Software Design
Second Opportunity Assignment
(2020-2021)

Student name: Beatrix-Valeria Csaki

Iterator Pattern

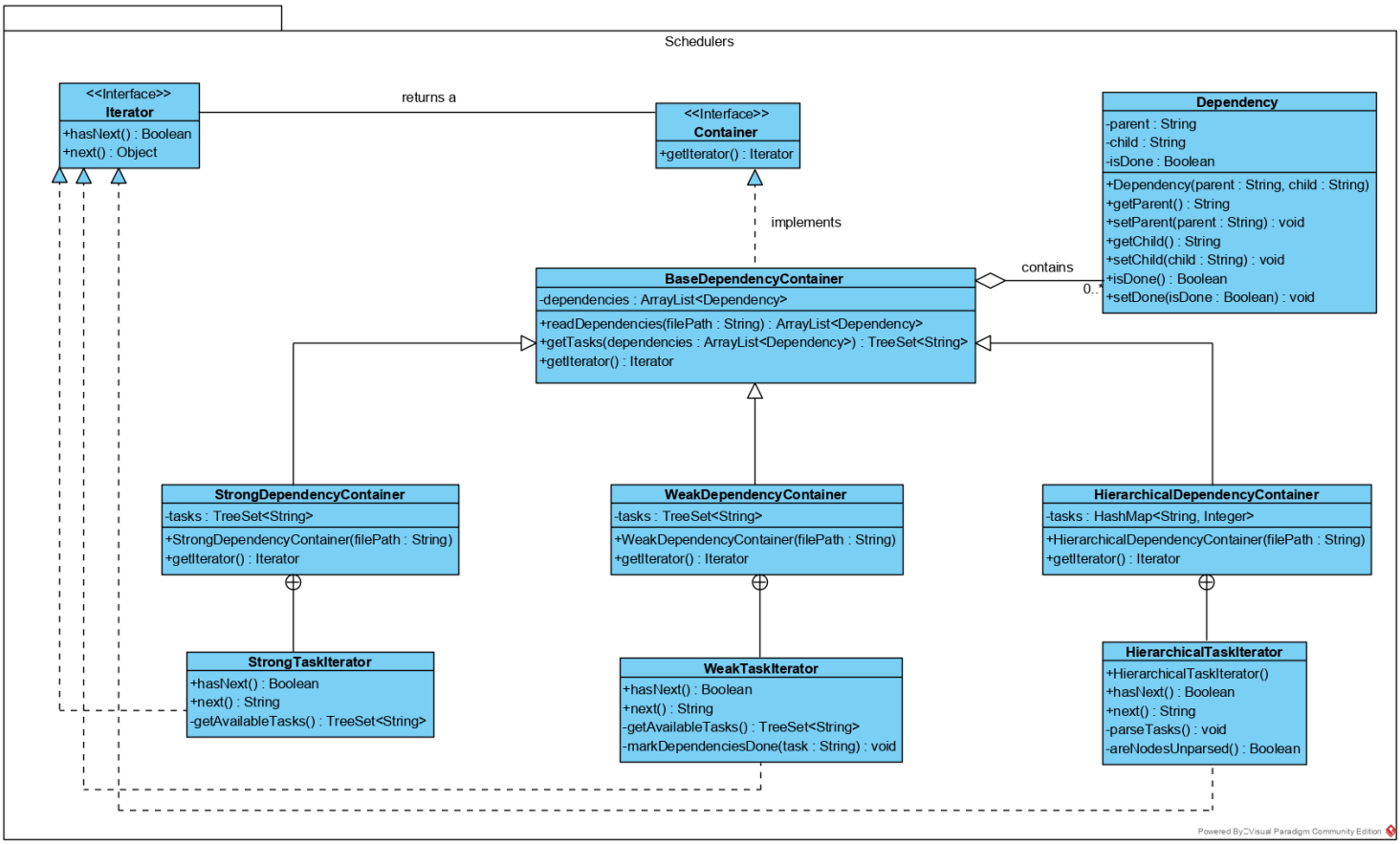
Brief explanation:

The iterator pattern takes part in the category of behavioral patterns and it provides a way to access the elements of a collection object in sequential manner without any need to know its underlying representation.

To solve the proposed problem, we first have to look at its requirements. The solution needs to provide a way that will allow as input a text file which contains the dependencies. The output will be represented by the read tasks in the correct order. To obtain the correct order (strong dependency, weak dependency or hierarchical order) we need a way to parse a container that can store the dependencies and the tasks. In this sense, the Iterator pattern gives us a good design start for the solution.

