深圳大学
Shenzhen University

第15-1讲

备忘录模式

软件体系结构与设计模式

Software Architecture & Design Pattern

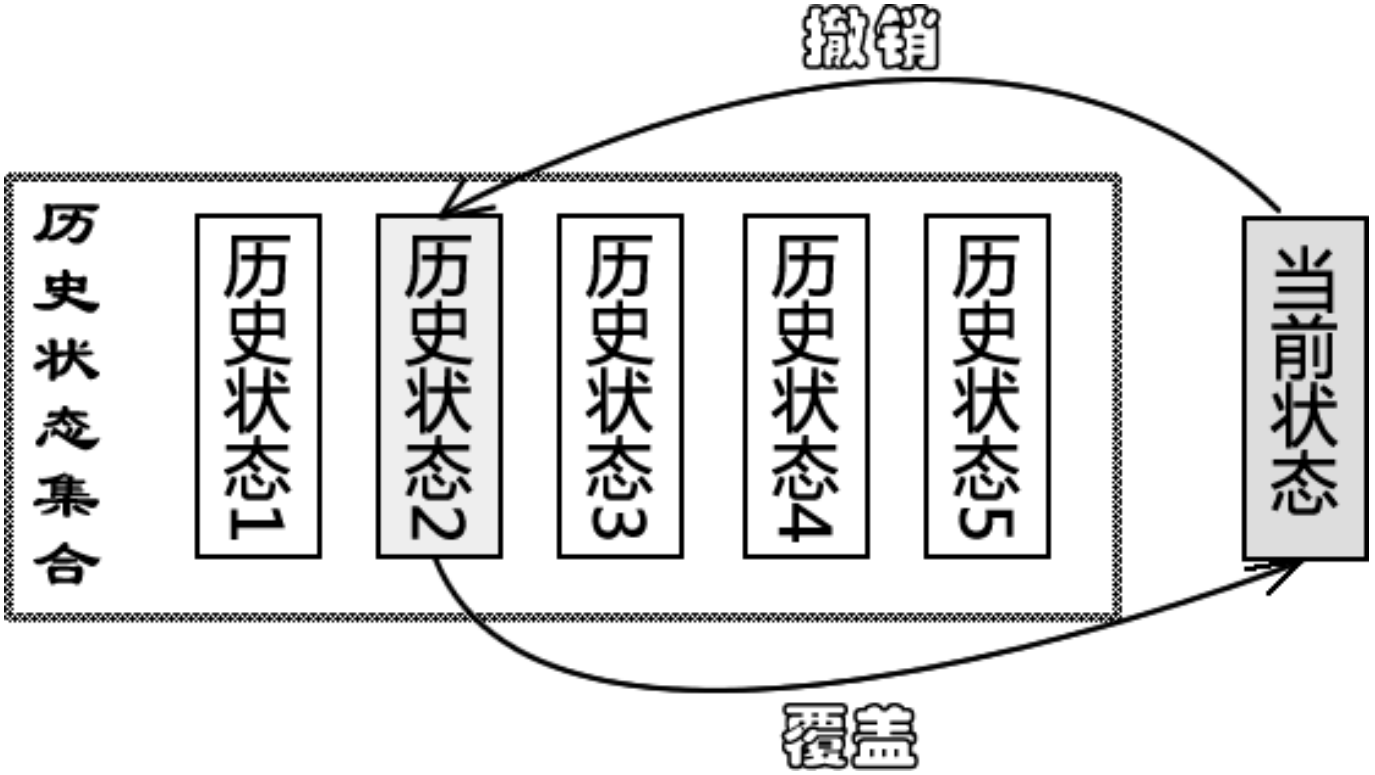
深圳大学计算机与软件学院

主要内容

- 备忘录模式动机与定义
- 备忘录模式结构与分析
- 备忘录模式实例与解析
- 备忘录模式效果与应用

备忘录模式动机

- **备忘录模式——软件中的“后悔药”——撤销(Undo)**



备忘录模式动机

- 通过使用备忘录模式可以让系统恢复到某一特定的历史状态
- 首先保存软件系统的历史状态，当用户需要取消错误操作并且返回到某个历史状态时，可以取出事先保存的历史状态来覆盖当前状态

