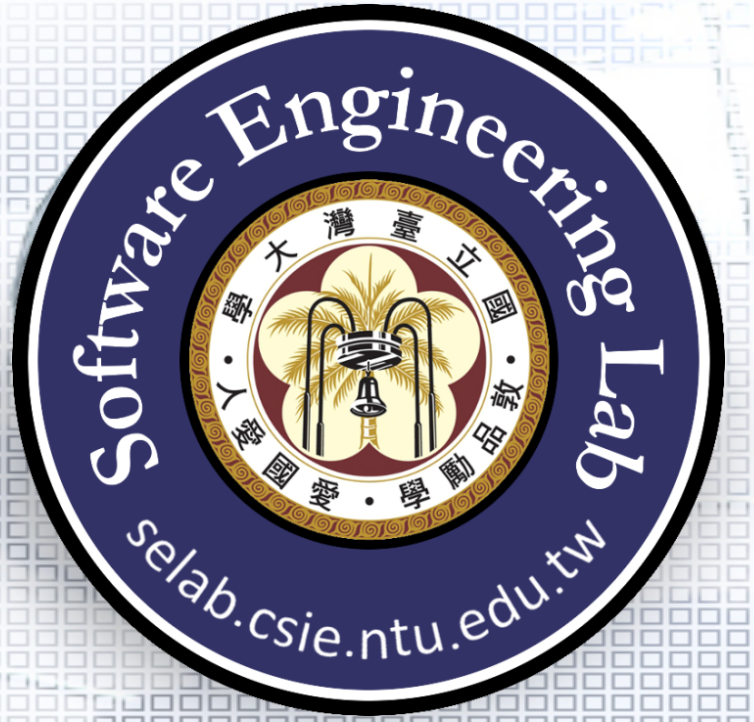


Memento Pattern

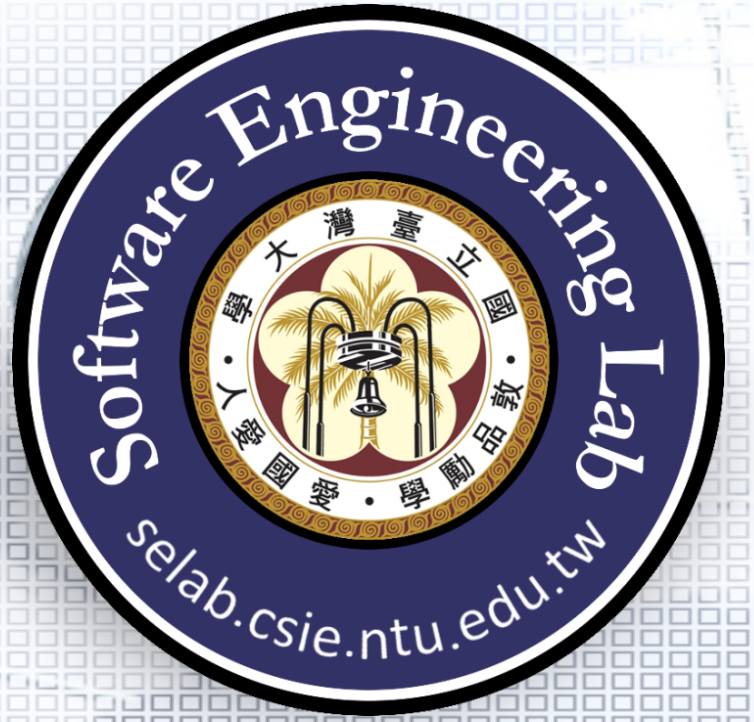
Prof. Jonathan Lee（李允中）
Department of CSIE
National Taiwan University

Design Aspect of Memento

what private information is stored outside an object, and when

Outline

- Master Game Requirements Statements
- Initial Design and Its Problems
- Design Process
- Refactored Design after Design Process
- Recurrent Problems
- Intent
- Memento Pattern Structure
- Saving a Graph: Another Example



Master Game

Prof. Jonathan Lee (李允中)

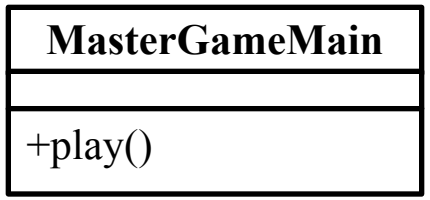
Department of Computer Science and Information Engineering
National Taiwan University

Requirements Statements

- The main program of Master Game is MasterGameMain, and the entry point is the play() operation.
- The current state of Master Game is also kept in MasterGameMain.
- In order to provide save/load functionalities, MasterGameMain should be able to store and restore the snapshots of Master Game.

Requirements Statements$_1$

- The main program of Master Game is MasterGameMain, and the entry point is the play() operation.



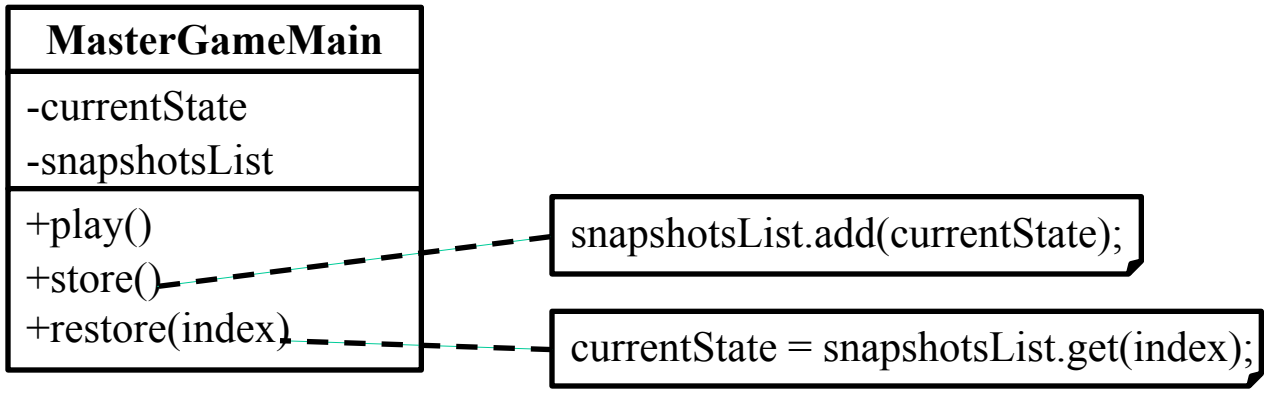
Requirements Statements$_2$

- The current state of Master Game is also kept in MasterGameMain.

