Նախագծման Ձևանմուշներ: Memento

Հրաչյա Թանդիլյան

2020

Memento

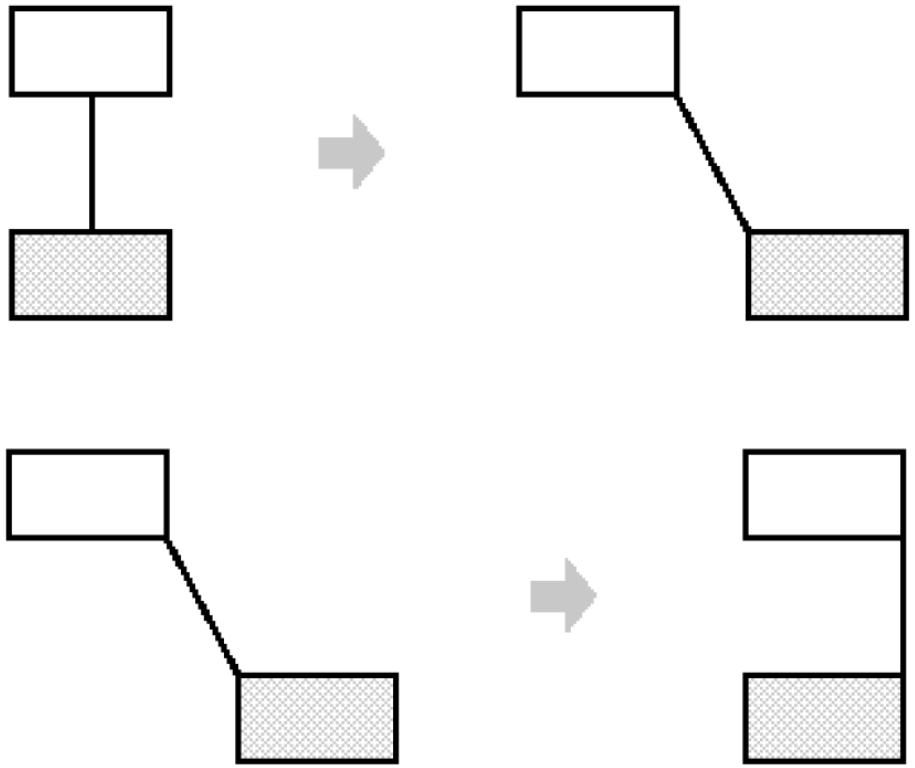
Նպատակը

Առանց ինկապսուլացիան խախտելու ֆիքսել և պահպանել օբյեկտի ներքին վիճակն այնպես, որ օբյեկտը հետագայում հնարավոր լինի բերել այդ վիճակի:

Նաև հայտնի է որպես

- Token

Մոտիվացիան

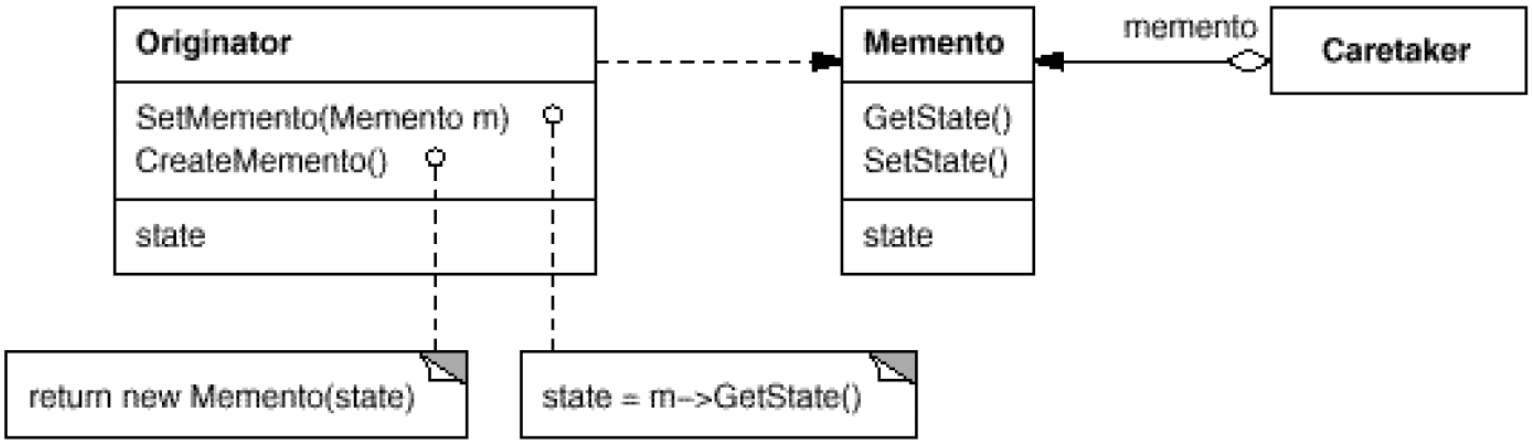


Կիրառելիությունը

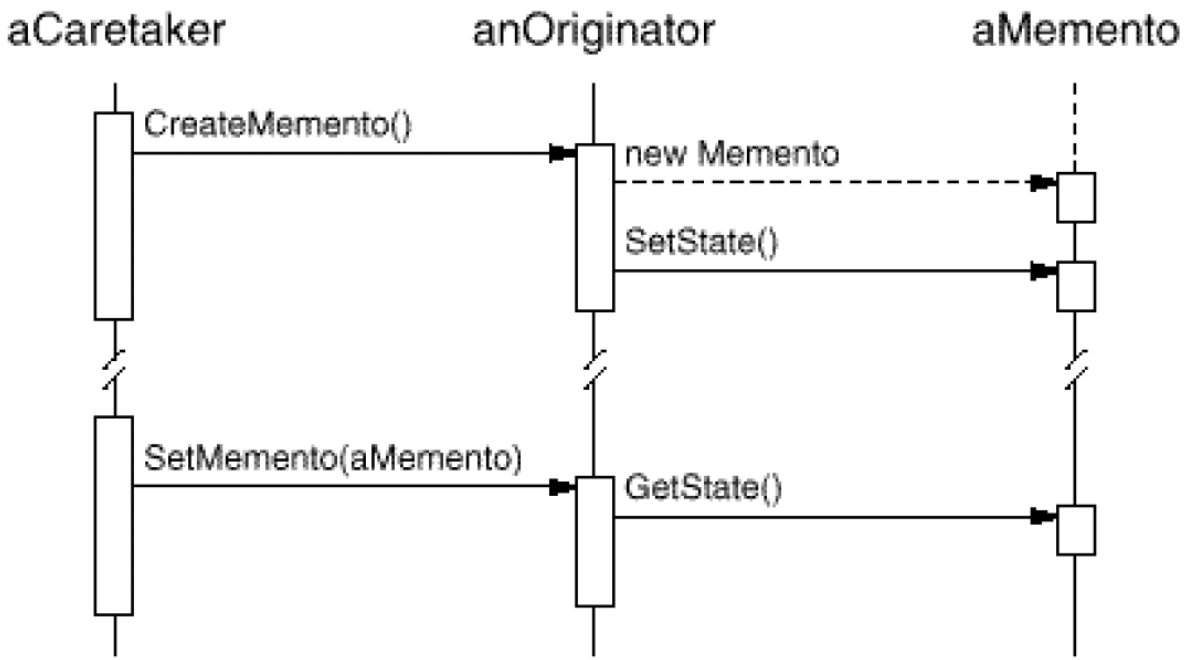
Այս Ն.Ձ. պետք է օգտագործել երբ.

- Ա Անհրաժեշտ է պահպանել օբյեկտի ֆիքսված պահի ներքին վիճակն այնպես, որ օբյեկտը հետագայում հնարավոր լինի վերադարձնել այդ վիճակի:
- Բ Օբյեկտի ներքին վիճակը վերադարձնող անմիջական ինտերֆեյսի տրամադրումը կբերի նրա ինկապսուլացիաի խախտման:

Կառուցվածքը



Կառուցվածքը



Հետևանքները

Այս Ն.Ձ. ունի հետևյալ առավելություններն ու թերությունները.

Ա Ինկապսուլացիաի պահպանում:

Բ Originator դասի պարզեցում:

Գ Memento-ի կիրառումը կարող է ծախսատար լինել:

Դ Լայն և նեղ ինտերֆեյսների սահմանումը:

Իրականացումը

- Ա Լեզվային ապահովում:
- Բ Հաջորդական (incremental) փոփոխությունների պահպանում:

Իրականացումը: Լեզվային ապահովում

```
class State;

class Originator {

public:
    Memento* CreateMemento();
    void SetMemento(const Memento*);

private:
    State* state;

    // Internal data structures
};
```

Իրականացումը: Լեզվային ապահովում

```
class Memento {

public:
    // Narrow public interface
    virtual ~Memento();

private:
    // Wide private interface
    friend class Originator;

    Memento();
    void SetState(State*);
    State* GetState();

private:
    State* state;
};
```

Օրինակ

```
// Base class for graphical objects in the graphical editor
class Graphic;

class MoveCommand {

public:
    MoveCommand(Graphic* t, const Point& d);
    void Execute();
    void Unexecute();

private:
    ConstraintSolverMemento* state;
    Point delta;
    Graphic* target;
};
```

Օրինակ

```
class ConstraintSolver {

public:
    static ConstraintSolver* Instance();
    void Solve();

    void AddConstraint(Graphic* startConnection,
                       Graphic* endConnection);

    void RemoveConstraint(Graphic* startConnection,
                          Graphic* endConnection);

    ConstraintSolverMemento* CreateMemento();
    void SetMemento(ConstraintSolverMemento*);

private:
    // Nontrivial state and operations for
    // enforcing connectivity semantics
};
```

Օրինակ

```
class ConstraintSolverMemento {

public:
    virtual ~ConstraintSolverMemento();

private:
    friend class ConstraintSolver;

    ConstraintSolverMemento();

    // Private constraint solver state
};
```

Օրինակ

```
void MoveCommand::Execute () {

    ConstraintSolver* solver = ConstraintSolver::Instance();

    state = solver->CreateMemento(); // Create a memento
    target->Move(delta);
    solver->Solve();
}

void MoveCommand::Unexecute () {

    ConstraintSolver* solver = ConstraintSolver::Instance();

    target->Move(-delta);
    solver->SetMemento(state); // Restore solver state
    solver->Solve();
}
```

Օրինակ 2

```
template <class Item>
class Collection {

public:
    Collection();
    IterationState* CreateInitialState();

    void Next(IterationState&);
    bool IsDone(const IterationState&) const;
    Item CurrentItem(const IterationState&) const;
    IterationState* Copy(const IterationState&) const;

    void Append(const Item&);
    void Remove(const Item&);
};
```

Օրինակ 2

```
class ItemType {

public:
    void process();
    // Other methods
};

Collection<ItemType> c;

std::auto_ptr<IterationState> state(c.CreateInitialState());

while (!c.IsDone(*state)) {

    c.CurrentItem(*state).process();
    c.Next(*state);
}
```

Առնչվող Նախագծման Ձևանմուշները

- Command
- Iterator