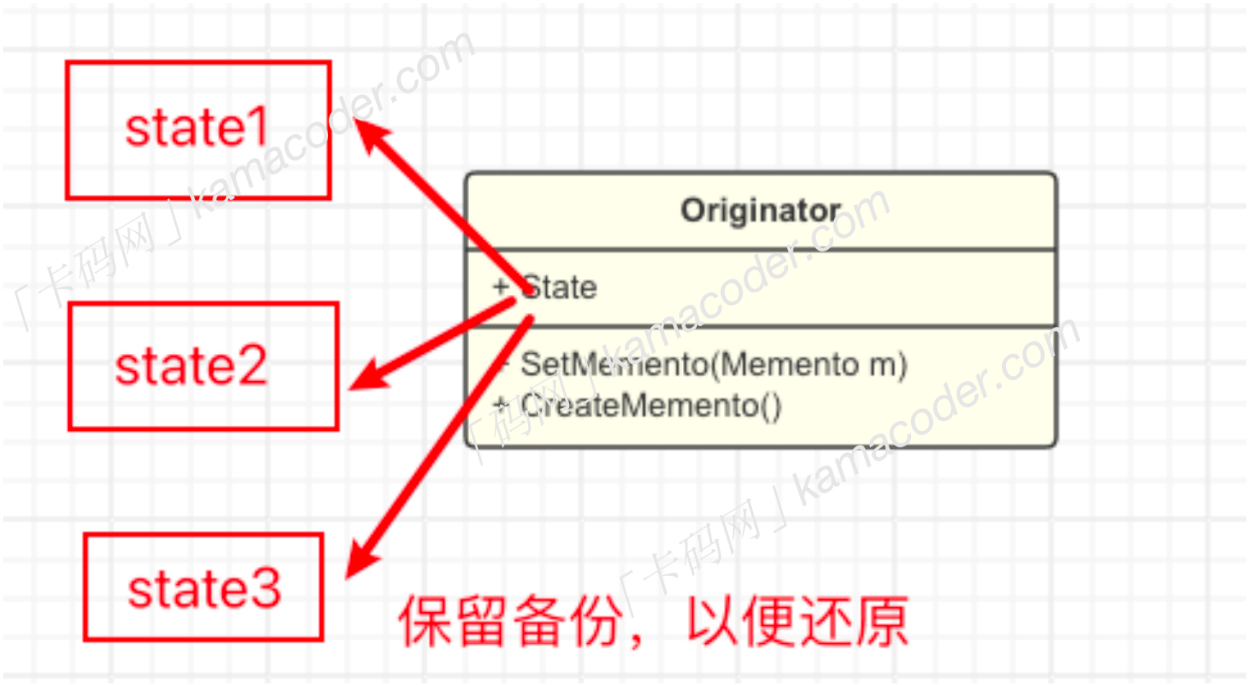
18. 备忘录模式-计数器应用

题目链接

备忘录模式-redo计数器应用

基本概念

备忘录模式（Memento Pattern）是一种行为型设计模式，它允许在**不暴露对象实现的情况下捕获对象的内部状态**并**在对象之外保存这个状态**，以便稍后可以将其还原到先前的状态。



