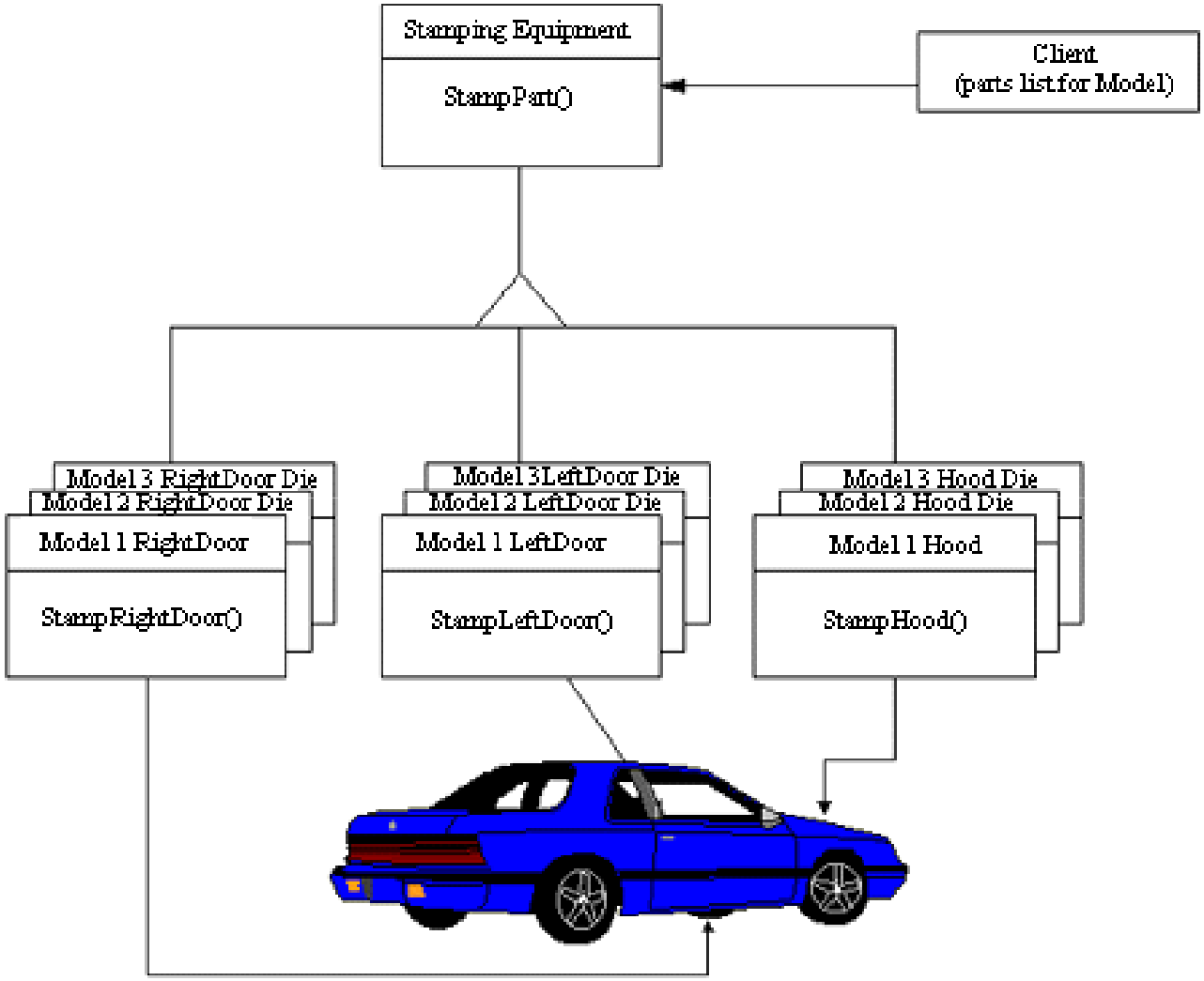
Abstract Factory

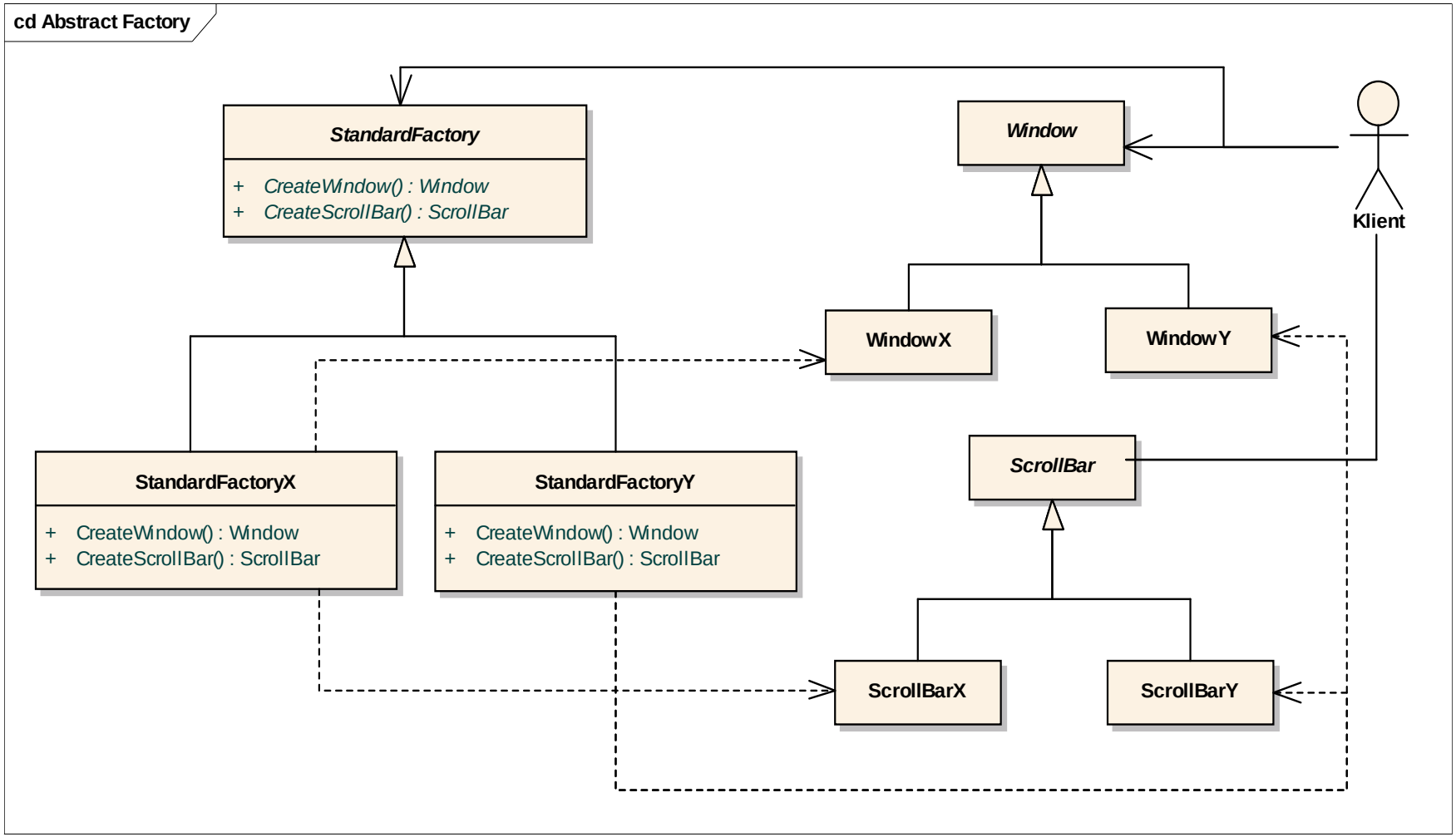


Klasifikácia a zmysel

- Object
- Creational

- Poskytuje rozhranie na vytváranie rodín súvisiacich alebo závislých objektov, pričom klient je izolovaný od volania týchto tried

Motív


cd Abstract Factory
StandardFactory
+ CreateWindow() : Window
+ CreateScrollBar() : ScrollBar
Window
Klient
WindowX
WindowY
StandardFactoryX
+ CreateWindow() : Window
+ CreateScrollBar() : ScrollBar
StandardFactoryY
+ CreateWindow() : Window
+ CreateScrollBar() : ScrollBar
ScrollBar
ScrollBarX
ScrollBarY

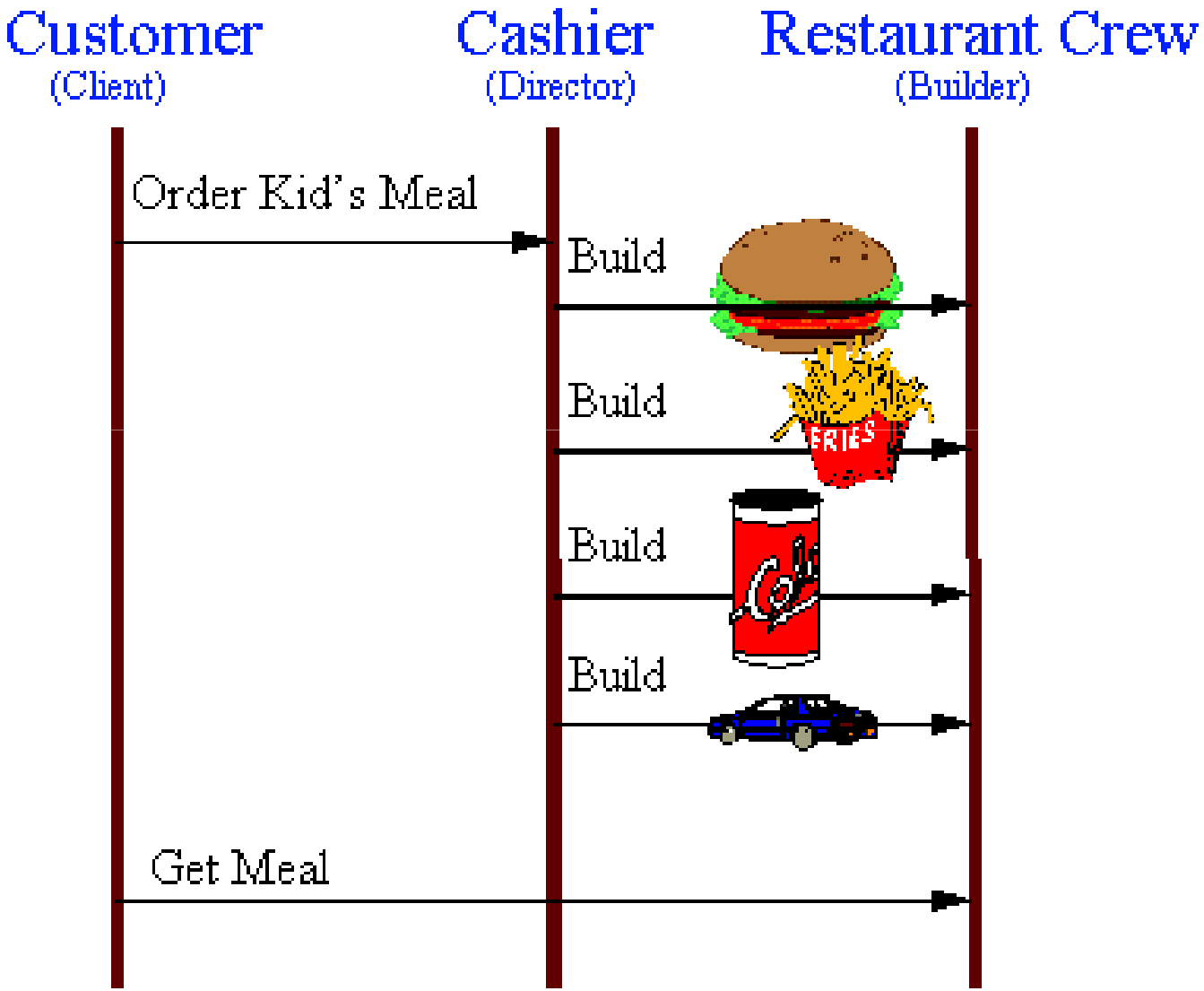
Použitie

- Izoluje konkrétne triedy
- Zjednodušuje výmenu rodiny produktov
- Napomáha konzistentnosti medzi produktmi
- Podpora nových druhov produktov je zložitá

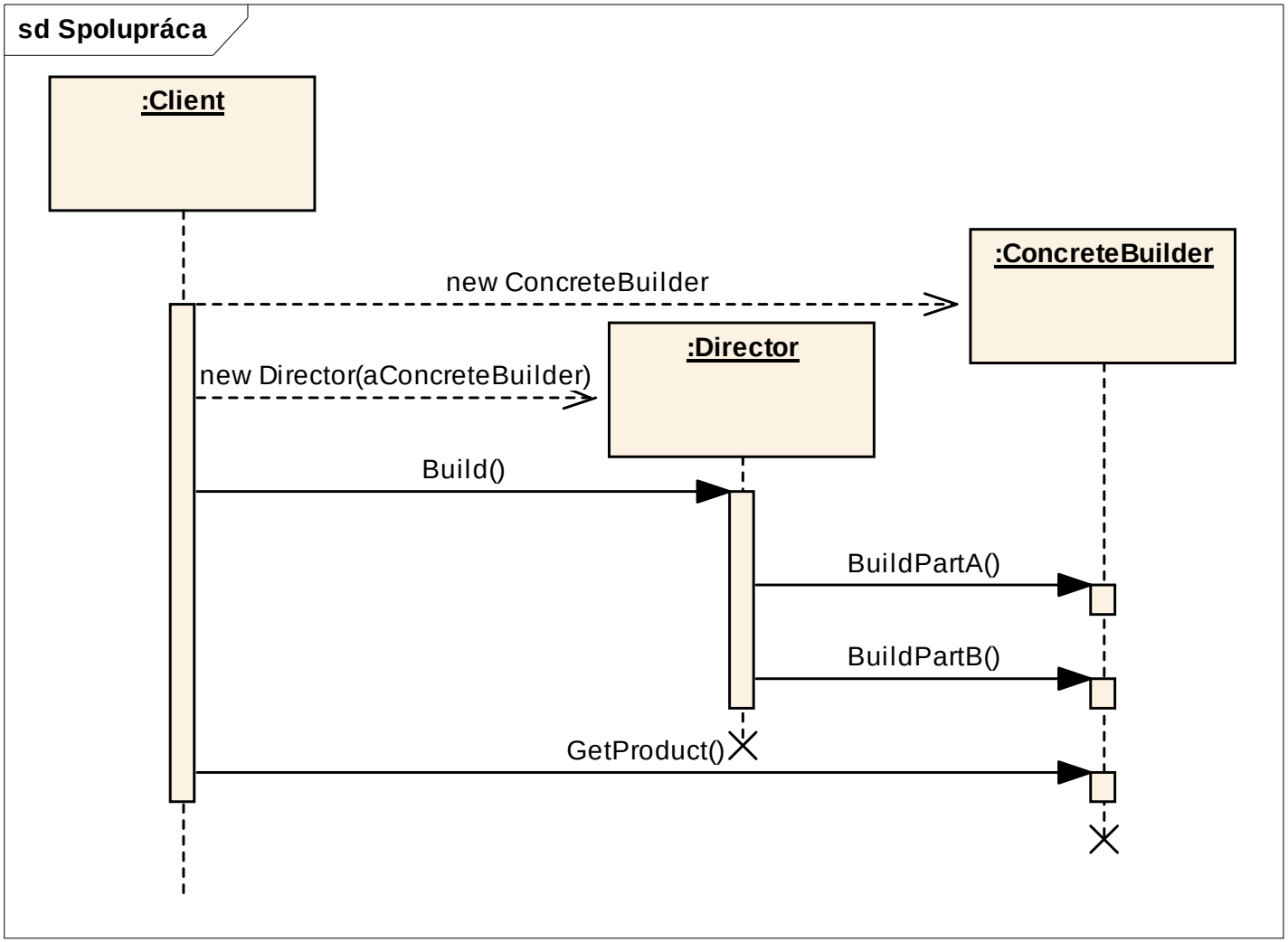
Implementácia

- ConcreteFactory ako Singletony
- Vytváranie produktov
  - Factory Method
  - Prototype
  - Katalóg tried
- Definícia extenzívnych factory
- Vzor Factory

Builder


Customer
(Client)
Cashier
(Director)
Restaurant Crew
(Builder)
Order Kid's Meal
Build
Build
Build
Build
Get Meal

Spolupráca


sd Spolupráca
:Client
:ConcreteBuilder
new ConcreteBuilder
:Director
new Director(aConcreteBuilder)
Build()
BuildPartA()
BuildPartB()
GetProduct()

