

Advanced Aspects of Object-Oriented Programming (SS 2009)

Practice Sheet 7

Date of Issue: 09.06.09
Deadline: 15.06.09
(until 10 a.m. as PDF via E-Mail)

Exercise 1 Aliasing

- a) Give a solution to the signers issue on slide 8 of chapter 4.
b) Analyze the following code with respect to aliasing issues. Give a solution.

```
interface Observer<T extends Subject<T,? extends E>, E> {
    void notifyMe(Subject<T,? extends E> subj, E event);
}

class Subject<T extends Subject<T,E>, E> {
    protected List<Observer<T,? super E>> observers =
        new ArrayList<Observer<T,? super E>>();

    public void register(Observer<T,? super E> obs) {
        observers.add(obs);
    }

    public void unregister(Observer<T,? super E> obs) {
        observers.remove(obs);
    }

    public void notifyObservers(E event) {
        for (Observer<T,? super E> obs : observers) {
            obs.notifyMe(this, event);
        }
    }

    public List<Observer<T,? super E>> getObservers() {
        return observers;
    }
}

class WindowEvent {}
class SpecialWindowEvent extends WindowEvent {}

interface WindowObserver extends Observer<WindowSubject, WindowEvent> {}

class WindowObserverImpl implements WindowObserver {
    public void notifyMe(
        Subject<WindowSubject,? extends WindowEvent> subj,
        WindowEvent event) {
        System.out.println("I've got notified");
    }
}

class WindowSubject extends Subject<WindowSubject, SpecialWindowEvent> {
    @Override
    public void register(
        Observer<WindowSubject,? super SpecialWindowEvent> obs) {
        System.out.println("WindowObserver registered");
        super.register(obs);
    }

    @Override
    public void unregister(
        Observer<WindowSubject,? super SpecialWindowEvent> obs) {
        System.out.println("WindowObserver registered");
        super.unregister(obs);
    }

    @Override
    public void notifyObservers(SpecialWindowEvent event) {
        System.out.println("Notifying WindowObserver");
        super.notifyObservers(event);
    }
}
```

Exercise 2 Confined Types

Examine the code, available with this practice sheet on the web, with respect to confinedness. The classes `ProofTreeNodeIt` und `ProofContainer` should provide the externally visible interfaces and are thus not confined.

- a) Examine the class `ConfinedList`. Can we declare this class as confined? Can we modify the implementation as to make this class confined?
- b) Examine the class `ProofTreeNode` and answer the same questions as above.
- c) Examine the class `PTNIterator` and answer the same questions as above.

Exercise 3 Confined Types for Stojas

- a) Enhance the `StoJas` language by packages. Also add the possibility to declare classes as `public` (or not). Add the possibility to declare classes as confined. Define additional context conditions (similar to the static encapsulation rules starting on slide 37).
- b) Show (informally) that no encapsulation errors can occur in your enhanced language.
- c) Modify the dynamic semantics of `StoJas` such that programs with encapsulation errors get stuck.