a value and isn’t null. While it isn’t null, the string gets set to the current node value and printed out. Current is set to the next node and the process repeats until there are no more string values to read. | It looks like a linked list type of data structure is used here, especially when considering the nodes. |

Q2: Although different data structures are used in three examples, the function of them are exactly the same – looping over a collection of city names. Try to identify some commonalities among these examples with regards to how to realize the looping mechanism, i.e., what are the necessary steps regardless of what data structure is used?.

**Commonality 1:**

In all of these examples, we would need to be able to reset the "pointer" (or cursor) to the first element.

**Commonality 2:**

In all of these examples, we would need to be able to determine if there are any more elements in the sequence.

**Commonality 3:**

In all of these examples, we would need to be able to move the "pointer" (or cursor) to the next element.

**Commonality 4:**

In all of these examples, we would need to be able to retrieve the "current" element.

Q3: Suppose you need to design the three data structures, given the fact that they share the above commonalities when it comes to looping, how would you design these data structures so that they are loosely-coupled, easy to maintain, and more likely to be reused?

Hint: Try to hide the internal data structures from the looping mechanism itself using abstraction.

Assuming, we want to hide the internal data structures from the looping mechanism using abstraction and still access the elements of an aggregate object while doing so, I would design these data structures using the iterator design pattern and, in this case, use the iterable or iterator interface. This interface will allow me to replicate the commonalities that all of the data structures above share in a way that is loosely-coupled, easier to maintain, and easier to reuse.

# Model 2 The Composite Pattern

Download file “*In-Class Activity 6.vsdx*”.



Q1: Open the Visio file. On worksheet “Coloring”, using the given shapes, try to create a drawing in Visio as close to the above interface as you can.

Hint: Try to use the “Group” function to speed up the process. How many “Groups” do you need to create?

I only needed to create 3 of them.

\*I have the completed drawing in a separate file that I submitted!

Q2: Suppose each button in the interface is an atomic shape (meaning it cannot contain any other shapes). “*The groups you used in Q1 are different, they are NOT atomic.*” Is this statement true? Justify your answer.

This statement is true. The reason for this is because groups can be modified to include more shapes and buttons if needed. Thus, I would say that the groups in Q1 are not atomic because of the fact that the groups can still be expanded to contain other shapes.

Q3: Now let’s think of the design of Visio. Since a *group* is sorts of a *shape*, like a *rectangle.* We can abstract out their commonalities into an interface, named “*Component*”, which contains a *show()* method. In the same visio file, on worksheet “*Design*”, please try to complete the following given design by adding more (if necessary) relationships, such as implementation, aggregation, etc.

Hint: How to implement the non-atomic aspect of *Group*? i.e. what is a group made of?



A picture containing diagram

Description automatically generated

Done!