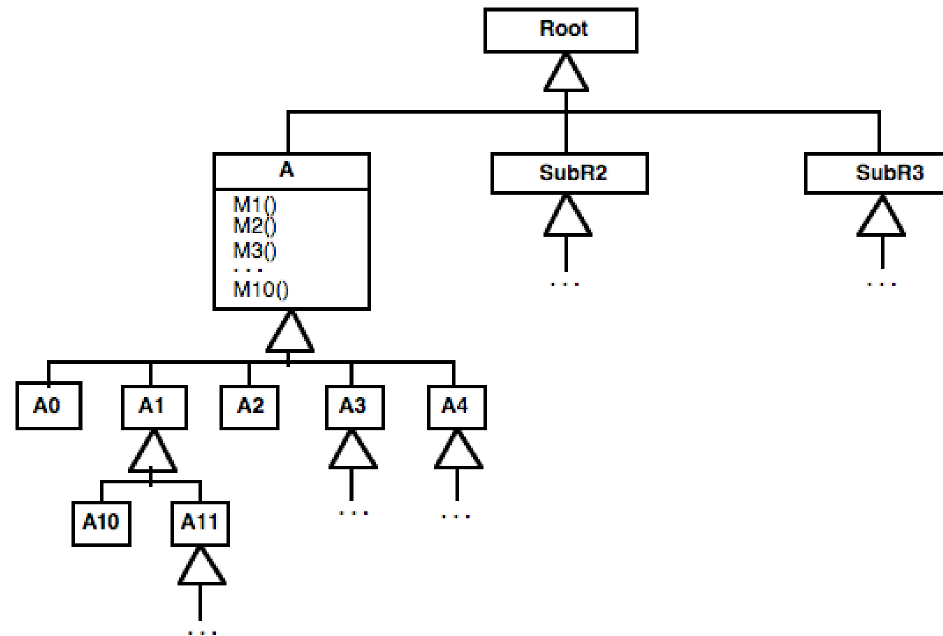
Objekt-Komposition versus Vererbung: Decorator-Design-Pattern

O.Univ.-Prof. Dipl.-Ing. Dr. Wolfgang Pree

Fachbereich Informatik
cs.uni-salzburg.at

UNIVERSITÄT SALZBURG

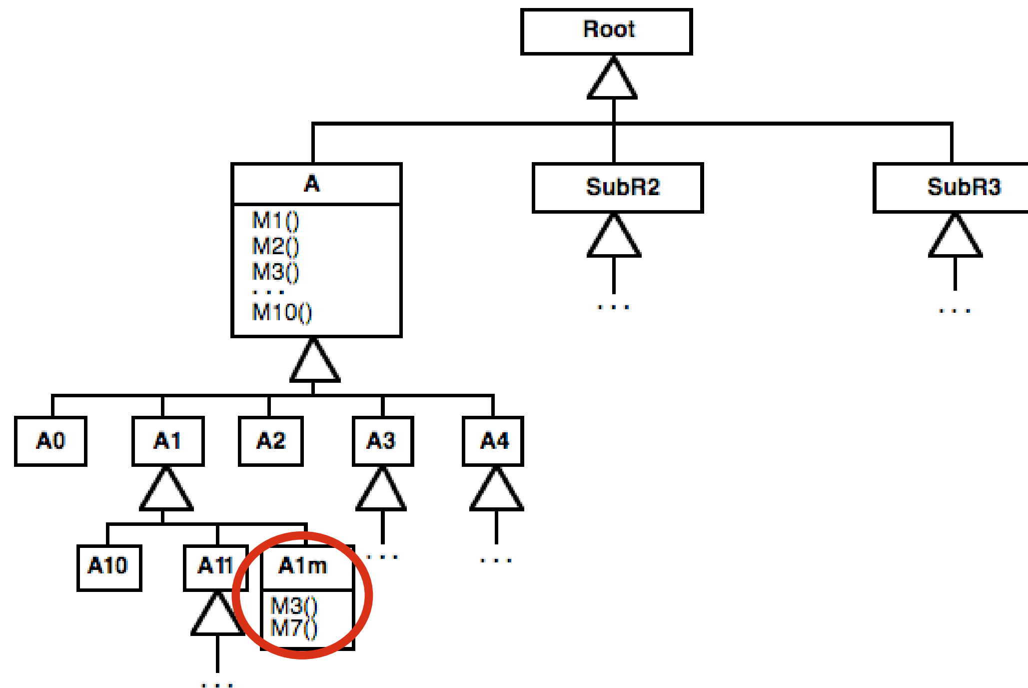
Motivation: Änderungen einer Klasse mit vielen Unterklassen (I)



