Quellcode für Beispiel 01

Block:

```
public interface Block {

}
```

EmitBlock:

```
//Only Observable
public abstract class EmitBlock implements Block {

    public abstract void create_stream() throws Exception;

    public abstract void create_stream(String filename) throws Exception;

}
```

IntermediateBlock:

```
public abstract class IntermediateBlock implements Block {

    public abstract void process_stuff(Object o) throws Exception;

    protected Double[] splitAndTurn(Object o) {
        //gets a line (string)
        //assumption: the numbers are separated by ,
        String s = String.valueOf(o);
        //o.toString();
        String[] arr = s.split(",");
        Double[] doubles = new Double[arr.length];

        //take three values and throw them into a container
        for (int j = 0; j < arr.length; j++) {
            arr[j] = arr[j].replace("[", "");
            arr[j] = arr[j].replace("]", "");

            doubles[j] = Double.parseDouble(arr[j]);
        }

        return doubles;
    }

}
```

TerminalBlock:

```
public abstract class TerminalBlock {

    abstract public void show_data(Observable source, Object o) throws
Exception;

}
```

FileReaderBlock:

```
public class FileReaderBlock extends EmitBlock implements Observable {

    public FileReaderBlock() {}

    private Set<Observer> registeredObservers = new HashSet<>();

    //reads filename from user input
    public void create_stream() throws Exception {
        try {
            BufferedReader reader = new BufferedReader(new
InputStreamReader(System.in));

            create_stream(reader.readLine());
        } catch (IOException e) {

        }
    }

    //overloaded version of create_stream
    public void create_stream(String filename) throws Exception {
            Scanner scanner = new Scanner(new File(filename));

            //reads line by line
            while (scanner.hasNext()) {
                //send one line to next block
                String line = scanner.nextLine();
                //System.out.println(line);
                this.notifyObservers(line);
            }

            scanner.close();
    }

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

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

    @Override
    public void notifyObservers(Object o) throws Exception {
        for (Observer observer : registeredObservers) {
            observer.update(this, o);
        }
    }
}
```

BufferBlock:

```java
//takes bufferSize of LINES and throws them at the next block
public class BufferBlock extends IntermediateBlock implements Observer,
Observable {

    //bufferSize is per default 10
    private int bufferSize = 10;
    //savedValues is a list of Objects (gets Strings)
    private List<Object> savedValues = new ArrayList<>();

    private Set<Observer> registeredObservers = new HashSet<>();

    //constructor for creating the bufferBlock
    public BufferBlock(int bufferSize) {
        this.bufferSize = bufferSize;
    }

    //functionalities from Observable
    @Override
    public void registerObserver(Observer observer) {
        if(observer != null) {
            this.registeredObservers.add(observer);
        }
    }

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

    @Override
    public void notifyObservers(Object o) throws Exception {
        for (Observer observer : registeredObservers) {
            observer.update(this, o);
        }
    }

    //functionality from Observer
    @Override
    public void update(Observable source, Object o) throws Exception {
        //store the object
        process_stuff(o);
    }

    //functionality from IntermediateBlock
    //gets a line and saves it, until the count reaches bufferSize
    @Override
    public void process_stuff(Object o) throws Exception {
        //count of elements is still smaller than bufferSize
        if (this.savedValues.size() < this.bufferSize) {
            this.savedValues.add(o);
        } else { //the bufferSize is reached
            //throw the list of elements to the registered Observer
            notifyObservers(savedValues);
            //clear the list
            savedValues.clear();
        }
    }
}
```