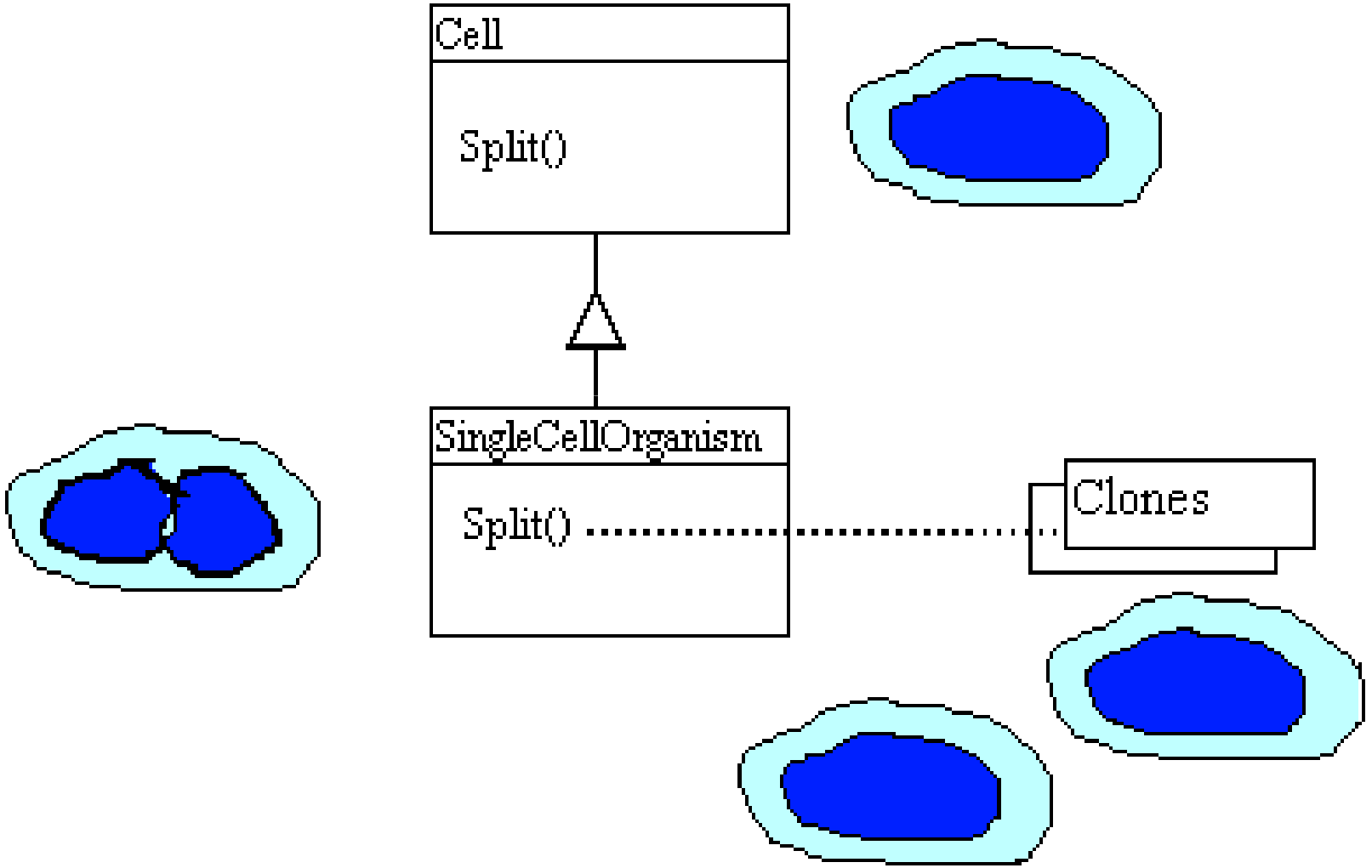
Použiteľnosť

- Uschovanie vnútornej štruktúry produktu
- Odtienenie konštruktorov a reprezentácie
- Lepšia kontrola nad konštrukčným procesom

Implementácia

- Medzivýsledky stavby
- Prečo nie abstraktnú triedu pre produkt?
- Builder musí byť úplny
- Nie abstraktné operácie na úrovni abstraktného buildra

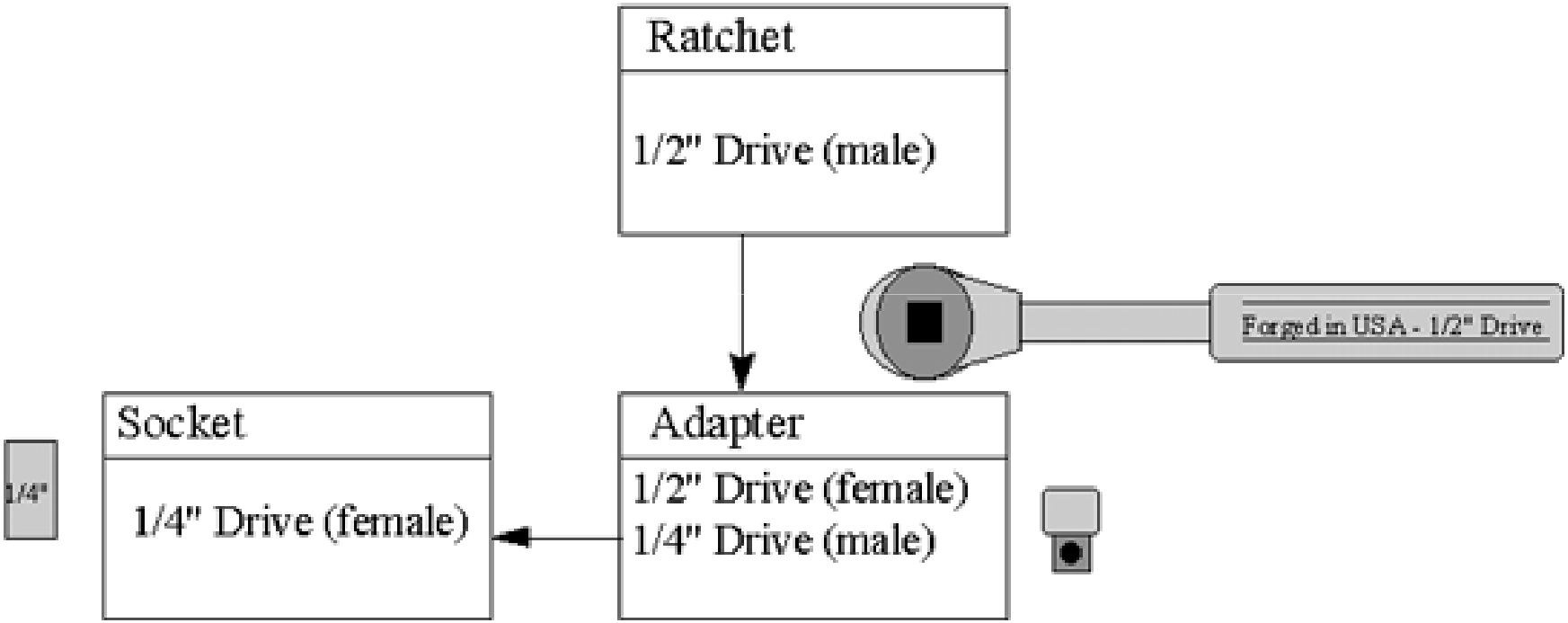
Prototype



Dôsledky

- Pridávanie a odoberanie prototypov počas behu programu
- Pseudotriedy ako implicitné stavy prototypov
- Redukuje dedenie
- Dynamická konfigurácia aplikácie triedami
- Každý potomok musí implementovať operaciu Clone

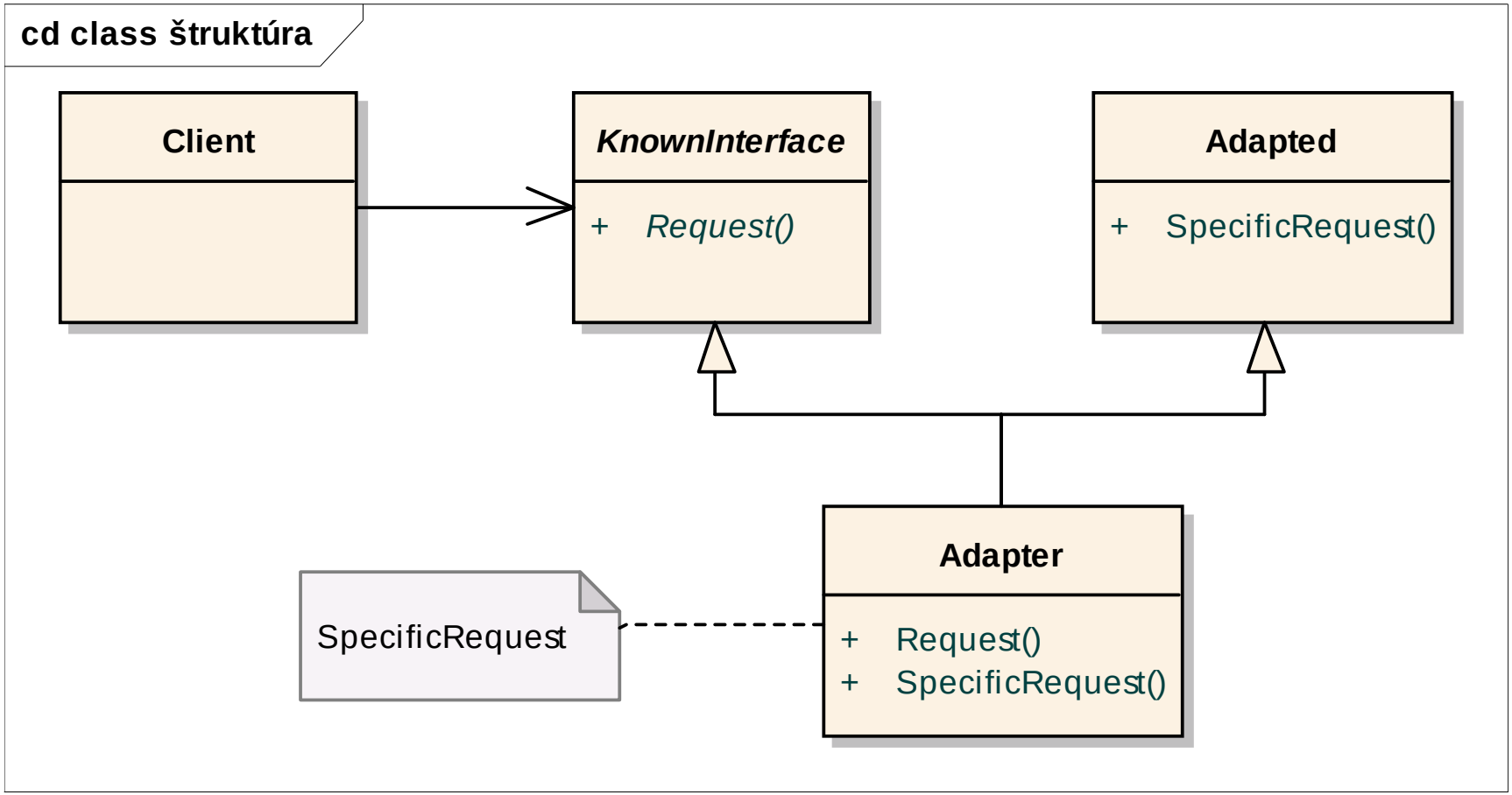
Adapter - Wrapper



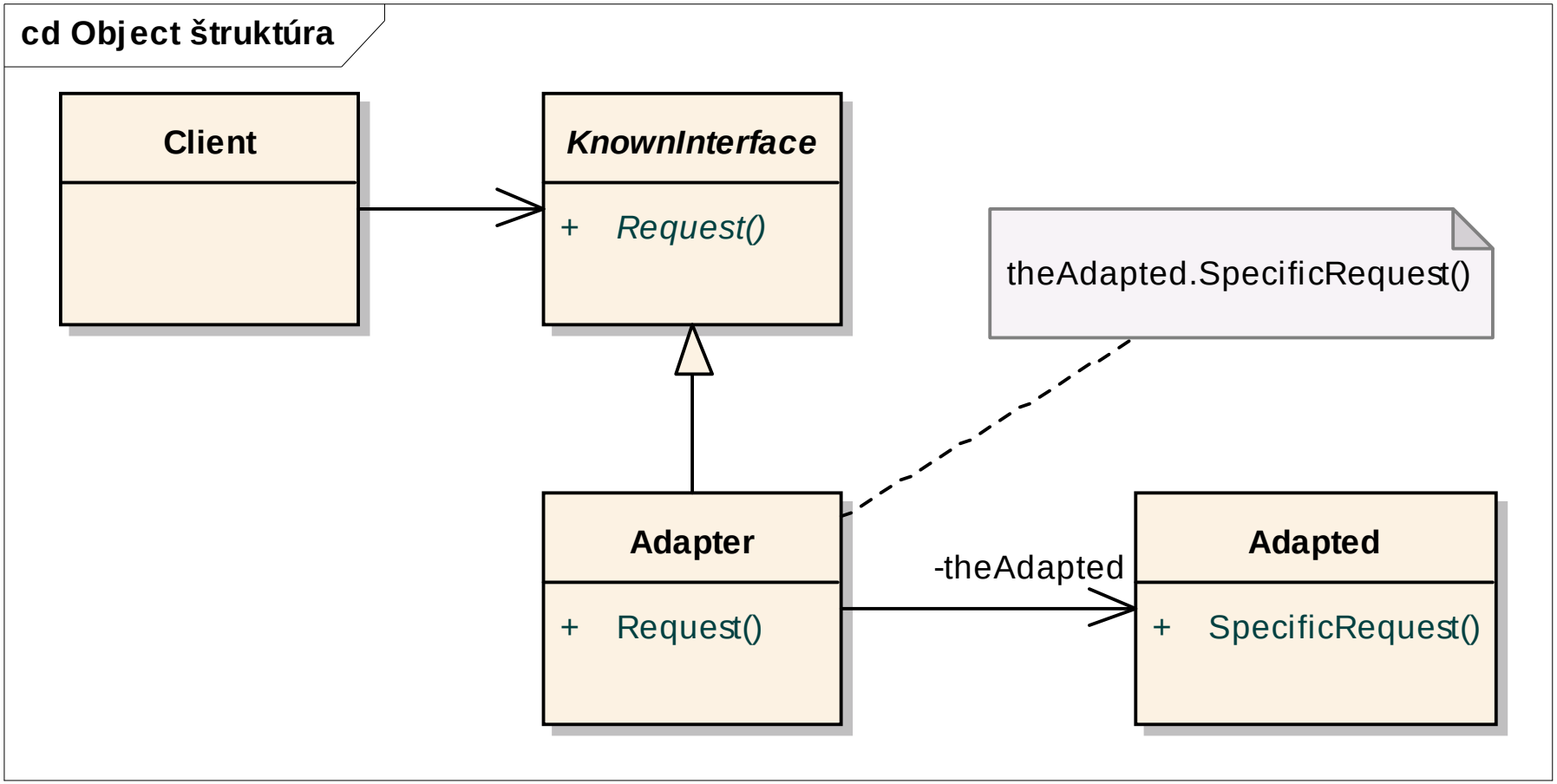
Klasifikácia a zmysel

- Object a class
- Structural
- Object: Zavádza objekt ako spojku medzi klienta, ktorý očakáva určitý interface a neznámy interface
- Class: Zavádza triedu spojujúcu neznámy a očakávaný interface

Štruktúra


cd class štruktúra
Client
KnownInterface
+ Request()
Adapted
+ SpecificRequest()
Adapter
+ Request()
+ SpecificRequest()
SpecificRequest