- Änderungen von M3() und M7() in Klasse A erforderlich
- Quelltext von A, wenn vorhanden, ändern?
- Änderung durch Vererbung?

© 2004, W. Pree

Motivation: Änderungen einer Klasse mit vielen Unterklassen (II)

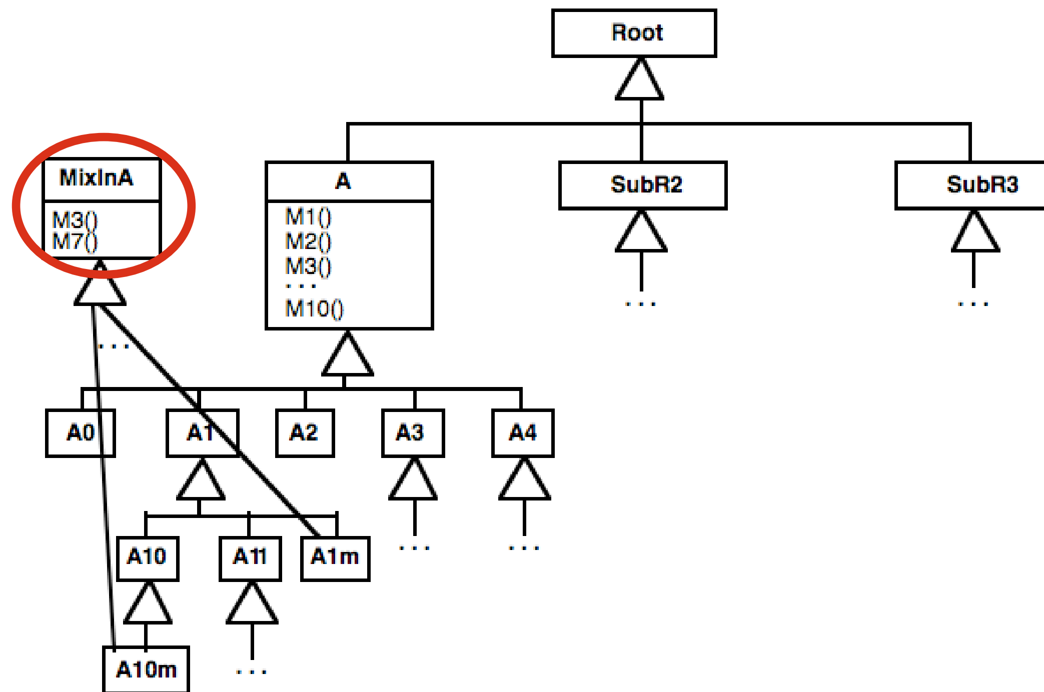


- für eine Klasse (zB A1m) ist die Anpassung sinnvoll
- für alle Unterklassen von A ist das aufwändig

