NIO Android Developer Co-Op - Preliminary Assignment

Summary

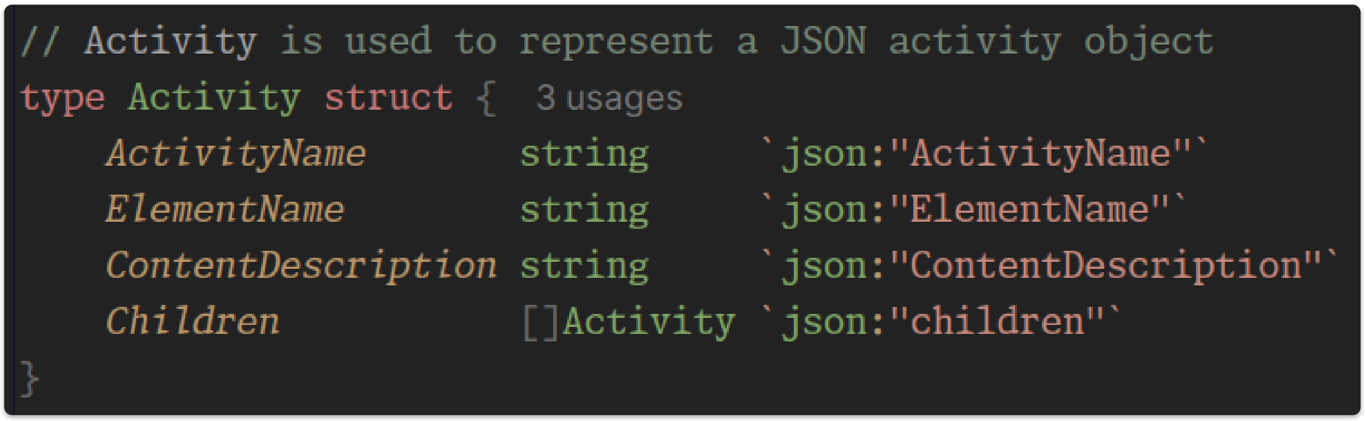
My implementation of the algorithm detailed in this assignment was done in the Go programming language. Go is a programming language I would like to get better at, and I saw this as an opportunity to get more practice with it.

The algorithm implemented is relatively simple and could be implemented in Java for use in an Android codebase with ease. When adapting this implementation to work in Java, the main work would be with deserializing the JSON (Google GSON could be used, or potentially a custom parser), and also with adapting test cases to use JUnit tests statically (local unit tests in Android Studio).

Implementation

To begin, I created a file to put my code in, `parse/parse.go`.

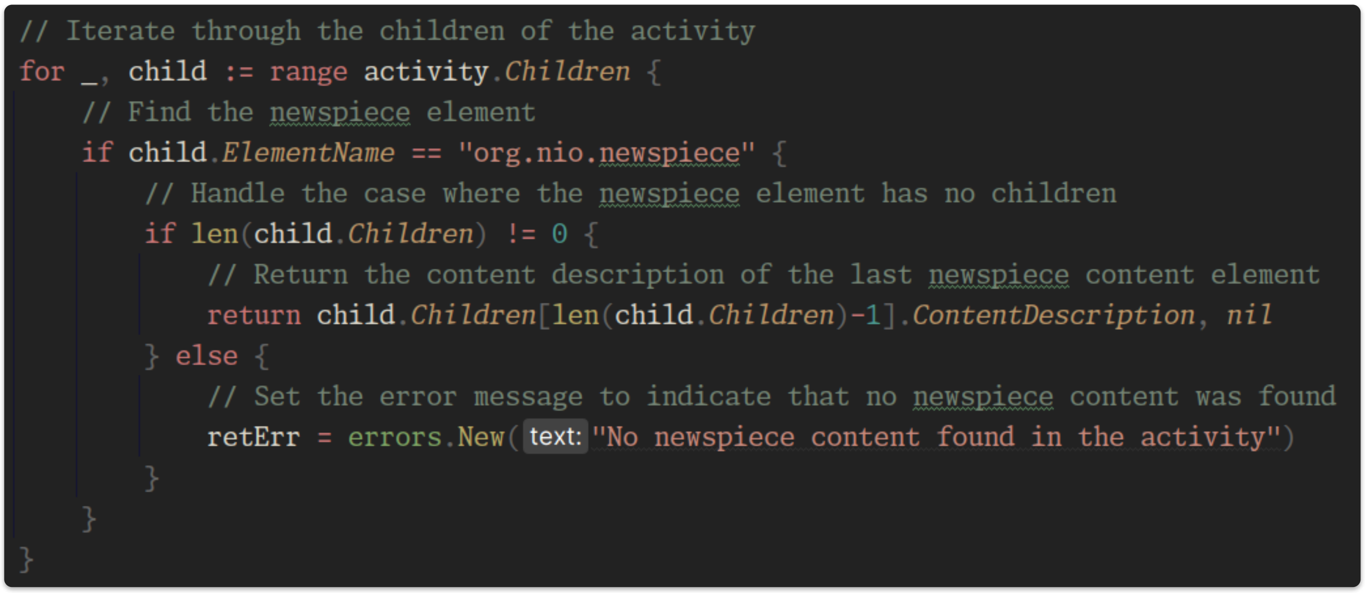
When breaking down this problem, the first thing I had to do was get the JSON string representation into an easily usable format. To do this, I created a my own `type`, "Activity". This type is a `struct` that has corresponding bindings for JSON keys to allow for easy deserialization.



```go
// Activity is used to represent a JSON activity object
type Activity struct {  3 usages
    ActivityName       string     `json:"ActivityName"`
    ElementName        string     `json:"ElementName"`
    ContentDescription string     `json:"ContentDescription"`
    Children           []Activity `json:"children"`
}
```