MedianBlock:

```
//gets buffered values and calculates median
public class MedianBlock extends IntermediateBlock implements Observable,
Observer {

    //list of observers
    private Set<Observer> registeredObservers = new HashSet<>();

    //size of window, default 3
    private int windowSize = 3;

    public MedianBlock(int windowSize) {
        this.windowSize = windowSize;
    }

    //funtionality from IntermediateBlock
    @Override
    public void process_stuff(Object o) throws Exception {
        //split string and turn to double
        Double[] doubles = splitAndTurn(o);
        //window with the values
        Double[] window = new Double[windowSize];
        //median will save all the medians
        Double[] median = new Double[doubles.length];

        int lower = 0;
        int upper = windowSize - 1;
        int edge = (int)(windowSize / 2);

        //goes through the array of doubles and calculates the median
        for (int i = 0; i < doubles.length; i++) {
            //do not change lower/upper if at the edge of array
            //e.g. windowSize = 3; (int)3/2 -> 1 > 0(do not change lower or
upper)
            if ((i + edge) < doubles.length - 1 && (i - edge) > 0) {
                lower++;
                upper++;
            }

            window = createSortedWindow(doubles, lower, upper);
            median[i] = calculateMedian(window);
        }

        List<Object> values = new ArrayList<>();

        for (int i = 0; i < median.length; i++) {
            values.add(median[i]);
        }

        //now we have a Double[] with the median
        //and now THROW IT AT THE NEXT BLOCK LIKE U DON'T CARE
        //print(median);
        notifyObservers(values);
    }

    //functionalities from Observable
    @Override
    public void registerObserver(Observer observer) {
        if(observer != null) {
            this.registeredObservers.add(observer);
```

```
        }
    }

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

    @Override
    public void notifyObservers(Object o) throws Exception {
        for (Observer observer : registeredObservers) {
            observer.update(this, o);
        }
    }

    //functionality from Observer
    @Override
    public void update(Observable source, Object o) throws Exception {
        process_stuff(o);
    }

    @Override
    protected Double[] splitAndTurn(Object o) {
        //gets a line (string)
        //assumption: the numbers are separated by ,
        String s = String.valueOf(o);
        //o.toString();
        String[] arr = s.split(",");
        Double[] doubles = new Double[arr.length / 3];

        int pos = 0;

        //take three values and throw them into a container
        for (int j = 0; j < arr.length; j += 3) {
            arr[j] = arr[j].replace("[", "");
            arr[j] = arr[j].replace("]", "");

            doubles[pos] = Double.parseDouble(arr[j]);
            pos++;
        }

        return doubles;
    }

    private Double[] createSortedWindow(Double[] doubles, int lower, int
upper) {
        Double[] sortedWindow = new Double[windowSize];

        int count = lower;
        int pos = 0;

        //add the needed values to the array
        while (count <= upper) {
            sortedWindow[pos] = doubles[count];

            pos++;
            count++;
        }
```

```
        //sort the values
        Arrays.sort(sortedWindow);

        return sortedWindow;
    }

    private double calculateMedian(Double[] sortedWindow) {
        double median = 0;

        if (windowSize % 2 == 0) { //it's even
            //median is the two values in the middle / 2
            median = (sortedWindow[windowSize/2 - 1] +
sortedWindow[windowSize/2]) / 2;
        } else { //it's odd
            //median is the middle value
            //double is cast to int, anything after the decimal point is
removed
            median = sortedWindow[(int)(windowSize/2)];
        }

        return median;
    }

}
```

DifferentialBlock:

```
//uses differential methods
public class DifferentialBlock extends IntermediateBlock implements
Observer, Observable {

    //list of observers
    private Set<Observer> registeredObservers = new HashSet<>();

    //functionality from IntermediateBlock
    @Override
    public void process_stuff(Object o) throws Exception {
        //gets a Double[] from MedianBlock
        Double[] doubles = splitAndTurn(o);
        Double[] diffs = differential(doubles);

        List<Object> values = new ArrayList<>();

        for (int i = 0; i < diffs.length; i++) {
            values.add(diffs[i]);
        }

        //System.out.println(o);

        //now throw it at the observers!
        //print(diffs);
        notifyObservers(values);
    }

    private Double[] differential(Double[] doubles) {
        Double[] diffs = new Double[doubles.length - 1];

        //one number is lost
        for (int i = 0; i < doubles.length - 1; i++) {
            diffs[i] = (doubles[i + 1] - doubles[i]);
```

```
        }

        return diffs;
    }

    //functionalities from Observable
    @Override
    public void registerObserver(Observer observer) {
        if(observer != null) {
            this.registeredObservers.add(observer);
        }
    }

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

    @Override
    public void notifyObservers(Object o) throws Exception {
        for (Observer observer : registeredObservers) {
            observer.update(this, o);
        }
    }

    //functionality from Observer
    @Override
    public void update(Observable source, Object o) throws Exception {
        process_stuff(o);
    }
}
```

MinBlock:

```
public class MinBlock extends IntermediateBlock implements Observable,
Observer {

    //list of observers
    private Set<Observer> registeredObservers = new HashSet<>();

    @Override
    public void process_stuff(Object o) throws Exception {
        Double[] doubles = splitAndTurn(o);
        double minVal = 100;

        for (int i = 0; i < doubles.length; i++) {
            if (doubles[i] < minVal) {
                minVal = doubles[i];
            }
        }

        notifyObservers(minVal);
    }

    //functionalities from Observable
    @Override
    public void registerObserver(Observer observer) {
        if(observer != null) {
```

```
            this.registeredObservers.add(observer);
        }
    }

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

    @Override
    public void notifyObservers(Object o) throws Exception {
        for (Observer observer : registeredObservers) {
            observer.update(this, o);
        }
    }

    //functionality from Observer
    @Override
    public void update(Observable source, Object o) throws Exception {
        //store the object
        process_stuff(o);
    }
}
```

