Design Patterns

宋 杰
Song Jie
东北大学 软件学院
Software College, Northeastern University

21. Memento Pattern

Intent

- Without violating encapsulation, capture and externalize an object's internal state so that the object can be restored to this state later.
  - Snapshot, Check Point
- 在不破坏封装的条件下，将一个对象的状态捕捉住，并外部化，存储起来，从而可以在将来合适的时候把这个对象还原到存储起来的状态。
- 备忘录对象是一个用来存储另外一个对象内部状态的快照的对象。

Example


CHEVROLET
FM / AM
CD
AUX
MUTE
EJECT
101.7 FM-A
SEEK
TUNE
SET
SOUND
SCAN / AST
AUDIO INPUT
1
2 RPT
3 RND
4
5
6

Step 1: Single Interface

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

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

Step 1: Single Interface

```
class Caretaker {
    private Memento memento;
    public Memento retrieveMemento() {
        return memento;
    }
    public void saveMemento(Memento memento) {
        this.memento = memento;
    }
}
```

Step 2: Multiple Checkpoints

```
class Originator {
    private String state;
    public void setMemeto(Memento memento) {
        this.state = memento.getState();
    }
    public Memento createMemento() {
        return new Memento(new Date().toString(), state);
    }
}

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

Step 2: Multiple Checkpoints

```
class Caretaker {
    private Map<String, Memento> mementoPool;
    public Caretaker() {
        mementoPool = new HashMap<String, Memento>();
    }
    public Memento retrieveMemento(String checkpoint) {
        return mementoPool.remove(checkpoint);
    }
    public void saveMemento(Memento memento) {
        mementoPool.put(memento.getCheckpoint(), memento);
    }
    public void clear() {
        mementoPool.clear();
    }
    public Iterator<String> checkpoints() {
        return mementoPool.keySet().iterator();
    }
    public Iterator<Memento> mementos() {
        return mementoPool.values().iterator();
    }
}
```

Step 3: Double interface

```
interface WideMemento {
    public String getCheckpoint();
    public String getState();
}

interface NarrowMemento {
    public String getCheckpoint();
}

class Originator {
    private String state;
    public void setMemeto(WideMemento memento) {
        this.state = memento.getState();
    }
    public WideMemento createMemento() {
        return new MementoImpl(new Date().toString(), state);
    }
}
```

Step 3: Double interface

```
class MementoImpl implements WideMemento, NarrowMemento {
    private String state;
    private String checkpoint;
    public MementoImpl(String checkpoint, String state) {
        this.checkpoint = checkpoint;
        this.state = state;
    }

    public String getCheckpoint() {
        return checkpoint;
    }

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

Step 3: Double interface

```
class Caretaker {
    private Map<String, NarrowMemento> mementoPool;
    public Caretaker() {
        mementoPool = new HashMap<String, NarrowMemento>();
    }
    public NarrowMemento retrieveMemento(String checkpoint) {
        return mementoPool.remove(checkpoint);
    }
    public void saveMemento(NarrowMemento memento) {
        mementoPool.put(memento.getCheckpoint(), memento);
    }
    public void clear() {
        mementoPool.clear();
    }
    public Iterator<String> checkpoints() {
        return mementoPool.keySet().iterator();
    }
    public Iterator<NarrowMemento> mementos() {
        return mementoPool.values().iterator();
    }
}
```

