Übung 01

Arbeitsaufwand insgesamt: 10h

Inhaltsverzeichnis

Teil 1 – Design Pattern Observer erweitern

In diesem Teil der Übung soll das im Unterricht besprochene Observer Pattern um einige Punkte erweitert werden.

Lösungsidee:

Der erste Punkt, den es zu lösen gilt, sind Notifications, welche über transitive Observer-Verbindungen mehrfach erhalten werden. Im in der Skizze gezeigten Beispiel würde z.B. S1 die Objekte S2/O1 und S3/O2 benachrichtigen, welche wiederum O3 benachrichtigen. Da die Benachrichtigungen vom selben Objekt ausgehen, bekommt O3 zweimal die gleiche Nachricht. Wird zusätzlich noch die Verbindung geschaffen, dass S1 nun O3 observiert, dann kann es zu niemals endenden Notification-Schleifen kommt.

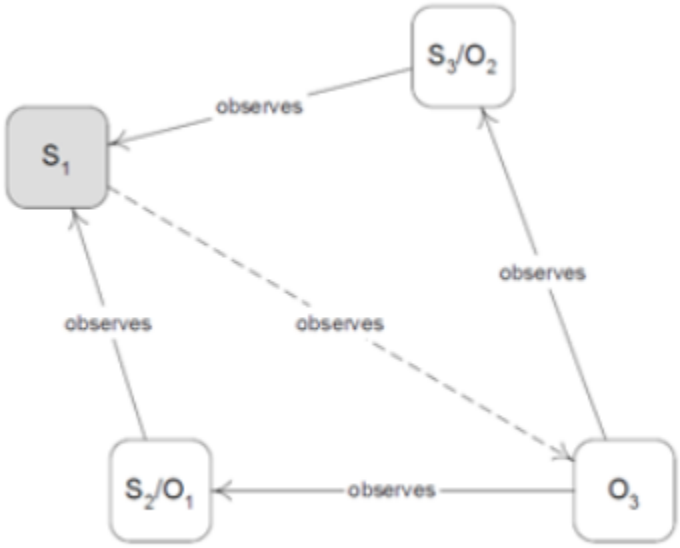


Abbildung 1: Objektgraph mehrerer Observer-Objekte.

Allem voran sieht es in der Grafik so aus, als wäre jedes Objekt gleichzeitig Subjekt und Observer. Hierfür würde ich eine eigene Klasse erstellen, welche vom AbstractSubject erbt und den Observer implementiert.

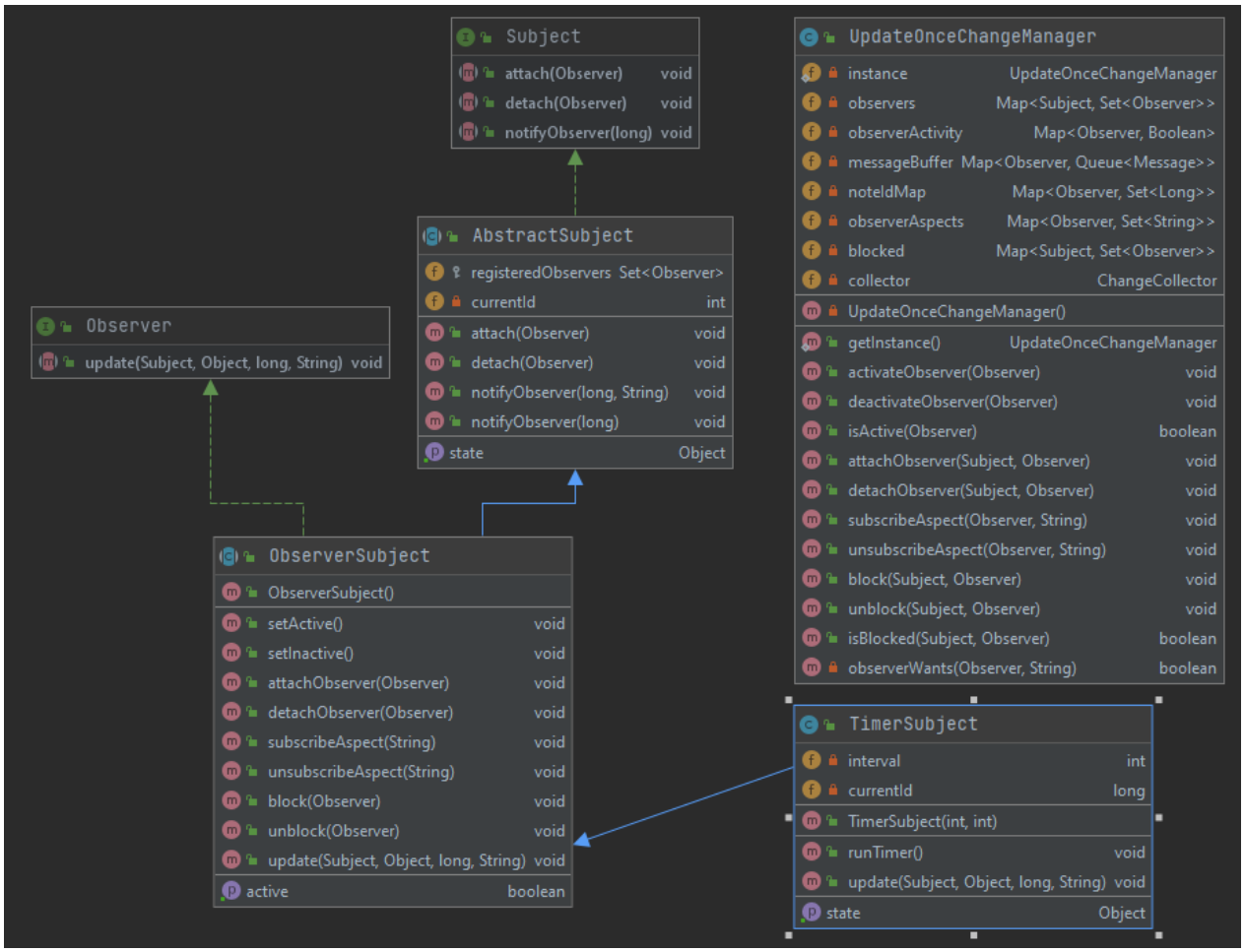
Als ersten Folgeschritt gilt es die Benachrichtigungen weiterzuleiten. Hierfür würde ich in der jeweiligen update Methode einfach wiederum alle Observer des neuen Subjekts mittels „notifyObservers()“ benachrichtigen. In einem einfachen Beispiel könnte man einfach den Status aller observierenden Objekte „synchronisieren“: Das heißt, sobald eine Benachrichtigung eintrifft, wird der Status (z.B. Ticks, beim TimerSubject aus dem Unterricht) des Objekts mit dem aus der Benachrichtigung überschrieben. Dadurch wird auch der richtige Wert bei „NotifyObservers()“ weitergegeben.