© 2004, W. Pree

UNIVERSITÄT SALZBURG

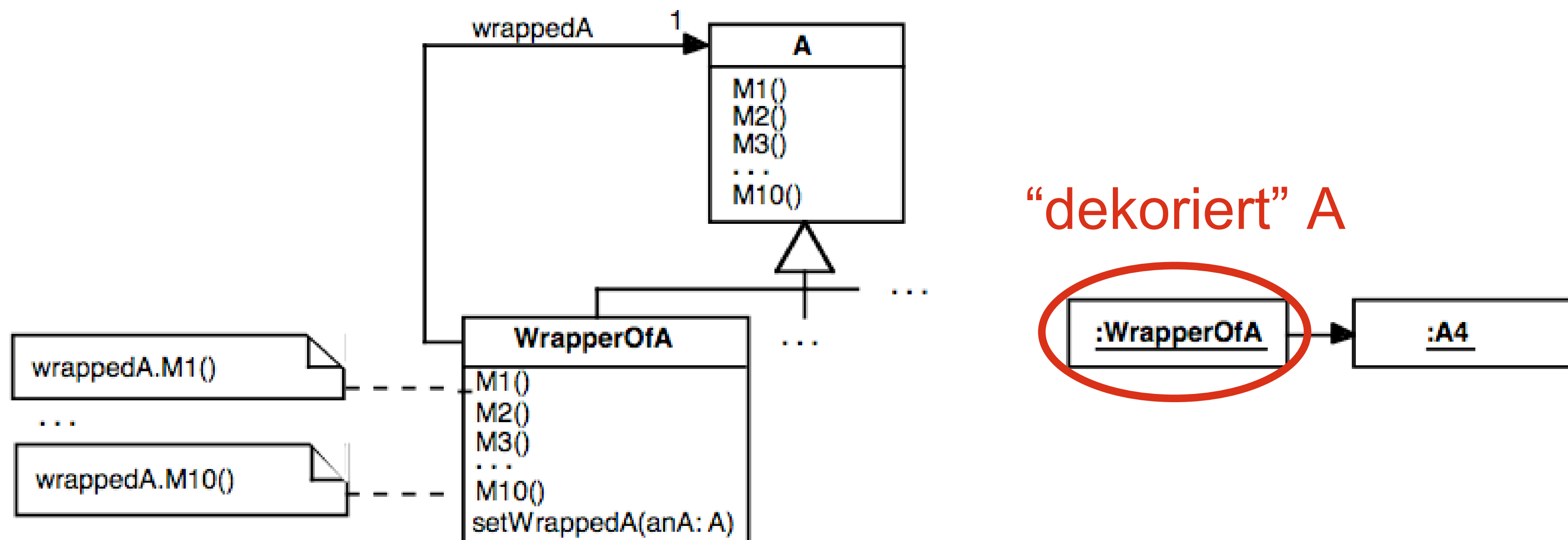
Motivation: Änderungen einer Klasse mit vielen Unterklassen (III)



- In Programmiersprachen, die mehrfache Vererbung unterstützen, können sogenannte Mix-In-Klassen definiert werden
- dennoch muss für jede Klasse, deren Verhalten angepasst werden soll, eine Unterklasse gebildet werden