Štruktúra

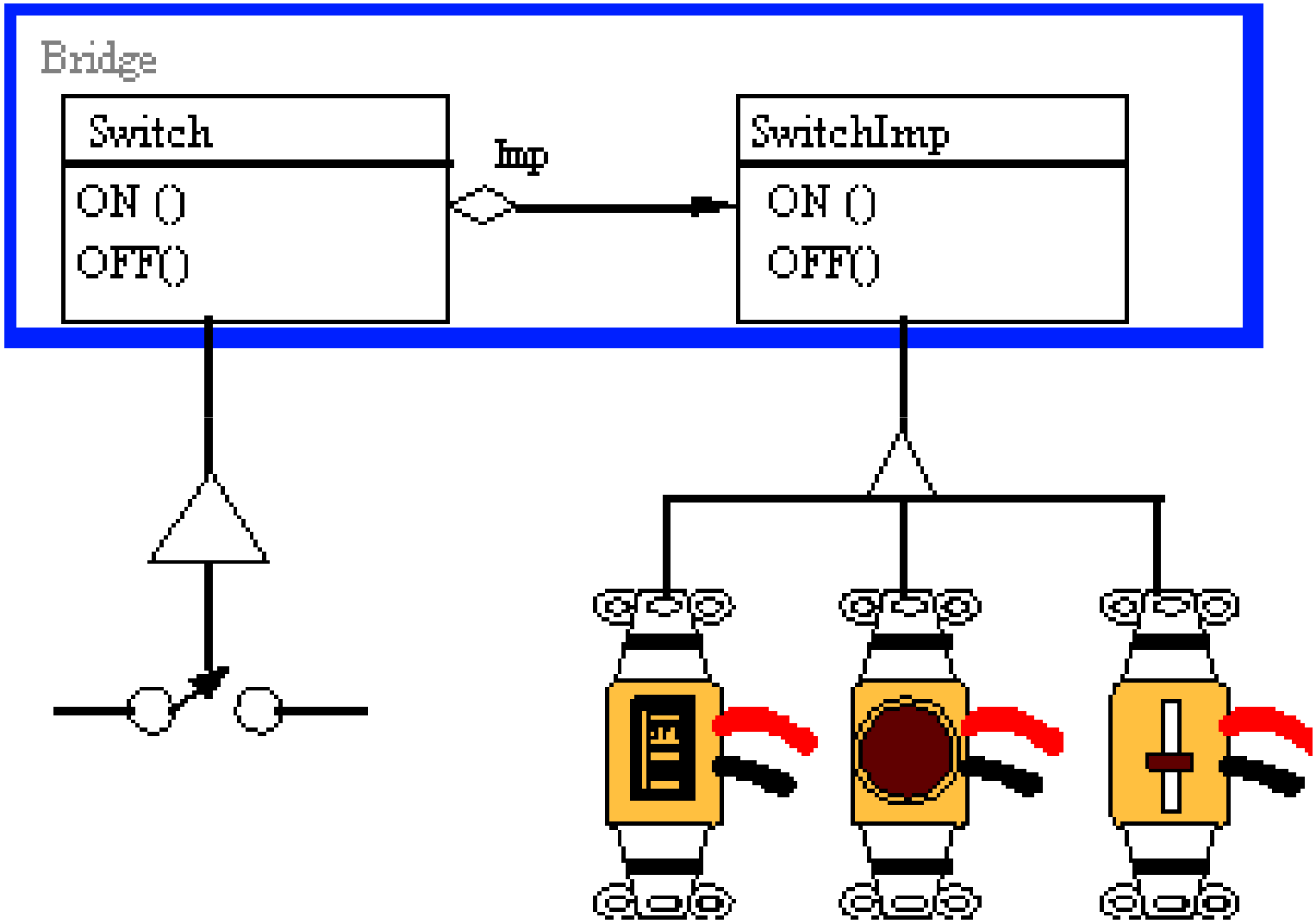

cd Object štruktúra
Client
KnownInterface
+ Request()
theAdapted.SpecificRequest()
Adapter
+ Request()
-theAdapted
Adapted
+ SpecificRequest()

Class vs Object

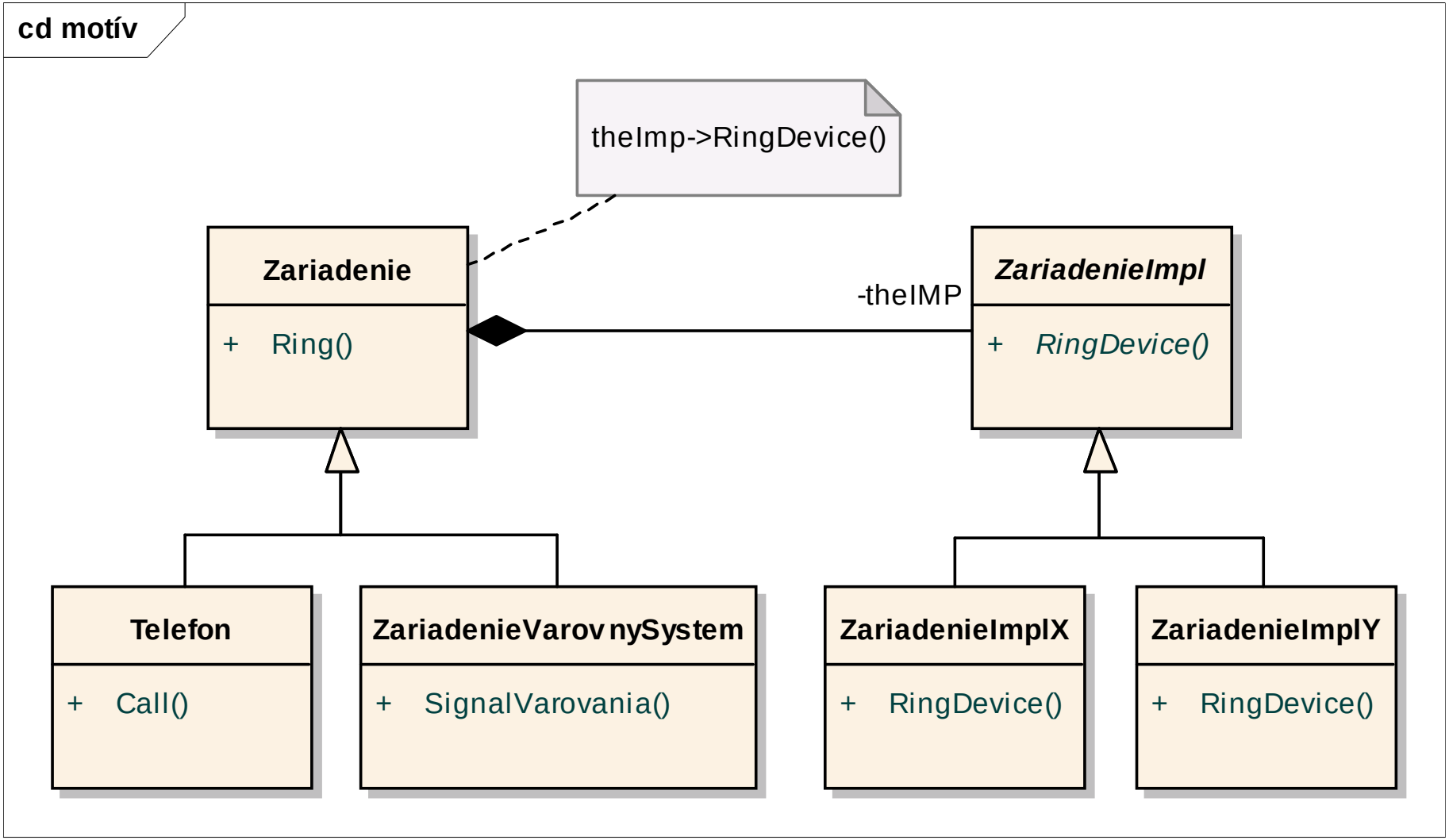
- Neumožňuje pracovať aj s potomkami triedy Adapted
- Umožňuje ľahko prepísať metódy triedy Adapted

- Umožňuje pracovať aj s potomkami triedy Adapted, pridať súčasne funkcionalitu všetkým potomkom
- Ťažšie sa prepisuje funkcionalita triedy Adapted

Bridge



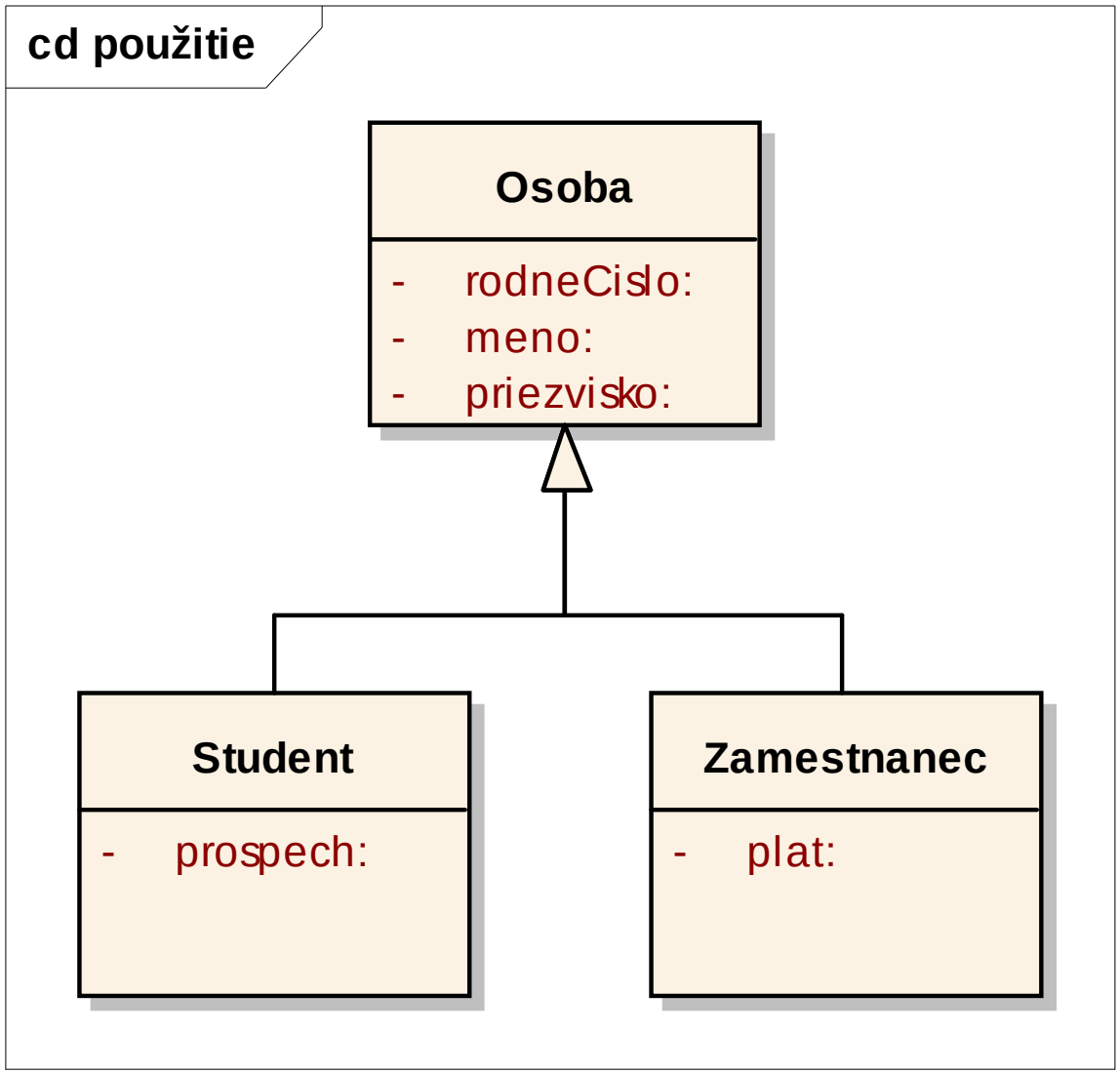
Motív


cd motív
theImp->RingDevice()
Zariadenie
+ Ring()
-theIMP
ZariadenieImpl
+ RingDevice()
Telefon
+ Call()
ZariadenieVarovnySystem
+ SignalVarovania()
ZariadenieImplX
+ RingDevice()
ZariadenieImplY
+ RingDevice()

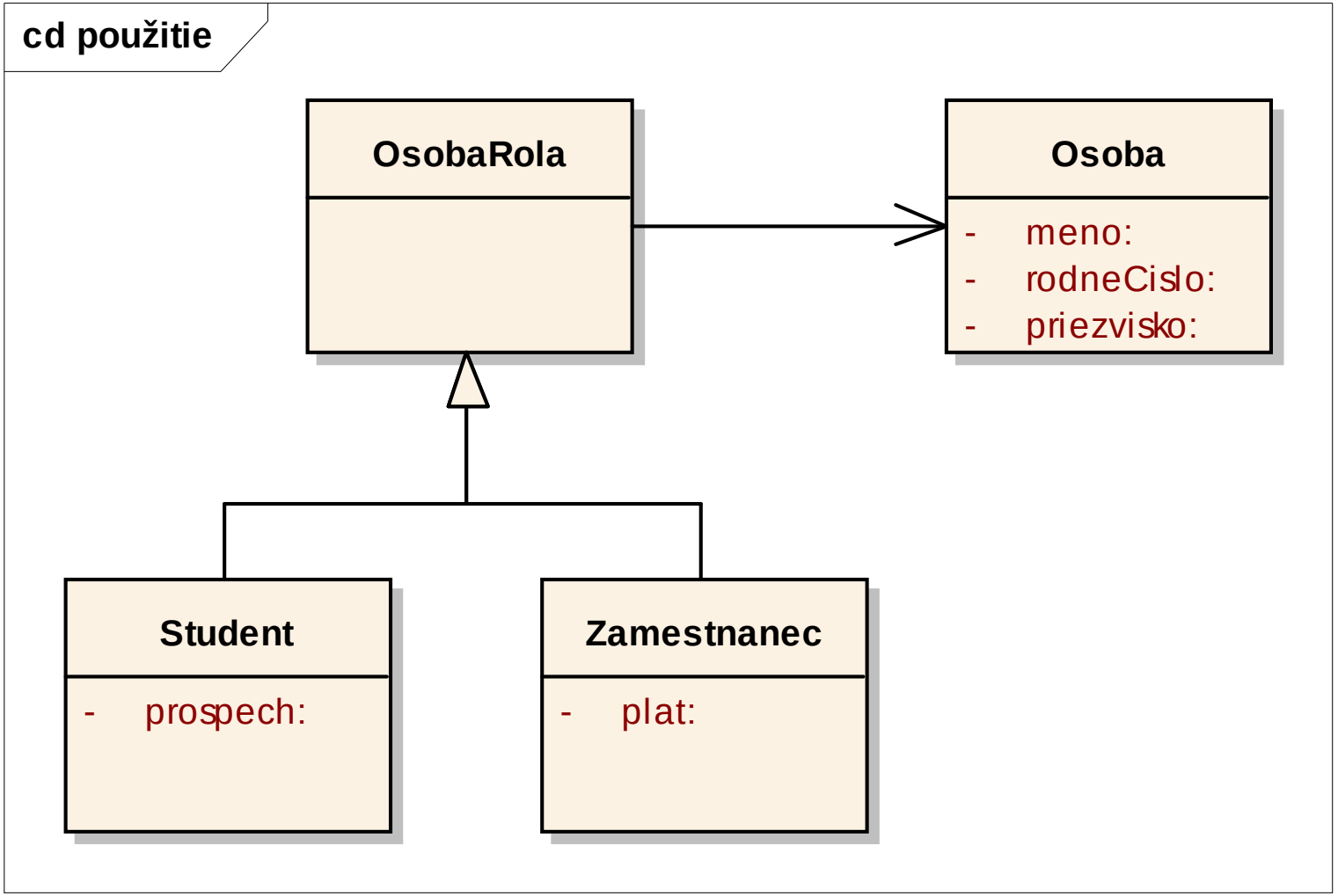
Použitie

- Abstrakcia a implementácia sú oddelené aj fyzicky
- Nezávislosť rozširovania stromov
- Riešenie chyby putovania inštancie a množenia tried