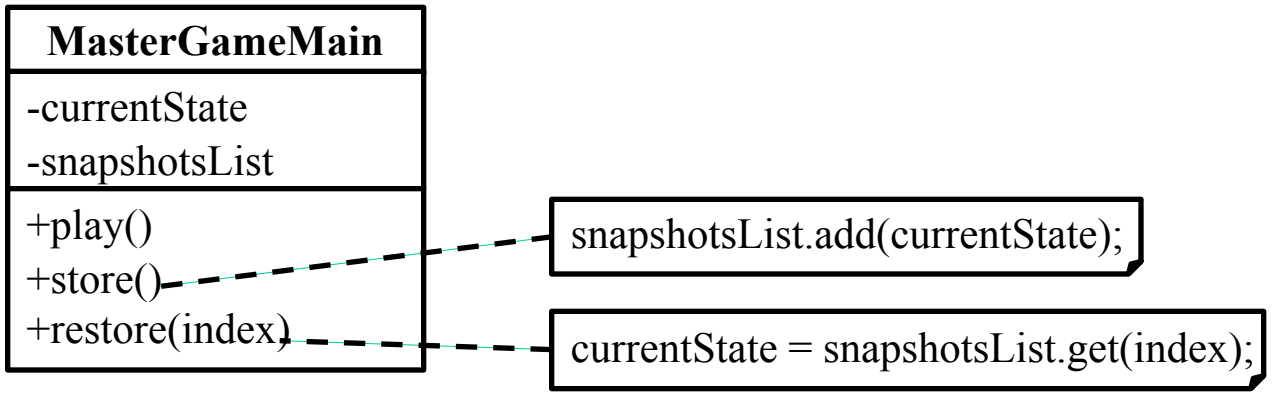
MasterGameMain
-currentState
+play()

Requirements Statements$_3$

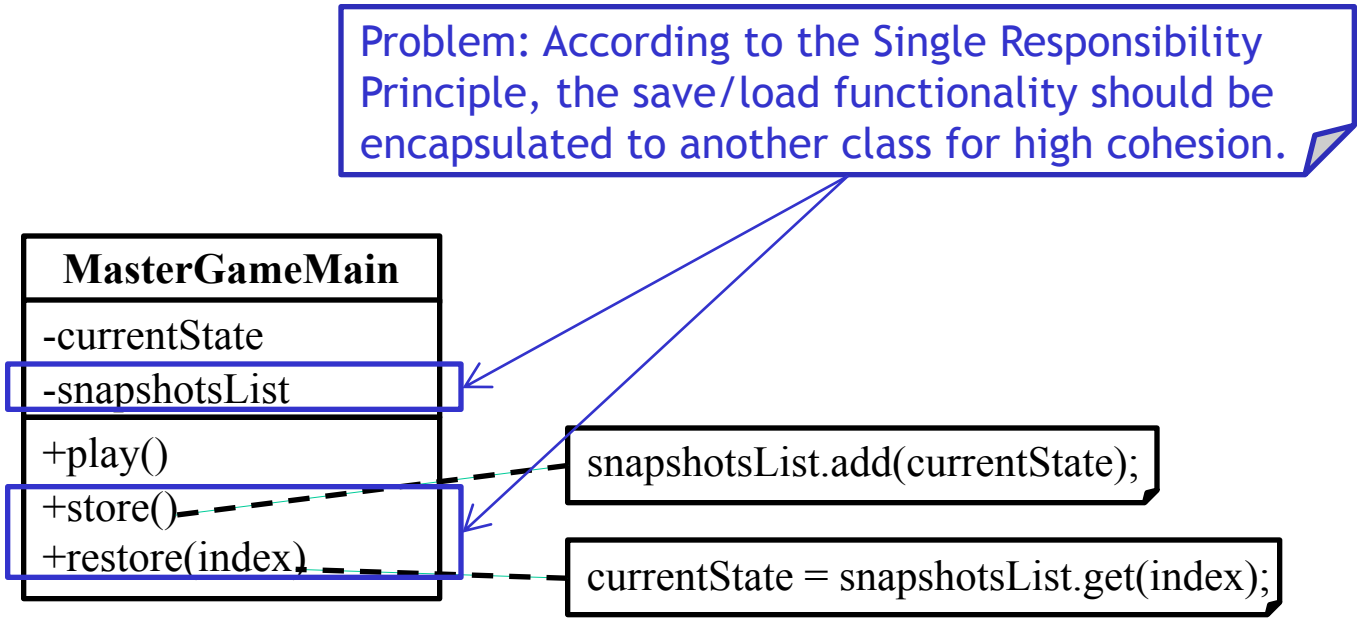
- In order to provide save/load functionalities, MasterGameMain should be able to store and restore the snapshots of Master Game.