Now, using the `json.Unmarshal()` function in the Go standard library, a variable of this type can be easily populated with data from the JSON string.

With that out of the way, it's time to actually find the desired string, the newspiece content. To do this, I iterate over all of the children in the root activity, searching for a child with the `elementName` of `org.nio.newspiece`. This will allow for the algorithm to determine the root of the newspiece.



```
// Iterate through the children of the activity
for _, child := range activity.Children {
    // Find the newspiece element
    if child.ElementName == "org.nio.newspiece" {
        // Handle the case where the newspiece element has no children
        if len(child.Children) != 0 {
            // Return the content description of the last newspiece content element
            return child.Children[len(child.Children)-1].ContentDescription, nil
        } else {
            // Set the error message to indicate that no newspiece content was found
            retErr = errors.New( text: "No newspiece content found in the activity")
        }
    }
}
```

After finding the element matching that condition, we first assert that the newspiece has children. This is important for handling inputs that may be invalid. If there are no children, we update an error value which will be returned unless another `org.nio.newspiece` is found.

In the event that a newspiece has children, we know there's likely a title and body. Still, in order to ensure compatibility in case there are more elements above the content, we simply return the content description of the last element. This allows for the algorithm to still work on inputs without titles or with additional fields in some cases.

Outside of this, there is a little bit more error handling to ensure safe operation.

Analysis

The implementation of the algorithm I created has a run-time complexity of $O(n+m)$, where n is the number of characters in the input string, and m is the number of children in the root activity.

The $O(n)$ run-time comes from the call to `json.Unmarshal()`, as it must iterate over each character in the input string and parse it into a data structure. This likely could be optimized, at a cost of the safety that `json.Unmarshal()` provides. Input validation is very important when designing algorithms and systems, so personally this is not something worth trading for a marginally more performant approach. By using this method to deserialize the input string to a `Activity` type, valid JSON input is ensured, and we should not be working with any input that is invalid format.

The $O(m)$ run-time comes from the for each loop that iterates over each child in the root activity.

Together, these make for a run-time complexity of $O(n+m)$.

Testing

In order to test my implementation, I used the Go programming language's built in unit testing functionality.

In the file `parse/parse_test.go`, I created a series of test cases that server to ensure the algorithm functions properly, even in the event of edge cases:

1. `TestParseNewsContentInvalidJSON`: Tests the algorithm when passed an invalid JSON string.
2. `TestParseNewsContentEmptyString`: Tests the algorithm when passed an empty string.
3. `TestParseNewsContentGivenString`: Tests the algorithm when passed valid JSON data.
4. `TestParseNewsContentNoNewsPiece`: Tests the algorithm when passed valid JSON data with no newspiece (`org.nio.newspiece` missing).
5. `TestParseNewsContentNoNewsPieceChildren`: Tests the algorithm when passed valid JSON data with no newspiece children.

These tests all pass and lead to 100% test coverage in my implementation.

Additional Testing

Additional test cases, for example, with missing JSON keys should still return an error properly from my algorithm, but I did not write these test cases. Given the assignment and what was provided, it seems as if it can be assumed that this would not be a possibility.