Verslag SWOP Groep 08: 11 april

Memento/Undo

Undo operatie is om use cases te reverten die het systeem op één ander manier veranderd hebben, dus zou het niet mogen dat een caretaker het huidig systeem steeds opslaat als een memento voor elke use case (Show project wijzigt bugtrap systeem niet). Dus moet er selectief opgeslagen worden: in elke use case moet een caretaker meegegeven worden de wijzigingen al dan niet op te slagen.

Dit zorgt ervoor dat de gekoppeld worden met de caretaker en dus weten de dat hun status opgeslagen wordt.

Een alternatief is dat we bij ALLE het systeem opslaan, en vervolgens de caretaker oproepen buiten de use cases. Gevolg: geen koppeling tussen caretaker en use case + hoge cohesie (use cases trekken zich niks aan van de caretaker).

```
SpecificUseCaseA class{
        private CareTaker caretaker;

        SpecificUseCase(CareTaker caretaker) { ...}
        void run(){
                caretaker.saveSystem();
                //do usecase
        }
}

SpecificUseCaseB class{
        private CareTaker caretaker;

        SpecificUseCase(CareTaker caretaker) { ...}
        void run(){
                //do usecase, geen saveSystem
        }
}
```

In plaats van:

```
class MainController{
        while(true){
                caretaker.saveSystem();
                callUseCase(int readIndex).run();
        }
}
```

Is deze koppeling aanvaarbaar ? Indien niet, kunnen we deze koppeling verlagen ? Alternatieven ?

Opslag van projectlijst en mailboxen

De objecten die moeten worden opgeslagen zijn List<Project> in projectService (bevat recursief de andere objecten zoals subsystem/bugreport ...) en mailboxen die gebonden zijn aan elke user (of zijn er nog andere objecten dan moet opgeslagen worden?).

Een eerste manier om het systeem op te slagen is door een deepcopy te voeren op listProject en users. Bij de undo-operatie vervangen we de referentie van listProject in projectService door de deepcopy ervan, hetzelfde voor users.

Het probleem is dat elke mailbox van elke user een lijst van registrations heeft (observers) die verwijzen naar subjects (project/subsystem/bugreport) en de subjects hebben hebben een verwijzing naar hun observers, maar deze bindingen worden niet consistent gekopieerd met deepcopy.

Er ontstaat dus een kopie van projecten en nog andere losse kopieën van projecten bij het kopiëren van de mailboxen. Is dit een aanvaardbaar design, ook al is dit geen memento ? Hoe kunnen we het probleem oplossen? Lost Serialization of Deserialization dit op of niet ? Mag dit gebruikt worden ?

Implementatie met memento pattern

Methode met memento kan gebeuren met een shallowcopy (of kan het ook met deepcopy/recursief ?). MementoListProject houdt een shallowcopy van listProject bij (Project referenties verwijzen wel naar de originele projecten).

Dan is er nog MementoProject, MementoSubsystem en MementoBugReport: elk kan het corresponderende object herstellen door te verwijzen naar het shallowcopy attribuut dat bijgehouden wordt.

De caretaker houdt een lijst bij van Snapshots, elk snapshot bevat een MementoListProject, een lijst van MementoProject, een lijst van MementoSubsystem etc ... Bij het herstellen zet caretaker de projectList in projectService terug naar de juiste referentie (shallowcopy uit MementoListProject).

Omdat de projecten zelf mogelijk gewijzigd werden, gaan we deze ook herstellen met de lijst van MementoProject, die de attributen van elk project herstelt en verwijst naar de shallowcopy van het subsystem. Hetzelfde idee wordt ook toegepast op subsystem/bugreport etc ...

Elk memento implementeert dan wel een interface memento:

```
class Snapshot{
        private List<Memento> mementos;
        void addMemento(Memento);
}
```

```
interface Memento{
	private Obect Originator;
	Object getOriginator();
	//Bij het implementeren bevat het dan de attributen van de originator die moeten worden
opgeslagen

}

ProjectMemento implementeert Memento{
	List<Subsystem> listsubsystem;
	String name;
	...
	ProjectMemento(..., List<Subsystem> list){
		..
		listsubsystem = new ArrayList<Subsystem>(list);
	}
}

Project  //Originator
{
	List<Subsystem> listsubsystem;
	ProjectMemento createMemento() {
		return new ProjectMemento(...,listsubsystem);
	}

	restore(Memento memento){
		this.x = memento.x;
		...
		this.listsubsystem = memento.listsubsystem;
	}
}

class caretaker{
	List<Snapshot> snapshots;
	public void restoreSystem(Snapshot){
		for(Memento memento : Snapshot.getMementos()
			memento.getObject().restore(memento);
		}

	Snapshot getSnapshot(index number);
```

```
        void saveSystem(){
                Snapshot snapshot = new Snapshot();
                snapshot.addMemento(projectService.createMemento());
                foreach (Project : projectService.getAllProjects())
                        snapshot.addMemento(project.createMemento());
                foreach (Subsystem : projectService.getAllSubSystem())
                        snapshot.addMemento(subsystem.createMemento());
                ...

                snapshots.add(Snapshot);
        }
}
```

Met deze methode houdt de caretaker een lijst van snapshots dat meerdere lijsten van MementoObjecten bijhoudt, terwijl bij de deepcopy methode, een snapshot enkel een lijst van projecten bevat. (+ user lijst maar dat moet ook bij memento methode).

Is de bovenstaande methode effectief een Memento pattern ? Waarom is de memento pattern nodig als men een simpele deepcopy kan uit voeren ? Welke design is te verkiezen ?

Komt het probleem dat voorkomt bij deepcopy ook voor bij memento methode?

In de bovenstaande code wordt er geprobeerd om gebruik te maken van polymorfisme van memento's maar dit zal niet lukken omdat de inhoud van elke memento sterk. Waardoor <<Interface>> Memento geen interface aanbiedt die breed genoeg is. Een slordige oplossing zou kunnen zijn om bij restore(Memento) in project, de Memento te casten naar ProjectMemento. Is deze cast dan oké ?

Zijn er nog alternatieven ?