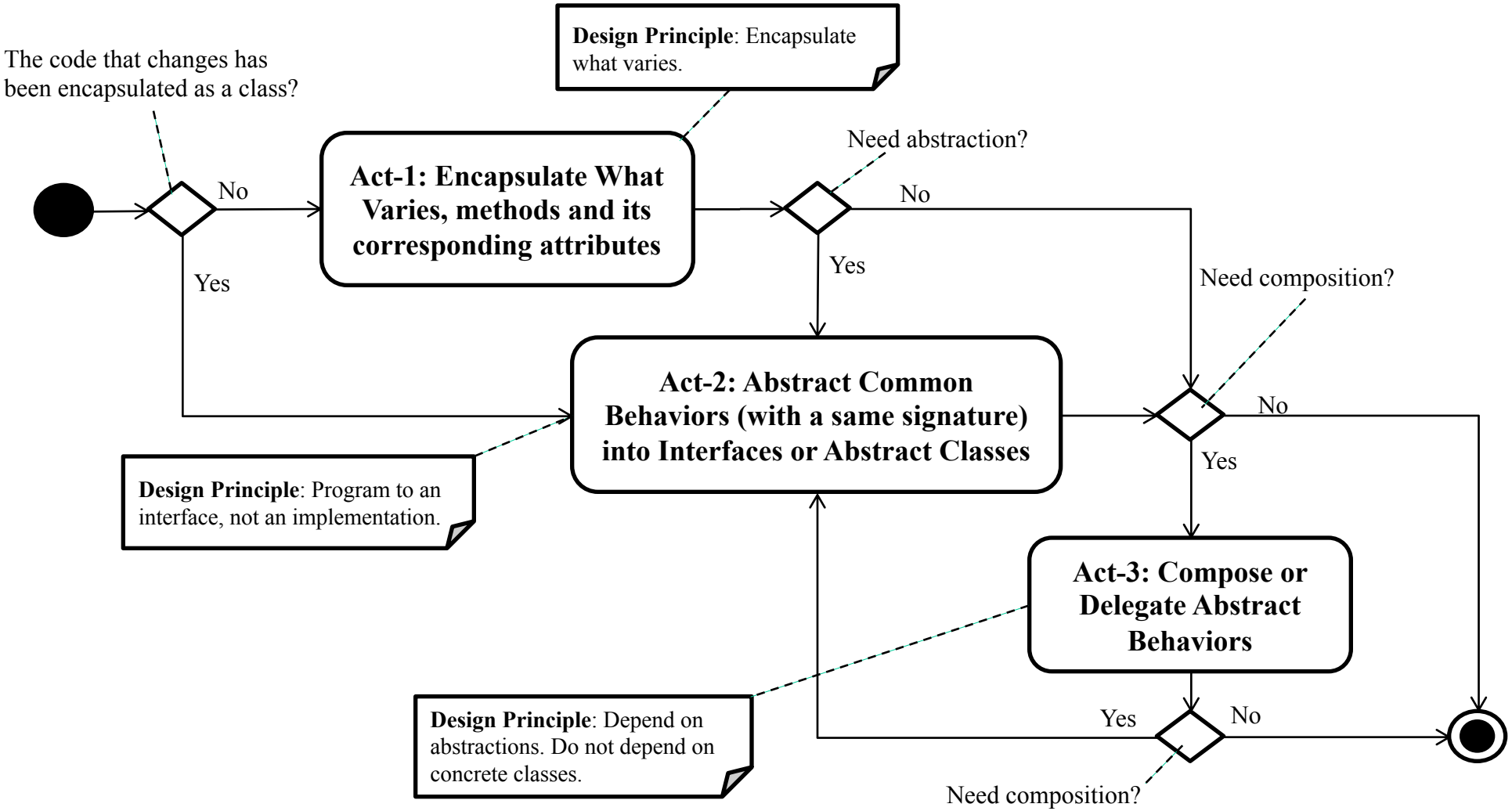
Initial Design - Class Diagram



The Problem with the Initial Design

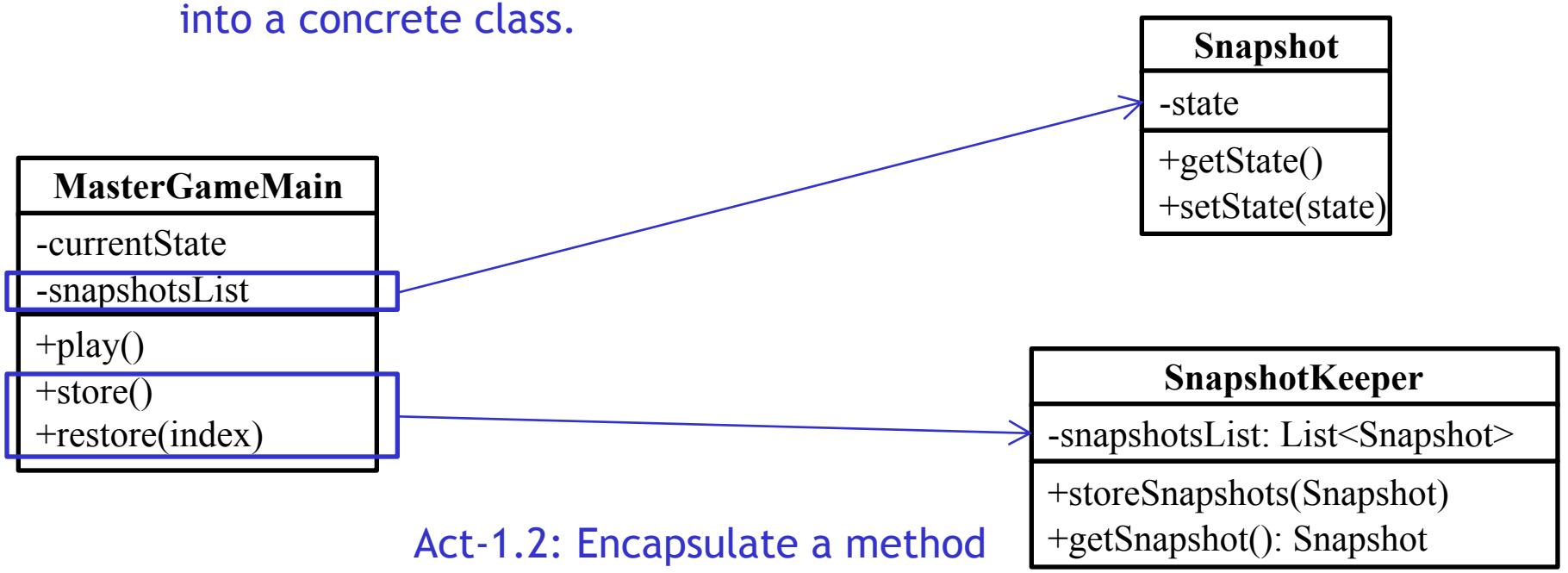


Design Process for Change



Act-1: Encapsulate What Varies

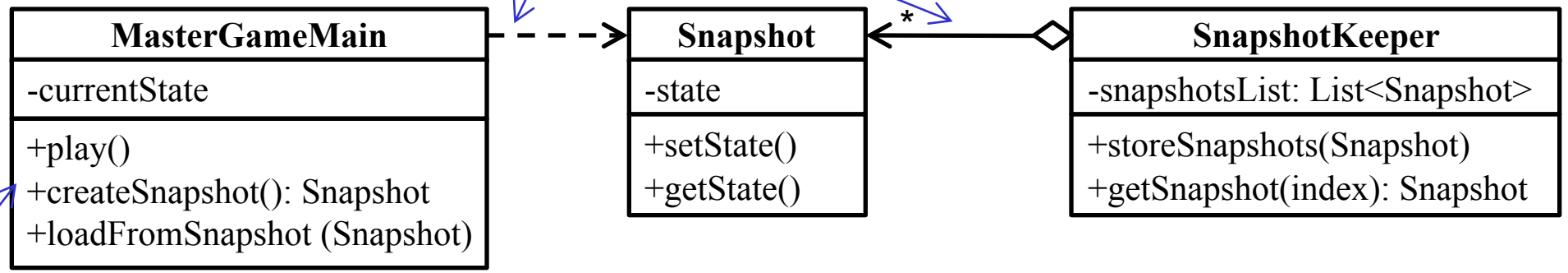
Act-1.1: Encapsulate an attribute into a concrete class.



Act-1.2: Encapsulate a method into a concrete class.

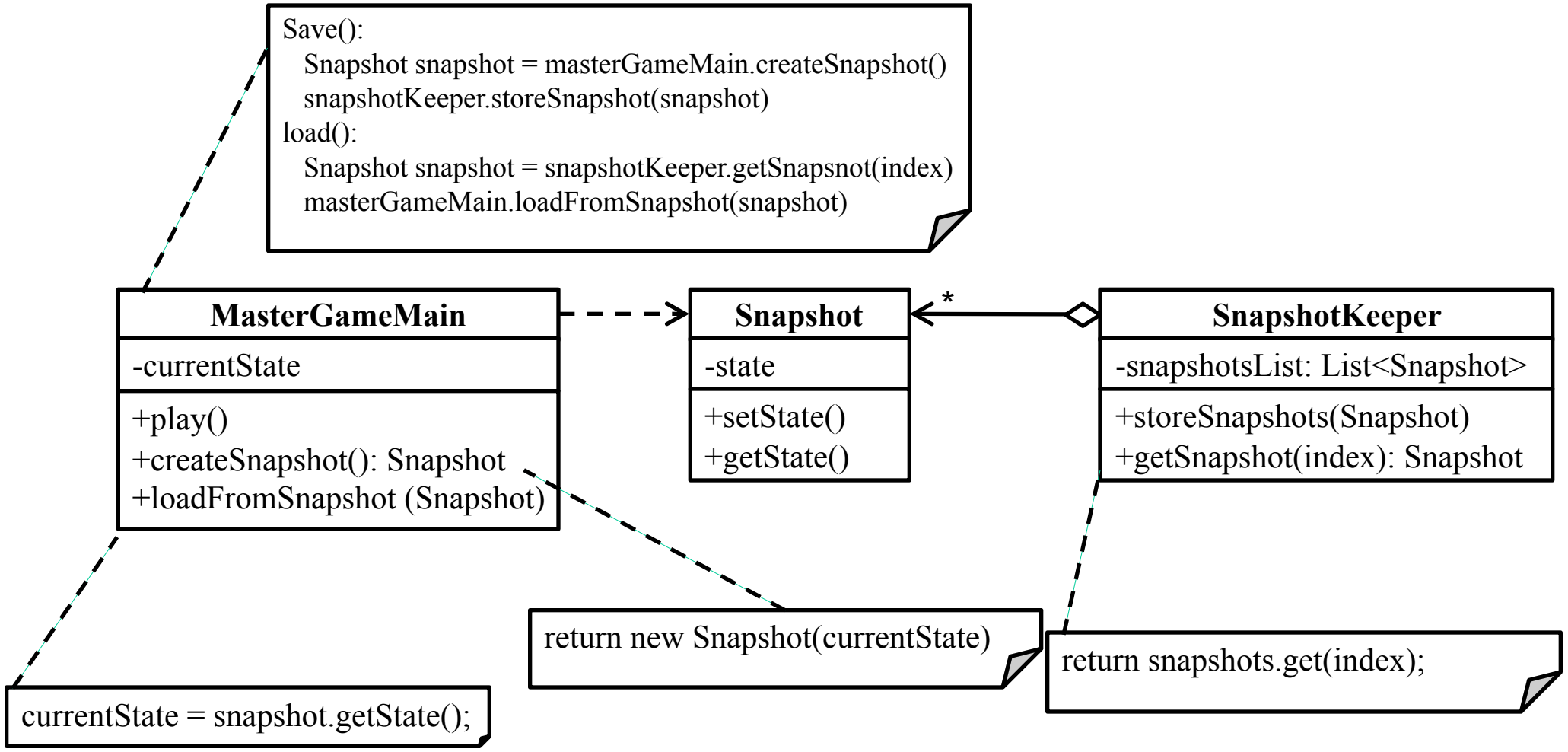
Refactored Design after Design Process

Act-3.4: Delegate behavior
to a concrete class.



create a Snapshot in order
to store the current state.