© 2004, W. Pree

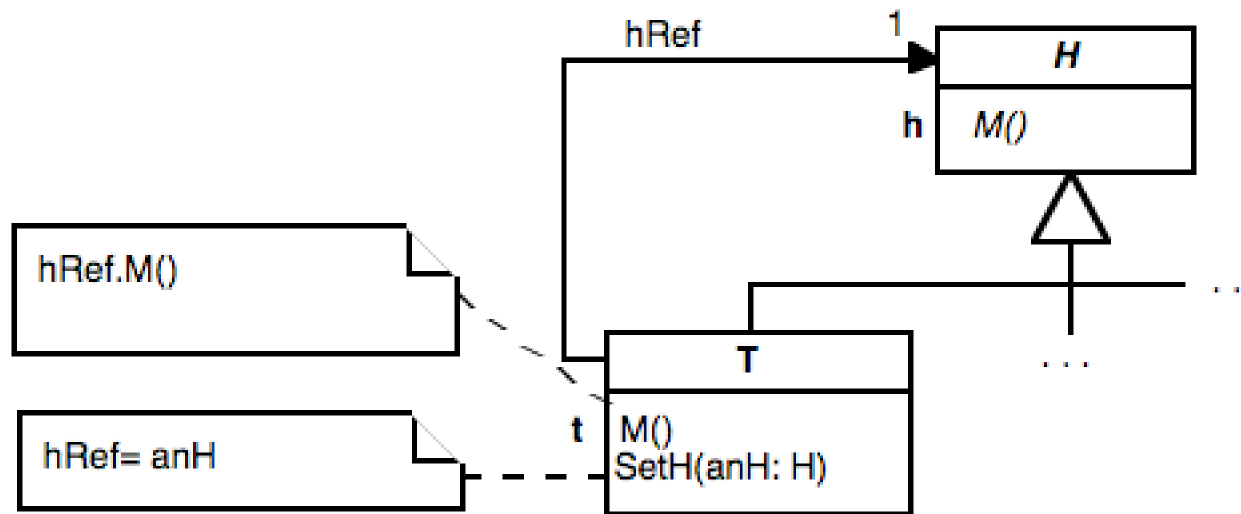
Änderung einer Klasse durch Komposition statt durch Vererbung



- In der Klasse WrapperOfA werden alle Methoden von A überschrieben, indem der Methodenaufruf jeweils an ein über die Instanzvariable wrappedA referenziertes Objekt delegiert wird – mit Ausnahme jener, die geändert werden.
- Da WrapperOfA eine Unterklasse von A ist, kann überall dort, wo ein Objekt vom statischen Typ A gefordert wird, eine Instanz der Klasse WrapperOfA verwendet werden. Da die Instanzvariable wrappedA den statischen Typ A hat, kann sie auf jedes Objekt einer Unterklasse von A verweisen.

© 2004, W. Pree

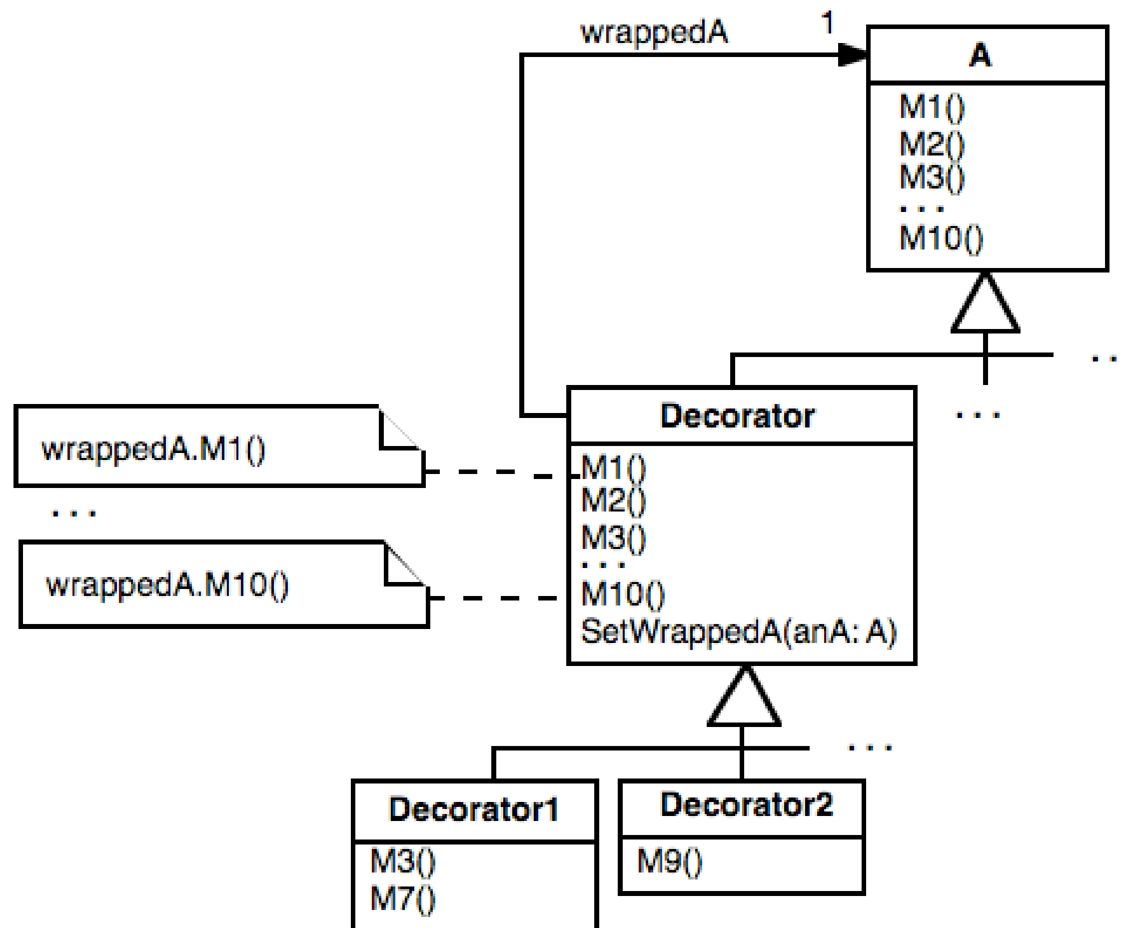
Decorator: Anpassung durch Komposition mit beliebig vielen Objekten statt durch Vererbung



