Programming Paradigms Ruby Homework

Johann Gamper Raditya Eko Prasojo Thomas Tschager

1st Semester 2018/19

Object-oriented design patterns typically show relationships and interactions between classes or objects, without specifying the final application classes or objects that are involved. In particular, the *Composite* design pattern describes how to treat a group of objects in the same way as a single object of this type. You can find more information about the Composite pattern in the wiki page: `https://en.wikipedia.org/wiki/Composite_pattern`.

We have already seen an example implementation of the Composite pattern in the lecture for implementing a tree. In this exercise you are asked to implement a file system data structure using the Composite design pattern. Your file system consists of objects that are either folders or files and structured in a tree-like hierarchy. A folder can contain one or many other folders and files, while a file is always a leaf node of the file system tree, i.e. it cannot have descendants in the file system. Files have a size that can be modified at any time. The size of a folder is defined as the sum of the sizes of all files in this folder and its subfolders.

You must implement at least the following classes:

1. An class *FileSystemObject* which models the objects (files or folders) of your file system. The class must have

   - a field `name`,
   - a field `size`, which represents the size of the object,
   - a field `children`, that is a list of `Objects`, and
   - a method `traverse`, which prints out all objects using a recursive breadth-first-search traverse the file system.

   Furthermore, ensure that objects are printed using their name (*hint:* `puts obj` uses `obj.to_s()` to convert the object to a string).

2. A class *FolderObject* that inherits from `FileSystemObject` and represents the folders of your file system. The class must have a method `add` that is used to add a file or a folder (updating also the size of itself and its ancestors), and a method `removeAll` that removes all files and folders that it contains and again updates the size of itself and its ancestors.

3. A class *FileObject* that inherits from `FileSystemObject` and represents the files of your file system. It cannot have children and must have a parent folder. The size of a file can be modified after initialization using a method `setSize` (again updating the parent folder's size).

Finally, create a module that checks the names of your files and folders. The names of files and folders must not include any of the following characters: / >< | : &. The module must contain a function `isValid?` to check the whether a given object name is valid or not. Include the module to the mixins of the `FileSystemObject` class. Modify the method `traverse` so that it additionally prints a warning for each FileSystemObject with a name that is not valid.

Deadline: 06.11.2018, 23:55 (OLE)