

Memento Pattern

It is sometimes necessary to capture the internal state of an object at some point and have the ability to restore the object to that state later in time. Such a case is useful in case of error or failure. Consider the case of a calculator object with an undo operation such a calculator could simply maintain a list of all previous operation that it has performed and thus would be able to restore a previous calculation it has performed. This would cause the calculator object to become larger, more complex, and heavyweight, as the calculator object would have to provide additional undo functionality and should maintain a list of all previous operations. This functionality can be moved out of the calculator class, so that an external (let's call it undo manager class) can collect the internal state of the calculator and save it. However providing the explicit access to every state variable of the calculator to the restore manager would be impractical and would violate the encapsulation principle.

Motivation

It is sometimes necessary to capture the internal state of an object at some point and have the ability to restore the object to that state later in time. Such a case is useful in case of error or failure. Consider the case of a calculator object with an undo operation such a calculator could simply maintain a list of all previous operation that it has performed and thus would be able to restore a previous calculation it has performed. This would cause the calculator object to become larger, more complex, and heavyweight, as the calculator object would have to provide additional undo functionality and should maintain a list of all previous operations. This functionality can be moved out of the calculator class, so that an external (let's call it undo manager class) can collect the internal state of the calculator and save it. However providing the explicit access to every state variable of the calculator to the restore manager would be impractical and would violate the encapsulation principle.

Intent

- The intent of this pattern is to capture the internal state of an object without violating encapsulation and thus providing a mean for restoring the object into initial state when needed.

Implementation

The figure below shows a UML class diagram for the Memento Pattern:

- Memento
 - Stores internal state of the Originator object. The state can include any number of state variables.
 - The Memento must have two interfaces, an interface to the caretaker. This interface must not allow any operations or any access to internal state stored by the memento and thus honors encapsulation. The other interface is to the originator and allows the originator to access any state variables necessary to for the originator to restore previous state.
- Originator
 - Creates a memento object capturing the originators internal state.
 - Use the memento object to restore its previous state.
- Caretaker

- Responsible for keeping the memento.
- The memento is opaque to the caretaker, and the caretaker must not operate on it.

A Caretaker would like to perform an operation on the Originator while having the possibility to rollback. The caretaker calls the `createMemento()` method on the originator asking the originator to pass it a memento object. At this point the originator creates a memento object saving its internal state and passes the memento to the caretaker. The caretaker maintains the memento object and performs the operation. In case of the need to undo the operation, the caretaker calls the `setMemento()` method on the originator passing the maintained memento object. The originator would accept the memento, using it to restore its previous state.

Applicability & Examples

The memento pattern is used when a snapshot of an object's state must be captured so that it can be restored to that state later and in situations where explicitly passing the state of the object would violate encapsulation.

Example - Simple Calculator with Undo Operation.

This simple example is a calculator that finds the result of addition of two numbers, with the additional option to undo last operation and restore previous result.

Source: [Click here to see java source code](#)

Specific problems and implementation

Database Transactions

Transactions are operations on the database that occur in an atomic, consistent, durable, and isolated fashion. A transaction can contain multiple operations on the database; each operation can succeed or fail, however a transaction guarantees that if all operations succeed, the transaction would commit and would be final. And if any operation fails, then the transaction would fail and all operations would rollback and leave the database as if nothing has happened.

This mechanism of rolling back uses the memento design pattern. Consider an object representing a database table, a transaction manager object which is responsible of performing transactions must perform operations on the table object while having the ability to undo the operation if it fails, or if any operation on any other table object fails. To be able to rollback, the transaction manager object would ask the table object for a memento before performing an operation and thus in case of failure, the memento object would be used to restore the table to its previous state.

Consequences

Memento protects encapsulation and avoids exposing originator's internal state and implementation. It also simplifies originator code such that the originator does not need to keep track of its previous state since this is the responsibility of the CareTaker.

Using the memento pattern can be expensive depending on the amount of state information that has to be stored inside the memento object. In addition the caretaker must contain additional logic to be able to manage mementos.

Related Patterns

Command Pattern - Commands can use mementos to maintain state for undoable operations.

Known Uses

Undo and restore operations in most software.

Database transactions discussed earlier.