# INFO0062 - Object-Oriented Programming

Project: practical use of the library

**Jean-François Grailet**

University of Liège

Faculty of Applied Sciences

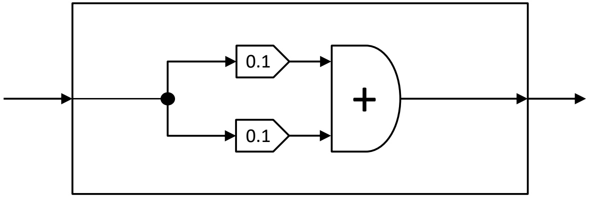
Academic Year 2019 - 2020

## Summary

- A toy composite filter
- Creating the filter with `CompositeFilter`
- Closing comments

## A toy composite filter

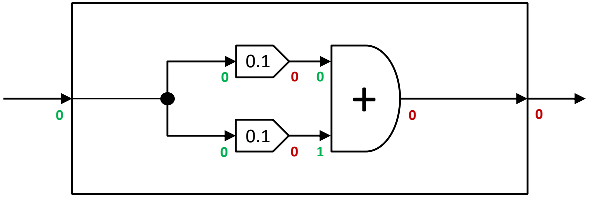
- The bottom of this slide shows a toy composite filter. [1]
- This filter doesn't do anything special.
  - It only lowers the volume of the initial audio sequence.
- Nevertheless, it involves two `GainFilter` and one `AdditionFilter`.



[1] FR: un filtre composite "jouet", c.-à-d. créé seulement à titre d'exemple.

## A toy composite filter (II)

- This filter has one input and one output.
- Among its components, only the `AdditionFilter` has two inputs.
  - Everything else has one input; all blocks have one output.
- Next figure shows the same figure as in the previous slide with annotations.
  - Green numbers show the numbered inputs.
  - Red numbers show the numbered outputs.



## Creating the filter with `CompositeFilter`

- First, we instantiate a new `CompositeFilter` object.
    - Arguments are **1** and **1**.
    - Indeed, there is **one input** and **one ouput**.
- Then, we instantiate the individual blocks making up the filter.

```
// ...
CompositeFilter audioFilter = new CompositeFilter(1, 1);

Filter mult1 = new GainFilter(0.1);
Filter mult2 = new GainFilter(0.1);
Filter add = new AdditionFilter();
// ...
```

## Creating the filter with `CompositeFilter` (II)

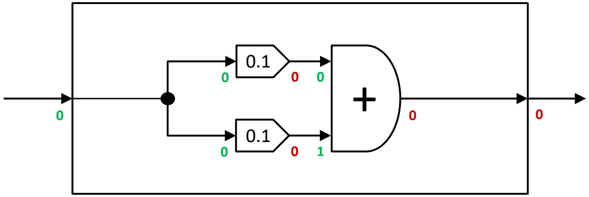
- Next, we *add the blocks* to the `CompositeFilter` object.
    - In order to connect blocks, a filter needs to *know* them first.
    - This is the purpose of the `addBlock()` method.
- Once all blocks are known, we can start to connect them.

```
// ...
audioFilter.addBlock(mult1);
audioFilter.addBlock(mult2);
audioFilter.addBlock(add);
// ...
```

## Creating the filter with `CompositeFilter` (III)

- We now need to use the 3 connection methods of `CompositeFilter`.
  - Cf. the statement.
- To connect blocks together, we have to use the right indexes.
  - In this simple example, we can use 0 everywhere with one exception.
  - This exception is the second input of the `AdditionFilter` object.
- Next slide shows the connections along with the annotated block diagram.
  - You will remark all indexes appear once, except the main input.
  - I.e., the input of the whole `CompositeFilter` (appears twice).

## Creating the filter with `CompositeFilter` (IV)



```
// ...
audioFilter.connectInputToBlock(0, mult1, 0);
audioFilter.connectInputToBlock(0, mult2, 0);
audioFilter.connectBlockToBlock(mult1, 0, add, 0);
audioFilter.connectBlockToBlock(mult2, 0, add, 1);
audioFilter.connectBlockToOutput(add, 0, 0);
// ...
```

## Creating the filter with `CompositeFilter` (V)

- The filter is now complete and can be applied to a WAV file.
  - Again via the `applyFilter()` method, cf. below.
  - Of course, it will work only if your implementation of `CompositeFilter` is complete.
- Note that instructions shown here should ideally be in a `try` block.
  - This is necessary to catch exceptions (any) and properly deal with them.
  - Cf. Chapter 6.
- A complete program with this filter is available online.
  - Download `CompositeExample.java` on the usual webpage. [2]

```
// ...
TestAudioFilter.applyFilter(audioFilter, "In.wav", "Out.wav");
```

[2] http://www.run.montefiore.ulg.ac.be/~grailet/INFO0062_proj_19-20.php

## Closing comments

- Your final project MUST compile with `CompositeExample.java`.
- Compilation failures would mean that
  - you didn't use the expected names for classes or
  - you didn't implement the interface documented in the statement.
- However, you remain completely free regarding
  - the inner workings of your `CompositeFilter` class,
    - i.e., you can create auxiliary classes suiting your needs,
  - exception handling.
- Don't forget to check (again) the main presentation slides.
  - They contain various tips for this project.