Das eigentliche Problem, dass nun mehrere gleiche Nachrichten über verschiedene Wege eintreffen können, würde ich über den Subjekten zugehörige IDs regeln. Jedes Mal, wenn ein Subjekt jemanden benachrichtigt, wird die ID mitgeschickt (über notifyObservers) und anschließend erhöht. Somit erhält jede Nachricht eine eindeutige ID. Diese IDs kann der ChangeManager für jeden Observer mitführen, und nur dann eine Benachrichtigung senden, falls dieser noch keine erhalten hat.

Ähnlich verhält es sich beim Unterdrücken von Nachrichten. Um generisch zu bleiben, wird zentral beim ChangeManager gespeichert, ob ein Observer Benachrichtigungen erhalten will oder nicht. Falls dieser keine erhalten will, werden zudem die Nachrichten je Observer in einer Liste gespeichert und beim Reaktivieren der Benachrichtigungen sofort ausgegeben.

Will ein Observer nur Nachrichten mit bestimmten Aspekten erhalten, so muss wiederum die NotifyObservers() und update() Methode angepasst werden. Welche Aspekte ein Observer erhalten will, wird wiederum in einer Map beim ChangeManager gespeichert. Zusätzlich muss jetzt natürlich beim Subjekt definiert werden, um welchen Aspekt es sich bei der Nachricht handelt. Hat ein Observer garkeinen Aspekt hinterlegt, so erhält dieser standardmäßig alle Nachrichten, unabhängig vom Aspekt.

Zu guter Letzt kann ein Subjekt noch bestimmte Observer blockieren. Diese Information wird wieder beim ChangeManager innerhalb einer Map eingetragen. Im Falle, dass ein blockierter Observer das Subjekt observieren will, so wird dessen Update-Methode aufgerufen.



Lösung:

Als IDEA Projekt im Archiv
Nachfolgender Source-Code

Source-Code:

ObserverTest.java

```
package ue02.part1.test;


import ue02.part1.patterns.impl.Timer;

public class ObserverTest {

    public static void test_reference() {
        Timer s1 = new Timer(10,1000);
        Timer s2o1 = new Timer(10,2000);
        Timer s3o2 = new Timer(10,1500);
        Timer o3 = new Timer(10,1000);

        s1.block(s2o1);
        s1.attachObserver(s2o1);
        s1.attachObserver(s3o2);
        s2o1.attachObserver(o3);
        s3o2.attachObserver(o3);
        o3.attachObserver(s1);
        s1.runTimer();
    }

    public static void test_suppress() {
        Timer s1 = new Timer(5,1000);
        Timer obs1 = new Timer(10, 1000);
        Timer obs2 = new Timer(10, 1000);

        s1.attachObserver(obs1);
        s1.attachObserver(obs2);
        obs2.setInactive();
        s1.runTimer();
        obs2.setActive();
    }

    public static void test_aspects() {
        Timer s1 = new Timer(5,1000);
        Timer obs1 = new Timer(10, 1000);
        Timer obs2 = new Timer(10, 1000);

        s1.attachObserver(obs1);
        s1.attachObserver(obs2);
        obs2.subscribeAspect("TimerEnd");
        obs2.setInactive();
        s1.runTimer();
        obs2.setActive();
    }

    public static void test_block() {
        Timer s1 = new Timer(5,1000);
        Timer obs1 = new Timer(10, 1000);

        s1.block(obs1);
        s1.attachObserver(obs1);
    }
```

```
    public static void main(String[] args) {
        test_reference();
        //test_suppress();
        //test_aspects();
        //test_block();
    }
}
```