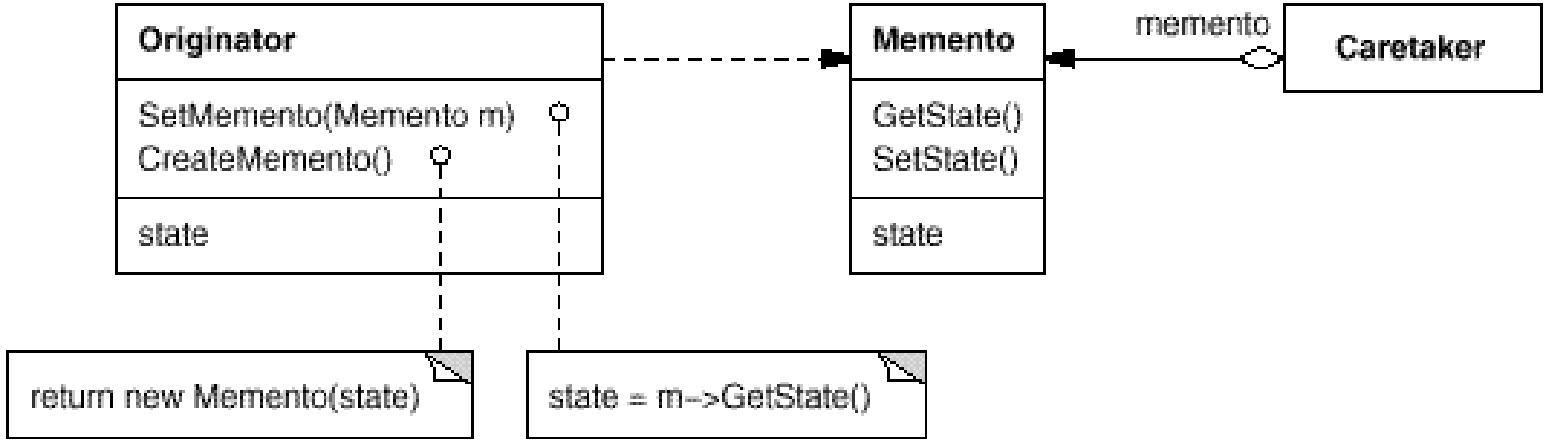
Step 4: Inner Class

```
interface Memento {
    public String getCheckpoint();
}

class Originator {
    private String state;
    public void setMemeto(Memento memento) {
        this.state = ((InnerMemento) memento).getState();
    }
    public Memento createMemento() {
        return new InnerMemento(new Date().toString(), state);
    }
    class InnerMemento implements Memento {
        private String stateCopy;
        private String checkpoint;

        public InnerMemento(String checkpoint, String state) {
            this.checkpoint = checkpoint;
            this.stateCopy = state;
        }
        public String getCheckpoint() {
            return checkpoint;
        }
        public String getState() {
            return stateCopy;
        }
    }
}
```

Step 4: Inner Class

```
class Caretaker {
    private Map<String, Memento> mementoPool;
    public Caretaker() {
        mementoPool = new HashMap<String, Memento>();
    }
    public Memento retrieveMemento(String checkpoint) {
        return mementoPool.remove(checkpoint);
    }
    public void saveMemento(Memento memento) {
        mementoPool.put(memento.getCheckpoint(), memento);
    }
    public void clear() {
        mementoPool.clear();
    }
    public Iterator<String> checkpoints() {
        return mementoPool.keySet().iterator();
    }
    public Iterator<Memento> mementos() {
        return mementoPool.values().iterator();
    }
}
```

Structure


Originator
SetMemento(Memento m)
CreateMemento()
state
Memento
GetState()
SetState()
state
memento
Caretaker
return new Memento(state)
state = m->GetState()

Participants

- Memento
  - Stores internal state of the Originator object.
  - Protects against access by objects other than the originator.
- Originator
  - Creates a memento containing a snapshot of its current internal state.
  - Uses the memento to restore its internal state.
- Caretaker
  - Be responsible for the memento's safekeeping.
  - Never operates on or examines the contents of a memento.

