Module 04:
"Decorator"

TEKNOLOGISK
INSTITUT

Agenda

- Introductory Example: Rental Vehicles
- Challenges
- Implementing the Decorator Pattern
- Pattern: Decorator
- Overview of Decorator Pattern
- To Decorate or Not To Decorate?

Introductory Example: Rental Vehicles

TEKNOLOGISK INSTITUT

```
interface IVehicle
{
    string Make { get; }
    VehicleColor Color { get; }
    int KilometersTravelled { get; }

    void Drive(int travelled);
}
```

```
abstract class Vehicle : IVehicle
{
    public string Make { get; }
    public VehicleColor Color { get; }
    public int KilometersTravelled
        { get; private set; }
    public void Drive(int travelled)
        { ... }
}
```

```
class Car : Vehicle
{
    public CarBodyStyle BodyStyle { get; }
    public int Doors { get; }
    ...
}
```

```
class Motorcycle : Vehicle
{
    public int Wheels { get; }
    public int Cc { get; }
    ...
}
```

Challenges

- How do we add Rental state and behavior?
- How would we then subsequently add
  - Shop state and behavior?
  - …?
- Can we uphold the Single Responsibility Principle?

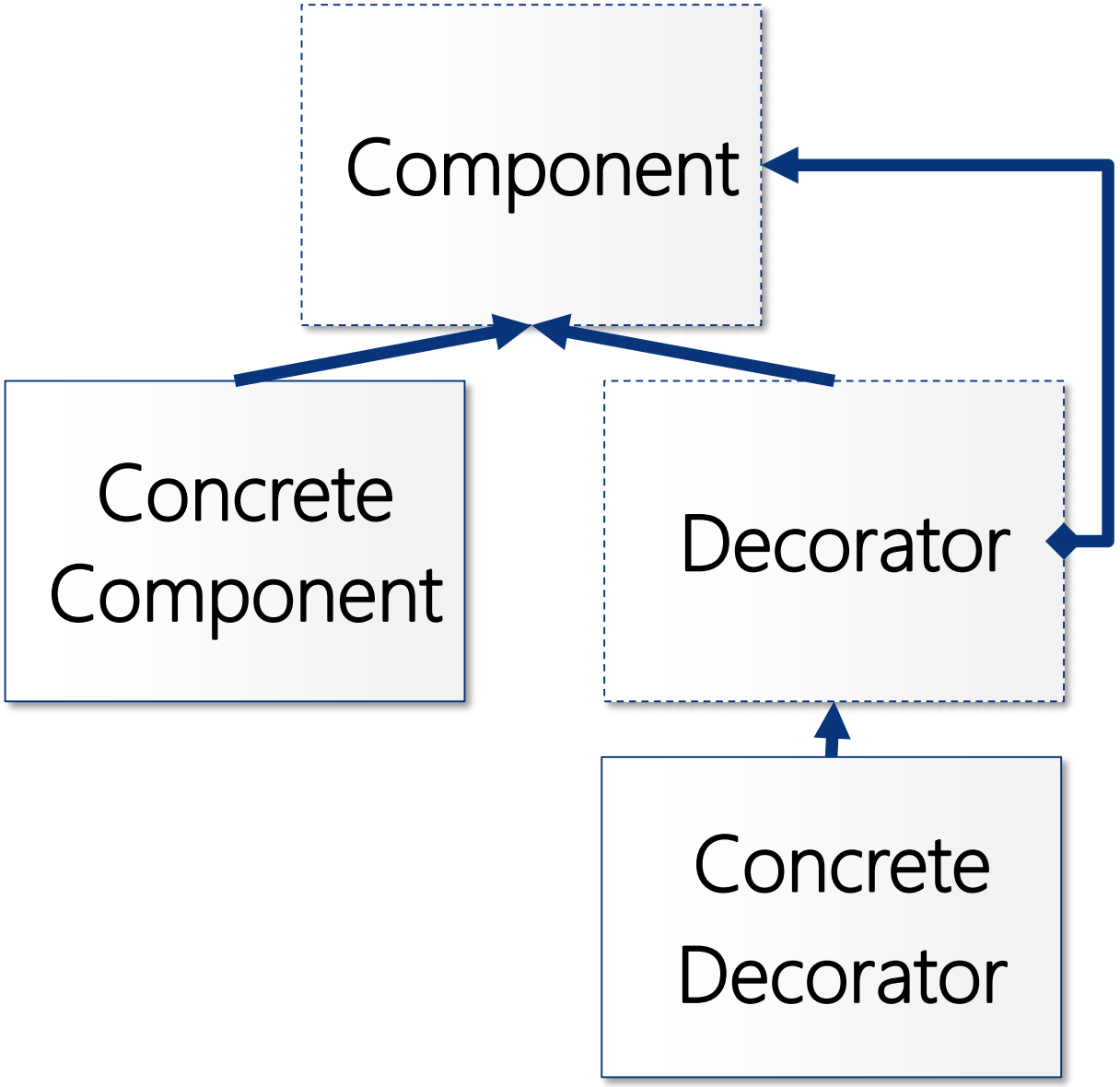
Pattern: Decorator

- *Attach additional responsibilities to an object dynamically. Decorators provide a flexible alternative to subclassing for extending functionality.*

- Outline
  - Extend functionality without modifying existing classes
  - Avoid "explosion" in number of subclasses
  - Create add-on classes adding "aspect"

- Origin: Gang of Four

Overview of Decorator Pattern



Overview of Decorator Pattern

- Component
  - Interface or abstract base class for class hierarchy
- Concrete Component
  - Concrete subclass in class hierarchy
- Decorator
  - Wraps an instance of Component
- Concrete Decorator
  - Adds concrete state or behavior

To Decorate or Not To Decorate?

- Pros
  - Decorator is central for upholding Single Responsibility Principle of SOLID
  - Can activate several decorators simultaneously
  - Avoids exponential explosion of subclasses
  - Can "wrap" legacy systems
  - Can create decorators for "aspects"
- Cons
  - System design can get increasingly complicated
  - End up with many quite similar classes


WINCUBATE
Jesper Gulmann Henriksen
PhD, MCT, MCSD, MCPD
Phone : +45 22 12 36 31
Email : jgh@wincubate.net
WWW : http://www.wincubate.net
Ringgårdsvej 4A
8270 Højbjerg
Denmark