A knight can have a
sword, to realize which
we used aggregation.

How can we distinguish
kinds of swords?

At a strike, a decision
needs to be taken based
on the kind of a knight
and kind of a sword.

Sword efficiency
depends on its kind,
but also on the kind of
the knight who
handles it.

In which class to make this
decision?

1. In the Knight class
2. In the Sword class
3. Outside these classes

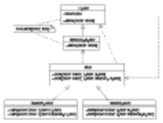
Flexibility problems
occur in object-
oriented code, too

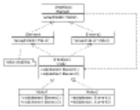
Code flexibility can
be improved by
applying design
patterns

A design pattern is being
chosen according to the
generalization of the
problem situation: the
nature of contradicting
forces that determine it

What if we would transfer
the responsibility for the
strike to the sword?

A sword, then, must know
what knight handles it.



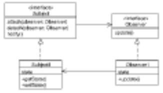


Visitor

adding operations to the
objects of certain classes
without having to
change them

contradicting
forces

Observer



observing objects have to be notified of a
change in the state of the observed object,
and it has to be possible to add them without
a need to modify the observed object

Christopher Alexander

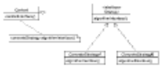
The Timeless Way of Building

A Pattern Language

Context

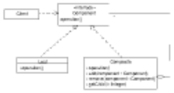
Independent Regions
...
House Cluster
House for a Small Family
Alcoves
...

Strategy (Policy)



there are (and can be added) different
strategies of solving a given problem,
and the context of their application
has to be ready for this

Composite



elements may be composite
and atomic,
and the way of accesing them
has to be uniform

Lecture 3:

Design Patterns

Valentino Vranić

Ústav informatiky, informačných systémov
a softvérového inžinierstva

STU
FIIT

vranic@stuba.sk

fiit.sk/~vranic

OOP 2018/19
26. 2. 2019

A knight can have a sword, to realize which we used aggregation.

How can we distinguish kinds of swords?

Sword efficiency depends on its kind, but also on the kind of the knight who handles it.

At a strike, a decision needs to be taken based on the kind of a knight and kind of a sword.

At a strike, a decision needs to be taken based on the kind of a knight and kind of a sword.

In which class to make this decision?

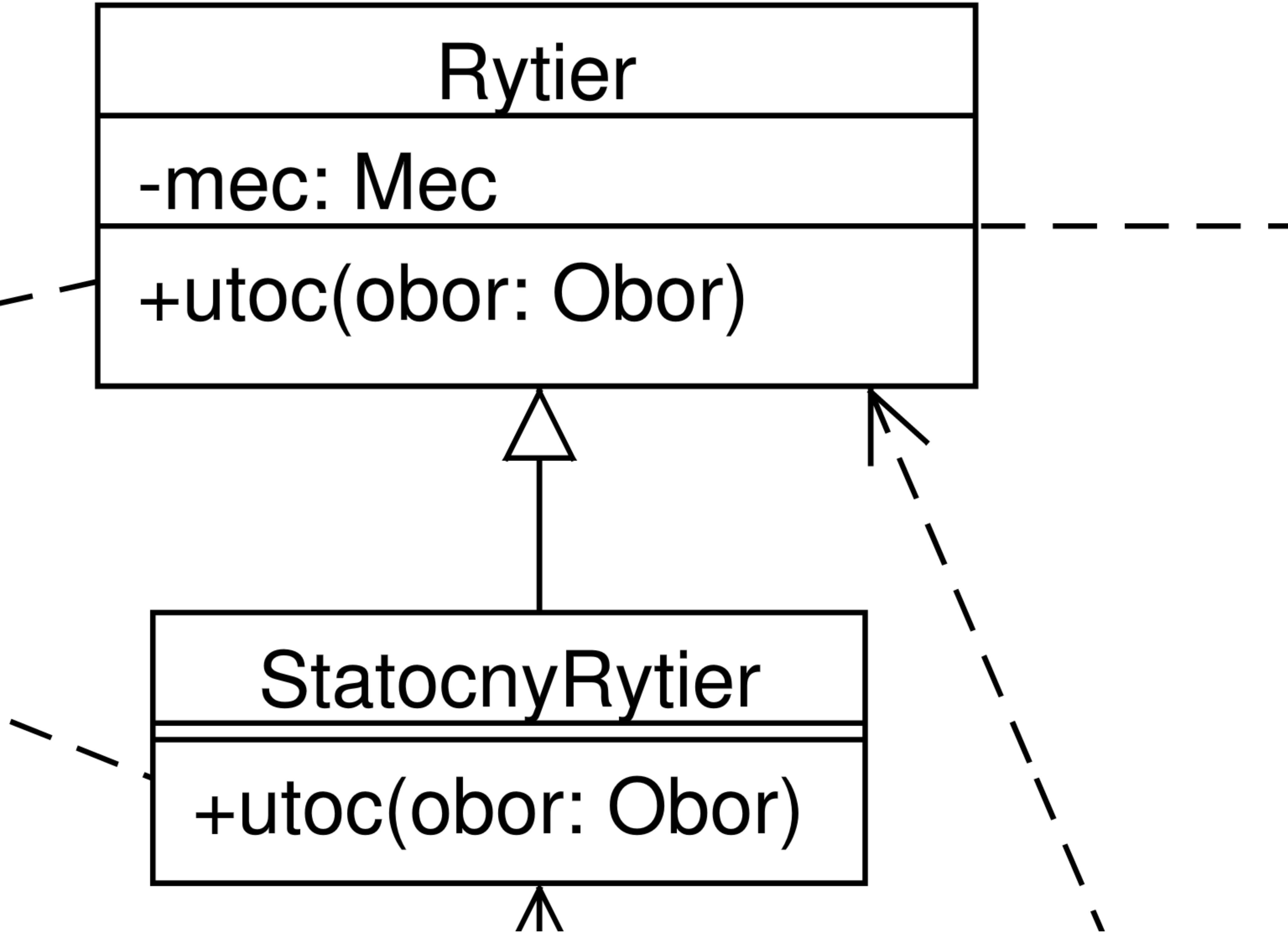
1. In the Knight class
2. In the Sword class
3. Outside these classes