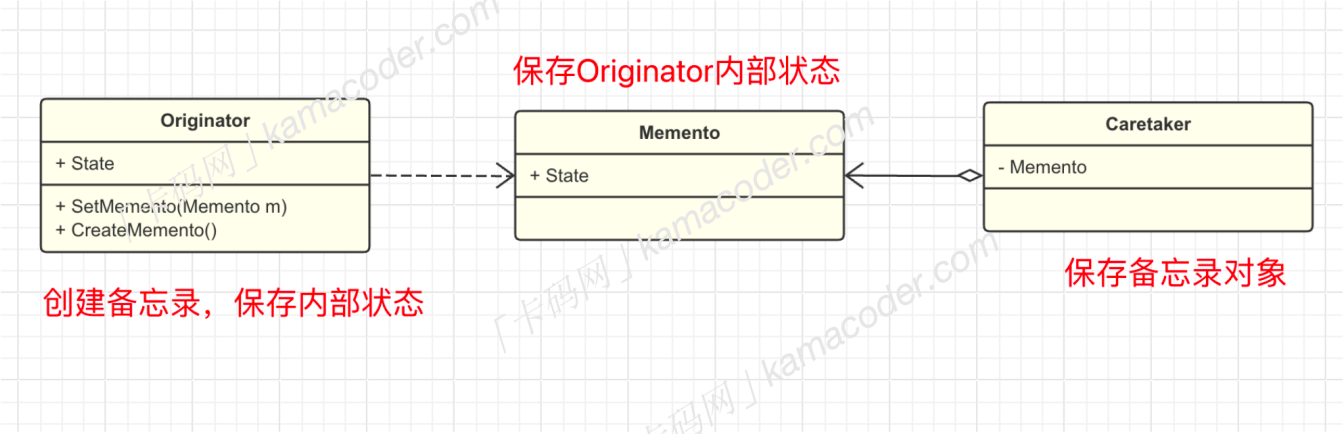
基本结构

备忘录模式包括以下几个重要角色：

- 发起人`Originator`： 需要还原状态的那个对象，负责创建一个【备忘录】，并使用备忘录记录当前时刻的内部状态。
- 备忘录`Memento`：存储发起人对象的内部状态，它可以包含发起人的部分或全部状态信息，但是对外部是不可见的，只有发起人能够访问备忘录对象的状态。

> *备忘录有两个接口，发起人能够通过宽接口访问数据，管理者只能看到窄接口，并将备忘录传递给其他对象。*

- 管理者Caretaker：负责存储备忘录对象，但并不了解其内部结构，管理者可以存储多个备忘录对象。
- 客户端：在需要恢复状态时，客户端可以从管理者那里获取备忘录对象，并将其传递给发起人进行状态的恢复。



基本实现

1. 创建发起人类：可以创建备忘录对象

```
class Originator {
    private String state;

    public void setState(String state) {
        this.state = state;
    }
    public String getState() {
        return state;
    }
    // 创建备忘录对象
    public Memento createMemento() {
        return new Memento(state);
    }
    // 通过备忘录对象恢复状态
    public void restoreFromMemento(Memento memento) {
        state = memento.getState();
    }
}
```

2. 创建备忘录类：保存发起人对象的状态

```
class Memento {

    private String state;
    // 保存发起人的状态
    public Memento(String state) {
        this.state = state;
    }

    public String getState() {
        return state;
    }
}
```

3. 创建管理者：维护一组备忘录对象

```
class Caretaker {
    private List<Memento> mementos = new ArrayList<>();

    public void addMemento(Memento memento) {
        mementos.add(memento);
    }

    public Memento getMemento(int index) {
        return mementos.get(index);
    }
}
```

4. 客户端使用备忘录模式

```
public class Main {
    public static void main(String[] args) {
        // 创建发起人对象
        Originator originator = new Originator();
        originator.setState("State 1");

        // 创建管理者对象
        Caretaker caretaker = new Caretaker();

        // 保存当前状态
        caretaker.addMemento(originator.createMemento());

        // 修改状态
        originator.setState("State 2");

        // 再次保存当前状态
```

```
        caretaker.addMemento(originator.createMemento());

        // 恢复到先前状态
        originator.restoreFromMemento(caretaker.getMemento(0));

        System.out.println("Current State: " + originator.getState());
    }
}
```

使用场景

备忘录模式在保证了对象内部状态的封装和私有性前提下可以轻松地添加新的备忘录和发起人，实现“备份”，不过 备份对象往往会消耗较多的内存，资源消耗增加。

备忘录模式常常用来实现撤销和重做功能，比如在Java Swing GUI编程中，`javax.swing.undo`包中的撤销（undo）和重做（redo）机制使用了备忘录模式。`UndoManager`和`UndoableEdit`接口是与备忘录模式相关的主要类和接口。

本题代码

```
import java.util.Scanner;
import java.util.Stack;

// 备忘录
class Memento {
    private int value;

    public Memento(int value) {
        this.value = value;
    }

    public int getValue() {
        return value;
    }
}

// 发起人（Originator）
class Counter {
    private int value;
    private Stack<Memento> undoStack = new Stack<>();
    private Stack<Memento> redoStack = new Stack<>();

    public void increment() {
        redoStack.clear();
        undoStack.push(new Memento(value));
```

```
        value++;
    }

    public void decrement() {
        redoStack.clear();
        undoStack.push(new Memento(value));
        value--;
    }

    public void undo() {
        if (!undoStack.isEmpty()) {
            redoStack.push(new Memento(value));
            value = undoStack.pop().getValue();
        }
    }

    public void redo() {
        if (!redoStack.isEmpty()) {
            undoStack.push(new Memento(value));
            value = redoStack.pop().getValue();
        }
    }

    public int getValue() {
        return value;
    }
}

// 客户端
public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        Counter counter = new Counter();

        // 处理计数器应用的输入
        while (scanner.hasNext()) {
            String operation = scanner.next();
            switch (operation) {
                case "Increment":
                    counter.increment();
                    break;
                case "Decrement":
                    counter.decrement();
                    break;
                case "Undo":
                    counter.undo();
```

```
                    break;
                case "Redo":
                    counter.redo();
                    break;
            }

            // 输出当前计数器的值
            System.out.println(counter.getValue());
        }

        scanner.close();
    }
}
```