备忘录模式定义

- 对象行为型模式

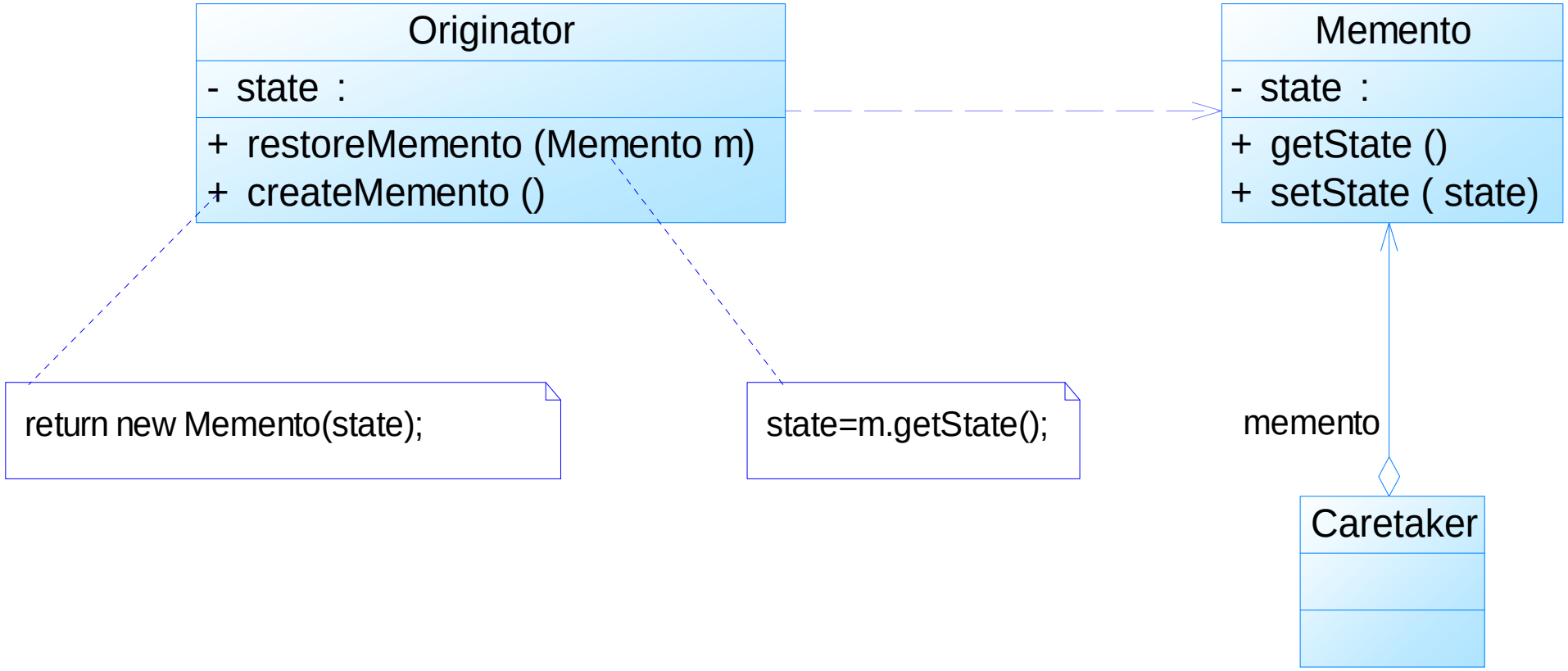
备忘录模式：在不破坏封装的前提下，捕获一个对象的内部状态，并在该对象之外保存这个状态，这样就可以在以后将对象恢复到原先保存的状态。

Memento Pattern: Without violating encapsulation, capture and externalize an object's internal state so that the object can be restored to this state later.

- 别名为标记(Token)模式
- 提供了一种状态恢复的实现机制，使得用户可以方便地回到一个特定的历史步骤
- 在很多软件所提供的撤销(Undo)操作中就使用了备忘录模式

备忘录模式结构

- **备忘录模式包含如下角色：**
  - **Originator: 原发器**
- **Memento: 备忘录**
- **Caretaker: 负责人**



备忘录模式分析

- 原发器示例代码：

```
package designpatterns.memento;
public class Originator {
    private String state;

    public Originator(){}

    //创建一个备忘录对象
    public Memento createMemento() {
        return new Memento(this);
    }

    //根据备忘录对象恢复原发器状态
    public void restoreMemento(Memento m) {
        state = m.state;
    }

    public void setState(String state) {
        this.state=state;
    }

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

备忘录模式分析

- 备忘录示例代码：

```
package designpatterns.memento;

//备忘录类，默认可见性，包内可见
class Memento {
    private String state;