Observer.java

```
package ue02.part1.patterns;

public interface Observer {

    void update(Subject source, Object argument, long noteId, String aspect);

}
```

Subject.java

```
package ue02.part1.patterns;

public interface Subject {
    void attach(Observer observer);
    void detach(Observer observer);
    void notifyObserver(long noteId);
}
```

AbstractSubject

```
package ue02.part1.patterns.impl;


import ue02.part1.patterns.Observer;
import ue02.part1.patterns.Subject;

import java.util.HashSet;
import java.util.Set;

public abstract class AbstractSubject implements Subject {

    protected Set<Observer> registeredObservers = new HashSet<>();
    private int currentId = 0;

    @Override
    public void attach(Observer observer) {
        if(observer != null) {
            registeredObservers.add(observer);
        }
    }

    @Override
    public void detach(Observer observer) {
        if(observer !=null) {
            registeredObservers.remove(observer);
        }
    }

    public void notifyObserver(long noteId, String aspect) {
        for(Observer observer : registeredObservers) {
            observer.update(this, getState(), noteId, aspect);
        }
        currentId++;
    }

    @Override
    public void notifyObserver(long noteId) {
        notifyObserver(noteId, "default");
    }


    public abstract Object getState();
}
```

ObserverSubject.java

```
package ue02.part1.patterns.impl;

import ue02.part1.patterns.Observer;
import ue02.part1.patterns.Subject;


public abstract class ObserverSubject extends AbstractSubject implements
Observer {


    public ObserverSubject() {

    }

    public void setActive() {
        UpdateOnceChangeManager.getInstance().activateObserver(this);
    }
    public void setInactive() {
        UpdateOnceChangeManager.getInstance().deactivateObserver(this);
    }
    public boolean isActive() {
        return UpdateOnceChangeManager.getInstance().isActive(this);
    }

    public void attachObserver(Observer observer) {
        UpdateOnceChangeManager.getInstance().attachObserver(this,observer);
    }
    public void detachObserver(Observer observer) {
        UpdateOnceChangeManager.getInstance().detachObserver(this,observer);
    }

    public void subscribeAspect(String aspect) {
        UpdateOnceChangeManager.getInstance().subscribeAspect(this,aspect);
    }
    public void unsubscribeAspect(String aspect) {
        UpdateOnceChangeManager.getInstance().unsubscribeAspect(this,aspect);
    }

    public void block(Observer observer) {
        UpdateOnceChangeManager.getInstance().block(this, observer);
    }
    public void unblock(Observer observer) {
        UpdateOnceChangeManager.getInstance().unblock(this, observer);
    }

    @Override
    public void update(Subject source, Object argument, long noteId, String
aspect) {
        System.out.println("Observer: " + this + " received "+aspect+"
notification (ID="+noteId+") from: " + source + " with argument: " +
argument);
    }

}
```

Timer.java

```
package ue02.part1.patterns.impl;

import ue02.part1.patterns.Subject;

public class Timer extends ObserverSubject {

    private int ticks = 0;
    private int interval = 0;
    private long currentId;

    public Timer(int ticks, int interval) {
        this.ticks = ticks;
        this.interval = interval;
    }

    public void runTimer() {
        if(ticks > 0 && interval > 0) {
            int tmpTicks = ticks;
            while(ticks > 0){
                try {
                    Thread.sleep(interval);
                } catch(InterruptedException e) {}
                ticks--;
                if(ticks == tmpTicks/2)
                {
                    notifyObserver(currentId++, "TimerHalf");
                }
                notifyObserver(currentId++, "TimerTick");
            }
            notifyObserver(currentId++, "TimerEnd");
            ticks = tmpTicks;
        }
    }

    @Override
    public void update(Subject source, Object argument, long noteId, String
aspect) {
        super.update(source, argument, noteId, aspect);
        if(noteId != -1)
        {
            ticks = (int)argument;
            notifyObserver(noteId, aspect);
        }
    }

    @Override
    public Object getState() {
        return ticks;
    }

}
```

