Simplify Path Problem Explanation

Understanding the Problem

In Unix-style file systems, paths can contain special symbols that need to be interpreted:

- `/` - directory separator
- `.` - current directory
- `..` - parent directory

We need to convert a complex path into its simplest canonical form by:

1. Removing redundant slashes (`//` → `/`)
2. Handling `.` (ignore) and `..` (go up one level)
3. Preserving valid directory names like `...`, `....`

Approach

We'll use a **stack** to keep track of the valid directories in the path:

1. **Split the path** by `/` to get all components
2. **Process each component**:
   - Ignore empty strings and `.` (they don't change the directory)
   - For `..`, pop from stack (go up one level)
   - For other valid names, push to stack
3. **Reconstruct the path** from the stack

Step-by-Step Solution

java Copy Download

```java
import java.util.*;

class Solution {
    public String simplifyPath(String path) {
        // Split the path by slashes
        String[] components = path.split("/");
        Stack<String> stack = new Stack<>();

        // Process each component
        for (String component : components) {
            // Skip empty strings and current directory "."
            if (component.isEmpty() || component.equals(".")) {
                continue;
            }

            // Handle parent directory ".."
            if (component.equals("..")) {
                if (!stack.isEmpty()) {
                    stack.pop(); // Go up one level
                }
            }
            // Handle valid directory/file names
            else {
                stack.push(component);
            }
        }

        // Reconstruct the canonical path
        StringBuilder result = new StringBuilder();
        for (String dir : stack) {
            result.append("/").append(dir);
        }
```

java Copy Download

```java
            // Handle valid directory/file names
            else {
                stack.push(component);
            }
        }

        // Reconstruct the canonical path
        StringBuilder result = new StringBuilder();
        for (String dir : stack) {
            result.append("/").append(dir);
        }

        // Handle root directory case
        return result.length() == 0 ? "/" : result.toString();
    }
}
```

Dry Run with Examples

Example 1: `/home/`

text Copy Download

```text
Components: ["", "home", ""]
Stack: ["home"]
Result: "/home"
```

Example 2: `/home//foo/`

text Copy Download

```text
Components: ["", "home", "", "foo", ""]
Stack: ["home", "foo"]
Result: "/home/foo"
```

Example 3: `/home/user/Documents/../Pictures`

text Copy Download

```text
Components: ["", "home", "user", "Documents", "..", "Pictures"]
Stack:
  Start: []
  Push "home" → ["home"]
  Push "user" → ["home", "user"]
  Push "Documents" → ["home", "user", "Documents"]
  ".." → pop → ["home", "user"]
  Push "Pictures" → ["home", "user", "Pictures"]
Result: "/home/user/Pictures"
```

Example 4: `/../`

text

```text
Components: ["", "..", ""]
Stack: [] (pop from empty stack does nothing)
Result: "/"
```

Example 5: `/.../a/../b/c/../d/./`

text Copy Download

```text
Components: ["", "...", "a", "..", "b", "c", "..", "d", ".", ""]
Stack:
  Push "..." → ["..."]
  Push "a" → ["...", "a"]
  ".." → pop → ["..."]
  Push "b" → ["...", "b"]
  Push "c" → ["...", "b", "c"]
  ".." → pop → ["...", "b"]
  Push "d" → ["...", "b", "d"]
  "." → skip
Result: "/.../b/d"
```

Key Concepts

1. **Stack Data Structure:** Perfect for handling directory navigation (LIFO)
2. **String Splitting**: `/` as delimiter helps isolate directory components
3. **Edge Case Handling**:
   - Root directory (`/`)
   - Empty components from multiple slashes
   - Valid multi-dot names (`...`, `....`)

Time Complexity

- **O(n)**: We process each character in the path once
- **O(n) space**: Stack stores directory names

This approach efficiently handles all the Unix path simplification rules while maintaining clean, readable code.