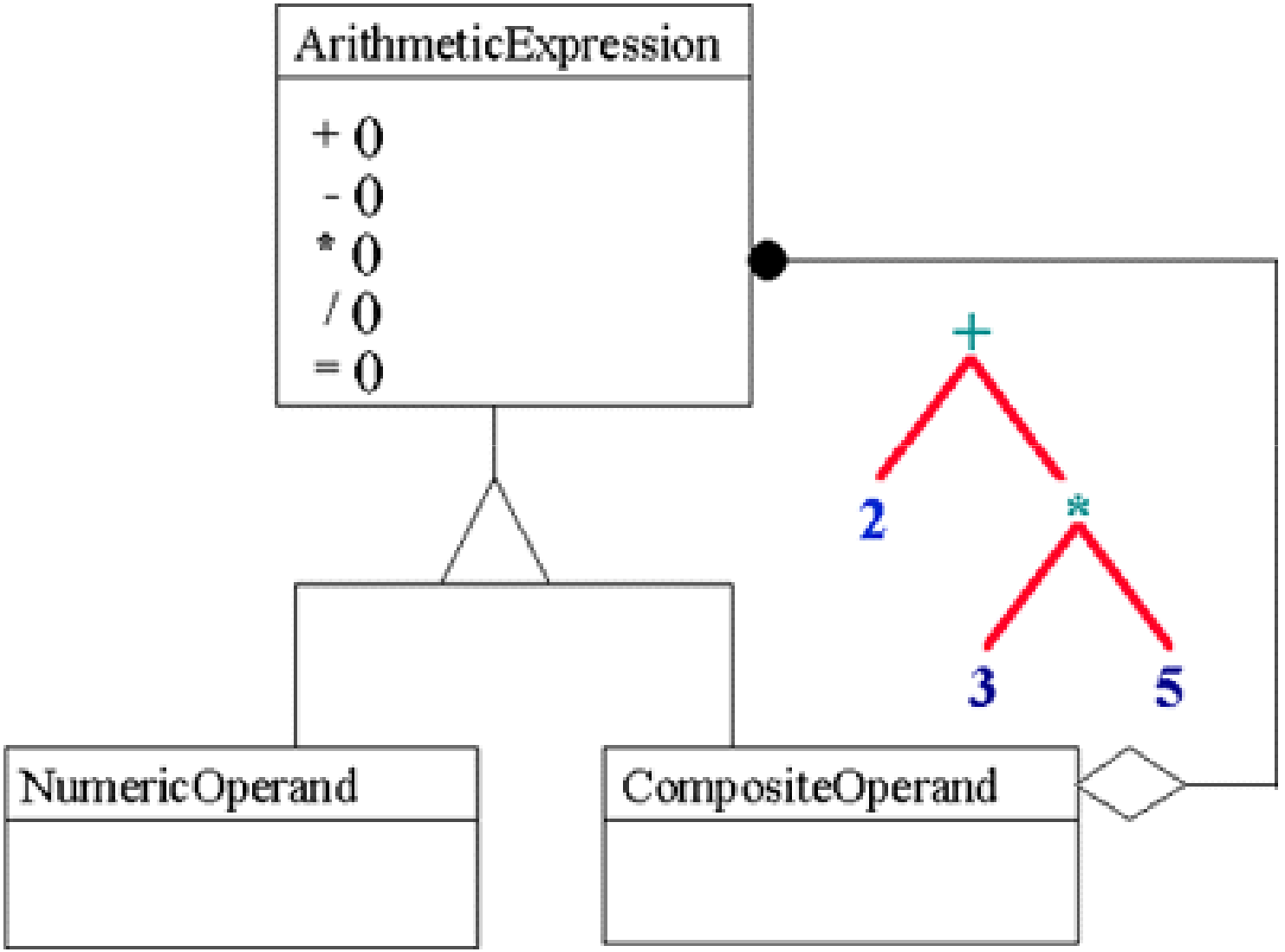
Explosion of subclasses



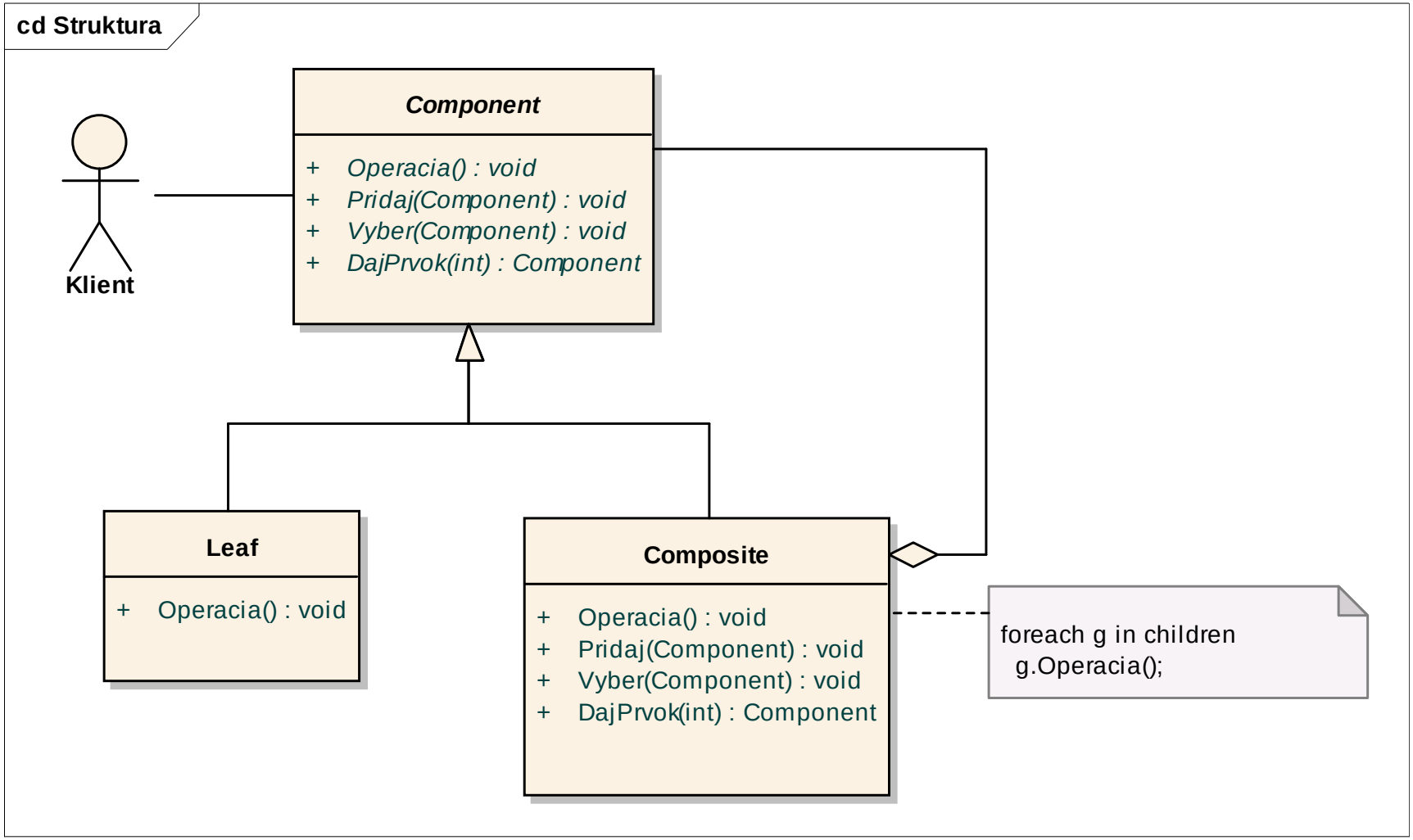
Explosion of subclasses



Composite


ArithmeticExpression
+ ()
- ()
* ()
/ ()
= ()
+
2
*
3
5
NumericOperand
CompositeOperand

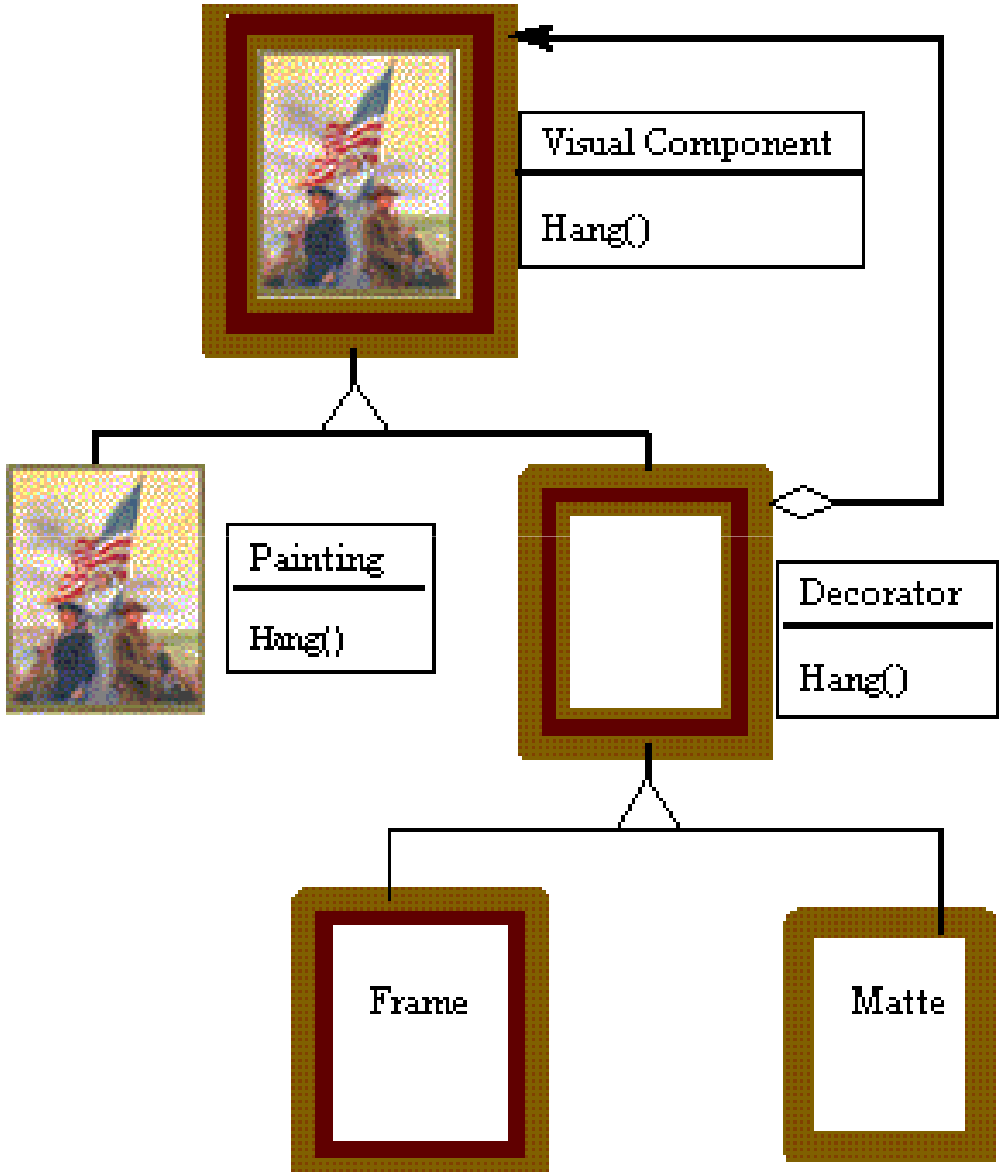
Štruktúra


cd Struktura
Component
+ Operacia() : void
+ Pridaj(Component) : void
+ Vyber(Component) : void
+ DajPrvok(int) : Component
Klient
Leaf
+ Operacia() : void
Composite
+ Operacia() : void
+ Pridaj(Component) : void
+ Vyber(Component) : void
+ DajPrvok(int) : Component
foreach g in children
g.Operacia();

Implementácia

- Explicitná referencia na nadriadený objekt
- Zdieľanie komponentov
- Maximalizácia rozhrania triedy Component
- Deklarácia operácií na správu vnorených objektov
  - Transparentnosť
  - Bezpečnosť
- Aká je najlepšia štruktúra na ukladanie komponentov?

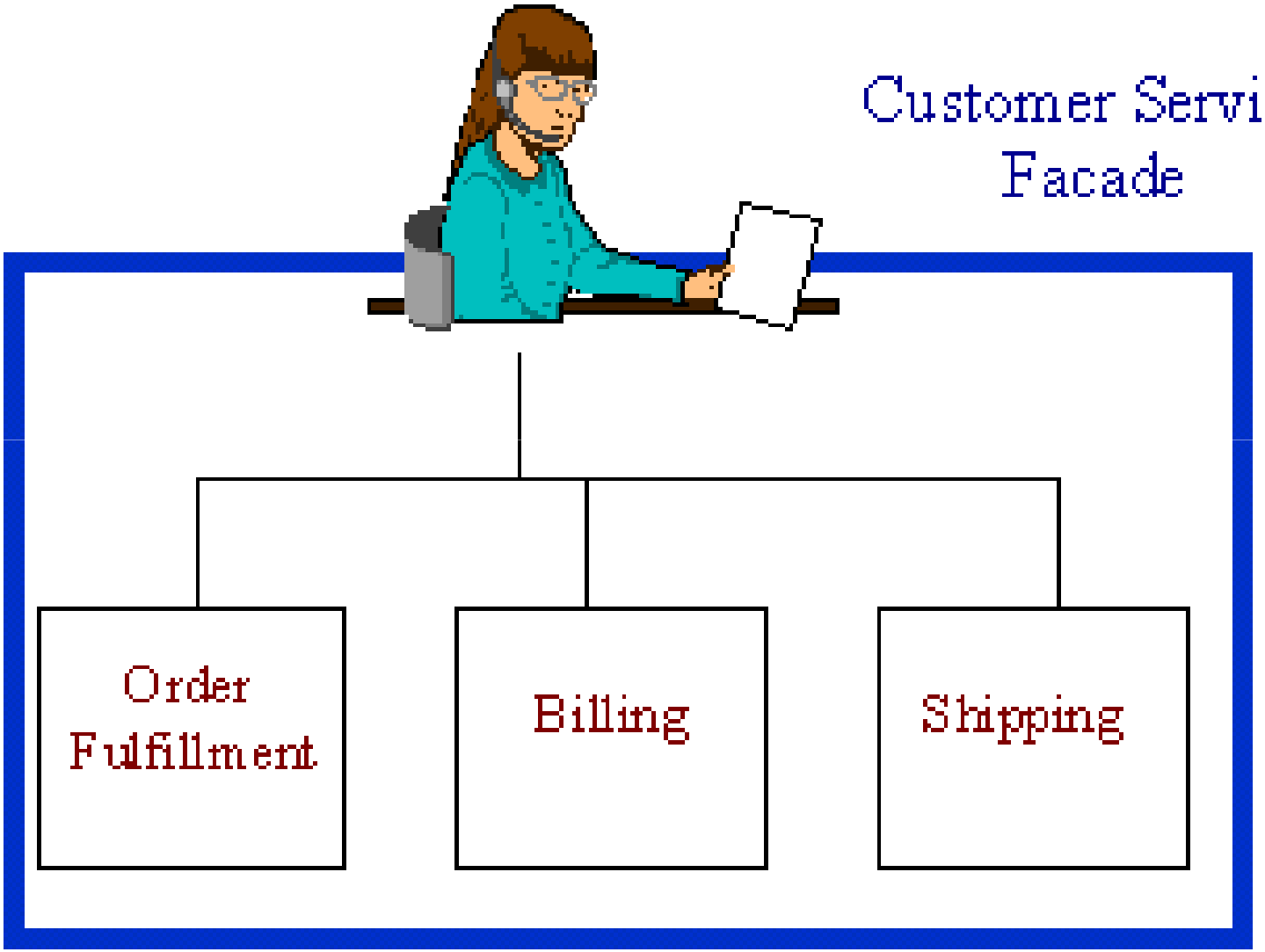
Decorator - Wrapper



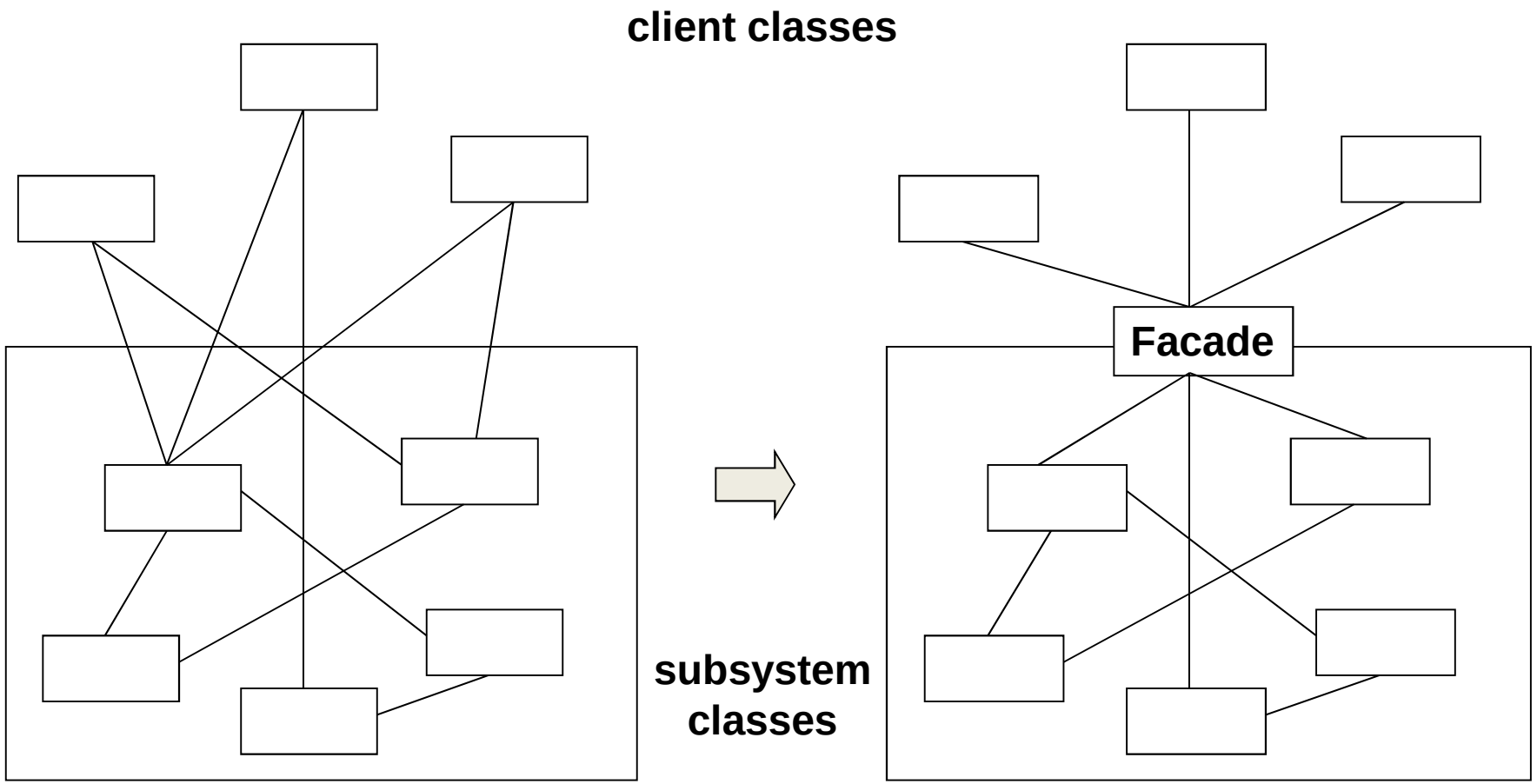
Použitie

- Flexibilita a počet tried
- Ľubovoľná kombinácia
- Rozdelenie funkcionality
- Komponent čo najjednoduchší

