# The adapter design pattern

The INFDEV team

Hogeschool Rotterdam
Rotterdam, Netherlands

# INFSEN02-2

# Introduction

# Introduction



- Today we are going to study the a behavioral pattern: the decorator design pattern
- Sometimes, we need to modify behaviors of an instance dynamically
- Sub-classing could be a solution, but excessive sub-classing is a pitfall
- The decorator pattern (also known as wrapper) solves this issues
- How? By limiting sub-typing through effective aggregation

# Example

# Example



## Iterator

- Consider the iterator interface

# Example



## Natural numbers

- Consider the natural number collection

# Example



## Iterating only the even numbers

- We wish now to iterate only the even numbers of our natural number list

# Example



## Adding an offset

- We wish now to iterate and while iterating adding an offset to our natural number list

# Example



## Adding an offset only on even numbers

- We wish now to iterate only even numbers of our natural number list and for each number add an offset
- It is getting hard :/

# Example



## UML discussion

# Example



## Iterating a range between two integers

- We now wish to implement a new data structure RangeBetween that takes two integers A and B (where A <= B)

# Example



## Ranging over even numbers and/or adding an offset

- We now wish our range to support the same behavior as for our natural numbers
- Trivial and time consuming

# Example



## Considerations

- Sub-typing solves our problem, but adds another one. Too many repetitions
- Every change/add requires lots of work

# Example



## Considerations

- A possible solution would see our numbers implementing offset and even
- This is not good..what about SOLID
- The resulting structure is a big, bulky class (a class whose functionality does not adapt to each instance instance)

# Example



## Considerations

- Abstract classes with a series of fields (which we can check to select an appropriate algorithm)
- But fields do not force appropriate behavior for each the roles

# Example



## Solution

- A possible could be that we define an intermediate class `Decorator`, which inherits our iterator and contains an instance of it
- It may seems strange, but this is a crucial moment!

# Example



## Solution

- As our behaviors described above: offset and even are general to all numbers we can define two distinct classes to represent them
- Such classes extends our decorator

# Example



## Solution

- We can think of the decorator as an iterator containing elements, but it does not know how to iterate such elements
- Who will teach the decorator how to iterate?
- Our concrete offset and even

# Example



## Solution

- See code

# Example



## Solution

- See UML

# Example



## Solution

- Compare with the initial solution

# Example



## Solution - improvements

- Offset simply transforms
- It is a map
- CODE

# Example



## Solution - improvements

- Even simply filters
- It is a filter
- CODE

# The decorator design pattern

# The decorator design pattern



## Formalism

- Formalism

# The decorator design pattern



## Formalism

- UML

# Conclusions

# This is it!



The best of luck, and thanks for the attention!