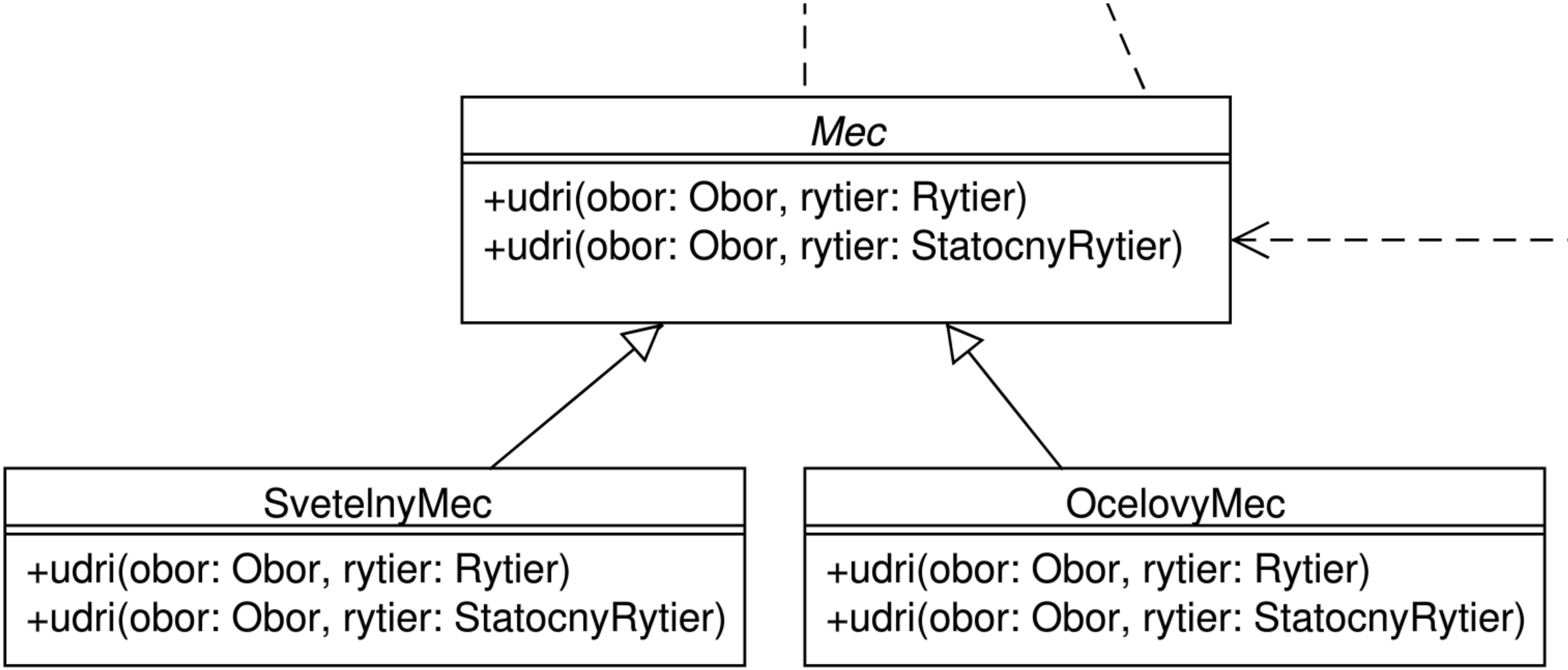
Flexibility problems occur in object-oriented code, too