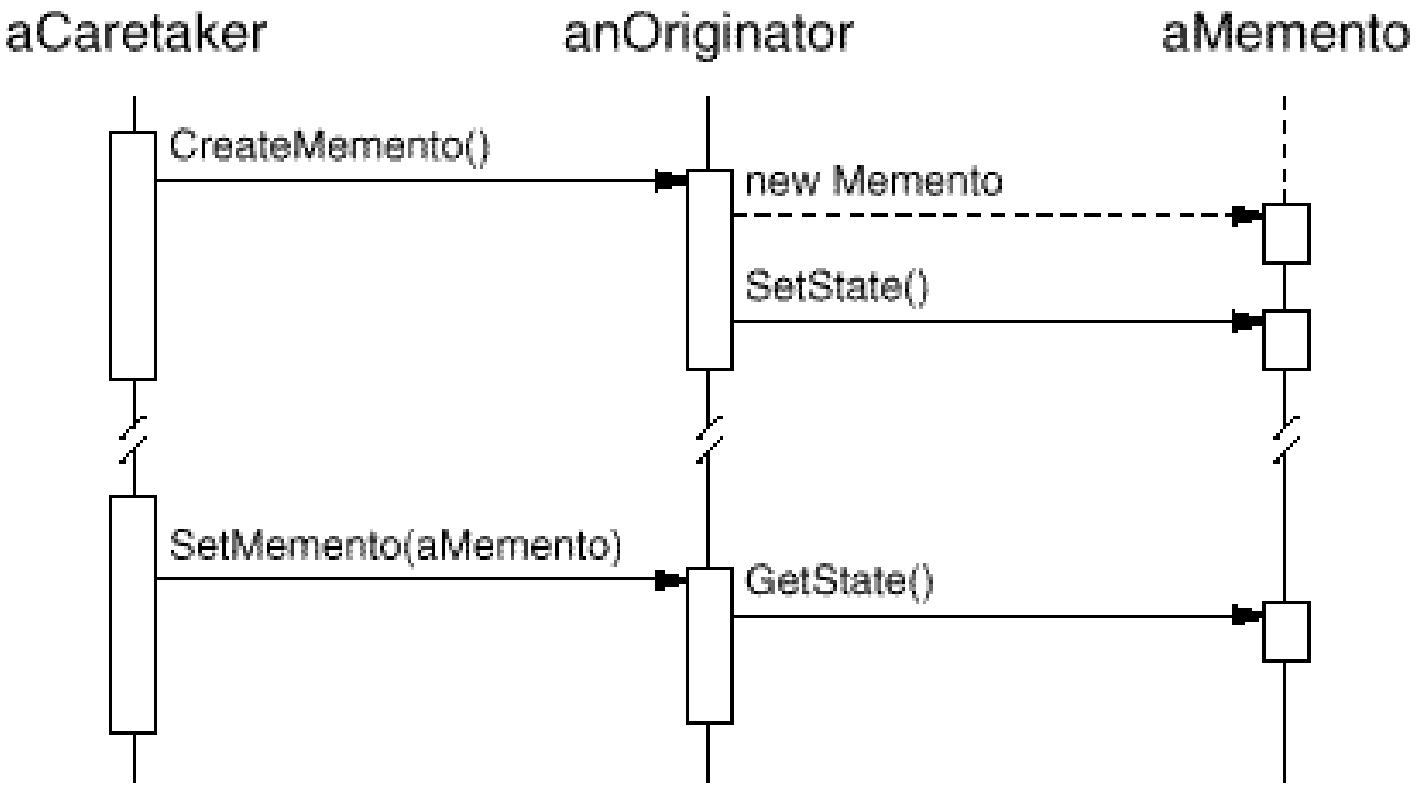
Two interfaces of Memento

- Narrow Interface: Caretaker sees a narrow interface to the Memento—it can only pass the memento to other objects.
- Wide Interface: Originator, in contrast, sees a wide interface, one that lets it access all the data necessary to restore itself to its previous state.
- Ideally, only the originator that produced the memento would be permitted to access the memento's internal state.

Collaborations

- A caretaker requests a memento from an originator, holds it for a time, and passes it back to the originator,
- Sometimes the caretaker won't pass the memento back to the originator, because the originator might never need to revert to an earlier state.
- Mementos are passive. Only the originator that created a memento will assign or retrieve its state.

Collaborations



Consequences – advantages

- Preserving encapsulation boundaries.
  - Memento avoids exposing information that only an originator should manage but that must be stored nevertheless outside the originator.
- It simplifies Originator.
  - Originator need not maintain and management the versions of internal state.