Facade


Customer Service
Facade
Order
Fulfillment
Billing
Shipping

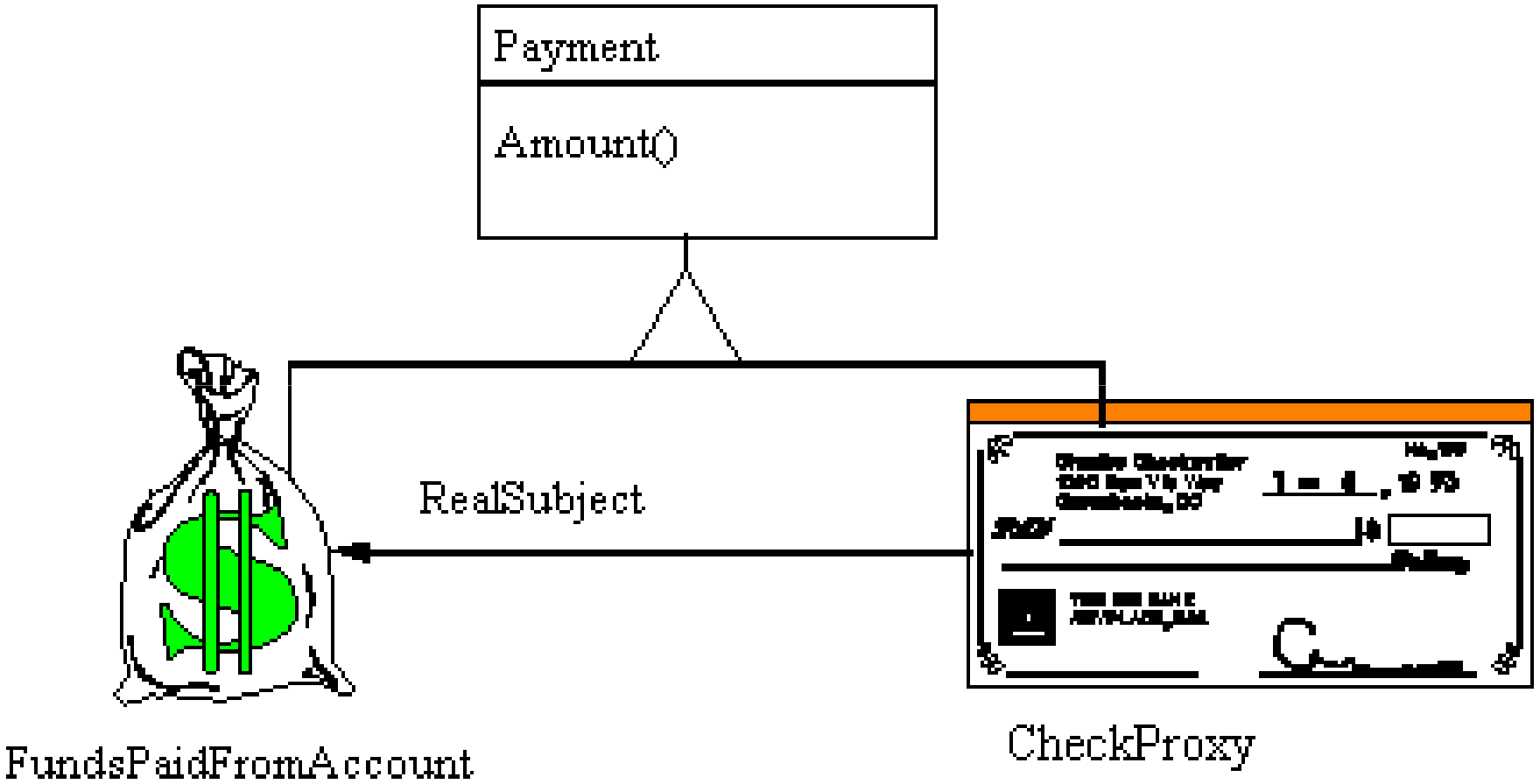
Motív



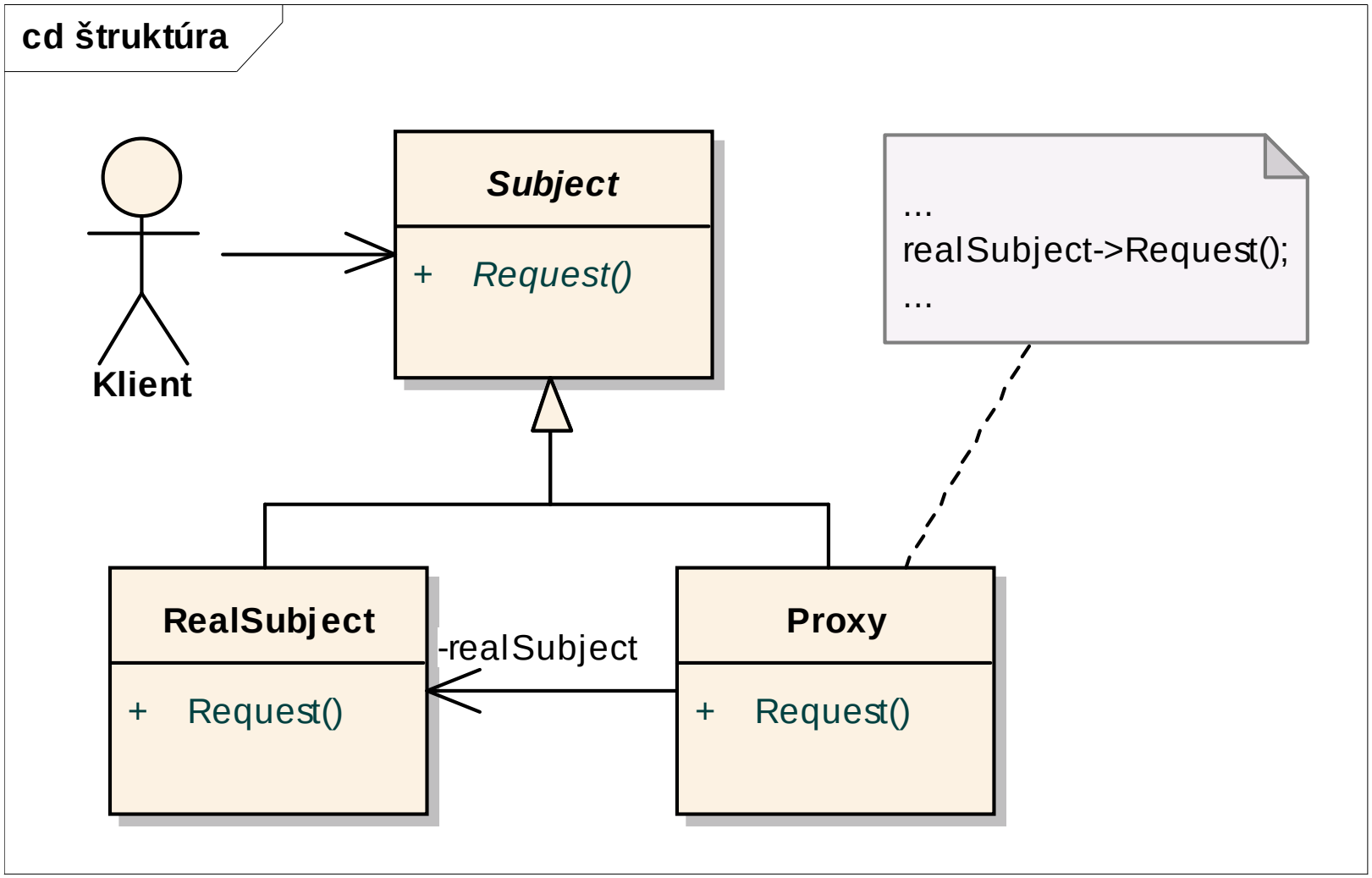
Použitie

- Na jednoduché rozhranie komplexných subsystémov
- Zníženie závislosti medzi triedami
- Zdieľané inštancie komponent
- Problémy s obecnosťou

Proxy - Surrogate



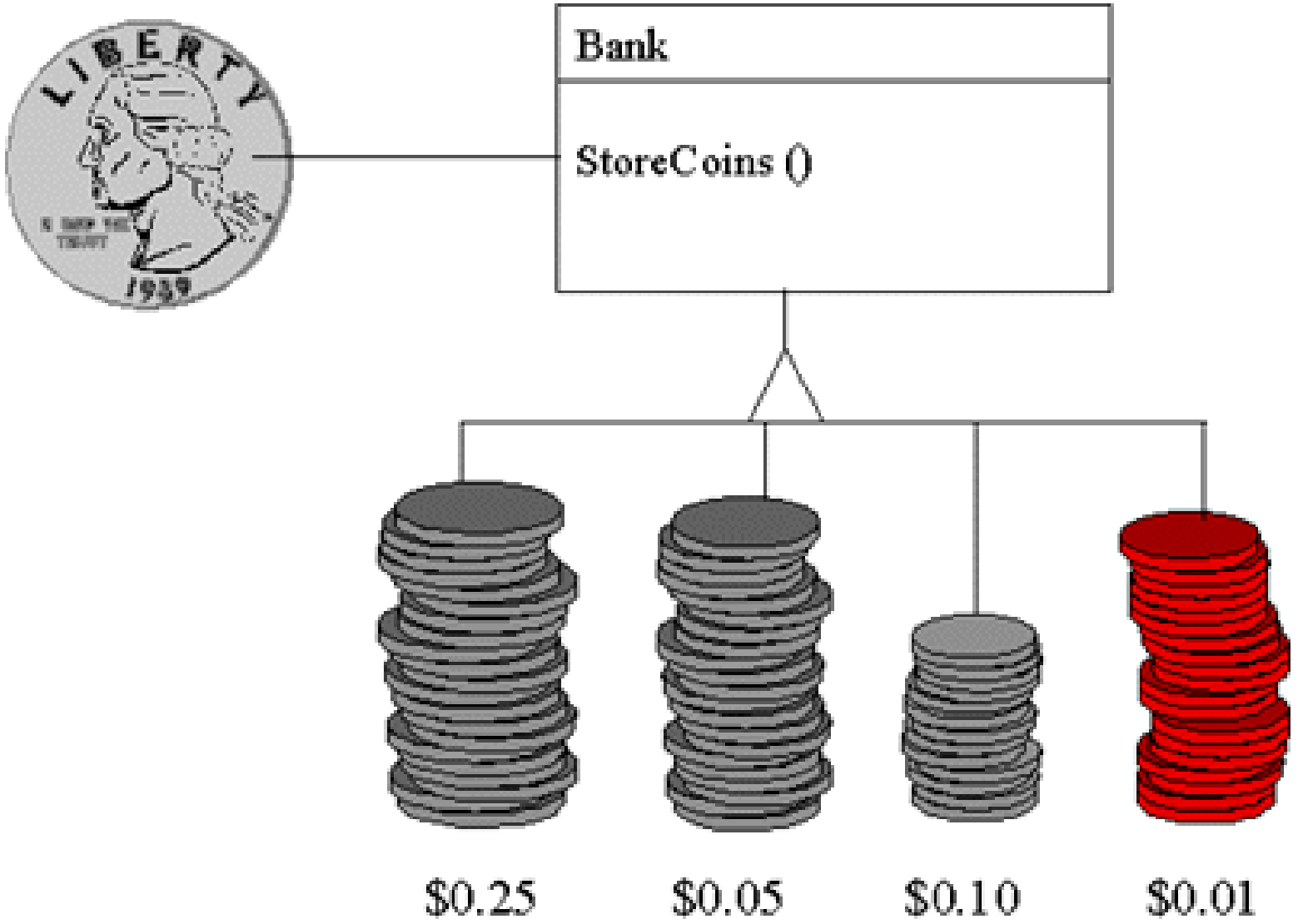
Štruktúra


cd štruktúra
Klient
Subject
+ Request()
...
realSubject->Request();
...
RealSubject
+ Request()
-realSubject
Proxy
+ Request()

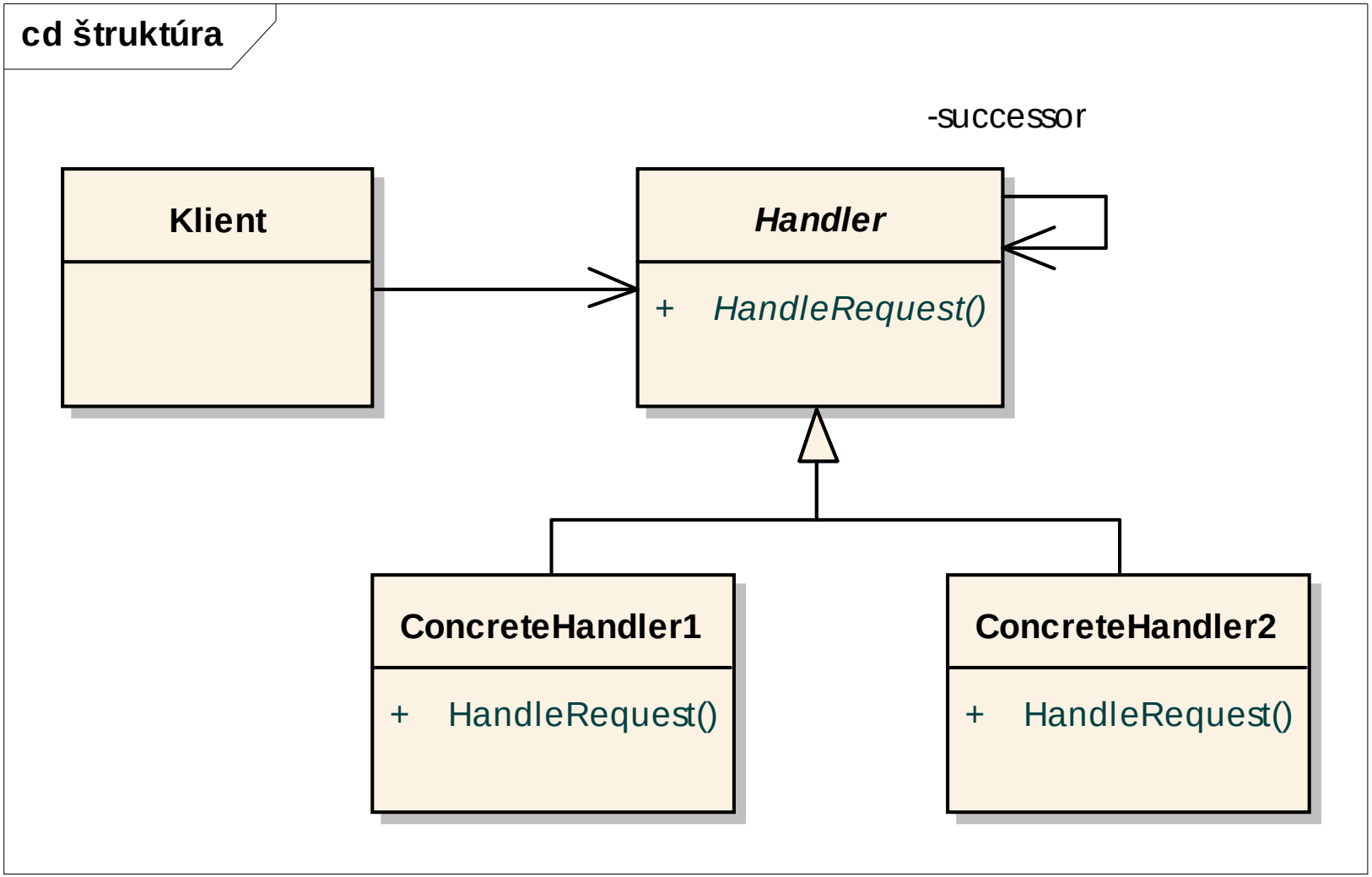
Použitie

- Remote proxy
- Protection proxy
- Virtual proxy
- Smart reference
- Copy-on-write