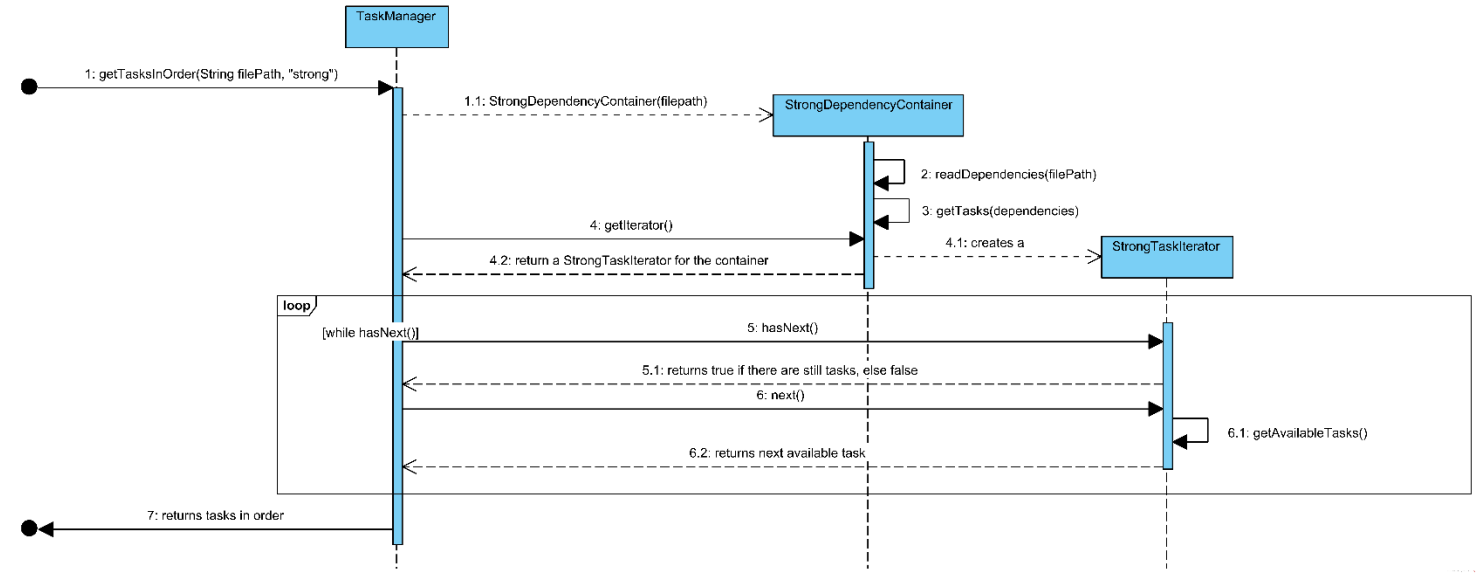
Each parsing order will be implemented by a container that has an internal iterator and a way to return this iterator. This internal component will implement the custom logic of the order. The classes StrongDependencyContainer, WeakDependencyContainer and HierarchicalDependencyContainer each has such an iterator. Those containers also have a parent class BaseDependencyContainer that provides a way to read the dependencies from the file and to get the tasks.

Class diagram:

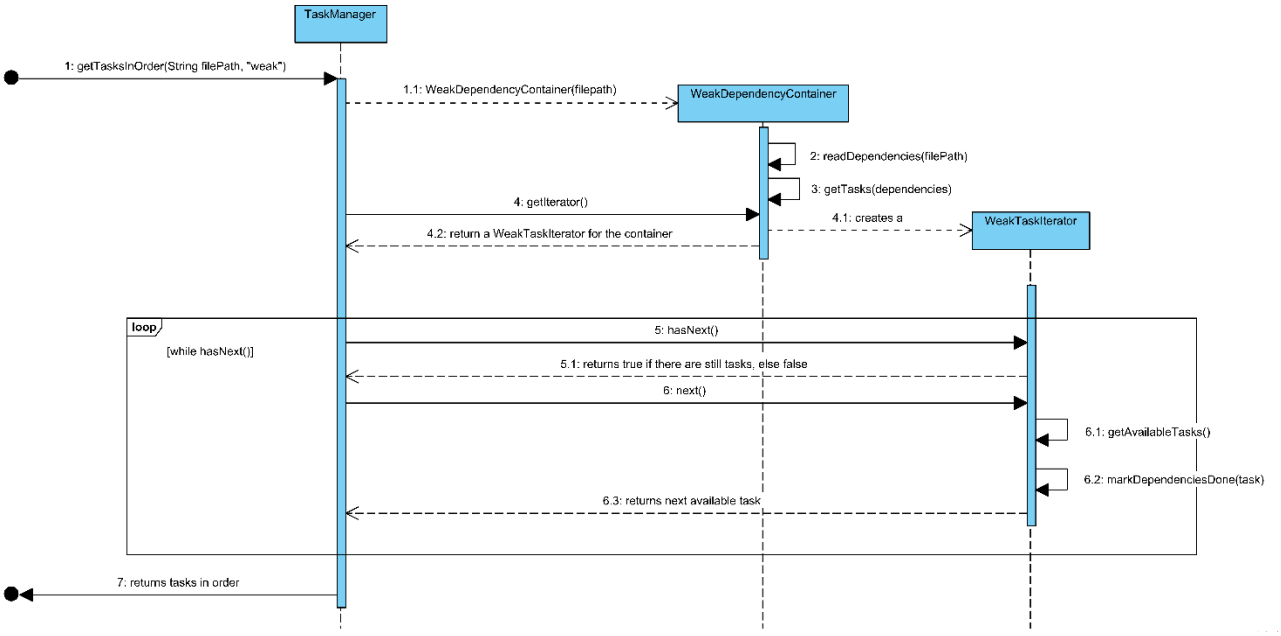


Sequence Diagram:

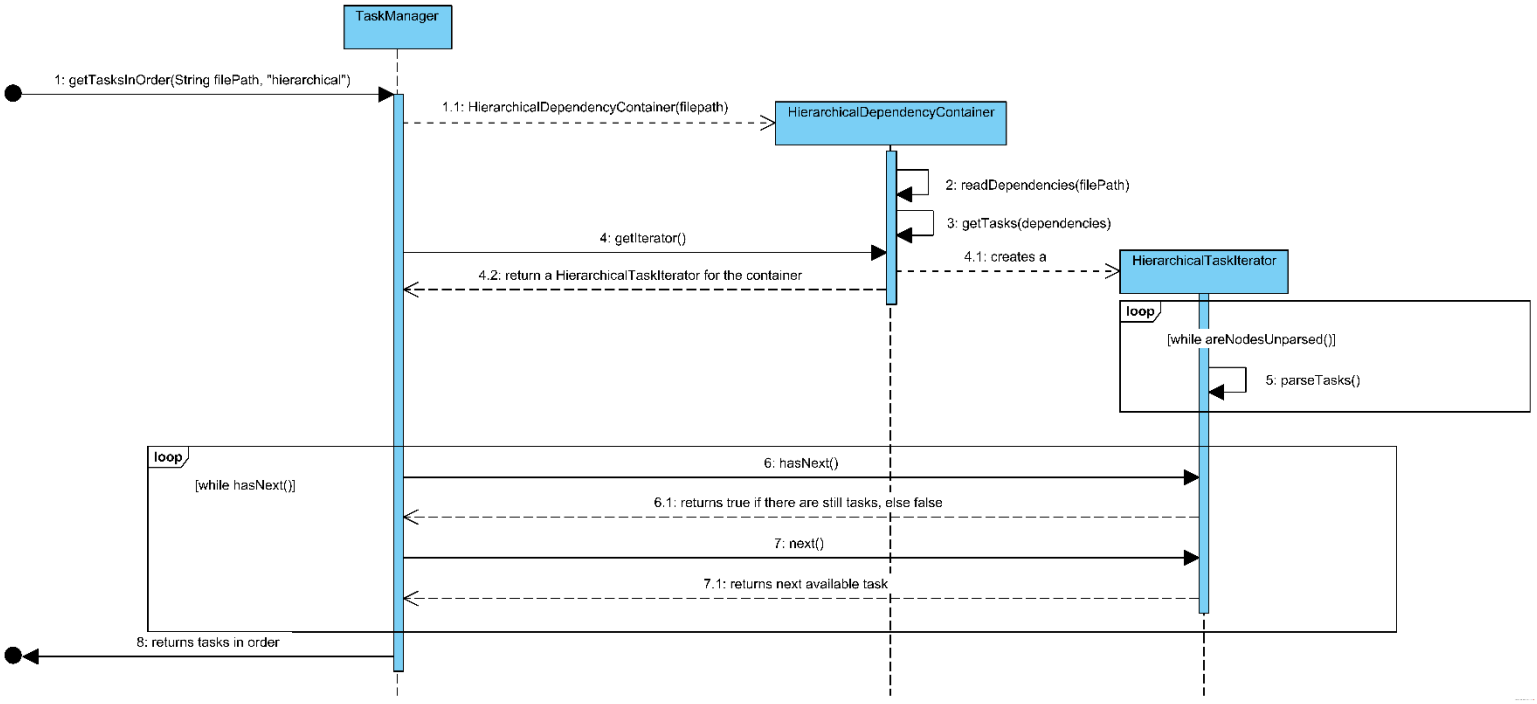
1. Strong Dependency Order



2. Weak Dependency Order



3. Hierarchical Order



SOLID principles:

Single Responsibility Principle

This principle is used by the three classes which inherit the BaseDependencyContainer class. Each (StrongDependencyContainer, WeakDependencyContainer and HierarchicalDependencyContainer) is responsible for executing the specific algorithm for the specified order.

Open Closed Principle

The current implementation allows the addition of a new specific order of task to be implemented without changing the current code. This can be done by inheriting the class BaseDependencyContainer and creating an inner class that implements the Iterators interface. Also, in the class Task manager another case needs to be added.

The Liskov Substitution Principle

The application of this principle can be seen in the TaskManager class in the function getTasksInOrder where a variable container of type BaseDependencyContainer is declared but as values gets objects of type: StrongDependencyContainer, WeakDependencyContainer, HierarchicalDependencyContainer. This allows the usage of the method getIterator that returns an Iterator which has the methods: hasNext() and next().