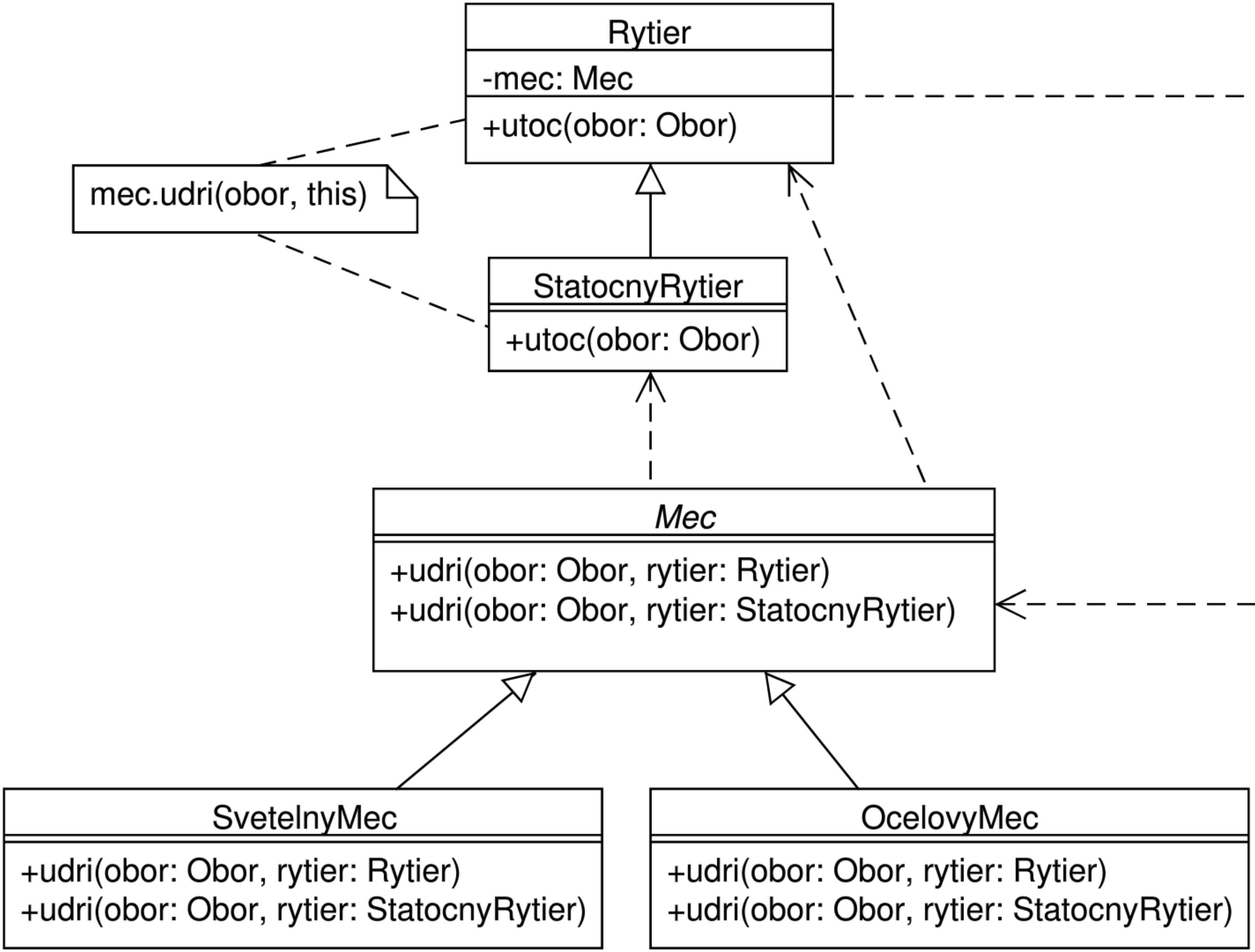
What if we would transfer the responsibility for the strike to the sword?

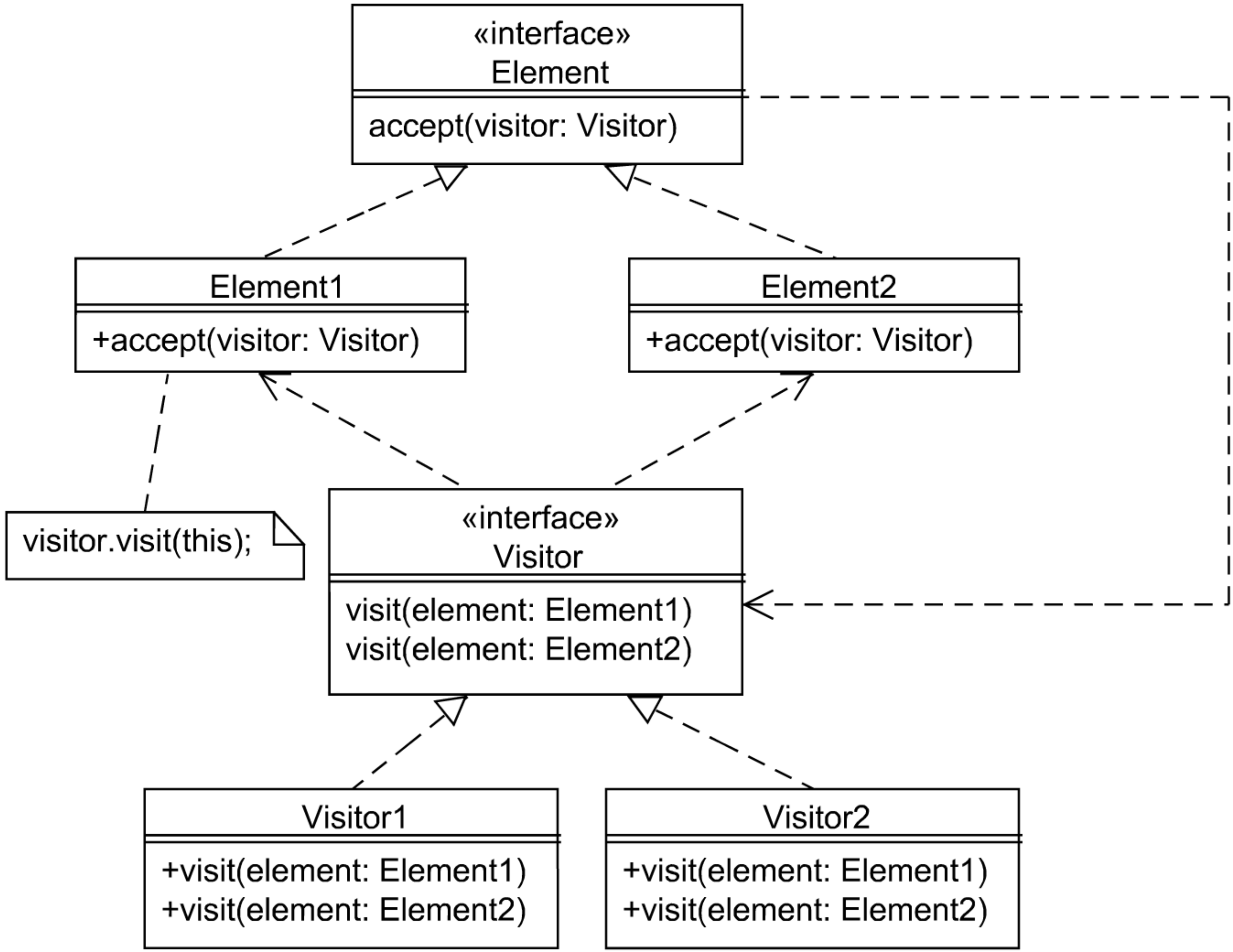
A sword, then, must know what knight handles it.