UpdateOnceChangeManager.java

```
package ue02.part1.patterns.impl;

import ue02.part1.patterns.Observer;
import ue02.part1.patterns.Subject;

import java.util.*;
import java.util.concurrent.ConcurrentLinkedDeque;
import java.util.concurrent.ConcurrentLinkedQueue;

public class UpdateOnceChangeManager {
    private static UpdateOnceChangeManager instance = null;
    private Map<Subject, Set<Observer>> observers = new HashMap<>();
    private Map<Observer, Boolean> observerActivity = new HashMap<>();
    private Map<Observer, Queue<Message>> messageBuffer = new HashMap<>();
    private Map<Observer, Set<Long>> noteIdMap = new HashMap<>();
    private Map<Observer, Set<String>> observerAspects = new HashMap<>();
    private Map<Subject, Set<Observer>> blocked = new HashMap<>();

    private ChangeCollector collector = new ChangeCollector();

    private UpdateOnceChangeManager() {}

    public static UpdateOnceChangeManager getInstance() {
        if(instance == null) {
            instance = new UpdateOnceChangeManager();
        }
        return instance;
    }

    public void activateObserver(Observer observer) {
        observerActivity.put(observer,true);

        // if observer was previously inactive, send all the buffered messages
        if(messageBuffer.containsKey(observer) &&
!messageBuffer.get(observer).isEmpty())
        {
            System.out.println("Notifications during inactivity:");
            while(!messageBuffer.get(observer).isEmpty()) {
                Message message = messageBuffer.get(observer).remove();
                collector.update(message.getSubject(), message.getArgument(),
message.getId(), message.getAspect());
            }
        }
    }
    public void deactivateObserver(Observer observer) {
        observerActivity.put(observer,false);
    }
    public boolean isActive(Observer observer) {
        if(observerActivity.containsKey(observer)) {
            return observerActivity.get(observer);
        } else
        {
            return true;
        }

    }
```

```
    public void attachObserver(Subject subject, Observer observer) {
        if(!observers.containsKey(subject))
        {
            observers.put(subject, new HashSet<>());
        }
        if(!isBlocked(subject, observer)) {
            Set<Observer> obs = observers.get(subject);
            obs.add(observer);
            subject.attach(collector);
        } else {
            observer.update(subject, "Blocked from observation.", -1,
"block");
        }


    }
    public void detachObserver(Subject subject, Observer observer) {
        if(observers.containsKey(subject))
        {
            observers.get(subject).remove(observer);
        }
    }

    public void subscribeAspect(Observer observer, String aspect) {
        if(!observerAspects.containsKey(observer)) {
            observerAspects.put(observer, new HashSet<>());
        }
        observerAspects.get(observer).add(aspect);
    }
    public void unsubscribeAspect(Observer observer, String aspect) {
        if(observerAspects.containsKey(observer)) {
            observerAspects.get(observer).remove(aspect);
        }
    }

    public void block(Subject subject, Observer observer) {
        if(!blocked.containsKey(subject)) {
            blocked.put(subject, new HashSet<>());
        }
        blocked.get(subject).add(observer);
    }
    public void unblock(Subject subject, Observer observer) {
        if(blocked.containsKey(subject)) {
            blocked.get(subject).remove(observer);
        }
    }
    public boolean isBlocked(Subject subject, Observer observer) {
        if(!blocked.containsKey(subject)) {
            return false;
        }

        return blocked.get(subject).size()>0 &&
blocked.get(subject).contains(observer);
    }

    private boolean observerWants(Observer observer, String aspect) {
        if(observerAspects.containsKey(observer) &&
observerAspects.get(observer).size()>0 &&
```

```
!observerAspects.get(observer).contains(aspect)) {
            return false;
        }

        return true;
    }

    private class ChangeCollector implements Observer {
        @Override
        public void update(Subject source, Object argument, long noteId,
String aspect) {
            Set<Observer> obs = observers.get(source);
            for(Observer observer : obs) {
                if(isActive(observer)) {
                    // forward update to actual observers
                    if(!noteIdMap.containsKey(observer))
                    {
                        noteIdMap.put(observer, new HashSet<>());
                    }

                    if(!noteIdMap.get(observer).contains(noteId))
                    {
                        noteIdMap.get(observer).add(noteId);
                        if(observerWants(observer, aspect)) {
                            System.out.print("From ChangeManager - ");
                            observer.update(source, argument, noteId, aspect);
                        }
                    }
                } else {
                    if(observerWants(observer, aspect)) {
                        if(!messageBuffer.containsKey(observer)) {
                            messageBuffer.put(observer, new
ConcurrentLinkedQueue<>());
                        }
                        messageBuffer.get(observer).add(new Message(source,
argument, noteId,aspect));
                    }
                }
            }
        }
    }

    private class Message {
        private Subject subject;
        private Object argument;
        private long id;
        private String aspect;

        public Message(Subject subject, Object argument, long id, String
aspect) {
            this.subject = subject;
            this.argument = argument;
            this.id = id;
            this.aspect = aspect;
        }

        public Subject getSubject() {
            return subject;
```

```
        }
        public Object getArgument() {
            return argument;
        }
        public long getId() { return id;}
        public String getAspect() {
            return aspect;
        }
    }
}
```