Refactored Design after Design Process



MasterGameMain

```
public class MasterGameMain {
    private int currentState = 0;

    public void play(){
        currentState++;
        System.out.println("Play~ Current State: " + currentState);
    }

    public Snapshot createSnapshot() { return new Snapshot(currentState); }

    public void loadFromSnapshot(Snapshot snapshot){
        this.currentState = snapshot.getState();
        System.out.println("Load!  Current State: " + currentState);
    }
}
```

Snapshot

```
public class Snapshot {
    private int state;

    public Snapshot(int state){
        this.state = state;
    }

    public int getState() { return state; }

}
```

SnapshotKeeper

```
public class SnapshotKeeper {
    private List<Snapshot> snapshots = new ArrayList<>();

    public void storeSnapshot(Snapshot snapshot){
        snapshots.add(snapshot);
    }

    public Snapshot getSnapsnot(int index){
        Snapshot snapshot = null;
        try{
            snapshot = snapshots.get(index);
        }
        catch (Exception e) {
            snapshot = null;
        }

        return snapshot;
    }
}
```

Input / Output format

Input:

```
play

save //save current [state_index]

load [record_index]

...
```

Output:

```
//show current state when input: play

Play~ Current State: [state_index]



// restore the state and show current state when input: load

Load! Current State: [state_index]

...
```

Test case

Sample.in

```
play
play
play
play
play
play
save
play
save
play
save
load 0
play
play
play
play
save
play
play
save
load 3
play
play
play
play
play
save
```

Sample.out

```
Play~ Current State: 1
Play~ Current State: 2
Play~ Current State: 3
Play~ Current State: 4
Play~ Current State: 5
Play~ Current State: 6
Play~ Current State: 7
Play~ Current State: 8
Load!  Current State: 6
Play~ Current State: 7
Play~ Current State: 8
Play~ Current State: 9
Play~ Current State: 10
Play~ Current State: 11
Play~ Current State: 12
Load!  Current State: 10
Play~ Current State: 11
Play~ Current State: 12
Play~ Current State: 13
Play~ Current State: 14
Play~ Current State: 15
```

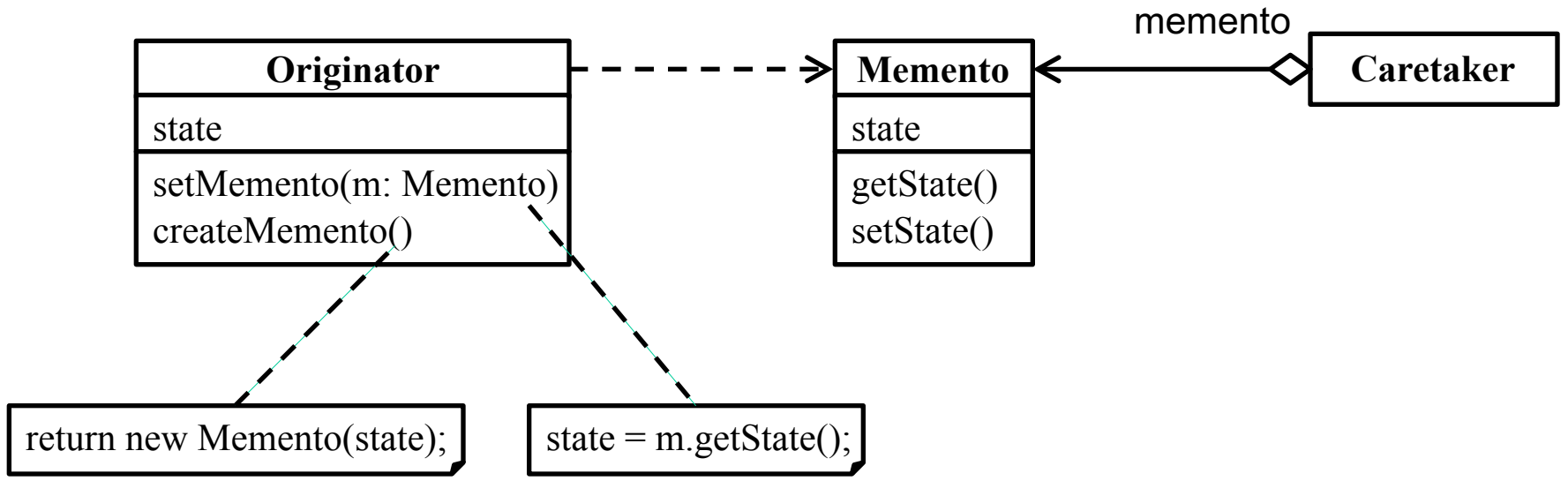
Recurrent Problem

- Sometimes it's necessary to record the internal state of an object. You must save state information somewhere so that you can restore objects to their previous states.
- But objects normally encapsulate some or all of their state, making it inaccessible to other objects and impossible to save externally. Exposing this state would violate encapsulation, which can compromise the application's reliability and extensibility.