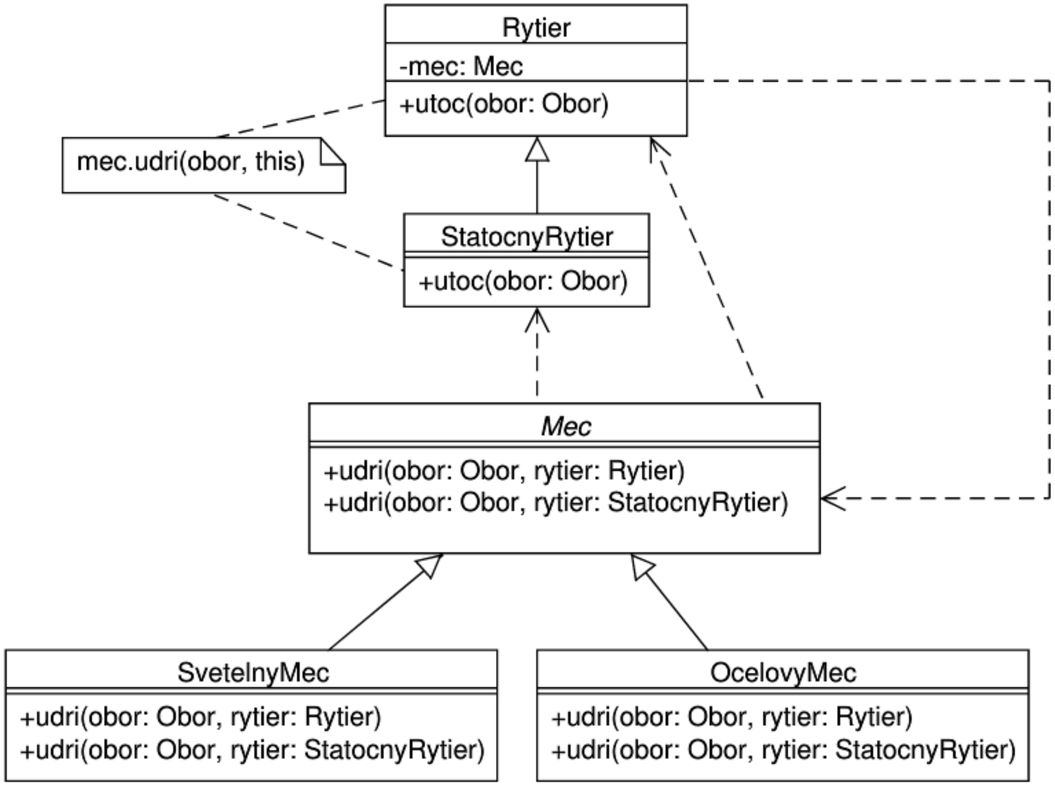
Code flexibility can be improved by applying design patterns