MaxBlock:

```
public class MaxBlock extends IntermediateBlock implements Observer,
Observable {

    //list of observers
    private Set<Observer> registeredObservers = new HashSet<>();

    @Override
    public void process_stuff(Object o) throws Exception {
        Double[] doubles = splitAndTurn(o);
        double maxVal = 0;

        for (int i = 0; i < doubles.length; i++) {
            if (doubles[i] > maxVal) {
                maxVal = doubles[i];
            }
        }

        notifyObservers(maxVal);
    }

    //functionalities from Observable
    @Override
    public void registerObserver(Observer observer) {
        if(observer != null) {
            this.registeredObservers.add(observer);
        }
    }

    @Override
    public void unregisterObserver(Observer observer) {
        if(observer != null) {
            this.registeredObservers.remove(observer);
```

```
        }
    }

    @Override
    public void notifyObservers(Object o) throws Exception {
        for (Observer observer : registeredObservers) {
            observer.update(this, o);
        }
    }

    //functionality from Observer
    @Override
    public void update(Observable source, Object o) throws Exception {
        //store the object
        process_stuff(o);
    }
}
```

AvgBlock:

```
public class AvgBlock extends IntermediateBlock implements Observer,
Observable {

    //list of observers
    private Set<Observer> registeredObservers = new HashSet<>();

    @Override
    public void process_stuff(Object o) throws Exception {
        Double[] doubles = splitAndTurn(o);
        double sum = 0;
        double avg = 0;

        for (int i = 0; i < doubles.length; i++) {
            sum += doubles[i];
        }

        avg = sum / (doubles.length - 1);

        notifyObservers(avg);
    }

    //functionalities from Observable
    @Override
    public void registerObserver(Observer observer) {
        if(observer != null) {
            this.registeredObservers.add(observer);
        }
    }

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

    @Override
    public void notifyObservers(Object o) throws Exception {
        for (Observer observer : registeredObservers) {
            observer.update(this, o);
```

```
        }
    }

    //functionality from Observer
    @Override
    public void update(Observable source, Object o) throws Exception {
        //store the object
        process_stuff(o);
    }
}
```

FileWriterBlock:

```
//gets a signal from DifferentialBlock
public class FileWriterBlock extends TerminalBlock implements Observer {

    //per default output.csv
    private String filename = "output.csv";

    public FileWriterBlock(String filename) {
        this.filename = filename;
    }

    @Override
    public void show_data(Observable source, Object o) throws Exception {
        String[] strings = splitAndTurnToString(o);

        FileWriter writer = new FileWriter(this.filename, true);

        for (int i = 0; i < strings.length; i++) {
            writer.write(strings[i]);

            writer.write("\n");
        }

        writer.close();
    }

    @Override
    public void update(Observable source, Object o) throws Exception {
        show_data(source, o);
    }

    //also has this cool function I totally did not steal
    protected String[] splitAndTurnToString(Object o) throws Exception {
        //gets a line (string)
        //assumption: the numbers are separated by ,
        String s = String.valueOf(o);
        //o.toString();
        String[] arr = s.split(",");

        //take three values and throw them into a container
        for (int j = 0; j < arr.length; j++) {
            arr[j] = arr[j].replace("[", "");
            arr[j] = arr[j].replace("]", "");
            arr[j] = arr[j].replace(" ", "");
        }

        return arr;
```

```
    }
}
```

ConsoleBlock:

```
//only observer
//takes data and throws it at the console
public class ConsoleBlock extends TerminalBlock implements Observer {

    @Override
    public void show_data(Observable source, Object o) {
        System.out.println("Received value from " + source.toString() + ":
" + o);
    }

    @Override
    public void update(Observable source, Object o) {
        show_data(source, o);
    }
}
```

SumBlock:

```
//is both Observer and Observable
//gets stuff from FileReaderBlock and throws it to ConsoleBlock
public class SumBlock extends IntermediateBlock implements Observable,
Observer {

    private Set<Observer> registeredObservers = new HashSet<>();

    @Override
    public void process_stuff(Object o) throws Exception {
        //gets a line (string)
        //assumption: the numbers are separated by ,
        //can also be ;
        String s = o.toString();
        String[] arr = s.split(",");
        double sum = 0;

        for (int j = 0; j < arr.length; j++) {
            sum += Double.parseDouble(arr[j]);
        }
        //also notify the observers after processing
        notifyObservers(sum);
    }

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

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