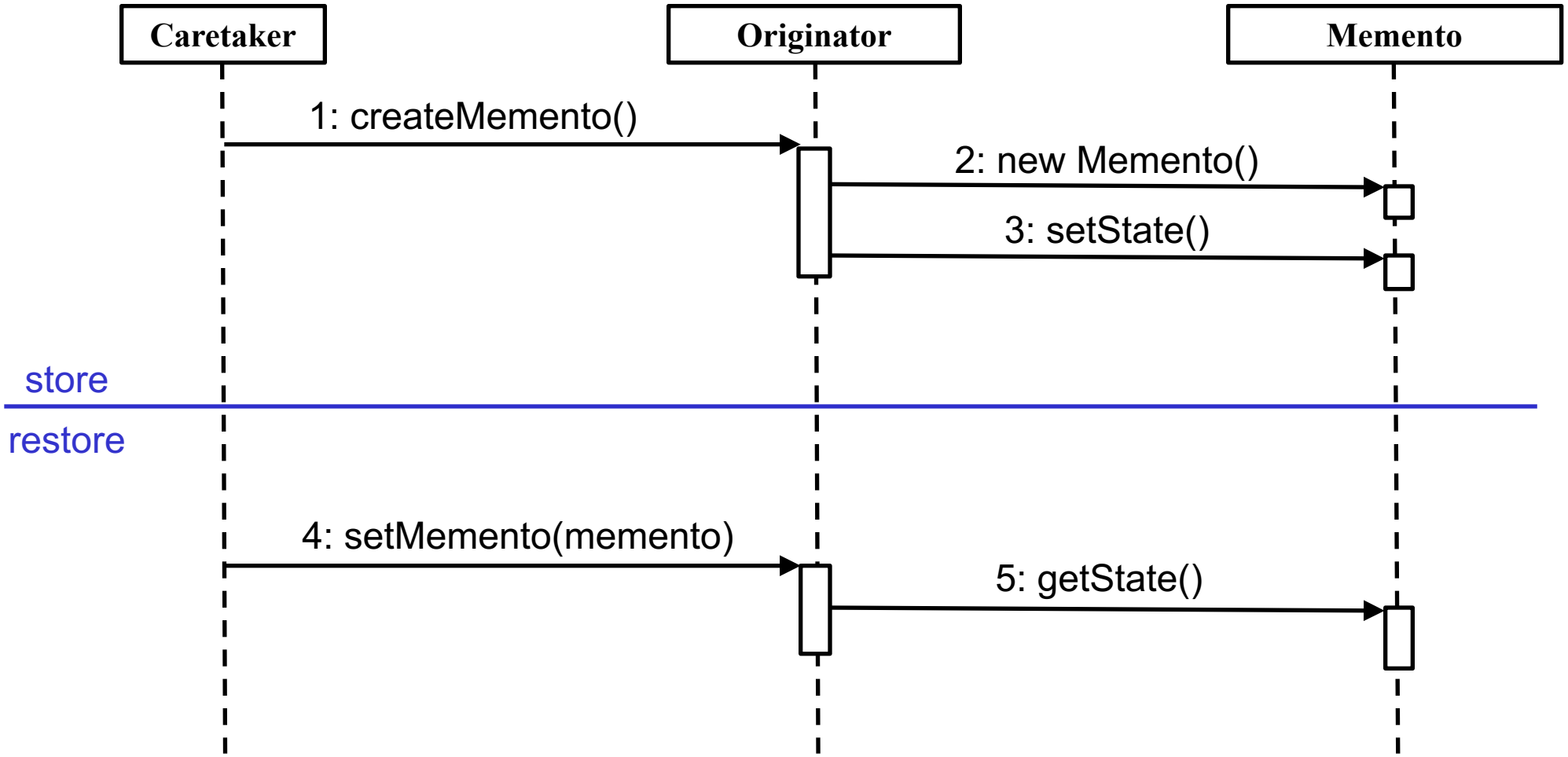
Intent

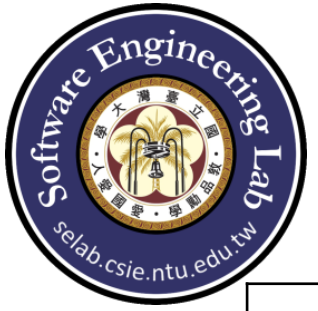
- Without violating encapsulation, capture and externalize an object's internal state so that the object can be restored to this state later.

Memento Pattern Structure$_1$



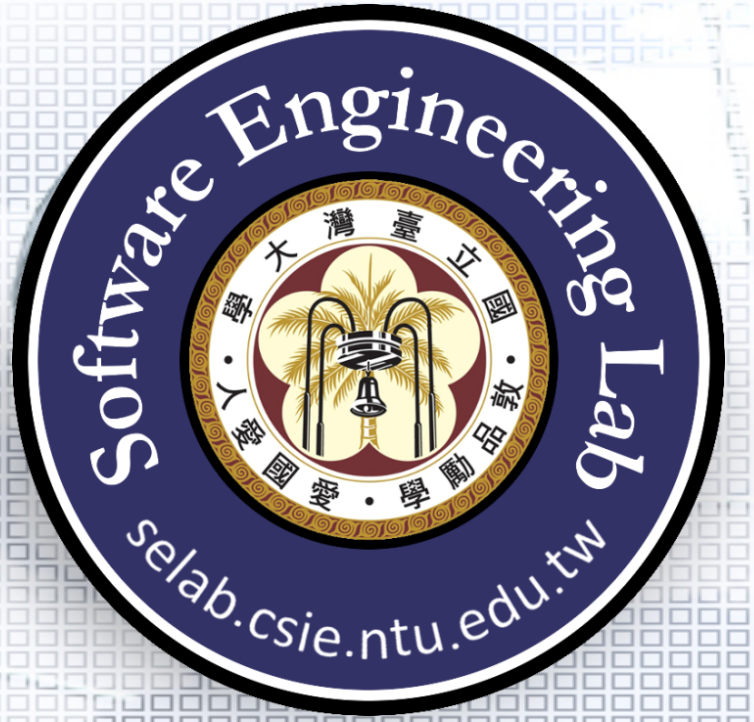
Memento Pattern Structure$_2$





Memento Pattern Structure$_3$

	Instantiation	Use	Termination
Memento	Originator	A new Memento is created by Originator, and Originator will set its internal state to Memento, and then Caretaker will keep the Memento safe. Once Originator needs the Memento again, it will be passed back to Originator by Caretaker.	Don't Care
Originator	Don't Care	Originator creates a Memento which contains a snapshot of its current internal state. Once Originator would like to restore its state, the Memento will be passed to the Originator to restore state.	Don't care
Caretaker	Don't Care	Caretaker is responsible for keeping memento safe. Caretaker requests Originator to create Memento and keeps it safe. When it needs to restore, Caretaker passes Memento to Originator.	Don't Care



Saving a Graph

Prof. Jonathan Lee (李允中)

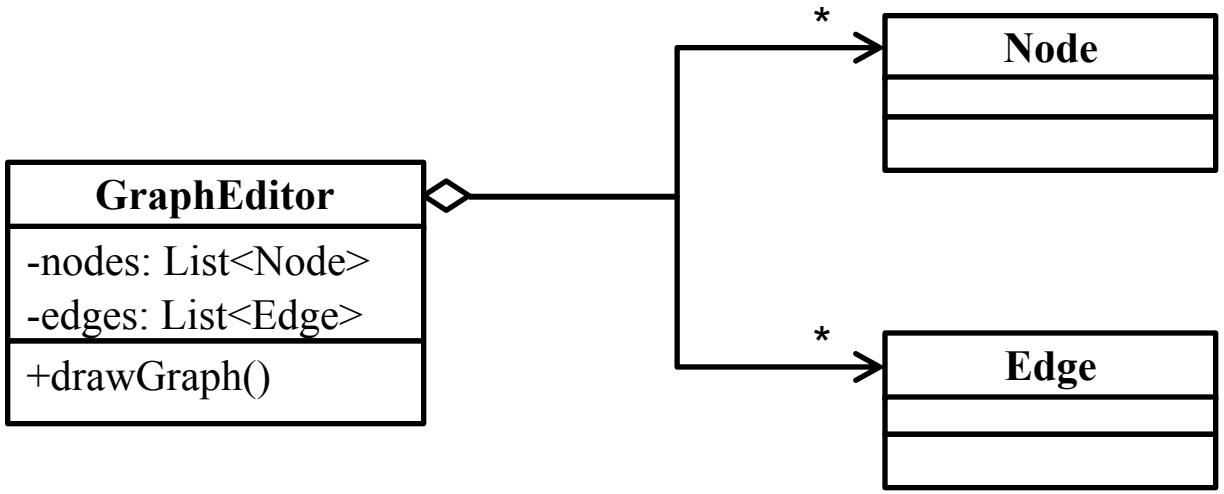
Department of Computer Science and Information Engineering
National Taiwan University

Requirements Statements

- A Graph Editor allows users to draw a graph which consists of nodes and edges.
- A node may have multiple edges while an edge must connect two nodes, and those connected nodes by an edge can be identical.
- This editor also provides the functionality for users to save/load graphs; the state of the graph will be kept while saving and restored to the editor while loading.

