Software Design Patterns

- Software Design and Construction
- GVHD: Nguyễn Thị Thu Trang
- SVTH: Nguyễn Đình Quang - 20146574

Contents

- Design Principles
- Design Patterns
  - Memento Pattern
- Example
- Q&A

Design Principles (OOP)

- Tại sao cần phải thiết kế ?
- Mối liên hệ giữa Design Principles & Design Patterns ?
- Class Design principles - 5 nguyên lý.
  - The Single Responsibility Principle - Nguyên lý đơn nhiệm.
    - Một class chỉ nên có một trách nhiệm duy nhất.
  - The Open Closed principle - Nguyên lý mở rộng/hạn chế.
    - Các thực thể phần mềm (các lớp, mô-đun, chức năng, v.v.) nên được Open để mở rộng, nhưng Closed để sửa đổi.
  - The Liskov Substitution Principle - Nguyên lý thay thế Liskov.
    - Class con có thể thay thế class cha mà không làm thay đổi tính đúng đắn của chương trình.
  - The Interface Segregation Principle - Nguyên lý phân tách giao tiếp.
    - Phân tách các Interface lớn thành nhiều các interface nhỏ với mục đích cụ thể. Khi đó việc implement và quản lý sẽ dễ hơn.
  - The Dependency inversion principle - Nguyên lý nghịch đảo phụ thuộc.
    - Các module cấp cao không nên phụ thuộc vào các modules cấp thấp. Cả 2 nên phụ thuộc vào những cái trừu tượng (abstractions).
    - Những cái trừu tượng không nên phụ thuộc vào chi tiết, mà ngược lại.

Design Patterns (OOP)

- Design Patterns là gì?
- Làm cách nào để mô tả các mẫu thiết kế?
- Phân loại các Design Patterns
- Memento Pattern

Design Patterns (Memento Pattern)...

- Mẫu Memento được sử dụng để khôi phục trạng thái của đối tượng về trạng thái trước đó.
- Mô hình Memento thuộc thể loại mô hình hành vi.

Design Patterns (Memento Pattern - Implementation)...

- Mô hình Memento sử dụng ba lớp actor.
- Memento chứa trạng thái của một đối tượng được khôi phục.
- Originator tạo và lưu trữ các trạng thái trong các đối tượng Memento.
- Caretaker có trách nhiệm khôi phục trạng thái đối tượng từ Memento.

Design Patterns (Memento Pattern - Intent)...

- Khôi phục trạng thái đối tượng trong một thời gian sau.
- Hữu ích khi thực hiện các điểm kiểm tra và hoàn tác các cơ chế cho phép người dùng thoát khỏi các hoạt động dự kiến hoặc khôi phục từ các lỗi.
- Nhập các đối tượng khác với thông tin cần thiết để trở lại trạng thái trước đó mà không để lộ cấu trúc và biểu diễn bên trong của nó.

Design Patterns (Memento Pattern - Motivation)...

- Memento là một đối tượng lưu trữ một snapshot về trạng thái bên trong của một đối tượng khác, memento's originator
- Cơ chế hoàn tác sẽ yêu cầu Memento từ originator khi cần kiểm tra trạng thái originator.
- Originator khởi tạo Memento với thông tin đặc trưng cho trạng thái hiện tại của nó.
- Chỉ Originator mới có thể lưu trữ và truy xuất thông tin từ Memento.

Design Patterns (Memento Pattern - Participants)...

- Lưu trữ trạng thái nội bộ của đối tượng Originator.
- Bảo vệ chống lại truy cập bởi các đối tượng khác ngoài Originator.
- Mementos có hai giao diện:
  - Caretaker
  - Originator