```
    @Override
    public void notifyObservers(Object o) throws Exception {
        for (Observer observer : registeredObservers) {
            observer.update(this, o);
        }
    }

    @Override
    public void update(Observable source, Object o) throws Exception {
        //process stuff
        process_stuff(o);
    }
}
```

Observer:

```
public interface Observer {

    void update(Observable source, Object o) throws Exception;

}
```

Observable:

```
public interface Observable {

    //attach Observer
    void registerObserver(Observer observer);

    //detach Observer
    void unregisterObserver(Observer observer);

    //notify
    void notifyObservers(Object o) throws Exception;

}
```

BlockTest:

```
public class BlockTest {

    public static void main(String[] args) throws Exception {
        test05();
    }

    //always used
    public static void createBlocks(String filename) throws Exception {
        FileReaderBlock b = new FileReaderBlock();
        SumBlock s = new SumBlock();
        ConsoleBlock c = new ConsoleBlock();

        b.registerObserver(s);
        s.registerObserver(c);

        //if there was no filename
        if (filename.equals("")) {
```

```
            b.create_stream();
        } else {
            b.create_stream(filename);
        }
    }

    public static void test01() throws Exception {
        createBlocks("");
    }

    public static void test02() throws Exception {
        createBlocks("IntegerTest.txt");
    }

    public static void test03() throws Exception {
        createBlocks("IntegerTest2.txt");
    }

    public static void test04() throws Exception {
        createBlocks("DoubleTest.txt");
    }

    public static void test05() throws Exception {
        createBlocks("StringTest.txt");
    }

}
```

FancyBlockTest:

```
public class FancyBlockTest {

    public static void main(String[] args) throws Exception {
        test05();
    }

    //always used
    public static void createBlocks(String filename) throws Exception {
        FileReaderBlock b = new FileReaderBlock();
        BufferBlock bu = new BufferBlock(10);
        MedianBlock m = new MedianBlock(3);
        DifferentialBlock d = new DifferentialBlock();
        MinBlock min = new MinBlock();
        MaxBlock max = new MaxBlock();
        AvgBlock avg = new AvgBlock();
        ConsoleBlock c = new ConsoleBlock();
        FileWriterBlock f = new FileWriterBlock("signal.csv");

        //delete file signal.csv

        b.registerObserver(bu);
        bu.registerObserver(m);
        m.registerObserver(d);
        d.registerObserver(min);
        d.registerObserver(max);
        d.registerObserver(avg);
        d.registerObserver(f);
        min.registerObserver(c);
        max.registerObserver(c);
        avg.registerObserver(c);
```

```
        //if there was no filename
        if (filename.equals("")) {
            b.create_stream();
        } else {
            b.create_stream(filename);
        }
    }

    public static void test01() throws Exception {
        createBlocks("sample_data.csv");
    }

    public static void test02() throws Exception {
        FileReaderBlock f = new FileReaderBlock();
        ConsoleBlock c = new ConsoleBlock();

        f.registerObserver(c);

        f.create_stream("sample_data.csv");
    }

    public static void test03() throws Exception {
        FileReaderBlock f = new FileReaderBlock();
        BufferBlock b = new BufferBlock(2);
        ConsoleBlock c = new ConsoleBlock();

        f.registerObserver(b);
        b.registerObserver(c);

        f.create_stream("sample_data.csv");
    }

    public static void test04() throws Exception {
        FileReaderBlock f = new FileReaderBlock();
        BufferBlock b = new BufferBlock(6);
        MedianBlock m = new MedianBlock(3);
        ConsoleBlock c = new ConsoleBlock();

        f.registerObserver(b);
        b.registerObserver(m);
        m.registerObserver(c);

        f.create_stream("sample_data.csv");
    }

    public static void test05() throws Exception {
        FileReaderBlock f = new FileReaderBlock();
        BufferBlock b = new BufferBlock(6);
        MedianBlock m = new MedianBlock(3);
        DifferentialBlock d = new DifferentialBlock();
        ConsoleBlock c = new ConsoleBlock();

        f.registerObserver(b);
        b.registerObserver(m);
        m.registerObserver(d);
        d.registerObserver(c);

        f.create_stream("sample_data.csv");
    }
```

```
}
```