其他语言版本

C++

```
#include <iostream>
#include <stack>

// 备忘录
class Memento {
private:
    int value;

public:
    Memento(int val) : value(val) {}

    int getValue() const {
        return value;
    }
};

// 发起人（Originator）
class Counter {
private:
    int value;
    std::stack<Memento> undoStack;
    std::stack<Memento> redoStack;

public:
    void increment() {
        redoStack = std::stack<Memento>(); // 清空 redoStack
        undoStack.push(Memento(value));
```

```
        value++;
    }

    void decrement() {
        redoStack = std::stack<Memento>(); // 清空 redoStack
        undoStack.push(Memento(value));
        value--;
    }

    void undo() {
        if (!undoStack.empty()) {
            redoStack.push(Memento(value));
            value = undoStack.top().getValue();
            undoStack.pop();
        }
    }

    void redo() {
        if (!redoStack.empty()) {
            undoStack.push(Memento(value));
            value = redoStack.top().getValue();
            redoStack.pop();
        }
    }

    int getValue() const {
        return value;
    }
};

int main() {
    Counter counter;

    // 处理计数器应用的输入
    std::string operation;
    while (std::cin >> operation) {
        if (operation == "Increment") {
            counter.increment();
        } else if (operation == "Decrement") {
            counter.decrement();
        } else if (operation == "Undo") {
            counter.undo();
        } else if (operation == "Redo") {
            counter.redo();
        }
```

```
        // 输出当前计数器的值
        std::cout << counter.getValue() << std::endl;
    }

    return 0;
}
```

Python

```
import sys
class Memento:
    def __init__(self, value):
        self.value = value

    def get_value(self):
        return self.value


class Counter:
    def __init__(self):
        self.value = 0
        self.undo_stack = []
        self.redo_stack = []

    def increment(self):
        self.redo_stack = []  # 清空 redo_stack
        self.undo_stack.append(Memento(self.value))
        self.value += 1

    def decrement(self):
        self.redo_stack = []  # 清空 redo_stack
        self.undo_stack.append(Memento(self.value))
        self.value -= 1

    def undo(self):
        if self.undo_stack:
            self.redo_stack.append(Memento(self.value))
            self.value = self.undo_stack.pop().get_value()

    def redo(self):
        if self.redo_stack:
            self.undo_stack.append(Memento(self.value))
            self.value = self.redo_stack.pop().get_value()

    def get_value(self):
```

```
        return self.value


# 客户端
counter = Counter()

# 处理计数器应用的输入
for line in sys.stdin:
    operation = line.strip()
    if operation == "Increment":
        counter.increment()
    elif operation == "Decrement":
        counter.decrement()
    elif operation == "Undo":
        counter.undo()
    elif operation == "Redo":
        counter.redo()

    # 输出当前计数器的值
    print(counter.get_value())
```

Go

```
package main

import (
    "bufio"
    "fmt"
    "os"
)

// 备忘录
type Memento struct {
    value int
}

// 发起人（Originator）
type Counter struct {
    value      int
    undoStack  []*Memento
    redoStack  []*Memento
}

func (c *Counter) increment() {
    c.redoStack = nil
    c.undoStack = append(c.undoStack, &Memento{value: c.value})
```

```
    c.value++
}

func (c *Counter) decrement() {
    c.redoStack = nil
    c.undoStack = append(c.undoStack, &Memento{value: c.value})
    c.value--
}

func (c *Counter) undo() {
    if len(c.undoStack) > 0 {
        c.redoStack = append(c.redoStack, &Memento{value: c.value})
        c.value = c.undoStack[len(c.undoStack)-1].value
        c.undoStack = c.undoStack[:len(c.undoStack)-1]
    }
}

func (c *Counter) redo() {
    if len(c.redoStack) > 0 {
        c.undoStack = append(c.undoStack, &Memento{value: c.value})
        c.value = c.redoStack[len(c.redoStack)-1].value
        c.redoStack = c.redoStack[:len(c.redoStack)-1]
    }
}

func (c *Counter) getValue() int {
    return c.value
}

// 客户端
func main() {
    scanner := bufio.NewScanner(os.Stdin)
    counter := &Counter{}

    // 处理计数器应用的输入
    for scanner.Scan() {
        operation := scanner.Text()
        switch operation {
        case "Increment":
            counter.increment()
        case "Decrement":
            counter.decrement()
        case "Undo":
            counter.undo()
        case "Redo":
            counter.redo()
```

```
            }

            // 输出当前计数器的值
            fmt.Println(counter.getValue())
        }

        if err := scanner.Err(); err != nil {
            fmt.Fprintln(os.Stderr, "reading standard input:", err)
        }
    }
```