Consequences – drawbacks

- Using mementos might be expensive.
  - Mementos might incur considerable overhead if Originator must copy large amounts of information to store in the memento or if clients create and return mementos to the originator often enough.
  - The caretaker has no idea how much state is in the memento. Hence an otherwise lightweight caretaker might incur large storage costs.

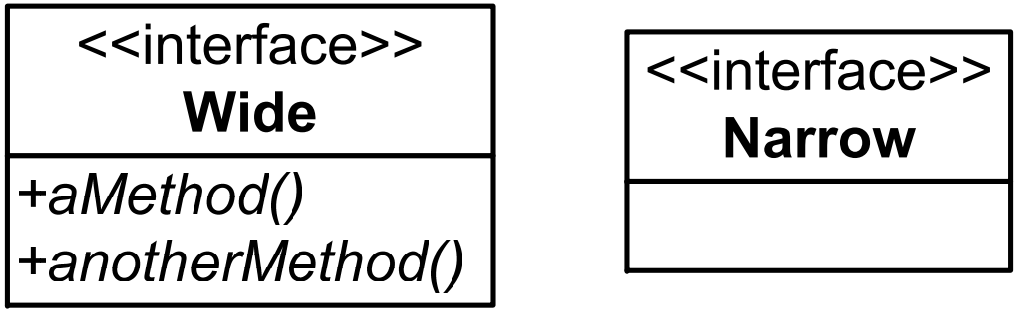
Applicability

- A snapshot of (some portion of) an object's state must be saved so that it can be restored to that state later, *and*
- A direct interface to obtaining the state would expose implementation details and break the object's encapsulation.

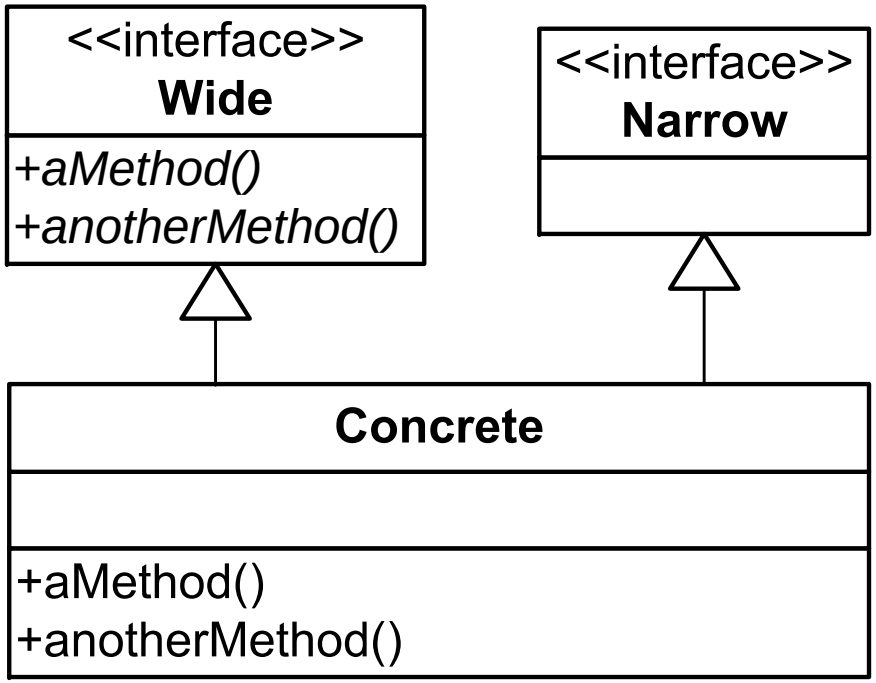
Implementation 1: Storing incremental changes.

- When mementos get created and passed back to their originator in a predictable sequence, then Memento can save just the incremental change to the originator's internal state.