Testfälle

- Referenzbeispiel
- Inaktive Observer
- Aspekte
- Observer blockieren

Referenzbeispiel:

```
From ChangeManager - Observer: ue02.part1.patterns.impl.Timer@15aeb7ab received TimerHalf
notification (ID=0) from: ue02.part1.patterns.impl.Timer@7ef20235 with argument: 1
From ChangeManager - Observer: ue02.part1.patterns.impl.Timer@7b23ec81 received TimerHalf
notification (ID=0) from: ue02.part1.patterns.impl.Timer@15aeb7ab with argument: 1
From ChangeManager - Observer: ue02.part1.patterns.impl.Timer@7ef20235 received TimerHalf
notification (ID=0) from: ue02.part1.patterns.impl.Timer@7b23ec81 with argument: 1
From ChangeManager - Observer: ue02.part1.patterns.impl.Timer@27d6c5e0 received TimerHalf
notification (ID=0) from: ue02.part1.patterns.impl.Timer@7ef20235 with argument: 1
From ChangeManager - Observer: ue02.part1.patterns.impl.Timer@15aeb7ab received TimerTick
notification (ID=1) from: ue02.part1.patterns.impl.Timer@7ef20235 with argument: 1
From ChangeManager - Observer: ue02.part1.patterns.impl.Timer@7b23ec81 received TimerTick
notification (ID=1) from: ue02.part1.patterns.impl.Timer@15aeb7ab with argument: 1
From ChangeManager - Observer: ue02.part1.patterns.impl.Timer@7ef20235 received TimerTick
notification (ID=1) from: ue02.part1.patterns.impl.Timer@7b23ec81 with argument: 1
From ChangeManager - Observer: ue02.part1.patterns.impl.Timer@27d6c5e0 received TimerTick
notification (ID=1) from: ue02.part1.patterns.impl.Timer@7ef20235 with argument: 1
From ChangeManager - Observer: ue02.part1.patterns.impl.Timer@15aeb7ab received TimerTick
notification (ID=2) from: ue02.part1.patterns.impl.Timer@7ef20235 with argument: 0
From ChangeManager - Observer: ue02.part1.patterns.impl.Timer@7b23ec81 received TimerTick
notification (ID=2) from: ue02.part1.patterns.impl.Timer@15aeb7ab with argument: 0
From ChangeManager - Observer: ue02.part1.patterns.impl.Timer@7ef20235 received TimerTick
notification (ID=2) from: ue02.part1.patterns.impl.Timer@7b23ec81 with argument: 0
From ChangeManager - Observer: ue02.part1.patterns.impl.Timer@27d6c5e0 received TimerTick
notification (ID=2) from: ue02.part1.patterns.impl.Timer@7ef20235 with argument: 0
From ChangeManager - Observer: ue02.part1.patterns.impl.Timer@15aeb7ab received TimerEnd
notification (ID=3) from: ue02.part1.patterns.impl.Timer@7ef20235 with argument: 0
From ChangeManager - Observer: ue02.part1.patterns.impl.Timer@7b23ec81 received TimerEnd
notification (ID=3) from: ue02.part1.patterns.impl.Timer@15aeb7ab with argument: 0
From ChangeManager - Observer: ue02.part1.patterns.impl.Timer@7ef20235 received TimerEnd
notification (ID=3) from: ue02.part1.patterns.impl.Timer@7b23ec81 with argument: 0
From ChangeManager - Observer: ue02.part1.patterns.impl.Timer@27d6c5e0 received TimerEnd
notification (ID=3) from: ue02.part1.patterns.impl.Timer@7ef20235 with argument: 0
```

In meinem Fall wird die Benachrichtigung auch zum Ursprungsobjekt weitergeleitet, es werden aber in jedem Fall Schleifen und doppelte Ausgaben verhindert. Das heißt: 2 Ausgaben für die Mittel-Elemente (S2O1 und S3O2), eine Ausgabe für O3 und eine für S1.

Inaktive Observer:

```
From ChangeManager - Observer: ue02.part1.patterns.impl.Timer@27d6c5e0 received TimerTick
notification (ID=0) from: ue02.part1.patterns.impl.Timer@7ef20235 with argument: 2
From ChangeManager - Observer: ue02.part1.patterns.impl.Timer@27d6c5e0 received TimerHalf
notification (ID=1) from: ue02.part1.patterns.impl.Timer@7ef20235 with argument: 1
From ChangeManager - Observer: ue02.part1.patterns.impl.Timer@27d6c5e0 received TimerTick
notification (ID=2) from: ue02.part1.patterns.impl.Timer@7ef20235 with argument: 1
From ChangeManager - Observer: ue02.part1.patterns.impl.Timer@27d6c5e0 received TimerTick
notification (ID=3) from: ue02.part1.patterns.impl.Timer@7ef20235 with argument: 0
From ChangeManager - Observer: ue02.part1.patterns.impl.Timer@27d6c5e0 received TimerEnd
notification (ID=4) from: ue02.part1.patterns.impl.Timer@7ef20235 with argument: 0
Notifications during inactivity:
From ChangeManager - Observer: ue02.part1.patterns.impl.Timer@15aeb7ab received TimerTick
notification (ID=0) from: ue02.part1.patterns.impl.Timer@7ef20235 with argument: 2
From ChangeManager - Observer: ue02.part1.patterns.impl.Timer@15aeb7ab received TimerHalf
notification (ID=1) from: ue02.part1.patterns.impl.Timer@7ef20235 with argument: 1
From ChangeManager - Observer: ue02.part1.patterns.impl.Timer@15aeb7ab received TimerTick
notification (ID=2) from: ue02.part1.patterns.impl.Timer@7ef20235 with argument: 1
From ChangeManager - Observer: ue02.part1.patterns.impl.Timer@15aeb7ab received TimerTick
notification (ID=3) from: ue02.part1.patterns.impl.Timer@7ef20235 with argument: 0
From ChangeManager - Observer: ue02.part1.patterns.impl.Timer@15aeb7ab received TimerEnd
notification (ID=4) from: ue02.part1.patterns.impl.Timer@7ef20235 with argument: 0
```