- Namen der Template- und Hook-Methoden sind gleich
- Hinzufügen von Decorator-Objekten mit SetH()
- Eine Instanz des Decorators (Filters) T zusammen mit der Instanz einer Unterklasse von H kann von Klienten wie ein H-Objekt verwendet werden. Allerdings wird das Verhalten des H-Objektes dadurch modifiziert.
- H + Decorator(s) sind als “ein Objekt” verwendbar (vgl. Composite)

© 2004, W. Pree

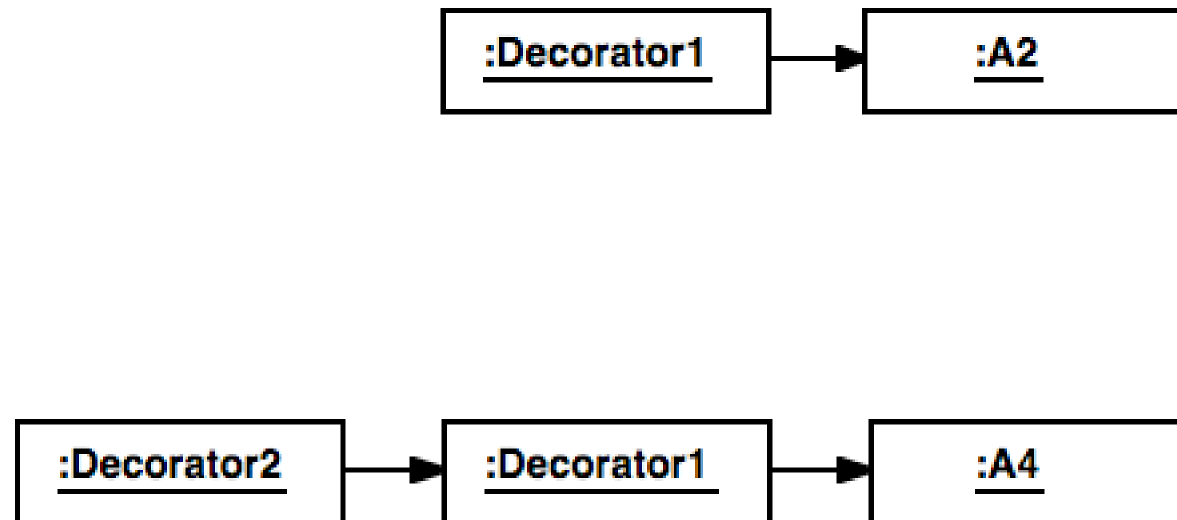
Vorschlag für einen Entwurf, wenn mehrere Decorator-Klassen verwendet werden