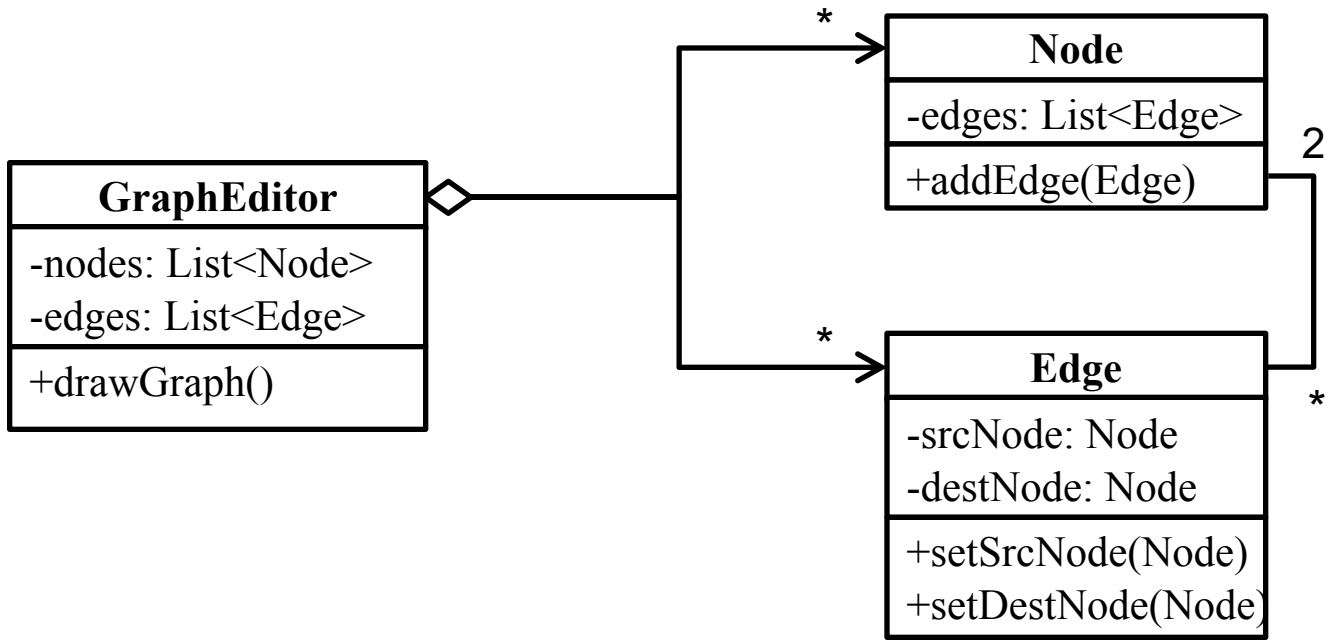
Requirements Statements$_1$

- A Graph Editor allows users to draw a graph which consists of nodes and edges.



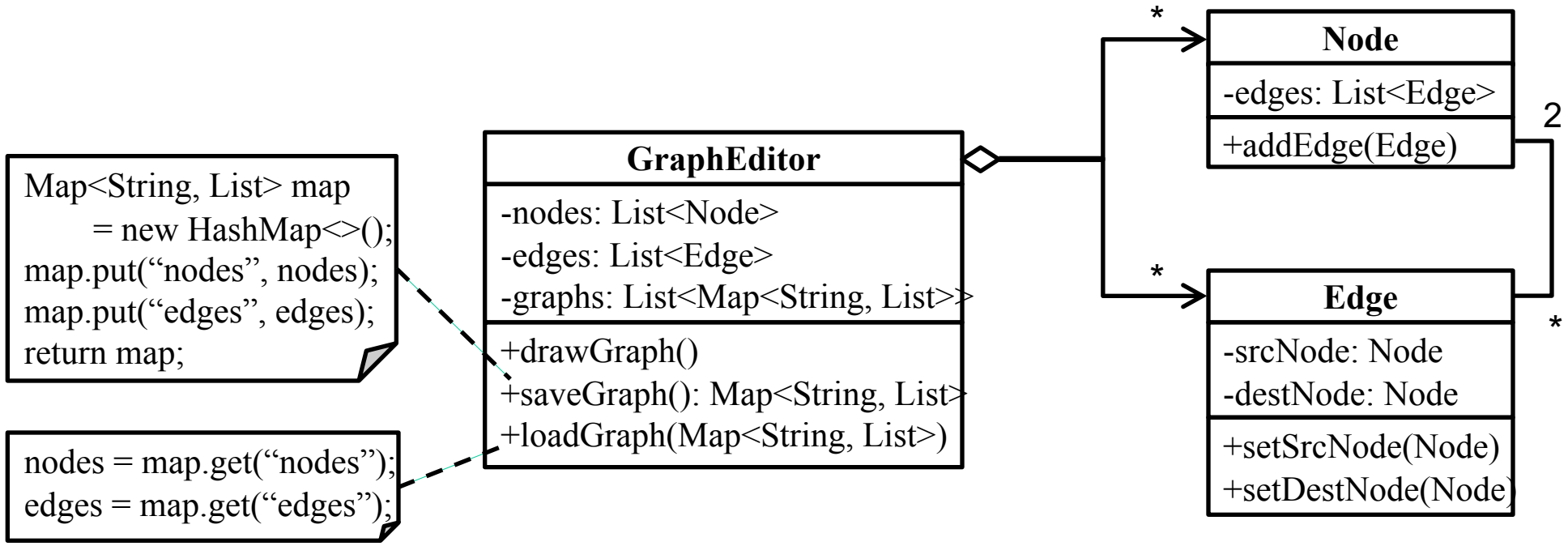
Requirements Statements$_2$

- A node may have multiple edges while an edge must connect two nodes, and those connected nodes by an edge can be identical.