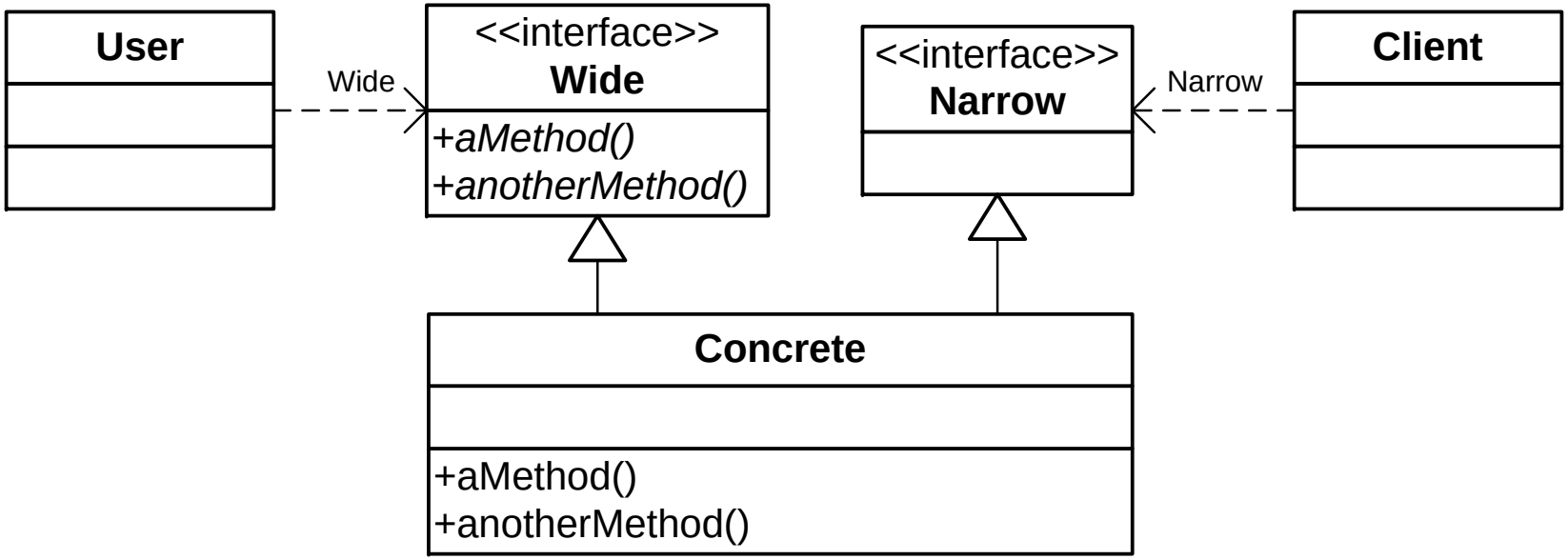
Implementation 2: Wide and narrow interfaces