© 2004, W. Pree

UNIVERSITÄT SALZBURG

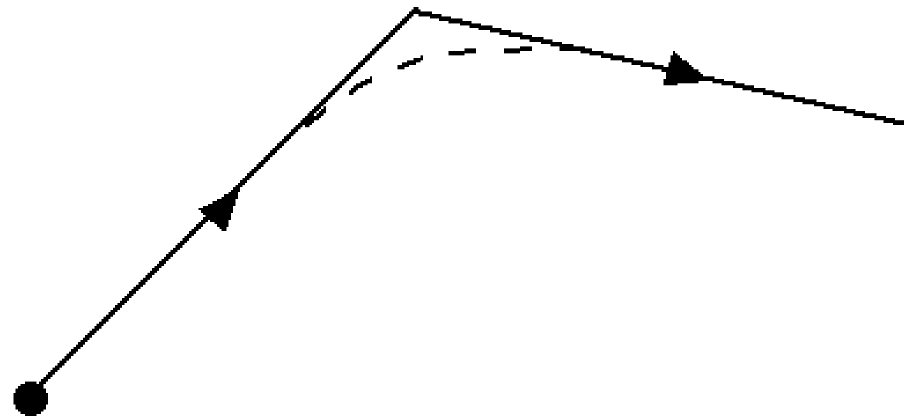
zwei exemplarische Kompositionen



© 2004, W. Pree

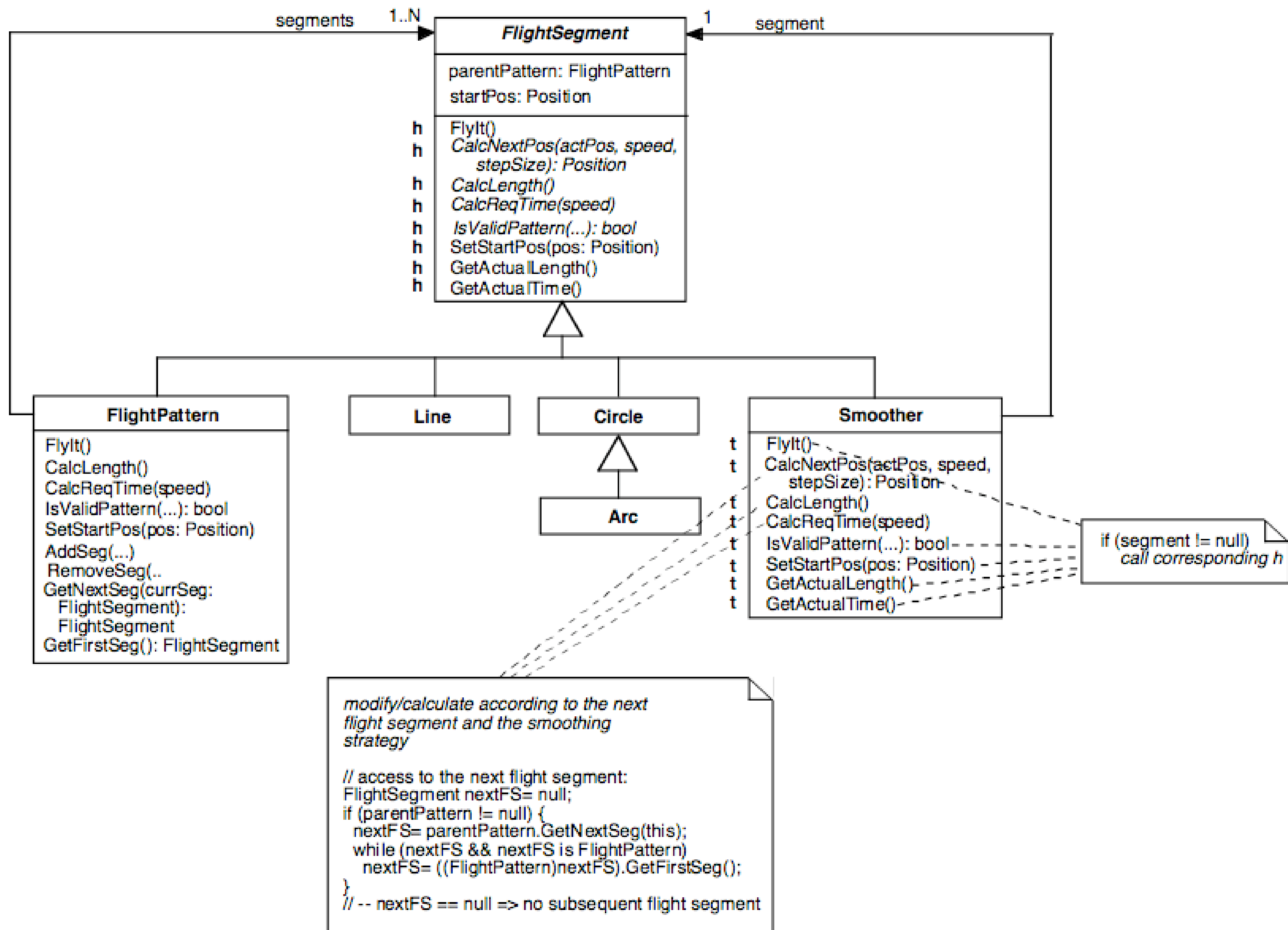
UNIVERSITÄT SALZBURG

Beispiel: Abrunden von Flugmustern



© 2004, W. Pree

UNIVERSITÄT SALZBURG


segments
1..N
FlightSegment
1
segment
parentPattern: FlightPattern
startPos: Position
h FlyIt()
h CalcNextPos(actPos, speed, stepSize): Position
h CalcLength()
h CalcReqTime(speed)
h IsValidPattern(...): bool
h SetStartPos(pos: Position)
h GetActualLength()
h GetActualTime()
FlightPattern
FlyIt()
CalcLength()
CalcReqTime(speed)
IsValidPattern(...): bool
SetStartPos(pos: Position)
AddSeg(...)
RemoveSeg(..
GetNextSeg(currSeg: FlightSegment): FlightSegment
GetFirstSeg(): FlightSegment
Line
Circle
Arc
Smoother
t FlyIt()
t CalcNextPos(actPos, speed, stepSize): Position
t CalcLength()
t CalcReqTime(speed)
t IsValidPattern(...): bool
t SetStartPos(pos: Position)
t GetActualLength()
t GetActualTime()
if (segment != null)
call corresponding h
modify/calculate according to the next flight segment and the smoothing strategy
// access to the next flight segment:
FlightSegment nextFS= null;
if (parentPattern != null) {
nextFS= parentPattern.GetNextSeg(this);
while (nextFS && nextFS is FlightPattern)
nextFS= ((FlightPattern)nextFS).GetFirstSeg();
}
// -- nextFS == null => no subsequent flight segment

Verwendung der Decorator-Klasse Smoother

```
FlightPattern triangle= new FlightPattern();
triangle.SetStartPos(...);
triangle.AddSeg(new Smoother(new Line(...)));
triangle.AddSeg(new Smoother(new Line(...)));
triangle.AddSeg(new Line(...));
```


© 2004, W. Pree

Rahmenbedingungen für die Anwendung des Decorator-Konstruktionsprinzips (I)