Chain of responsibility



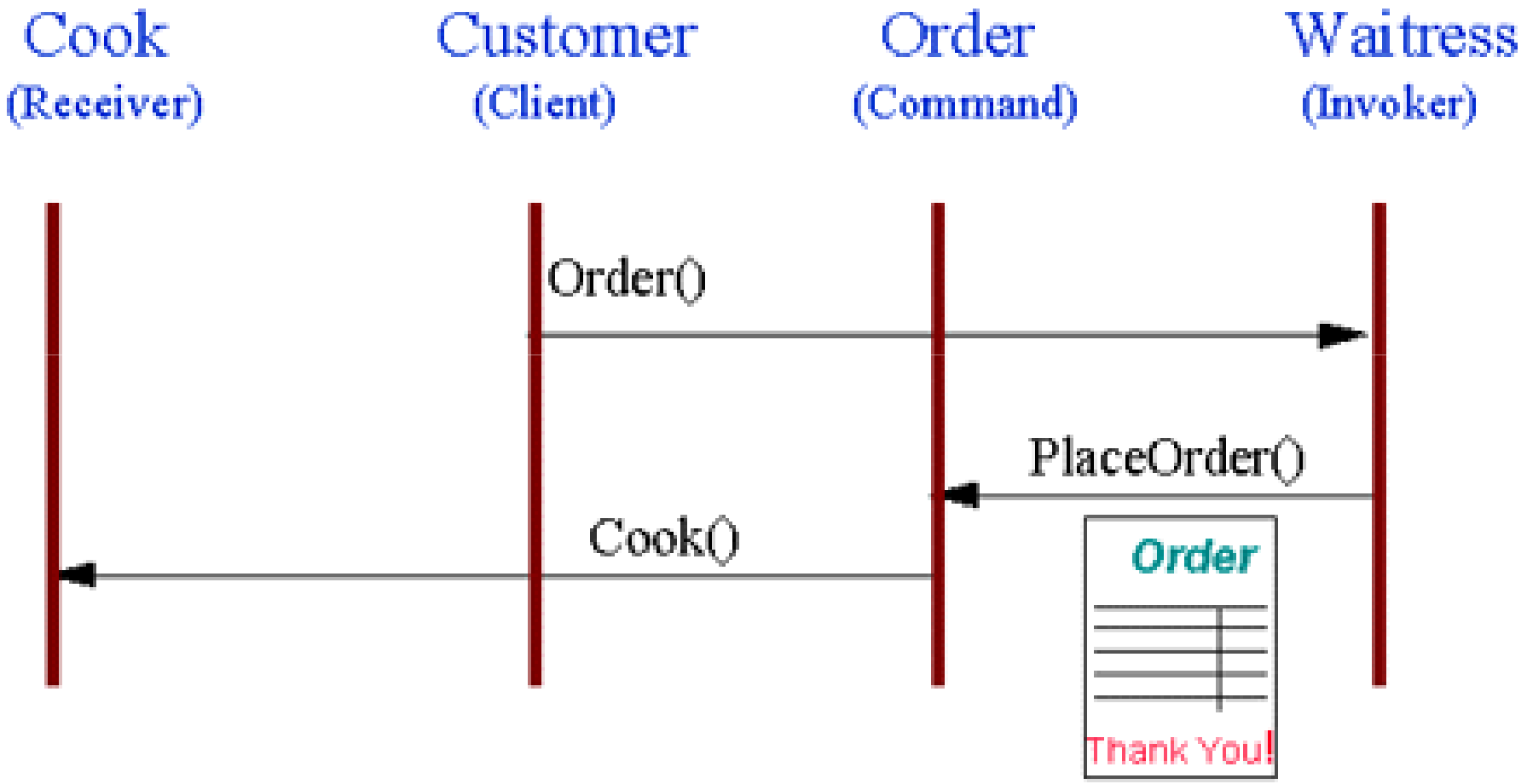
Štruktúra


cd štruktúra
-successor
Klient
Handler
+ HandleRequest()
ConcreteHandler1
+ HandleRequest()
ConcreteHandler2
+ HandleRequest()

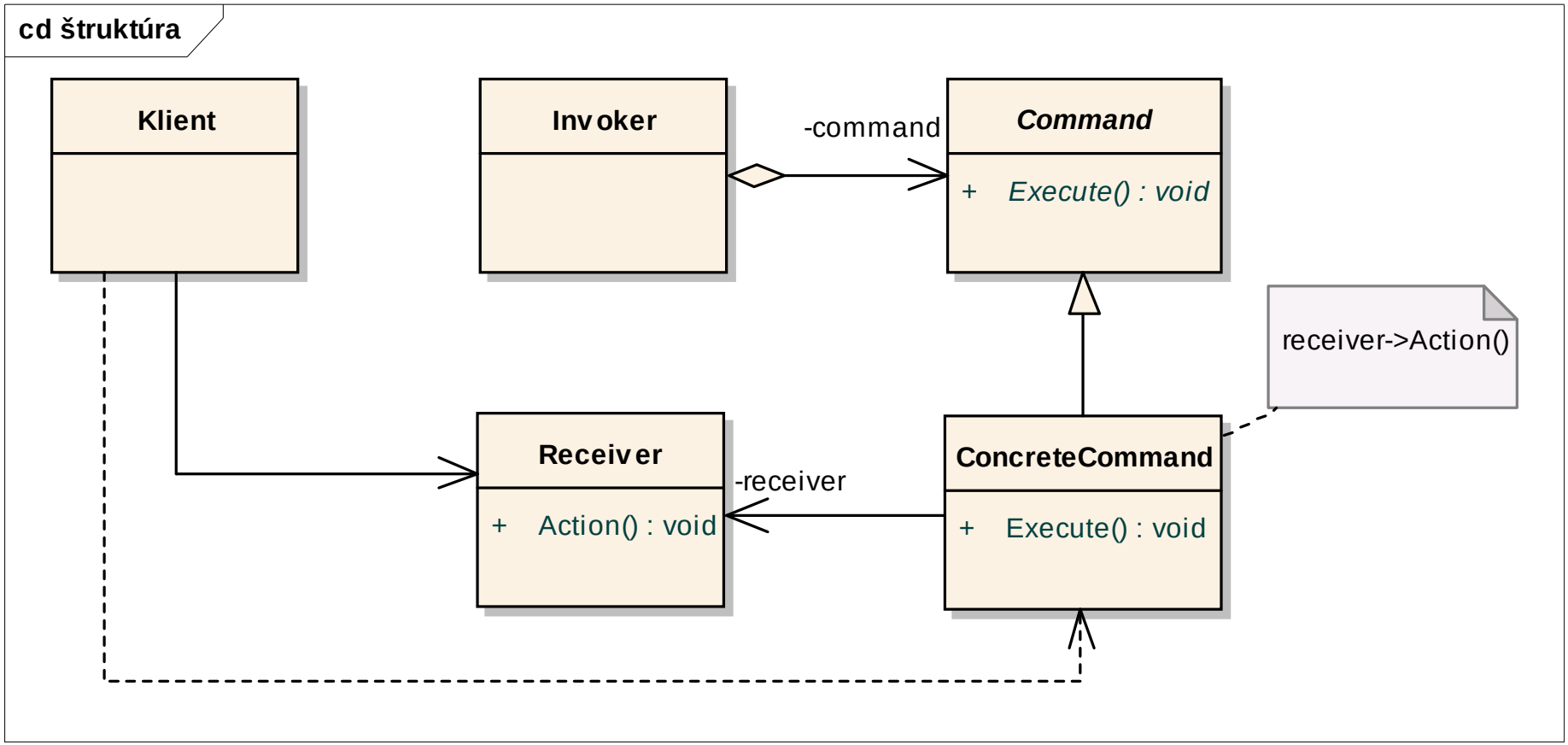
Použitie

- Flexibilita a dynamicky tvorené zreťazenia
- Zjednodušuje väzby medzi objektmi
- Chybné zreťazenia

Command - Action, Transaction



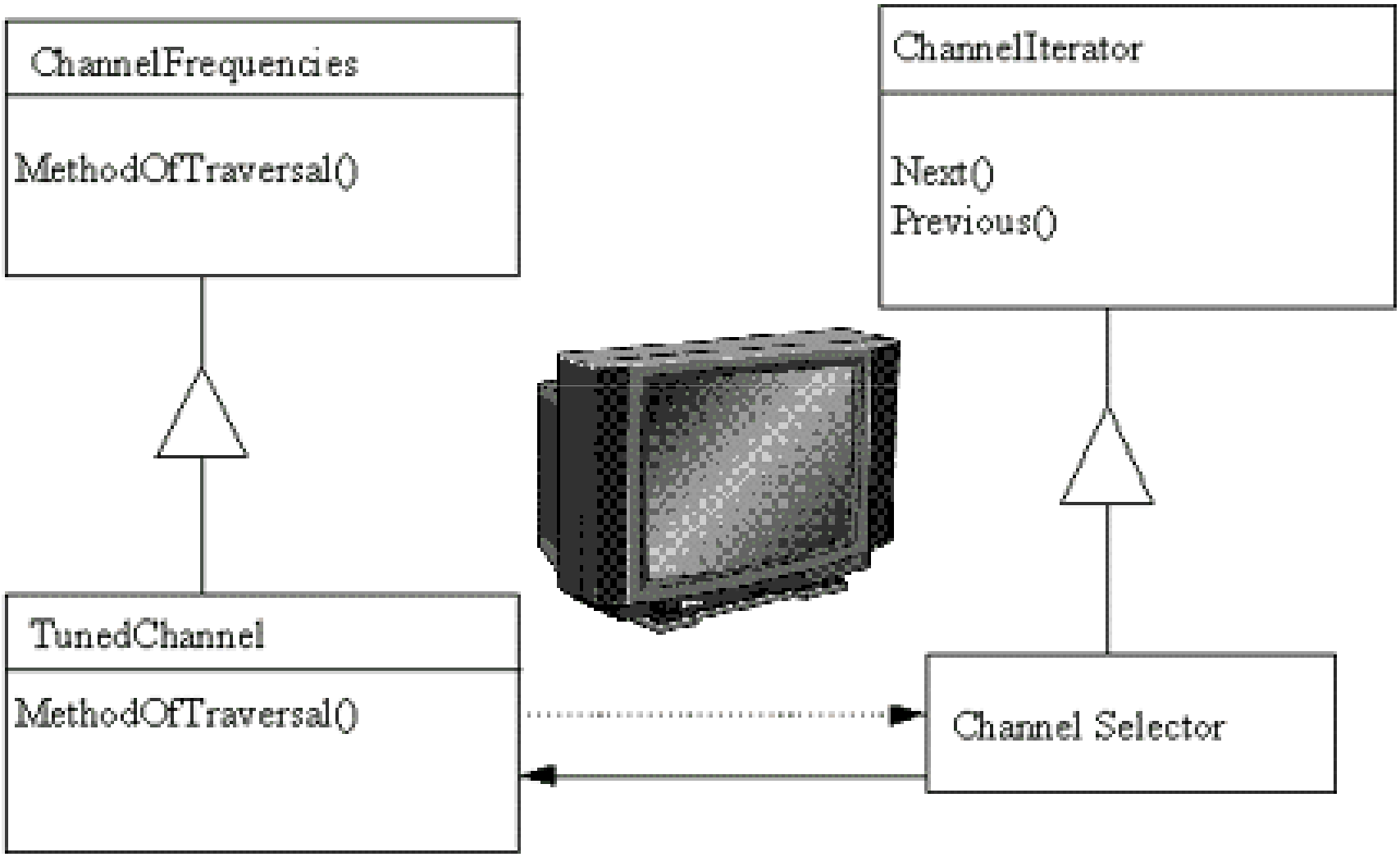
Štruktúra


cd štruktúra
Klient
Invoker
-command
Command
+ Execute() : void
receiver->Action()
Receiver
+ Action() : void
-receiver
ConcreteCommand
+ Execute() : void

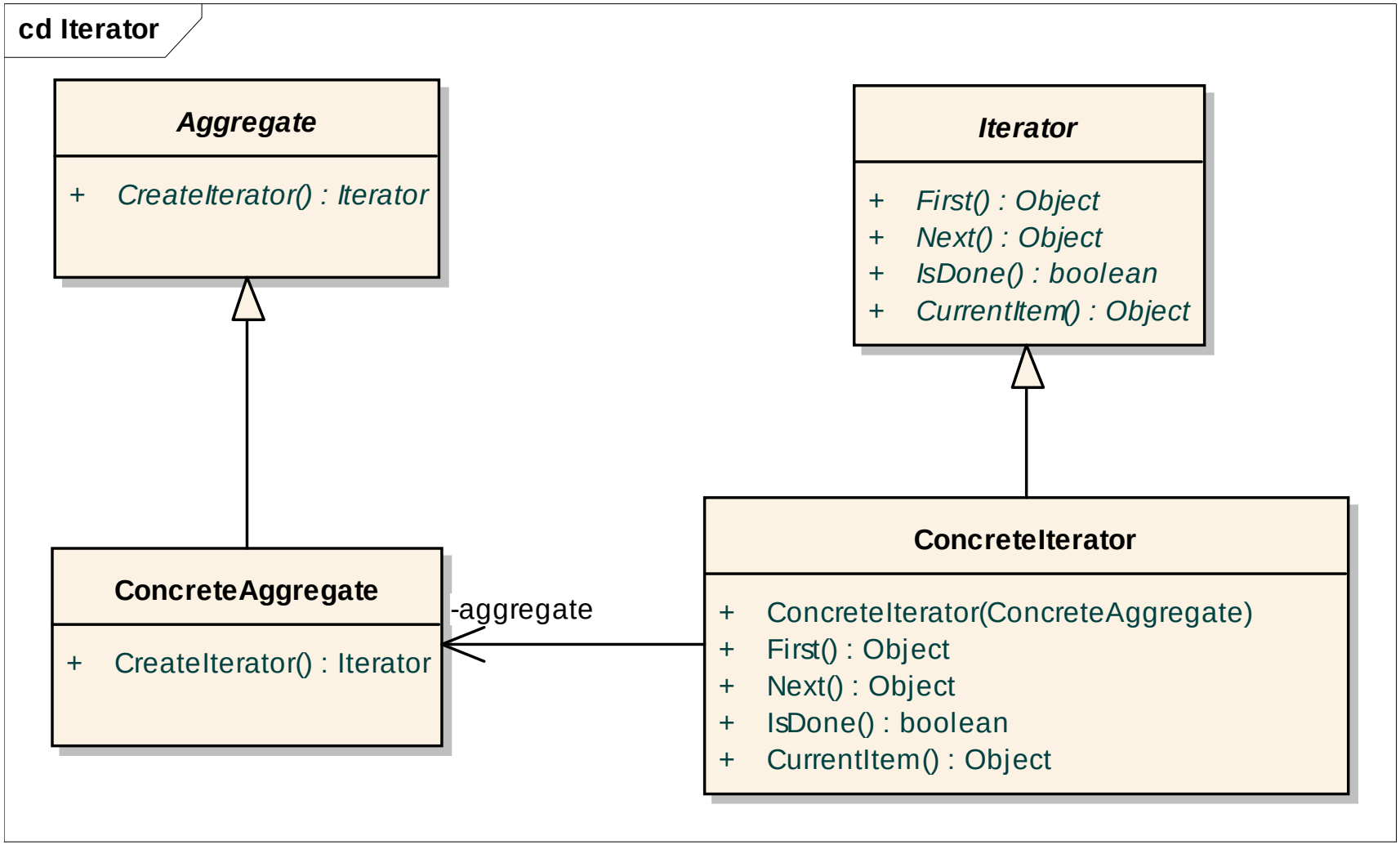
Výhody

- Umožňuje flexibilne pridávať nové triedy Command
- Možnosť pracovať s Commandami ako s premennými
- Oddeľuje klienta od realizácie požiadavky
- Možnosť vytvárať „MacroCommand“