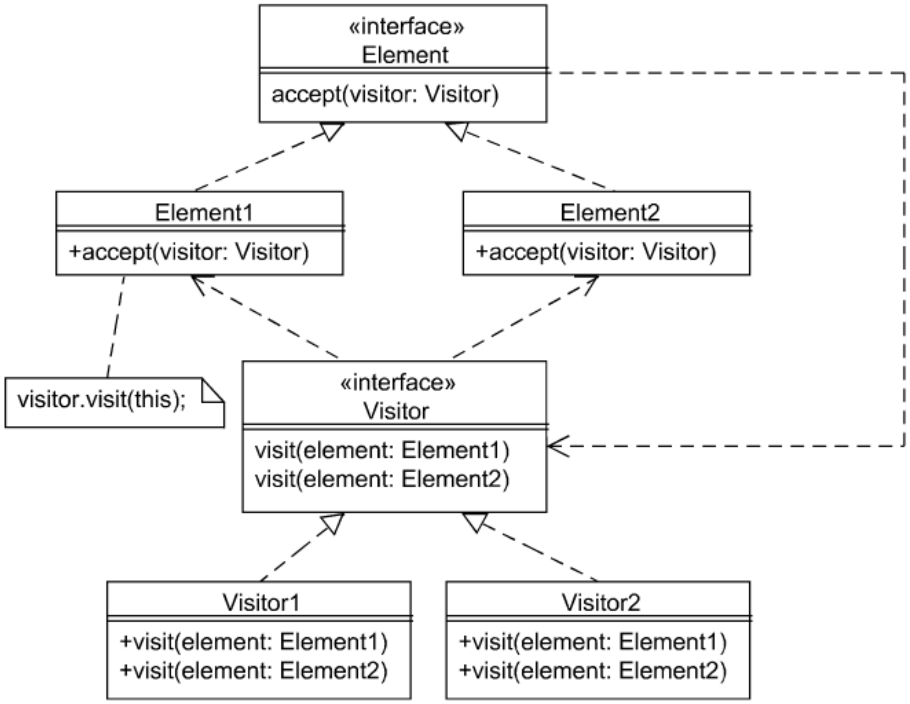

Rytier
-mec: Mec
+utoc(obor: Obor)
StatocnyRytier
+utoc(obor: Obor)


Mec
+udri(obor: Obor, rytier: Rytier)
+udri(obor: Obor, rytier: StatocnyRytier)
SvetelnyMec
+udri(obor: Obor, rytier: Rytier)
+udri(obor: Obor, rytier: StatocnyRytier)
OcelovyMec
+udri(obor: Obor, rytier: Rytier)
+udri(obor: Obor, rytier: StatocnyRytier)


Rytier
-mec: Mec
+utoc(obor: Obor)
mec.udri(obor, this)
StatocnyRytier
+utoc(obor: Obor)
Mec
+udri(obor: Obor, rytier: Rytier)
+udri(obor: Obor, rytier: StatocnyRytier)
SvetelnyMec
+udri(obor: Obor, rytier: Rytier)
+udri(obor: Obor, rytier: StatocnyRytier)
OcelovyMec
+udri(obor: Obor, rytier: Rytier)
+udri(obor: Obor, rytier: StatocnyRytier)


«interface»
Element
accept(visitor: Visitor)
Element1
+accept(visitor: Visitor)
Element2
+accept(visitor: Visitor)
visitor.visit(this);
«interface»
Visitor
visit(element: Element1)
visit(element: Element2)
Visitor1
+visit(element: Element1)
+visit(element: Element2)
Visitor2
+visit(element: Element1)
+visit(element: Element2)

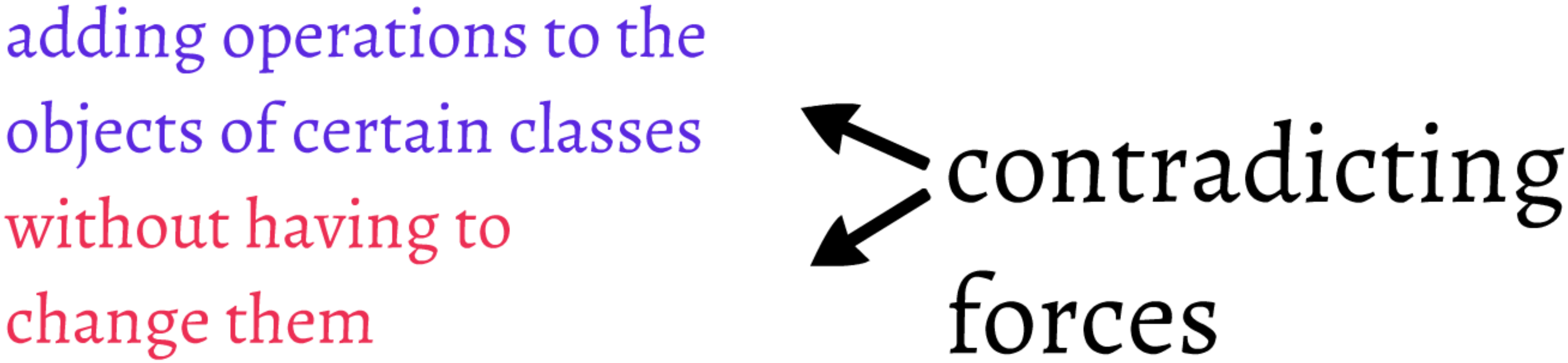

Rytier
-mec: Mec
+utoc(obor: Obor)
mec.udri(obor, this)
StatocnyRytier
+utoc(obor: Obor)
Mec
+udri(obor: Obor, rytier: Rytier)
+udri(obor: Obor, rytier: StatocnyRytier)
SvetelnyMec
+udri(obor: Obor, rytier: Rytier)
+udri(obor: Obor, rytier: StatocnyRytier)
OcelovyMec
+udri(obor: Obor, rytier: Rytier)
+udri(obor: Obor, rytier: StatocnyRytier)