    Memento(Originator o) {
        state = o.getState();
    }

    void setState(String state) {
        this.state=state;
    }

    String getState() {
        return this.state;
    }
}
```

备忘备模式分析

- Java语言实现：
  - 将Memento类与Originator类定义在同一个包(package)中来实现封装，使用默认可见性定义Memento类，即保证其在包内可见
  - 将备忘录类作为原发器类的内部类，使得只有原发器才可以访问备忘录中的数据，其他对象都无法使用备忘录中的数据

备忘备模式分析

- 负责人类示例代码：

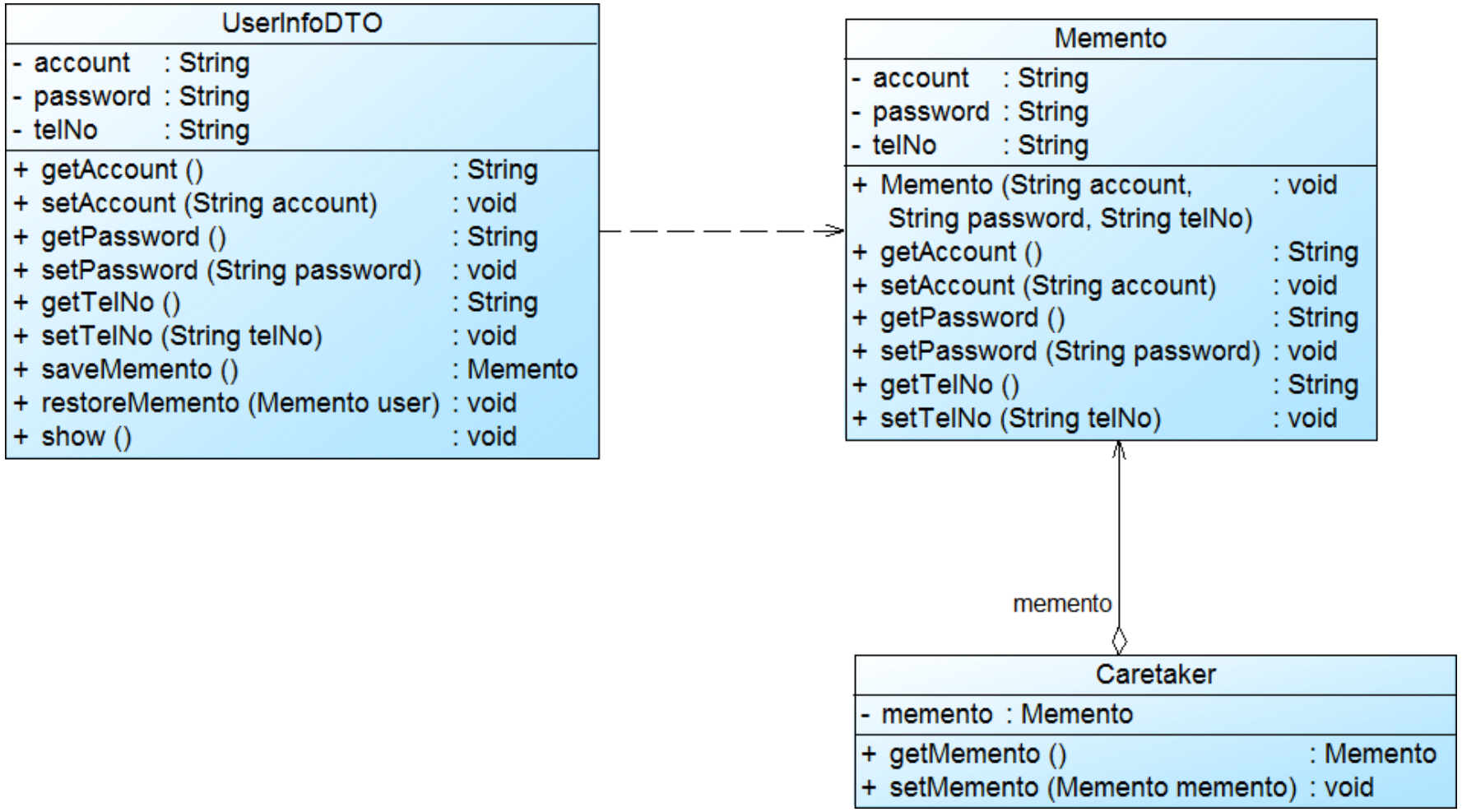
```
package designpatterns.memento;