Iterator - Cursor



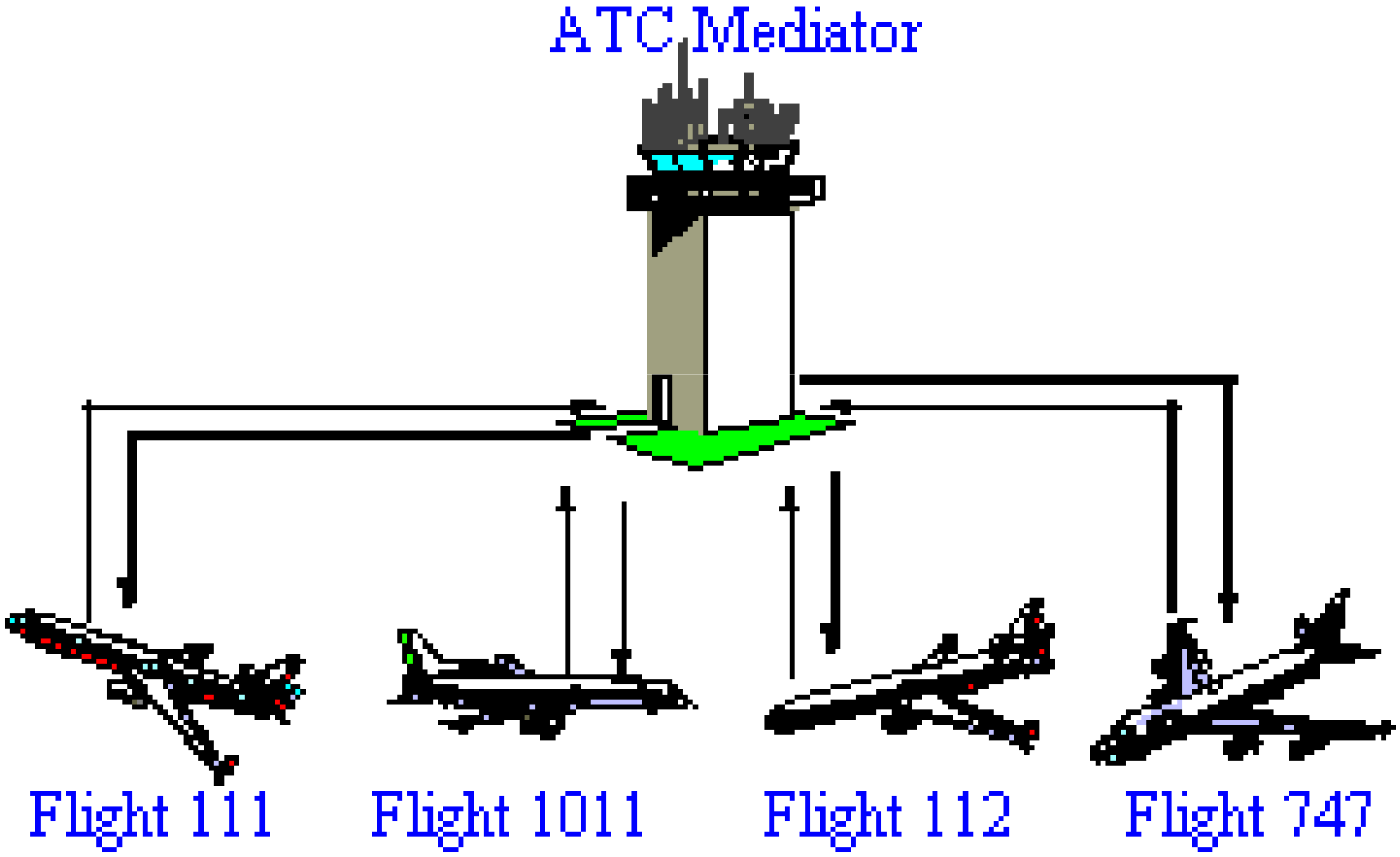
Štruktúra


cd Iterator
Aggregate
+ CreateIterator() : Iterator
Iterator
+ First() : Object
+ Next() : Object
+ IsDone() : boolean
+ CurrentItem() : Object
ConcreteAggregate
+ CreateIterator() : Iterator
-aggregate
ConcreteIterator
+ ConcreteIterator(ConcreteAggregate)
+ First() : Object
+ Next() : Object
+ IsDone() : boolean
+ CurrentItem() : Object

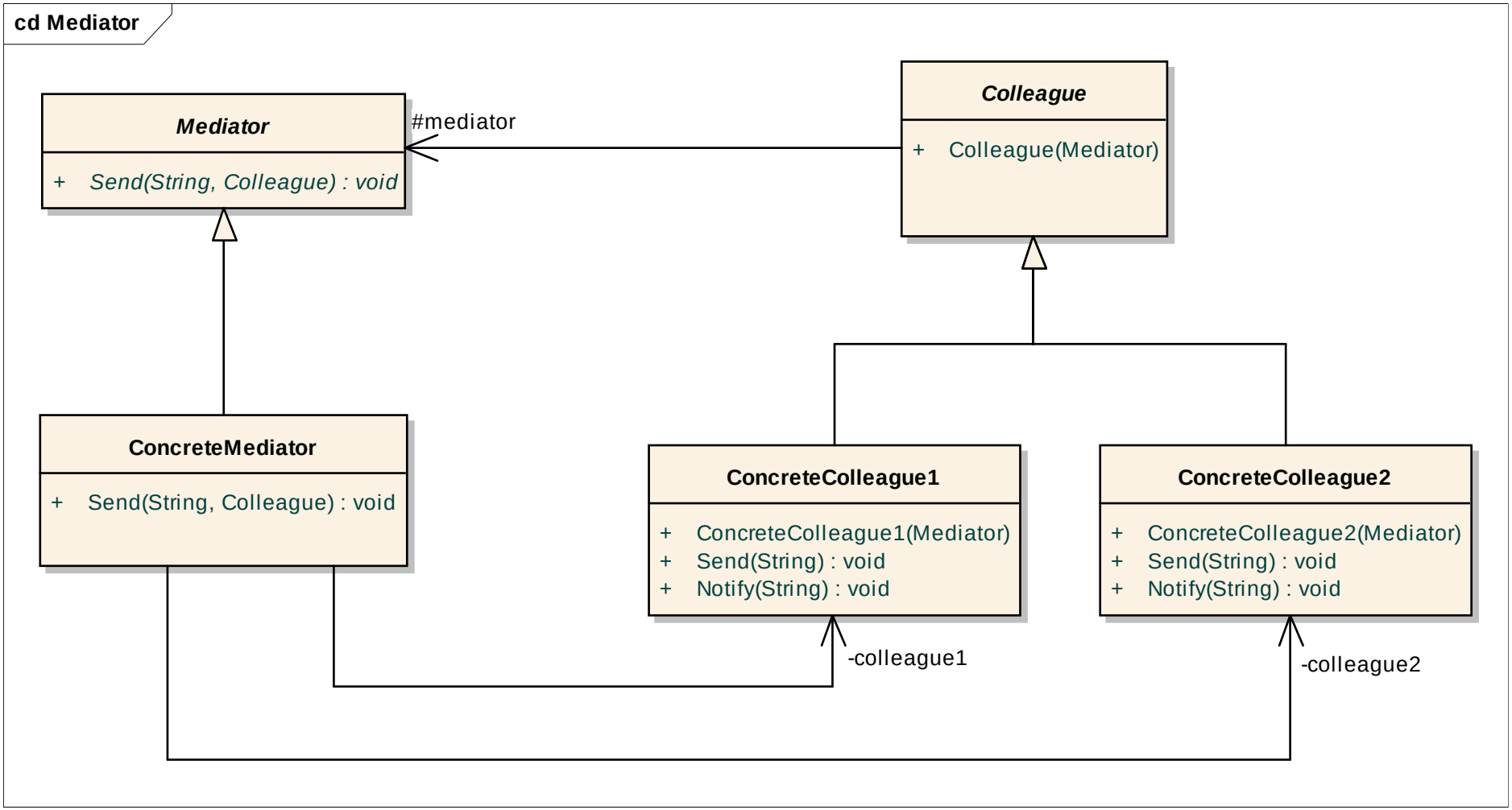
Použitie

- Chceme ukryť zložitú štruktúru objektu
- Násobné kurzory
- Polymorfný prechod rôznymi štruktúrami
- Môže podporovať aj iné metódy
- Pozor na zmeny v štruktúre
- Kto ovláda iterácie
- Kto definuje iteračný algoritmus

Mediator - Controller



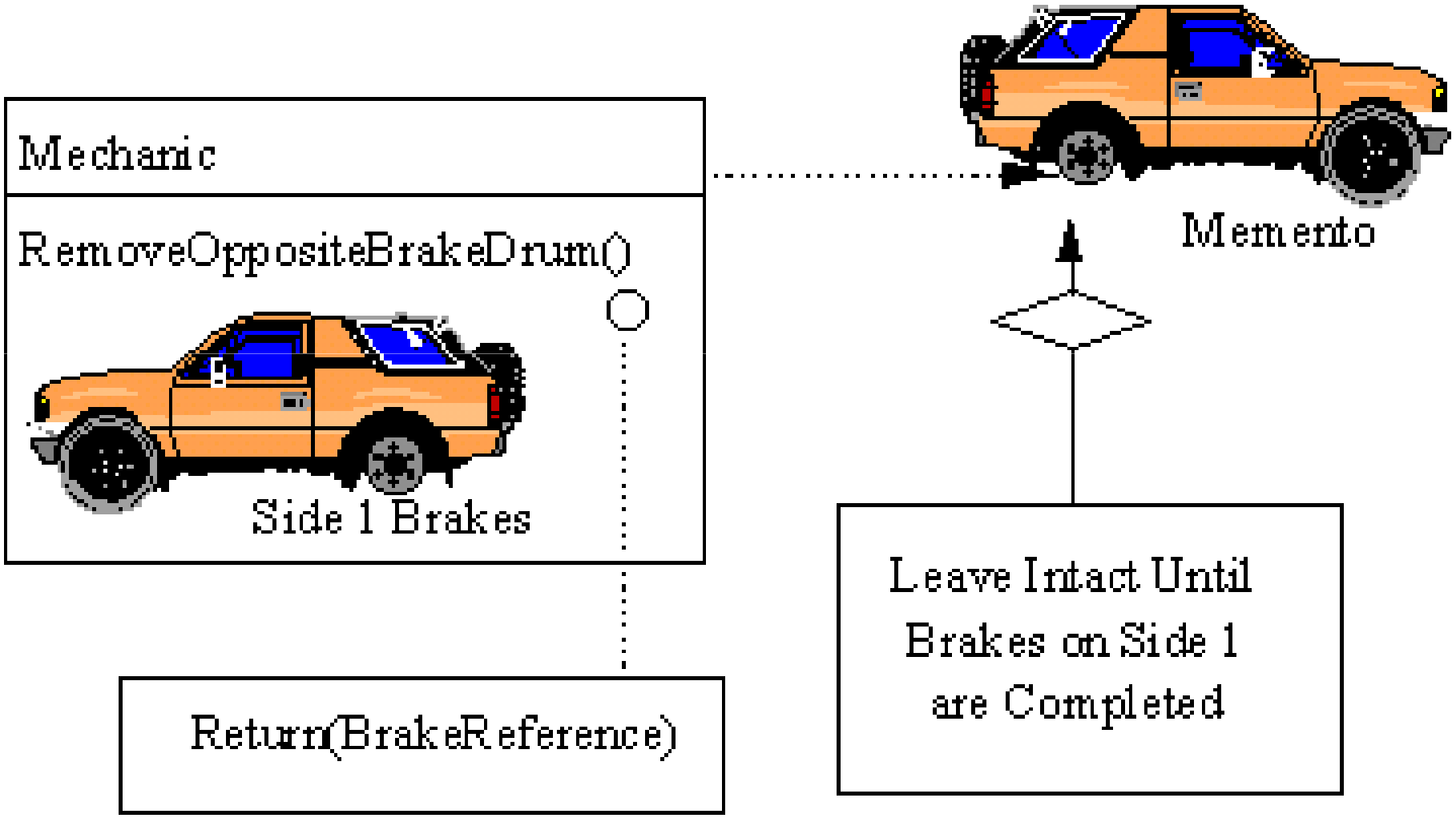
Štruktúra


cd Mediator
Mediator
+ Send(String, Colleague) : void
#mediator
Colleague
+ Colleague(Mediator)
ConcreteMediator
+ Send(String, Colleague) : void
ConcreteColleague1
+ ConcreteColleague1(Mediator)
+ Send(String) : void
+ Notify(String) : void
ConcreteColleague2
+ ConcreteColleague2(Mediator)
+ Send(String) : void
+ Notify(String) : void
-colleague1
-colleague2

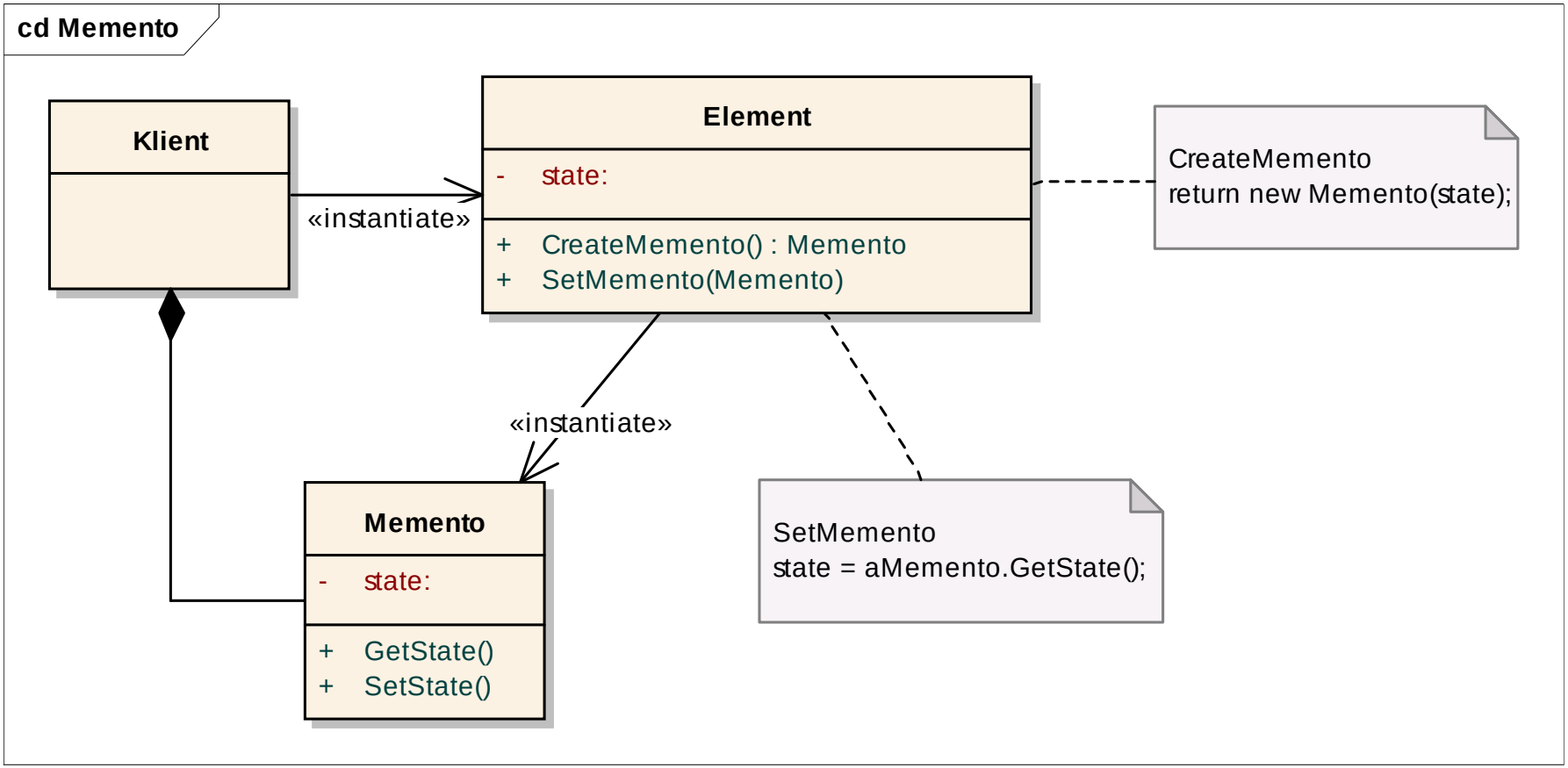
Použitie

- Udalosti
- Limituje dedičnosť a prepojenie medzi objektmi
- Zjednodušuje protokol
- Vynechanie abstraktnej triedy Mediator