«interface»
Element
accept(visitor: Visitor)
Element1
+accept(visitor: Visitor)
Element2
+accept(visitor: Visitor)
visitor.visit(this);
«interface»
Visitor
visit(element: Element1)
visit(element: Element2)
Visitor1
+visit(element: Element1)
+visit(element: Element2)
Visitor2
+visit(element: Element1)
+visit(element: Element2)

Visitor

adding operations to the objects of certain classes without having to change them

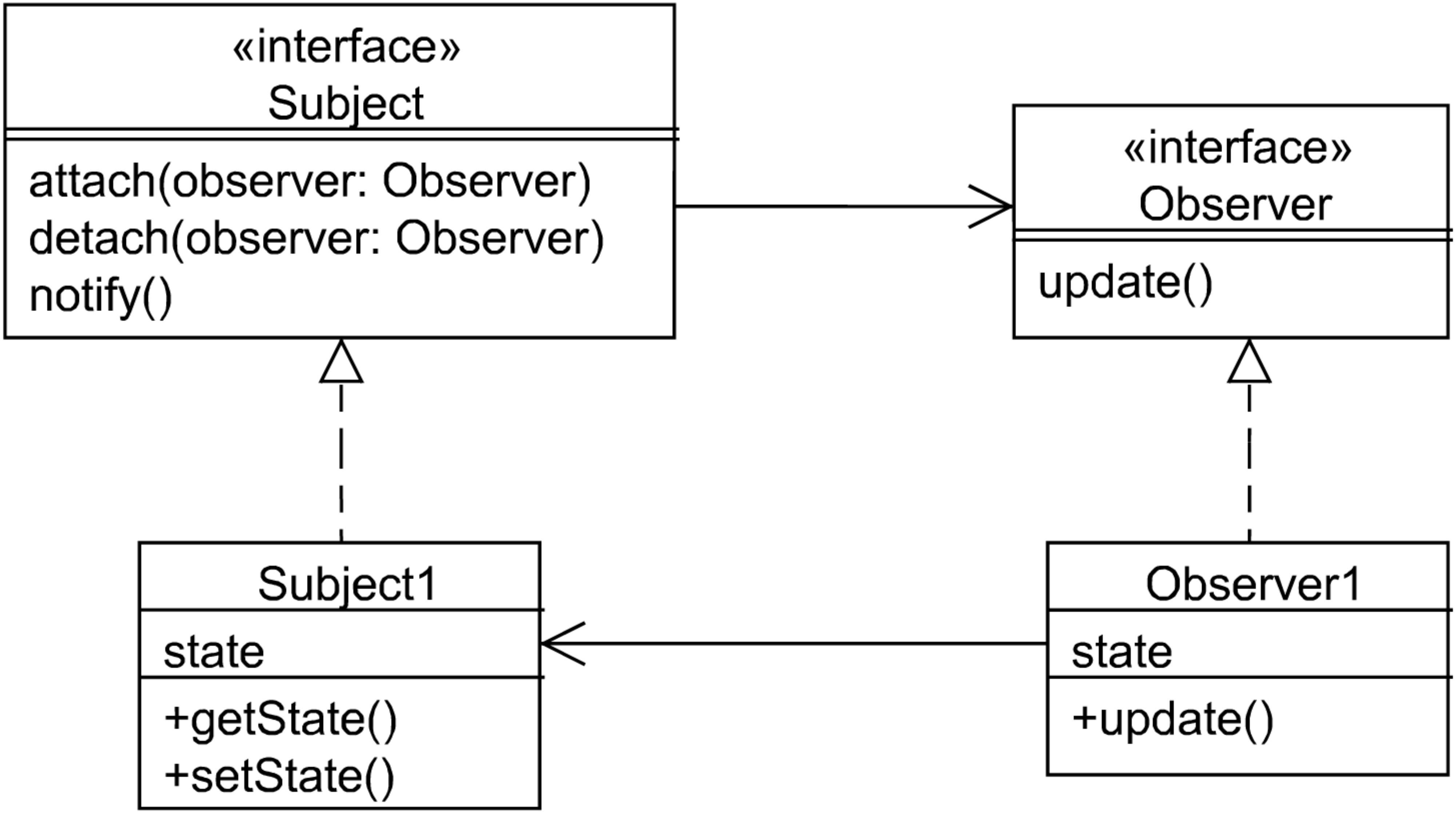
Visitor



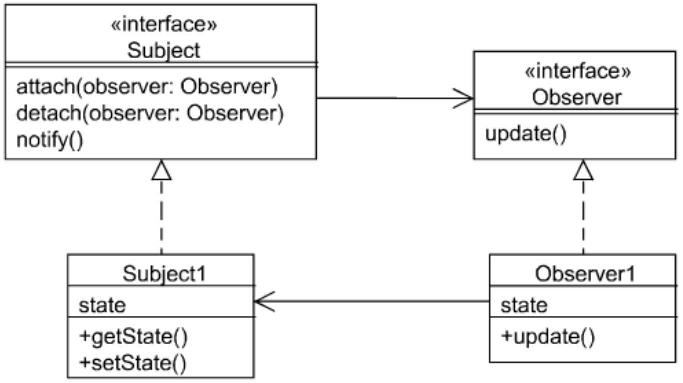
adding operations to the objects of certain classes