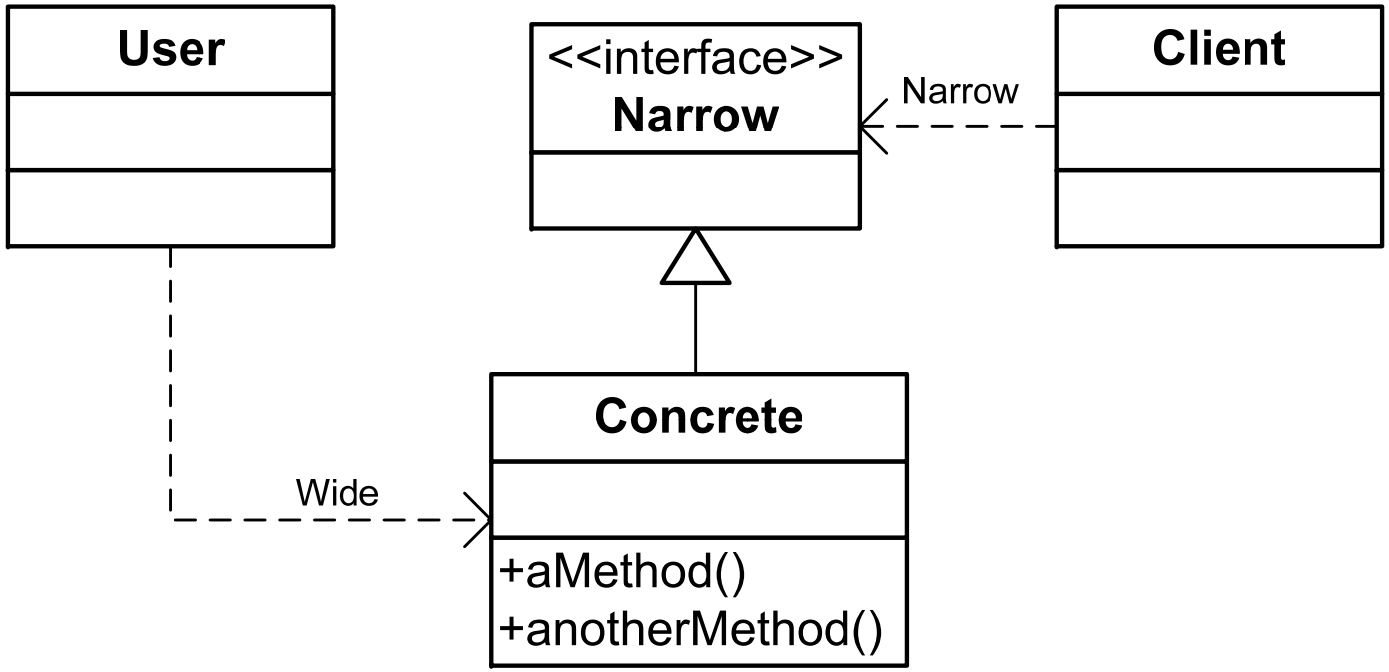
Implementation 2: Wide and narrow interfaces



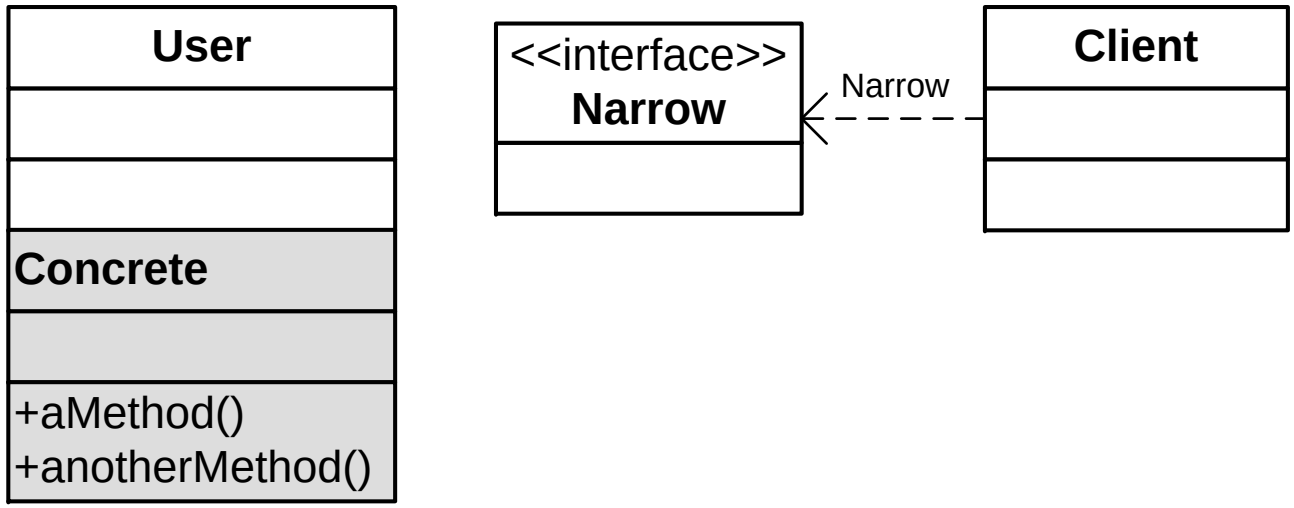
Implementation 2: Wide and narrow interfaces



Implementation 2: Wide and narrow interfaces



Implementation 2: Wide and narrow interfaces



Let’s go to next…