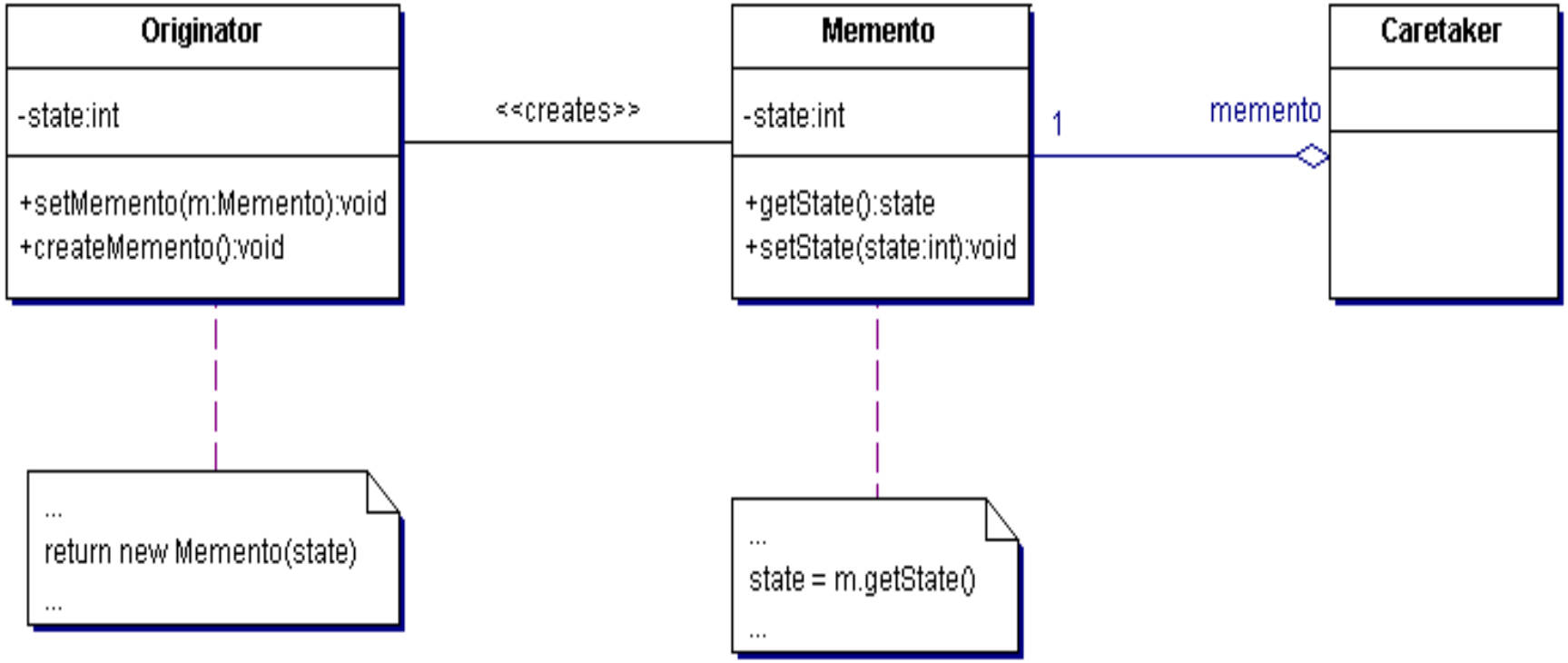
Design Patterns (Memento Pattern - Participants)...

- Originator
- Tạo memento object có chứa một ảnh chụp nhanh về trạng thái bên trong hiện tại của nó.
- Sử dụng memento để khôi phục trạng thái bên trong của nó.

Design Patterns (Memento Pattern - Collaborations)...

- Caretaker yêu cầu memento từ Originator, giữ nó trong một thời gian và chuyển lại cho Originator.
- Mementos là thụ động. Chỉ Originator đã tạo ra memento sẽ gán hoặc lấy trạng thái của nó.

Design Patterns (Memento Pattern - Structure)...



Example

```
public class Memento {
   private String state;
   public Memento(String state){
      this.state = state;
   }
   public String getState(){
      return state;
   }
}
```

```
public class Originator {
   private String state;
   public void setState(String
state){
      this.state = state;
   }
   public String getState(){
      return state;
   }
   public Memento
saveStateToMemento(){
      return new Memento(state);
   }
   public void
getStateFromMemento(Memento
memento){
      state = memento.getState();
   }
}
```

```
import java.util.ArrayList;
import java.util.List;
public class CareTaker {
   private List<Memento> mementoList = new ArrayList<Memento>();
   public void add(Memento state){
      mementoList.add(state);
   }
   public Memento get(int index){
      return mementoList.get(index);
   }
}
```

```
public class MementoPatternDemo {
   public static void main(String[] args) {

      Originator originator = new Originator();
      CareTaker careTaker = new CareTaker();

      originator.setState("State #1");
      originator.setState("State #2");
      careTaker.add(originator.saveStateToMemento());

      originator.setState("State #3");
      careTaker.add(originator.saveStateToMemento());

      originator.setState("State #4");
      System.out.println("Current State: " + originator.getState());


      originator.getStateFromMemento(careTaker.get(0));
      System.out.println("First saved State: " + originator.getState());
      originator.getStateFromMemento(careTaker.get(1));
      System.out.println("Second saved State: " + originator.getState());
   }
}
```

Q&A

References

- [SOLID] https://en.wikipedia.org/wiki/SOLID.
- [OOdesign] https://www.oodesign.com.

Cám ơn ...

- … cô và các bạn
- … đã chú ý lắng nghe!