Der zweite Observer ist inaktiv während runTimer() läuft, weswegen die Nachrichten erst beim Aktiv setzen ausgegeben werden.

Aspekte:

```
From ChangeManager - Observer: ue02.part1.patterns.impl.Timer@27d6c5e0 received TimerHalf
notification (ID=0) from: ue02.part1.patterns.impl.Timer@7ef20235 with argument: 1
From ChangeManager - Observer: ue02.part1.patterns.impl.Timer@27d6c5e0 received TimerTick
notification (ID=1) from: ue02.part1.patterns.impl.Timer@7ef20235 with argument: 1
From ChangeManager - Observer: ue02.part1.patterns.impl.Timer@27d6c5e0 received TimerTick
notification (ID=2) from: ue02.part1.patterns.impl.Timer@7ef20235 with argument: 0
From ChangeManager - Observer: ue02.part1.patterns.impl.Timer@27d6c5e0 received TimerEnd
notification (ID=3) from: ue02.part1.patterns.impl.Timer@7ef20235 with argument: 0
Notifications during inactivity:
From ChangeManager - Observer: ue02.part1.patterns.impl.Timer@15aeb7ab received TimerEnd
notification (ID=3) from: ue02.part1.patterns.impl.Timer@7ef20235 with argument: 0
```

Der erste Observer erhält alle Benachrichtigungen, da er keinen Aspekt bestimmt hat. Der zweite Observer (welcher für Übersicht vorerst auf Inaktiv gestellt wurde) erhält nur die TimerEnd Benachrichtigung.

Observer blockieren:

```
Observer: ue02.part1.patterns.impl.Timer@27d6c5e0 received block notification (ID=-1)
from: ue02.part1.patterns.impl.Timer@7ef20235 with argument: Blocked from observation.
```

Teil 2 – Datei-Utilities

Im zweiten Teil der Übung sollen die Utility-Programme head, tail, LOC und TreeSize implementiert werden, welche großteils bereits aus Linux bekannt sind.

Lösungsidee:

Gleich zu Beginn möchte ich Head beschreiben, weil es von allen Programmen am simpelsten ist. Hierfür muss einfach ein File geöffnet werden und dessen Zeilen ausgegeben werden, bis eine bestimmte Zeilenanzahl erreicht ist.

Ähnlich dazu ist Tail: Es sollen die letzten X Zeilen im File ausgegeben werden. Problem hierbei ist, dass wir, ohne vorher alle Zeilen durchgearbeitet zu haben, nicht wissen, wie lange die Datei ist. Meine Lösung dafür wäre, dass ich das File zuerst öffne, die Zeilenanzahl bestimme und danach nochmals lese und ab der richtigen Zeile die Ausgabe starte.

Das Tool LOC soll folgende Statistiken zu Files ermitteln: Anzahl der Zeilen, Wörter und Zeichen (ohne Leerzeichen). Die Zeilenanzahl ist dabei am einfachsten zu ermitteln, da Klassen wie der BufferedReader die File-Inhalte zeilenweise liefern können und das Inkrementieren scheinbar einfach ist. Danach kann die Zeile Zeichen für Zeichen durchsucht werden. Wird ein normaler Buchstabe gefunden (kein Space), so wird der Zähler für Zeichen erhöht. Liegt das Zeichen zudem nach einem oder mehreren Leerzeichen, oder am Zeilenanfang, dann wird der Wort-Zähler erhöht. Diese Zähler können alle gesammelt ausgegeben werden.

Zu guter Letzt soll noch TreeSize implementiert werden, welche ein Verzeichnis rekursiv durchwandert und die Größen aller Dateien und Directories schön visualisiert ausgibt. Hier ergibt sich ein ähnliches Problem wie bei „Tail“. Es ist unmöglich die Ordnergröße zu bestimmen, ohne vorher alle Unterordner zu durchwandern und dessen Files aufzusummieren. Um die gewollte Reihenfolge beizubehalten, müssen die Ergebnisse zwischengespeichert werden und dürfen erst nach Abarbeitung ausgegeben werden. Bei der Speicherung ist es außerdem wichtig die Reihenfolge einzuhalten, sowie eine Einrückung je Element zu definieren.