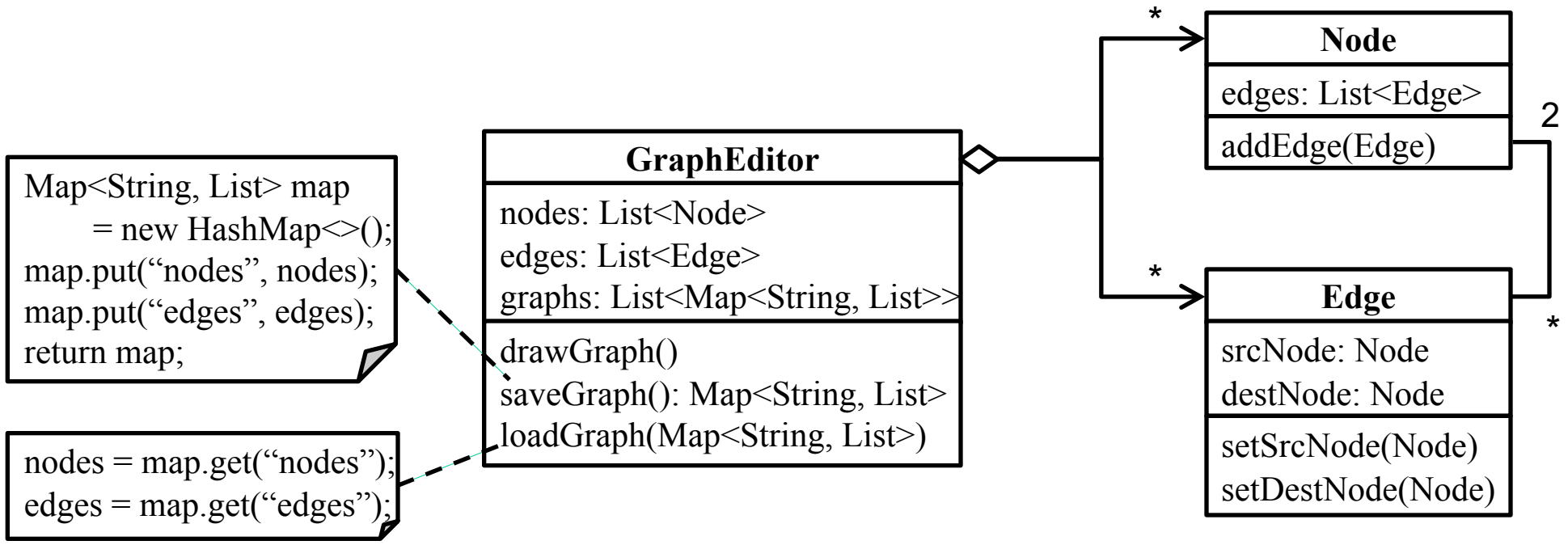


Requirements Statements$_3$

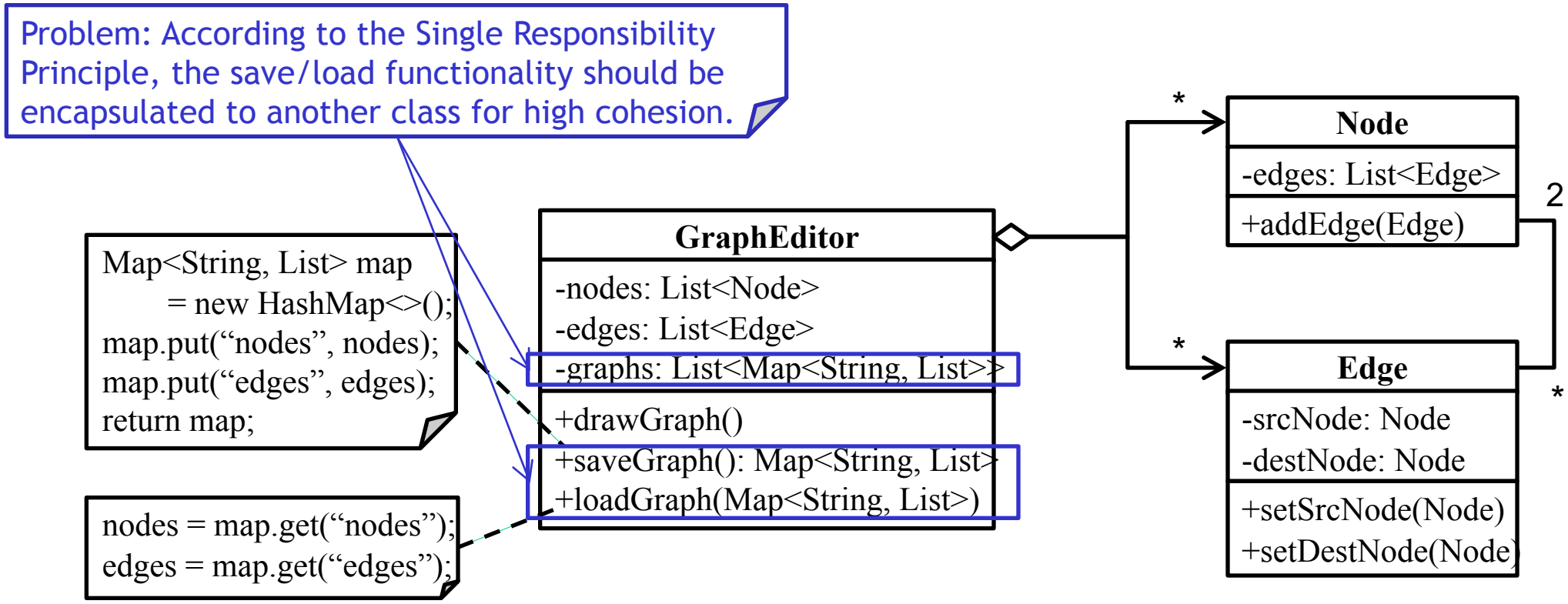
- This editor also provides the functionality for users to save/load graphs; the state of the graph will be kept while saving and restored to the editor while loading.



