Memento in C++

Memento is a behavioral design pattern that allows making snapshots of an object’s state and restoring it in future.

The Memento doesn’t compromise the internal structure of the object it works with, as well as data kept inside the snapshots.

Learn more about Memento

Complexity:

Popularity:

Usage examples: The Memento’s principle can be achieved using serialization, which is quite common in C++. While it’s not the only and the most efficient way to make snapshots of an object’s state, it still allows storing state backups while protecting the originator’s structure from other objects.

Conceptual Example

This example illustrates the structure of the **Memento** design pattern. It focuses on answering these questions:

- What classes does it consist of?
- What roles do these classes play?
- In what way the elements of the pattern are related?

main.cc: Conceptual example

```
/**
 * The Memento interface provides a way to retrieve the memento's metadata, such
 * as creation date or name. However, it doesn't expose the Originator's state.
 */
class Memento {
 public:
  virtual ~Memento() {}
```

```cpp
  virtual std::string GetName() const = 0;
  virtual std::string date() const = 0;
  virtual std::string state() const = 0;
};

/**
 * The Concrete Memento contains the infrastructure for storing the Originator's
 * state.
 */
class ConcreteMemento : public Memento {
 private:
  std::string state_;
  std::string date_;

 public:
  ConcreteMemento(std::string state) : state_(state) {
    this->state_ = state;
    std::time_t now = std::time(0);
    this->date_ = std::ctime(&now);
  }
  /**
   * The Originator uses this method when restoring its state.
   */
  std::string state() const override {
    return this->state_;
  }
  /**
   * The rest of the methods are used by the Caretaker to display metadata.
   */
  std::string GetName() const override {
    return this->date_ + " / (" + this->state_.substr(0, 9) + "...)";
  }
  std::string date() const override {
    return this->date_;
  }
};

/**
 * The Originator holds some important state that may change over time. It also
 * defines a method for saving the state inside a memento and another method for
 * restoring the state from it.
 */
class Originator {
  /**
   * @var string For the sake of simplicity, the originator's state is stored
   * inside a single variable.
   */
 private:
  std::string state_;

  std::string GenerateRandomString(int length = 10) {
    const char alphanum[] =
        "0123456789"
        "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
        "abcdefghijklmnopqrstuvwxyz";
    int stringLength = sizeof(alphanum) - 1;

    std::string random_string;
    for (int i = 0; i < length; i++) {
      random_string += alphanum[std::rand() % stringLength];
    }
    return random_string;
  }

 public:
  Originator(std::string state) : state_(state) {
    std::cout << "Originator: My initial state is: " << this->state_ << "\n";
  }
  /**
   * The Originator's business logic may affect its internal state. Therefore,
   * the client should backup the state before launching methods of the business
   * logic via the save() method.
   */
  void DoSomething() {
    std::cout << "Originator: I'm doing something important.\n";
    this->state_ = this->GenerateRandomString(30);
    std::cout << "Originator: and my state has changed to: " << this->state_ << "\n";
  }

  /**
   * Saves the current state inside a memento.
   */
  Memento *Save() {
    return new ConcreteMemento(this->state_);
  }
  /**
   * Restores the Originator's state from a memento object.
   */
  void Restore(Memento *memento) {
    this->state_ = memento->state();
    std::cout << "Originator: My state has changed to: " << this->state_ << "\n";
  }
};

/**
 * The Caretaker doesn't depend on the Concrete Memento class. Therefore, it
 * doesn't have access to the originator's state, stored inside the memento. It
 * works with all mementos via the base Memento interface.
 */
class Caretaker {
  /**
   * @var Memento[]
   */
 private:
  std::vector<Memento *> mementos_;

  /**
   * @var Originator
   */
  Originator *originator_;

 public:
     Caretaker(Originator* originator) : originator_(originator) {
     }

     ~Caretaker() {
         for (auto m : mementos_) delete m;
     }

  void Backup() {
    std::cout << "\nCaretaker: Saving Originator's state...\n";
    this->mementos_.push_back(this->originator_->Save());
  }
  void Undo() {
    if (!this->mementos_.size()) {
      return;
    }
    Memento *memento = this->mementos_.back();
    this->mementos_.pop_back();
    std::cout << "Caretaker: Restoring state to: " << memento->GetName() << "\n";
    try {
      this->originator_->Restore(memento);
    } catch (...) {
      this->Undo();
    }
  }
  void ShowHistory() const {
    std::cout << "Caretaker: Here's the list of mementos:\n";
    for (Memento *memento : this->mementos_) {
      std::cout << memento->GetName() << "\n";
    }
  }
};
/**
 * Client code.
 */

void ClientCode() {
  Originator *originator = new Originator("Super-duper-super-puper-super.");
  Caretaker *caretaker = new Caretaker(originator);
  caretaker->Backup();
  originator->DoSomething();
  caretaker->Backup();
  originator->DoSomething();
  caretaker->Backup();
  originator->DoSomething();
  std::cout << "\n";
  caretaker->ShowHistory();
  std::cout << "\nClient: Now, let's rollback!\n\n";
  caretaker->Undo();
  std::cout << "\nClient: Once more!\n\n";
  caretaker->Undo();

  delete originator;
  delete caretaker;
}

int main() {
  std::srand(static_cast<unsigned int>(std::time(NULL)));
  ClientCode();
  return 0;
}
```

Output.txt: Execution result

```
Originator: My initial state is: Super-duper-super-puper-super.

Caretaker: Saving Originator's state...
Originator: I'm doing something important.
Originator: and my state has changed to: uOInE8wmckHYPwZS7PtUTwuwZfCIbz

Caretaker: Saving Originator's state...
Originator: I'm doing something important.
Originator: and my state has changed to: te6RGmykRpbqaWo5MEwjji1fpM1t5D

Caretaker: Saving Originator's state...
Originator: I'm doing something important.
Originator: and my state has changed to: hX5xWDVljcQ9ydD7StUfbBt5Z7pcSN

Caretaker: Here's the list of mementos:
Sat Oct 19 18:09:37 2019
 / (Super-dup...)
Sat Oct 19 18:09:37 2019
 / (uOInE8wmc...)
Sat Oct 19 18:09:37 2019
 / (te6RGmykR...)

Client: Now, let's rollback!

Caretaker: Restoring state to: Sat Oct 19 18:09:37 2019
 / (te6RGmykR...)
Originator: My state has changed to: te6RGmykRpbqaWo5MEwjji1fpM1t5D

Client: Once more!
```

```
Caretaker: Restoring state to: Sat Oct 19 18:09:37 2019
 / (uOInE8wmc...)
Originator: My state has changed to: uOInE8wmckHYPwZS7PtUTwuwZfCIbz
```