Lösung:

Als Java Projekt im Archiv
Nachfolgender Source-Code

Source-Code:

FileTest.java

```
package ue02.part2.test;

import ue02.part2.utility.FileUtility;

import java.io.File;
import java.sql.SQLOutput;

public class FileTest {

    public static void nonexistant() {
        System.out.println("Testing nonexistant file (applies to all
utilities):");
        FileUtility.head("Iwasneverhere.txt");
        System.out.println();
    }
    public static void head_test() {
        System.out.println("Test mit 6 Zeilen und Head 5: ");
        FileUtility.head("testfolder/test.txt",5);
        System.out.println();

        System.out.println("Test mit 6 Zeilen und Head 30: ");
        FileUtility.head("testfolder/test.txt",30);
        System.out.println();

        System.out.println("Test mit 6 Zeilen und Head ohne Parameter:");
        FileUtility.head("testfolder/test.txt");
        System.out.println();

        System.out.println("Test mit leerem File und Head 5: ");
        FileUtility.head("testfolder/empty.txt",5);
        System.out.println();

        System.out.println("Test mit Head -1: ");
        FileUtility.head("testfolder/test.txt",-1);
        System.out.println();
    }
    public static void tail_test() {
        System.out.println("Test mit 6 Zeilen und Tail 3:");
        FileUtility.tail("testfolder/test.txt",3);
        System.out.println();

        System.out.println("Test mit 6 Zeilen und Tail 42:");
        FileUtility.tail("testfolder/test.txt",42);
        System.out.println();

        System.out.println("Test mit 6 Zeilen und Tail -1:");
        FileUtility.tail("testfolder/test.txt",-1);
        System.out.println();

        System.out.println("Test mit leerem File und Tail 3:");
        FileUtility.tail("testfolder/empty.txt",3);
        System.out.println();
    }
    public static void test_loc() {
        System.out.println("LOC for file with 6 lines, 9 words and 19
```

```
characters:");
        FileUtility.LOC("testfolder/test.txt");
        System.out.println();

        System.out.println("LOC for empty file:");
        FileUtility.LOC("testfolder/empty.txt");
        System.out.println();
    }
    public static void test_treeSize() {
        System.out.println("Test with file:");
        FileUtility.treeSize("testfolder/test.txt");
        System.out.println();

        System.out.println("Test with src/ue02/part2 directory:");
        FileUtility.treeSize("src\\ue02\\part2");
        System.out.println();

        System.out.println("Test with nonexistant directory:");
        FileUtility.treeSize("empty");
        System.out.println();
    }


    public static void main(String[] args) {
        nonexistant();
        //head_test();
        //tail_test();
        //test_loc();
        //test_treeSize();
    }
}
```