without having to change them

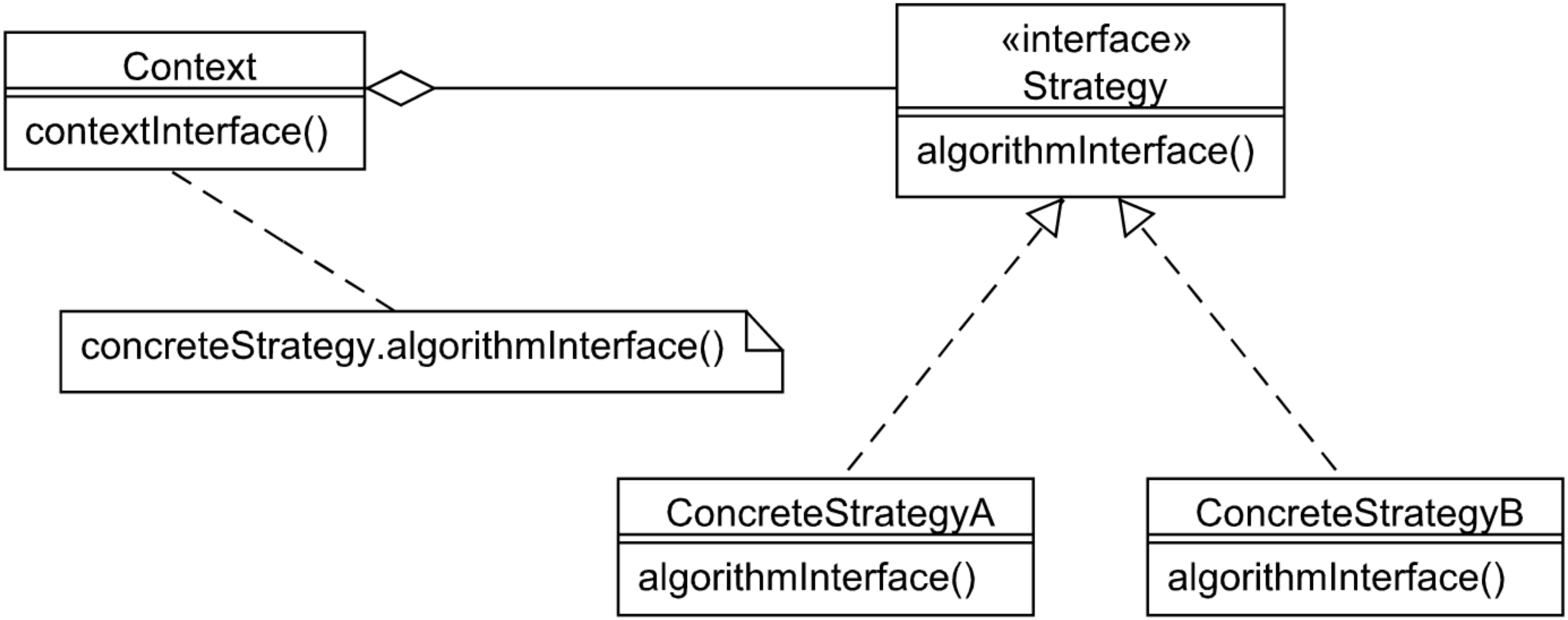
contradicting forces

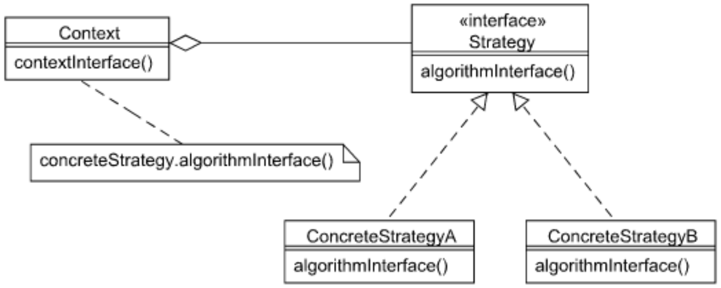

«interface»
Subject
attach(observer: Observer)
detach(observer: Observer)
notify()
«interface»
Observer
update()
Subject1
state
+getState()
+setState()
Observer1
state
+update()