- Die Signatur von H, also von der Wurzelklasse des Teilbaums, soll von den Unterklassen von H nicht erweitert werden. Der Grund dafür ist, dass zusätzliche Methoden in den Unterklassen, die die Signatur der Wurzelklasse erweitern, in den Decorator-Klassen nicht berücksichtigt werden können.
- Um die Erfüllung dieser Forderung sicherzustellen, ist es notwendig, dass die Gemeinsamkeiten aller Unterklassen von H in die Wurzelklasse transferiert, das heißt faktorisiert werden. Bei vielen Klassenbibliotheken ist diese Forderung nicht erfüllt. Die Anwendung des Decorator-Konstruktionsprinzips ist daher in solchen Fällen nicht immer in vollem Umfange möglich. (vgl. Decorator Smoother)

© 2004, W. Pree

Rahmenbedingungen für die Anwendung des Decorator-Konstruktionsprinzips (II)

In unserem Beispiel müssen daher die spezifischen Methoden für die erwähnten Objekte – bei Line SetLine(); bei Circle SetDirection(), sowie die Methoden zur Festlegung der Ebene, in der der Kreis liegt: bei FlightPattern AddSeg() und RemoveSeg() – explizit aufrufen werden, da sie von einer Smoother-Instanz nicht weitergeleitet werden können:

```
Circle circle= new Circle(...);
circle.SetDirection(cRight);
Smoother smoother= new Smoother(circle);
```

Wäre die erwähnte Forderung erfüllt, könnte eine Smoother-Instanz wie jedes spezifische FlightSegment-Objekt behandelt werden:

```
Smoother smoother= new Smoother(new Circle(...));
smoother.SetDirection(cRight);
```

Eine Möglichkeit, die Flugsegment-spezifischen Methoden zu eliminieren, ist, alle Eigenschaften nur über den Konstruktor der jeweiligen Klasse angeben zu lassen:

```
Smoother smoother= new Smoother(new Circle(..., cRight));
```


© 2004, W. Pree

UNIVERSITÄT SALZBURG

Anwendung des Decorator-Konstruktionsprinzips zum Entwurf “leichtgewichtiger” Wurzelklassen

- Das Decorator-Konstruktionsprinzip kann dazu herangezogen werden, Klassen nahe der Wurzel des Klassenbaumes leichtgewichtiger zu machen. Funktionalität, die nicht in allen Klassen benötigt wird, wird in Decorator-Klassen implementiert. Nur jene Objekte, welche die spezielle Funktionalität benötigen, erhalten diese durch Komposition mit der entsprechenden Decorator-Instanz.
- Das Decorator-Konstruktionsprinzip kann daher sowohl beim (Erst-)Entwurf einer Klassenhierarchie als auch bei der Erweiterung von Klassenhierarchien nutzbringend eingesetzt werden.

© 2004, W. Pree

Beispiel: Clipping-Mechanismus bei GUI-Bibliotheken

- Clipping-Mechanismus: das Zuschneiden eines GUI-Elements auf dessen festgelegte Größe
- Da der Clipping-Mechanismus nicht für alle GUI-Elemente benötigt wird, ist es sinnvoll, den Clipping-Mechanismus nicht in der Wurzel des Teilbaumes vorzusehen, sondern eine Decorator-Klasse Clipper einzuführen. (vgl. Decorator-Beispiel in Gamma et al., 1995)

© 2004, W. Pree

Zusammenfassung *Decorator*

+ einfache Anpassung durch Objektkomposition
+ neue Decorator-Elemente (Template-Klassen, die Unterklassen der Hook-Klasse sind) können definiert werden, ohne die Unterklassen der Hook-Klasse verändern zu müssen;
+ „leichtgewichtige“ Klassen können damit elegant realisiert werden
– allerdings sollte die Hook-Klasse die erwähnte Rahmenbedingung erfüllen (faktorisiert Verhalten aus allen Unterklassen heraus)
– zusätzliche Indirektion bei Methodenaufrufen
– komplexe Interaktionen zwischen beteiligten Objekten

© 2004, W. Pree
UNIVERSITÄT SALZBURG