public class Caretaker {
    private Memento memento;

    public Memento getMemento() {
        return memento;
    }

    public void setMemento(Memento memento) {
        this.memento=memento;
    }
}
```

备忘录模式实例

- **用户信息操作撤销：实例说明**
  - 某系统提供了用户信息操作模块，用户可以修改自己的各项信息。为了使操作过程更加人性化，现使用备忘录模式对系统进行改进，使得用户在进行了错误操作之后可以恢复到操作之前的状态。

备忘录模式实例与解析

- **用户信息操作撤销：参考类图**



备忘录模式实例与解析

- **备忘录模式实例**
  - **用户信息操作撤销：参考代码**
- DesignPatterns之memoto包

Caretaker.java

```
3  public class Caretaker
4  {
5      private Memento memento;
6      public Memento getMemento()
7      {
8          return memento;
9      }
10     public void setMemento(Memento memento)
11     {
12         this.memento=memento;
13     }
14 }
```

Memento.java

```
3  class Memento
4  {
5      private String account;
6      private String password;
7      private String telNo;
8
9      public Memento(String account,String password,String telNo)
10     {
11         this.account=account;
12         this.password=password;
13         this.telNo=telNo;
14     }
15     public String getAccount()
16     {
17         return account;
18     }
19
20     public void setAccount(String account)
21     {
22         this.account=account;
23     }
24
25     public String getPassword()
26     {
27         return password;
28     }
```

Memento.java

```
29
30      public void setPassword(String password)
31      {
32          this.password=password;
33      }
34
35      public String getTelNo()
36      {
37          return telNo;
38      }
39
40      public void setTelNo(String telNo)
41      {
42          this.telNo=telNo;
43      }
44  }
```

UserInfoDTO.java

```
3  public class UserInfoDTO
4  {
5      private String account;
6      private String password;
7      private String telNo;
8
9      public String getAccount()
10     {
11         return account;
12     }
13
14     public void setAccount(String account)
15     {
16         this.account=account;
17     }
18
19     public String getPassword()
20     {
21         return password;
22     }
23
24     public void setPassword(String password)
25     {
26         this.password=password;
27     }
```

UserInfoDTO.java

```
28
29      public String getTelNo()
30      {
31          return telNo;
32      }
33      public void setTelNo(String telNo)
34      {
35          this.telNo=telNo;
36      }
37      public Memento saveMemento()
38      {
39          return new Memento(account,password,telNo);
40      }
41      public void restoreMemento(Memento memento)
42      {
43          this.account=memento.getAccount();
44          this.password=memento.getPassword();
45          this.telNo=memento.getTelNo();
46      }
47
48      public void show()
49      {
50          System.out.println("Account:" + this.account);
51          System.out.println("Password:" + this.password);
52          System.out.println("TelNo:" + this.telNo);
53      }
54  }
```

Client.java

```java
public class Client
{
    public static void main(String a[])
    {
    UserInfoDTO user=new UserInfoDTO();
    Caretaker c=new Caretaker();

    user.setAccount("zhangsan");
    user.setPassword("123456");
    user.setTelNo("13000000000");
    System.out.println("状态一: ");
    user.show();
    c.setMemento(user.saveMemento());//保存备忘录
    System.out.println("----------------------------");

    user.setPassword("111111");
    user.setTelNo("13100001111");
    System.out.println("状态二: ");
    user.show();
    System.out.println("----------------------------");

    user.restoreMemento(c.getMemento());//从备忘录中恢复
    System.out.println("回到状态一: ");
    user.show();
    System.out.println("----------------------------");
    }
}
```

备忘录模式效果与应用

- **备忘录模式优点：**
  - 提供了一种状态恢复的实现机制，使得用户可以方便地回到一个特定的历史步骤
  - 实现了对信息的封装，一个备忘录对象是一种原发器对象状态的表示，不会被其他代码所改动

备忘录模式效果与应用

- **备忘录模式缺点：**
  - 资源消耗过大，如果需要保存的原发器类的成员变量太多，就不可避免地需要占用大量的存储空间，每保存一次对象的状态都需要消耗一定的系统资源

备忘录模式效果与应用

- **在以下情况下可以使用备忘录模式：**
  - 保存一个对象在某一个时刻的全部状态或部分状态，这样以后需要时能够恢复到先前的状态，实现撤销操作
  - 防止外界对象破坏一个对象历史状态的封装性，避免将对象历史状态的实现细节暴露给外界对象

谢谢