Observer



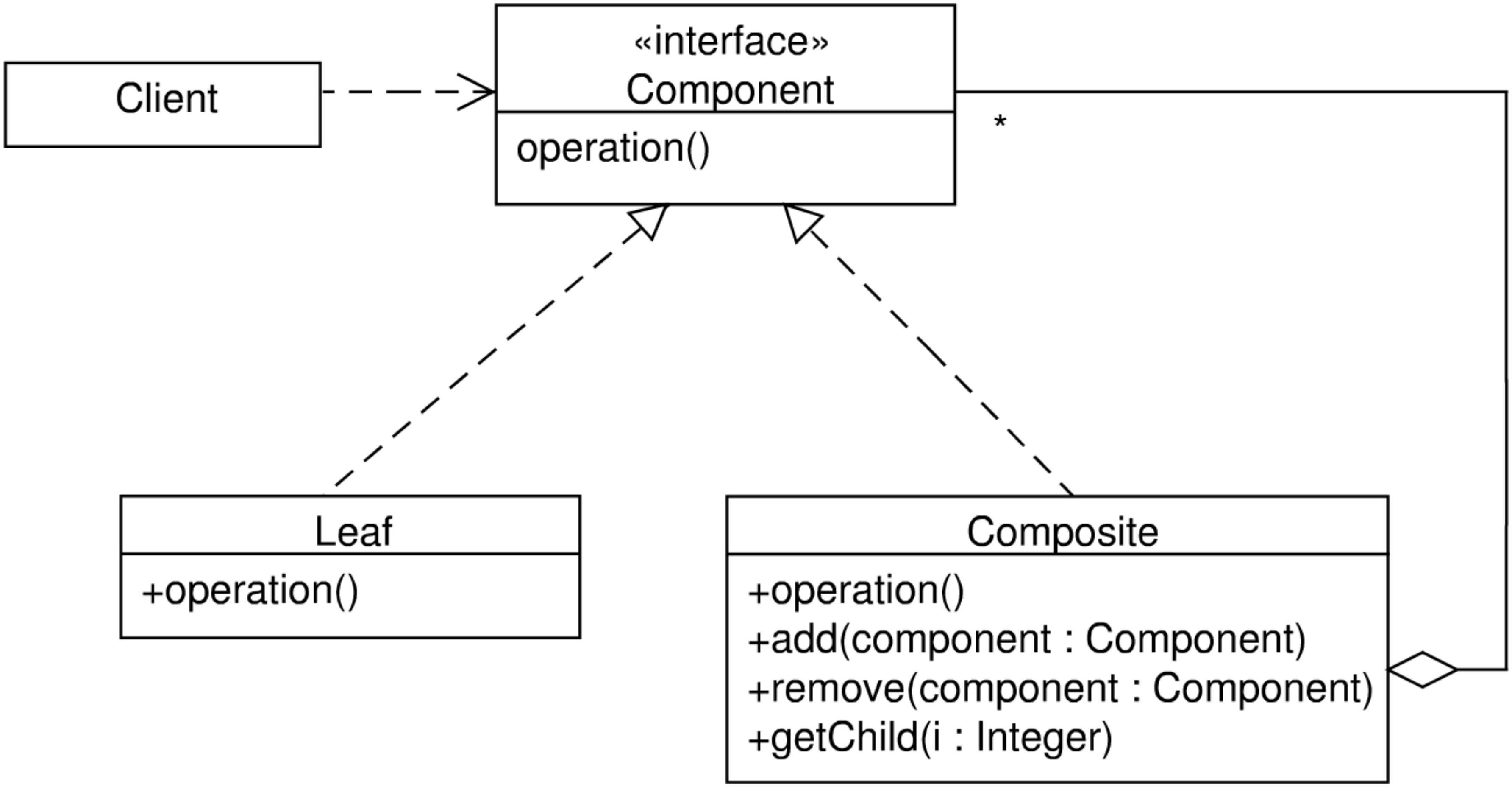
observing objects have to be notified of a change in the state of the observed object, and it has to be possible to add them without a need to modify the observed object

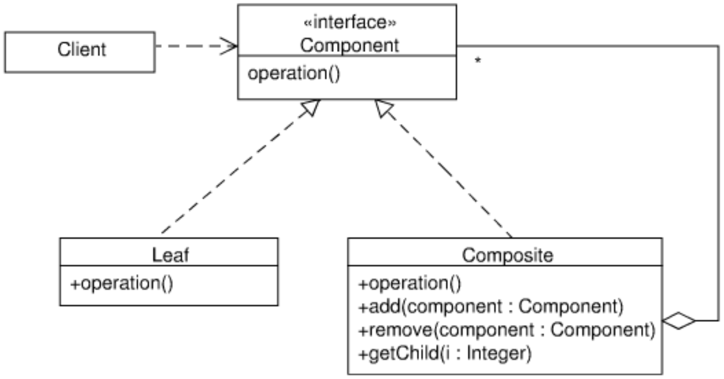

Context
contextInterface()
«interface»
Strategy
algorithmInterface()
concreteStrategy.algorithmInterface()
ConcreteStrategyA
algorithmInterface()
ConcreteStrategyB
algorithmInterface()



Strategy (Policy)

there are (and can be added) different strategies of solving a given problem, and the context of their application has to be ready for this


Client
«interface»
Component
operation()
*
Leaf
+operation()
Composite
+operation()
+add(component : Component)
+remove(component : Component)
+getChild(i : Integer)



Composite

elements may be composite and atomic,

and the way of accesing them has to be uniform

Christopher Alexander

The Timeless Way of Building

A Pattern Language

Christopher Alexander

The Timeless Way of Building

A Pattern Language

Context

- Independent Regions
  - ...
  - House Cluster
    - House for a Small Family
      - Alcoves
        - ...

A design pattern is being chosen according to the generalization of the problem situation: the nature of contradicting forces that determine it

Flexibility problems occur in object-oriented code, too

Code flexibility can be improved by applying design patterns

A design pattern is being chosen according to the generalization of the problem situation: the nature of contradicting forces that determine it