Creative Software Design, Assignment 13-2

Deadline: 2024-11-26 23:59 (No score for late submission)

- Submit your homework by uploading your zip file to the LMS assignment section. Below is an example.

```
13178_Assignment1-1_2024123456.zip
├─ 1.cc
├─ 2.cc
├─ 3.cc
└─ ...
```

- Your zip file name should follow this format:
  13178_Assignment[Assignment-number]_[Student-ID].zip
  - Ex. 13178_Assignment1-1_2024123456.zip
- Source files should be named as **<filename>.cc** *or* **<filename>.cpp**
- **You must submit your solution in the zip file before the deadline.**

1. **File System Simulation with Smart Pointers**

A. Task Overview

Develop a file system simulation in C++ that challenges your understanding of smart pointers, particularly in managing ownership and preventing memory leaks due to cyclic references using std::weak_ptr. This assignment involves creating a hierarchical file system with directories and files, implementing functionalities as demonstrated in the provided code sample. The focus is on effectively using std::shared_ptr and std::weak_ptr to manage ownership, ensuring proper resource management without memory leaks.

B. Implement Classes

1. `FileSystemException` Class

i. Contains an error message.

ii. Implement a method `print()` to print the error message.

2. `FileSystemObject` Class

- Attributes
  - i. `std::string name`
  - ii. `std::weak_ptr<Directory> parent`: references the parent directory.
- Methods
  - i. Constructor to initialize the `name`.
  - ii. Virtual destructor.
  - iii. `void display(int indentLevel) const`: pure virtual function that displays the object with indentation.
  - iv. `bool isDirectory() const`: pure virtual function that returns whether the object is a directory or not.
  - v. `std::string getName() const`: gets the name of the object.
  - vi. `void setParent(const std::weak_ptr<Directory>& parentDir)`: sets the parent directory.

3. `File` Class (Derived from `FileSystemObject`)

- Attributes
  - i. `int size`: representing the file size in bytes.

- Methods
  i. Constructor to initialize the `name` and `size`.
  ii. Virtual destructor.
  iii. Override `void display(int indentLevel) const` to display the file's name and size with proper indentation.

4. **`Directory` Class** (Derived from `FileSystemObject`)
   - Attributes
     i. `std::vector<std::shared_ptr<FileSystemObject>> contents`
   - Methods
     i. Constructor to initialize the `name`.
     ii. Virtual destructor.
     iii. `void add(const std::shared_ptr<FileSystemObject>& obj, const std::shared_ptr<Directory>& self)`: adds a file or directory to the contents.
       a. Check for duplicate names in the current directory.
       b. Set the `parent` of `obj` to this directory using `std::weak_ptr`.
       c. Add `obj` to `contents`.
     iv. `void remove(const std::string& name)`: removes an object by name.
     v. Override `void display(int indentLevel) const` to display the directory's name and **recursively** display its contents.
     vi. `std::shared_ptr<FileSystemObject> find(const std::string& name)`: finds an object by name relative to this directory.
       - Search **recursively** in **subdirectories**.
       - Return *nullptr* if not found.
       - Hint: `std::dynamic_pointer_cast<T>()`

C. Implement Functionalities

1. **File System Construction**:
   i. Create a root directory.

ii. Add multiple files and subdirectories to the root and its subdirectories.

iii. When adding an object, set its `parent` pointer appropriately.

2. **Display** Functionality:

   i. Implement the `display` method to show the entire file system hierarchy with indentation representing directory levels.

3. **Removal** Functionality:

   i. Remove files and directories by name.

   ii. Ensure that removing a directory also removes all its contents and doesn't leave dangling references.

4. **Find** Functionality:

   i. Implement the `find()` method to search for files or directories by name, searching recursively through subdirectories

D. Exception Handling

1. **Adding Objects**:

   When attempting to add an object with a duplicate name in the same directory, throw a `FileSystemException` with an appropriate error message (e.g., "Cannot add 'filename': Duplicate name.").

2. **Removing Objects**:

   If the object to be removed is not found in the directory, throw a `FileSystemException` with an appropriate error message (e.g., "Cannot remove 'filename': File or directory not found.").

E. Additional Requirements

1. Ensure no memory leaks occur due to improper handling of smart pointers.

2. Be cautious with object lifetimes and dangling references.

F. Example of the `main()` function:

```
int main() {
    try {
        // Create root directory
        std::shared_ptr<Directory> root(new Directory("root"));

        // Add files to root
        root->add(std::shared_ptr<File>(new File("file1.txt", 100)), root);
        root->add(std::shared_ptr<File>(new File("file2.txt", 200)), root);

        // Add subdirectory to root
        std::shared_ptr<Directory> subDir(new Directory("subdir"));
        root->add(subDir, root);

        // Add files to subdirectory
        subDir->add(std::shared_ptr<File>(new File("file3.txt", 300)), subDir);

        // Display file system
        std::cout << "File system structure:\n";
        root->display(0);

        // Attempt to add a duplicate file to root (should throw exception)
        try {
            root->add(std::shared_ptr<File>(new File("file1.txt", 150)), root);
        } catch (const FileSystemException& e) {
            e.print();
        }

        // Attempt to add a duplicate directory to root (should throw exception)
        try {
            root->add(std::shared_ptr<Directory>(new Directory("subdir")), root);
        } catch (const FileSystemException& e) {
            e.print();
        }

        // Attempt to remove a non-existent file (should throw exception)
        try {
            root->remove("nonexistent.txt");
        } catch (const FileSystemException& e) {
            e.print();
        }

        // Remove a file
        root->remove("file1.txt");

        // Remove a subdirectory
        root->remove("subdir");

        // Display file system after removals
        std::cout << "\nFile system after removals:\n";
        root->display(0);

        // Attempt to display a removed directory (should not find it)
        std::shared_ptr<FileSystemObject> removedDir = root->find("subdir");
        if (!removedDir) {
            std::cout << "\nSubdirectory 'subdir' not found after removal." << std::endl;
        }

    } catch (const FileSystemException& e) {
        e.print();
    }

    return 0;
}
```

G. Example output of your program (Bold text indicates user input):

```
File system structure:
Directory: root
  File: file1.txt (100 bytes)
  File: file2.txt (200 bytes)
  Directory: subdir
    File: file3.txt (300 bytes)

Checking isDirectory method:
root is a directory.
file2.txt is a file.
Cannot add 'file1.txt': Duplicate name.
Cannot add 'subdir': Duplicate name.
Cannot remove 'nonexistent.txt': File or directory not found.

File system after removals:
Directory: root
  File: file2.txt (200 bytes)

Subdirectory 'subdir' not found after removal.
```

H. Submission file: one C++ source file (File name: 1.**cc** or 1.**cpp**)