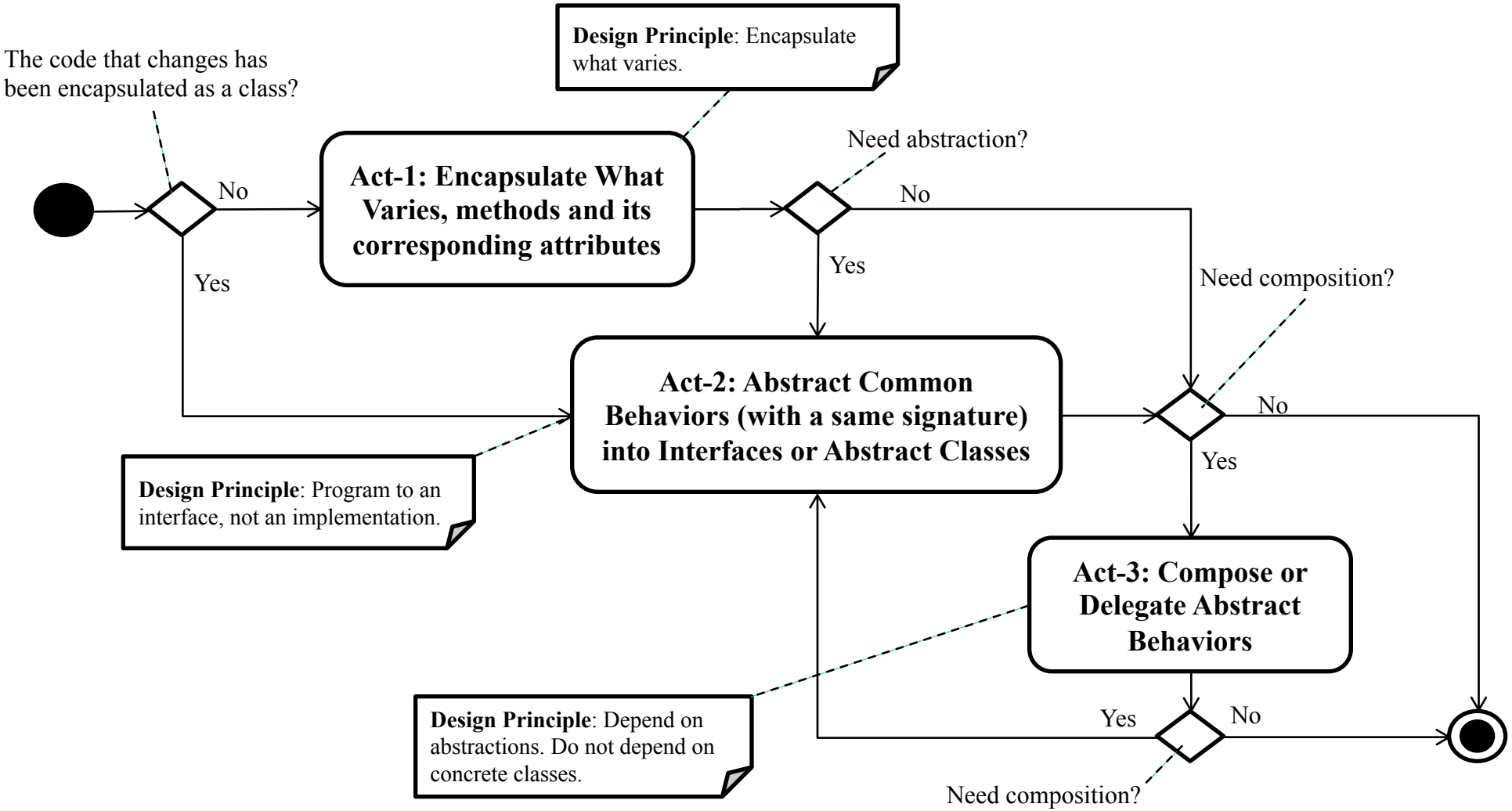
Initial Design - Class Diagram



The Problem with the Initial Design

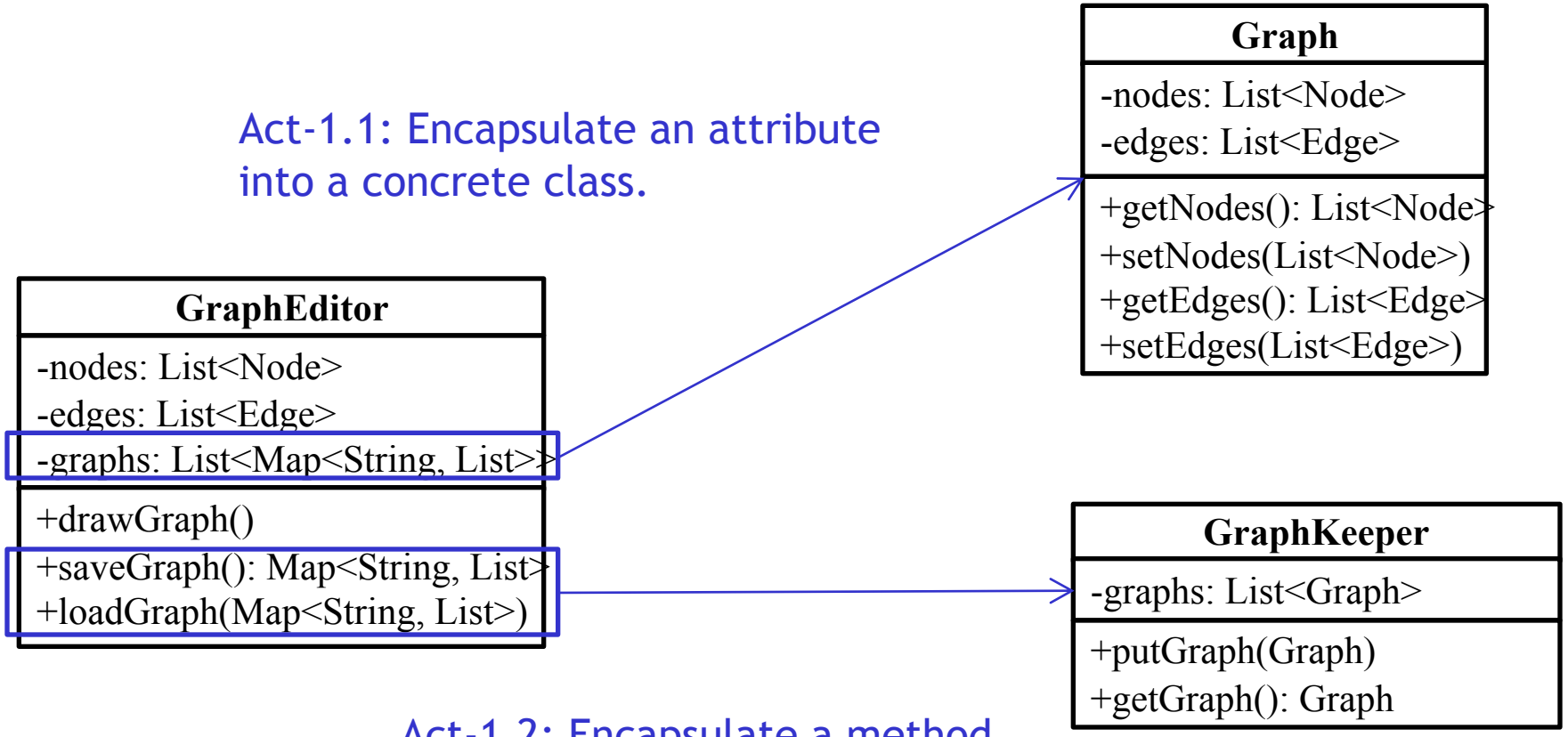


Design Process for Change



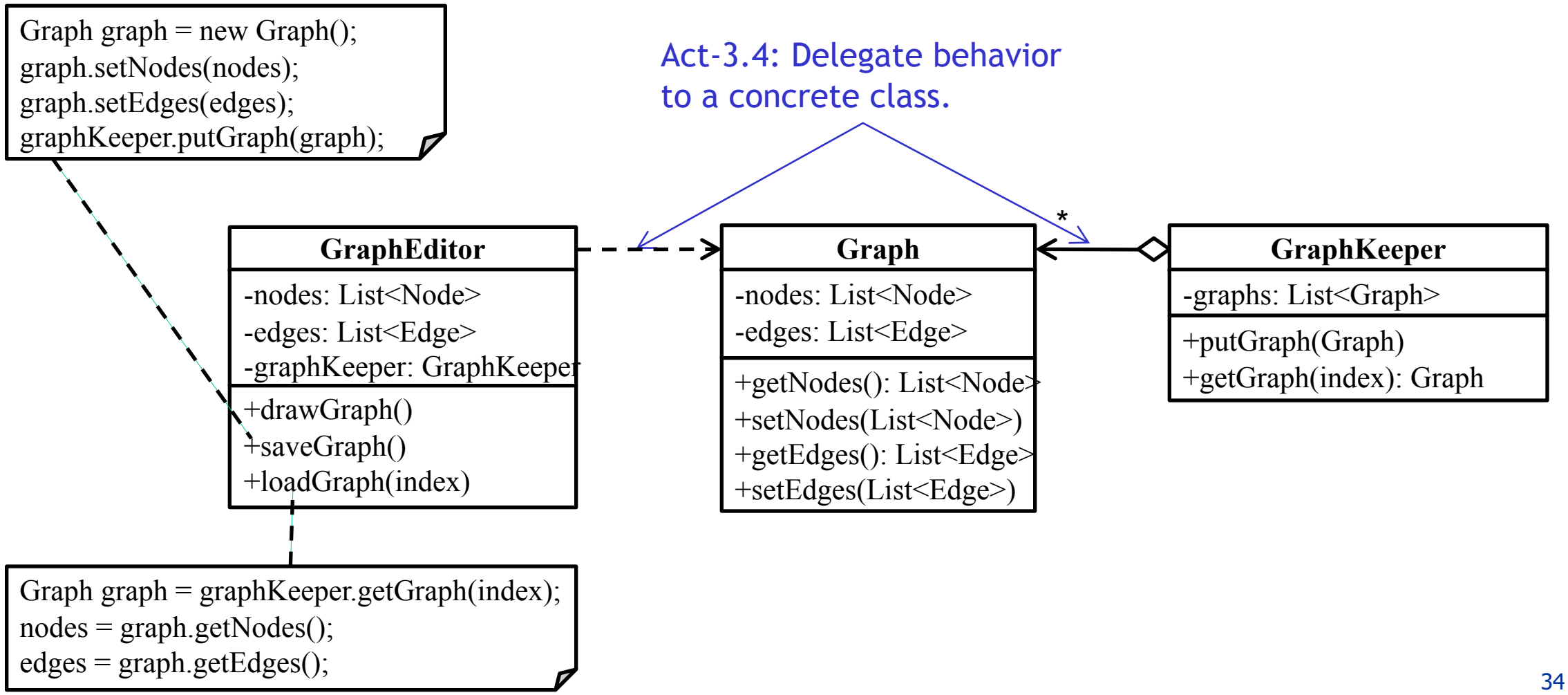
Act-1: Encapsulate What Varies

Act-1.1: Encapsulate an attribute into a concrete class.



Act-1.2: Encapsulate a method into a concrete class.

Act-3: Compose Abstract Behaviors



Refactored Design after Design Process