TECHNISCHE UNIVERSITÄT DRESDEN

Faculty of Computer Science Institute of Software and Multimedia Technology, Software Technology Group

WS2019/20 – Design Patterns and Frameworks

Role-Based Modeling for Design Patterns (Part 2)

Professor: Prof. Dr. Uwe Aßmann
Lecturer: Dr.-Ing. Sebastian Götz
Tutor: Dr. rer. nat. Marvin Triebel

Task 1 Role-based Design Pattern Catalog

This exercise focuses on finally applying the *Compartment Role Object Model* (CROM) [3] to formalize and compose the various design patterns discussed in this course. In detail, the first task is to create role-based models of various design patterns, ultimately, creating a role-based design pattern catalog similar to Riehle's design pattern catalog [4].

a) Design a *compartment type* and role model for the `Observer` design pattern.

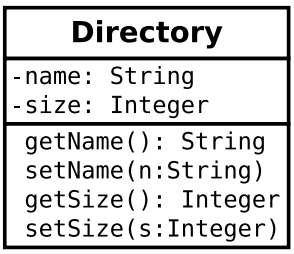
b) Design a *compartment type* and role model for the `Bridge` design pattern.

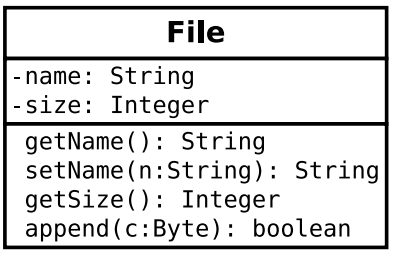
c) Design a *compartment type* and role model for the `Composite` design pattern.

Task 2 Applying and Composing Design Patterns

Applying a role-based design pattern entails assigning the domain classes to play the appropriate role types from the design pattern.

In detail, this task focuses on a **file system** including *files* and *directories*. Both files and directories have a modifiable name and return their size in Bytes. While files return the size of their content, directories should return the accumulated size of all containing directories and files. The properties of the `File` and `Directory` classes are depicted below, whereas both are considered natural types.





Thus far, the `Directory` class does not encompass files and directories. Yet, neither `Directories` nor `Files` do notify changes to their name or size to their parent directories. Moreover, size changes should be propagated up to the root of the directory hierarchy.

a) First, apply the role-based `Composite` design pattern to organize the file system's structure, such that `Directories` can contain both `Files` and other `Directories`.

b) Second, apply the role-based `Observer` design pattern to propagate any size changes of a `File` up to its parent `Directory`.

c) Combine the role models of both the `Observer` and the `Composite` compartment type by means of *role constraints*, to create a sound combination of both design patterns in a new compartment type.

d) Reapply the combined design pattern to the `File` and `Directory` classes of the file system. Compare this solutions to their separate application.

References

[1] Dirk Bäumer, Dirk Riehle, Wolf Siberski, and Martina Wulf. The role object pattern. In *Washington University Dept. of Computer Science*. Citeseer, 1998.

[2] Erich Gamma. Extension object. In *Pattern languages of program design 3*, pages 79–88. Addison-Wesley Longman Publishing Co., Inc., 1997. URL https://www.ecs.syr.edu/faculty/fawcett/handouts/CSE776/PatternPDFs/ExtensionObject.pdf.

[3] Thomas Kühn, Böhme Stephan, Sebastian Götz, and Uwe Aßmann. A combined formal model for relational context-dependent roles. In *Proceedings of the 2015 ACM SIGPLAN International Conference on Software Language Engineering*, pages 113–124. ACM, 2015. doi: 10.1145/2814251.2814255. URL http://dl.acm.org/citation.cfm?id=2814255.

[4] Dirk Riehle. A role-based design pattern catalog of atomic and composite patterns structured by pattern purpose. Ubilab Technical Report 97.1. 1., Union Bank of Switzerland, Zürich, Switzerland, 1997. URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.496.7976&rep=rep1&type=pdf.

[5] Yannis Smaragdakis and Don Batory. Implementing layered designs with mixin layers. In *European Conference on Object-Oriented Programming*, pages 550–570. Springer, 1998. URL citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.23.7182&rep=rep1&type=pdf.