FileUtility.java

```java
package ue02.part2.utility;

import java.io.File;
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.text.DecimalFormat;
import java.text.NumberFormat;
import java.util.ArrayList;
import java.util.Locale;
import java.util.Scanner;

public class FileUtility {

    private static ArrayList<TreeInfo> treeInfoList = new ArrayList<>();

    public static void head(String path, int count) {
        // Open file and go through the lines and count them
        try(Scanner scanner = new Scanner(new File(path))) {
            if(count > 0) {
                while (scanner.hasNext() && count > 0) {
                    System.out.println(scanner.nextLine());
                    count--;
                }
            }
        } catch(IOException e) {
            System.err.println("Could not read from file: " + path);
            e.printStackTrace();
        }
    }

    public static void head(String path) {
        // Print every line using scanner
        try(Scanner scanner = new Scanner(new File(path))) {
                while (scanner.hasNext()) {
                    System.out.println(scanner.nextLine());
            }
        } catch(IOException e) {
            System.err.println("Could not read from file: " + path);
            e.printStackTrace();
        }
    }

    public static void tail(String path, int count) {
        try {
            if(count > 0) {
                // determine lines (+ linecount)
                Object[] lines = Files.lines(Paths.get(path)).toArray();

                int position = lines.length < count ? 0 : lines.length-count;

                // print the wanted lines
                while (position<lines.length) {
                    System.out.println(lines[position]);
                    position++;
                }
            }
```

```
        } catch(IOException e) {
            System.err.println("Could not read from file: " + path);
            e.printStackTrace();
        }
    }

    public static void LOC(String path) {
        try(Scanner scanner = new Scanner(new File(path))) {
            // intialize counters
            int linecount = 0;
            int wordcount = 0;
            int charcount = 0;

            String line;
            while (scanner.hasNext()) {
                // count every line
                line = scanner.nextLine();
                linecount++;

                boolean word = false;
                for(char c:line.toCharArray()) {
                    if(!Character.isSpaceChar(c)) {
                        // count every non-space-char
                        charcount++;
                        if(!word) {
                            // if we're not inside a word, and a character
appears, increment word counter
                            wordcount++;
                            word = true;
                        }
                    } else {
                        word = false;
                    }
                }
            }

            System.out.println("Printing statistics for file: "+path);
            System.out.println("Lines: "+linecount);
            System.out.println("Words: "+wordcount);
            System.out.println("Chars: "+charcount);

        } catch(IOException e) {
            System.err.println("Could not read from file: " + path);
            e.printStackTrace();
        }
    }

    public static void treeSize(String path) {
        long totalSize = treeSizeWorker(path,0);
        // If its a single file:
        if(treeInfoList.size()==1) {
            System.out.println("File "+path+":
"+String.format(Locale.GERMAN,"%,d",totalSize)+" bytes");
        } else if (treeInfoList.size()>1) { // if it was a directory
            System.out.println("Directory " + path + ": " +
String.format(Locale.GERMAN,"%,d",totalSize)+" bytes");
            printTree();
        }
```

```
        treeInfoList.clear();
    }
    private static long treeSizeWorker(String path, int depth) {
        File root = new File(path);

        // if its a directory, get the files and recurse
        if(root.isDirectory()) {
            File[] files = root.listFiles();
            int bytes = 0;
            for(File f : files) {
                // create treeInfo objects for each file/directory
                TreeInfo treeInfo = new TreeInfo(f.getName(), depth);
                treeInfoList.add(treeInfo);
                int index = treeInfoList.size()-1;
                long elementSize = treeSizeWorker(f.getPath(), depth+1);
                treeInfoList.get(index).setSize(elementSize);
                bytes += elementSize;
            }
            return bytes;
        } else {
            // if its a file, just return its length.
            return root.length();
        }
    }
    private static void printTree() {
        // Print each tree using the filename, size and indentation
        for(TreeInfo ti : treeInfoList) {
            String line = "";
            for(int i=0; i<ti.getDepth(); i++) {
                line += "\t";
            }
            line += ti.getName() + " - " +
String.format(Locale.GERMAN,"%,d",ti.getSize());
            System.out.println(line);
        }
    }

    private static class TreeInfo {
        private String name;
        private long size;
        private int depth;

        public TreeInfo(String name, int depth) {
            this.name = name;
            this.depth = depth;
        }

        public void setSize(long size) {
            this.size = size;
        }

        public String getName() {
            return this.name;
        }

        public long getSize() {
            return this.size;
        }
```

```
        public int getDepth() {
            return this.depth;
        }
    }

}
```

Testfälle

- Nichtexistentes File
- Head-Test
    - Weniger Zeilen als im File
    - Mehr Zeilen als im File
    - Negative Zeilenanzahl
    - Leeres File
    - Aufruf ohne Zeilenanzahl
- Tail-Test
    - Weniger Zeilen als im File
    - Mehr Zeilen als im File
    - Negative Zeilenanzahl
    - Leeres File
- LOC-Test
    - File mit bekanntem Aufbau
    - Leeres File
- Treesize-Test
    -

Nichtexistentes File (Exception Handling):

```
Testing nonexistant file (applies to all utilities):

Could not read from file: Iwasneverhere.txt
java.io.FileNotFoundException: Iwasneverhere.txt (Das System kann die angegebene Datei
nicht finden)
	at java.base/java.io.FileInputStream.open0(Native Method)
	at java.base/java.io.FileInputStream.open(FileInputStream.java:211)
	at java.base/java.io.FileInputStream.<init>(FileInputStream.java:153)
	at java.base/java.util.Scanner.<init>(Scanner.java:639)
	at ue02.part2.utility.FileUtility.head(FileUtility.java:32)
	at ue02.part2.test.FileTest.nonexistant(FileTest.java:12)
	at ue02.part2.test.FileTest.main(FileTest.java:78)
```

Head-Tests:

```
Test with 6 lines and head 5:
Ich bin ein Text.
2
3
4
5

Test with 6 lines and head 30:
Ich bin ein Text.
2
3
4
5
6

Test with 6 lines and head without parameter:
Ich bin ein Text.
2
3
4
5
6

Test with empty file and head 5:

Test with head -1:
```

Tail-Tests:

```
Test with 6 lines and tail 3:
4
5
6

Test with 6 lines and tail 42:
Ich bin ein Text.
2
3
4
5
6

Test with 6 lines and tail -1:

Test with empty file and tail 3:
```

LOC-Tests:

```
LOC for file with 6 lines, 9 words and 19 characters:
Printing statistics for file: testfolder/test.txt
Lines: 6
Words: 9
Chars: 19

LOC for empty file:
Printing statistics for file: testfolder/empty.txt
Lines: 0
Words: 0
Chars: 0
```

TreeSize-Tests:

```
Test with file:
File testfolder/test.txt: 27 bytes

Test with src/ue02/part2 directory:
Directory src\ue02\part2: 8.424 bytes
test - 2.707
       FileTest.java - 2.707
utility - 5.717
       FileUtility.java - 5.717
```