Memento - Token



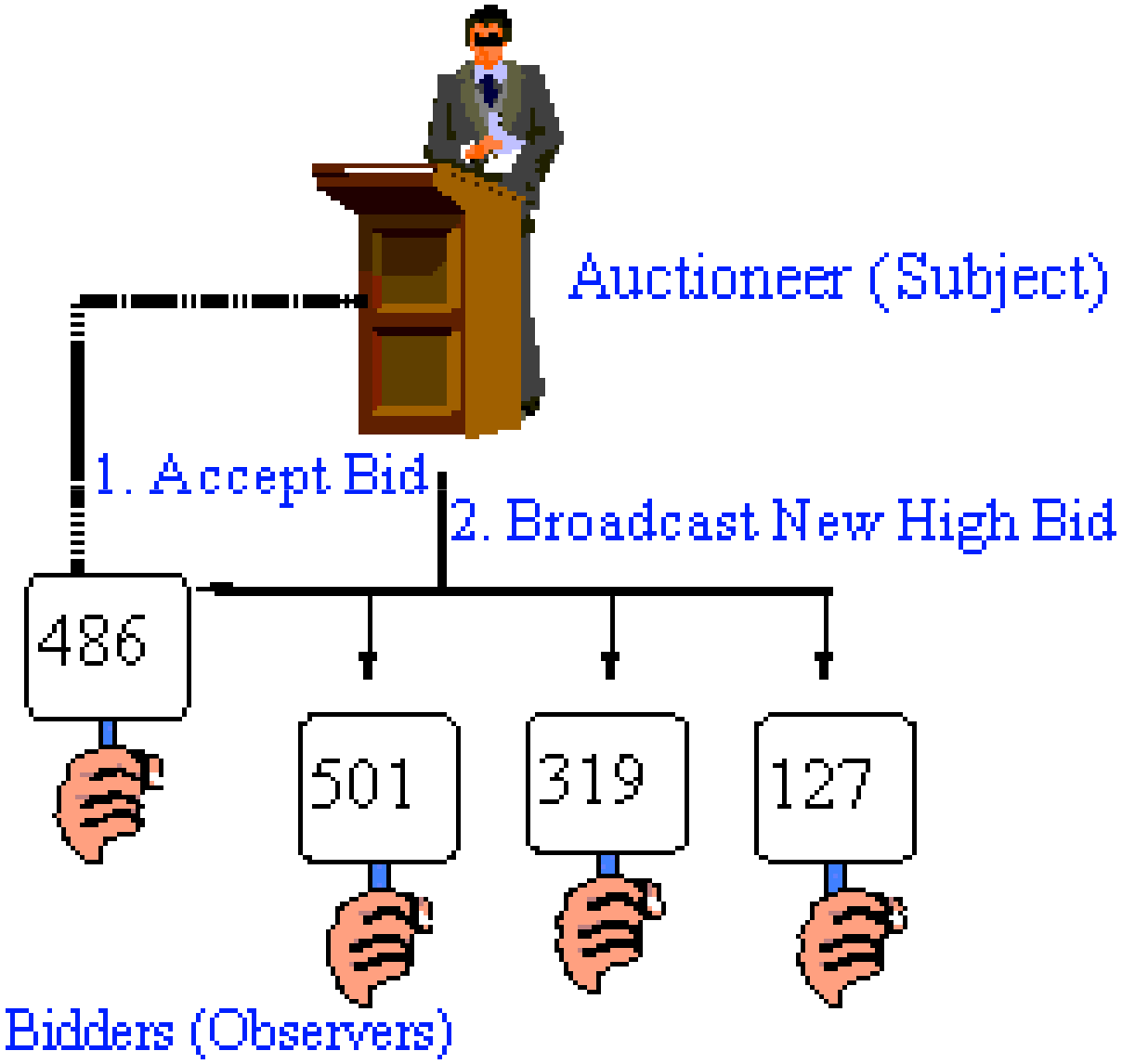
Štruktúra


cd Memento
Klient
Element
- state:
+ CreateMemento() : Memento
+ SetMemento(Memento)
«instantiate»
CreateMemento
return new Memento(state);
«instantiate»
Memento
- state:
+ GetState()
+ SetState()
SetMemento
state = aMemento.GetState();

Použitie

- Zapuzdrenie
- Klient aj Element sú jednoduchší
- Nečakané problémy u klienta
- Inkrementálne memento
- Command

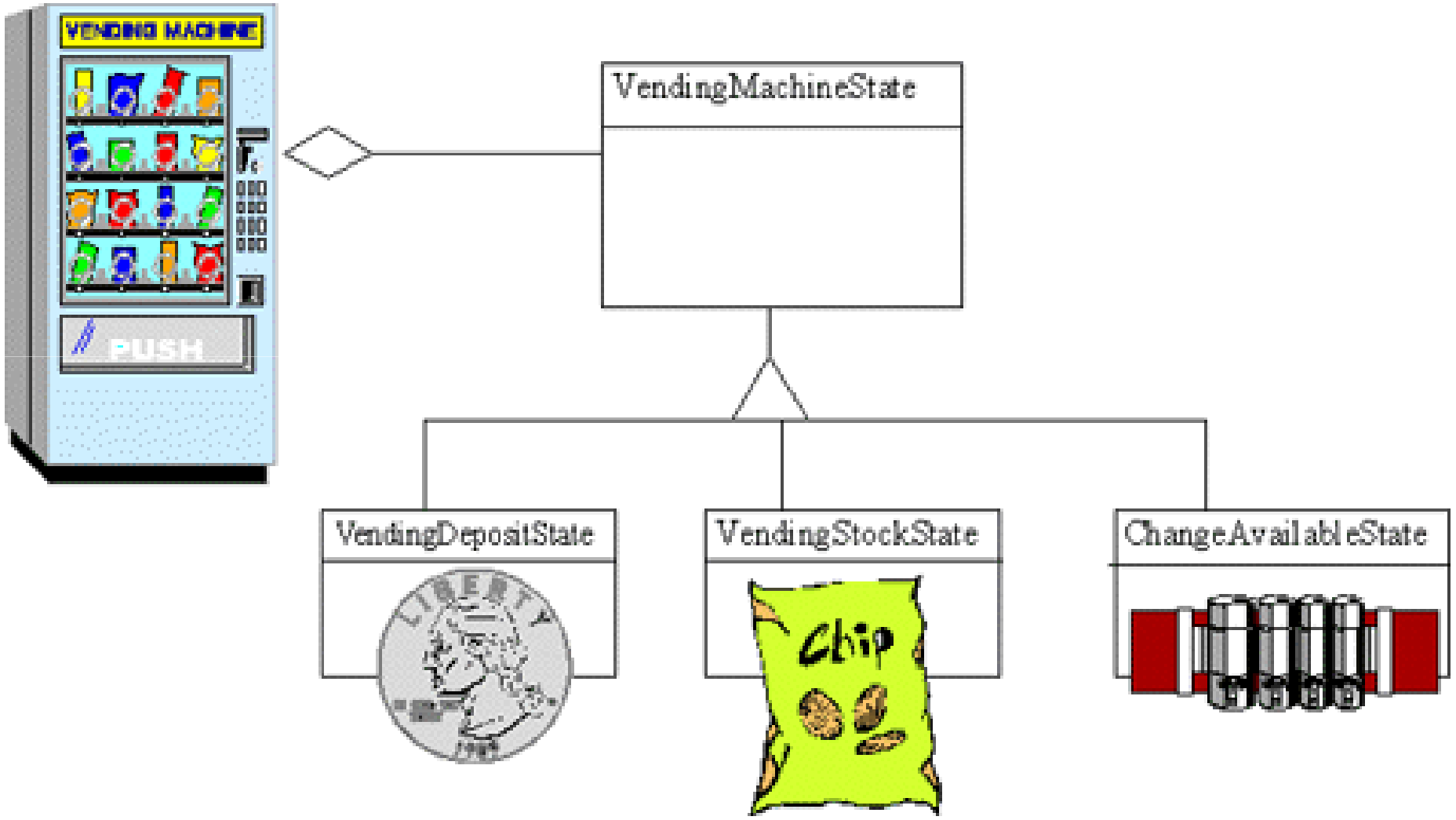
Observer - Dependents, Publish-Subscribe



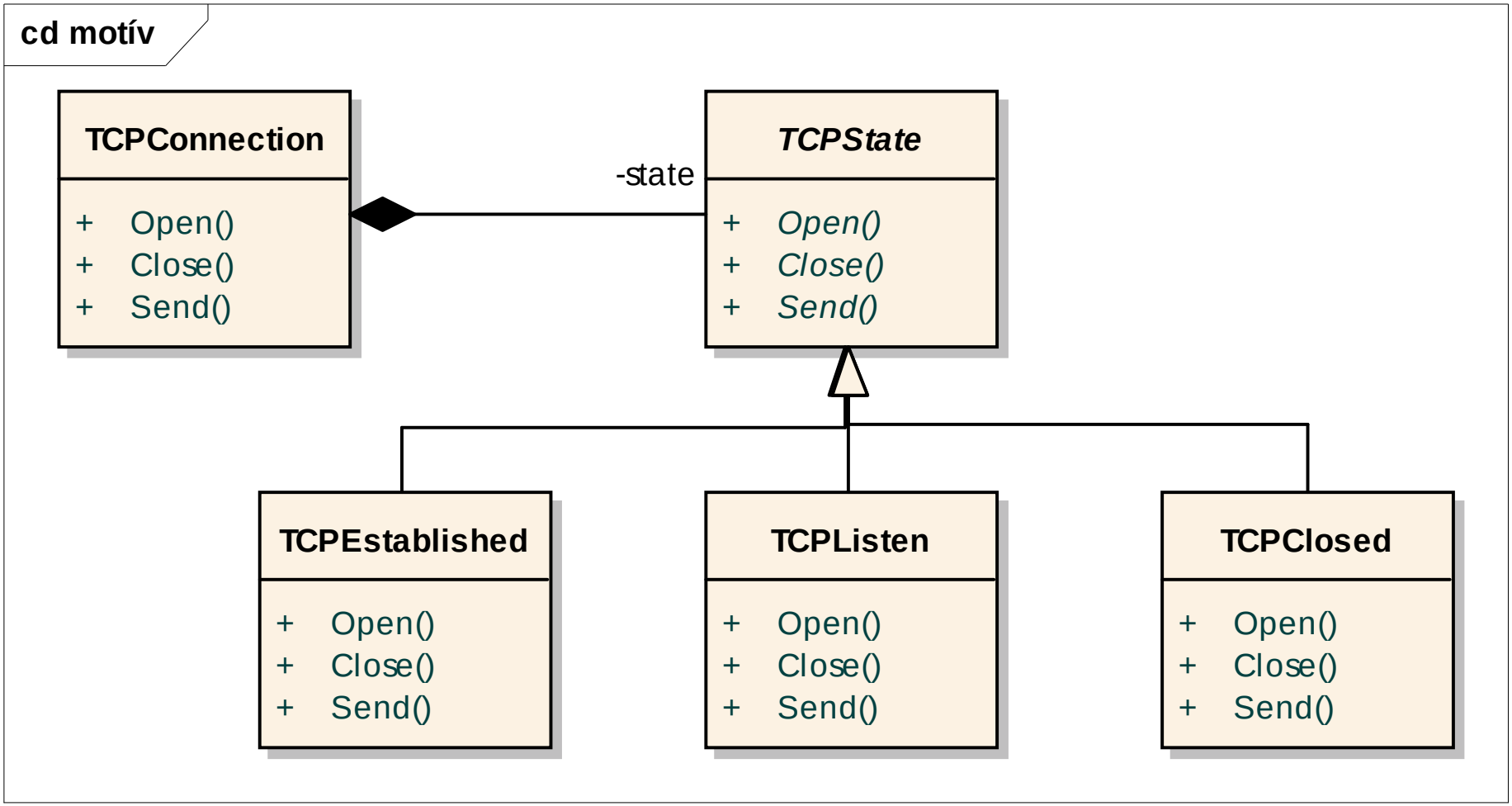
Použitie

- Odtienenie a nepriame volanie objektov
- Pozor na príliš veľkú flexibilitu
- Zacyklený Update
- Pozor na konzistenciu stavu konkrétneho subjektu
- Vymazanie observera

State


VENDING MACHINE
PUSH
VendingMachineState
VendingDepositState
VendingStockState
ChangeAvailableState
LIBERTY
chip

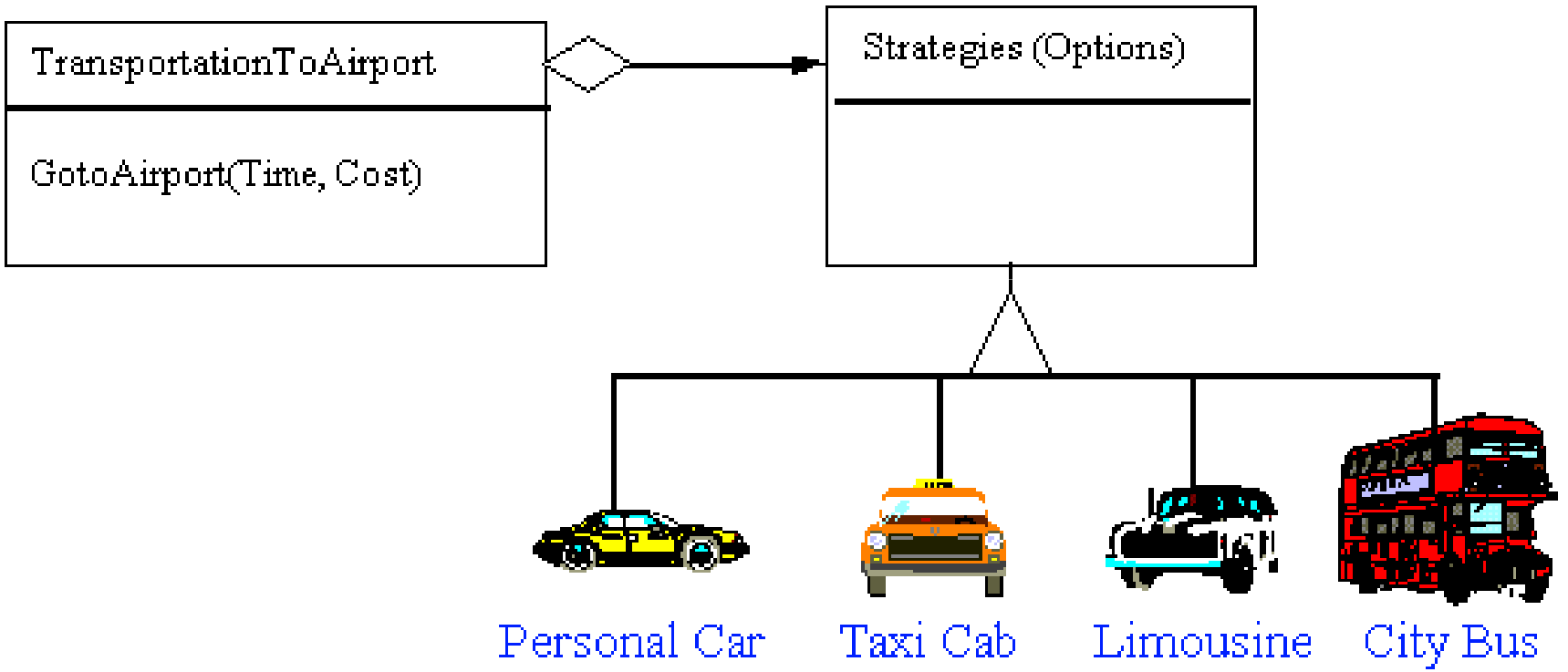
Motív


cd motív
TCPConnection
+ Open()
+ Close()
+ Send()
-state
TCPState
+ Open()
+ Close()
+ Send()
TCPEstablished
+ Open()
+ Close()
+ Send()
TCPListen
+ Open()
+ Close()
+ Send()
TCPClosed
+ Open()
+ Close()
+ Send()

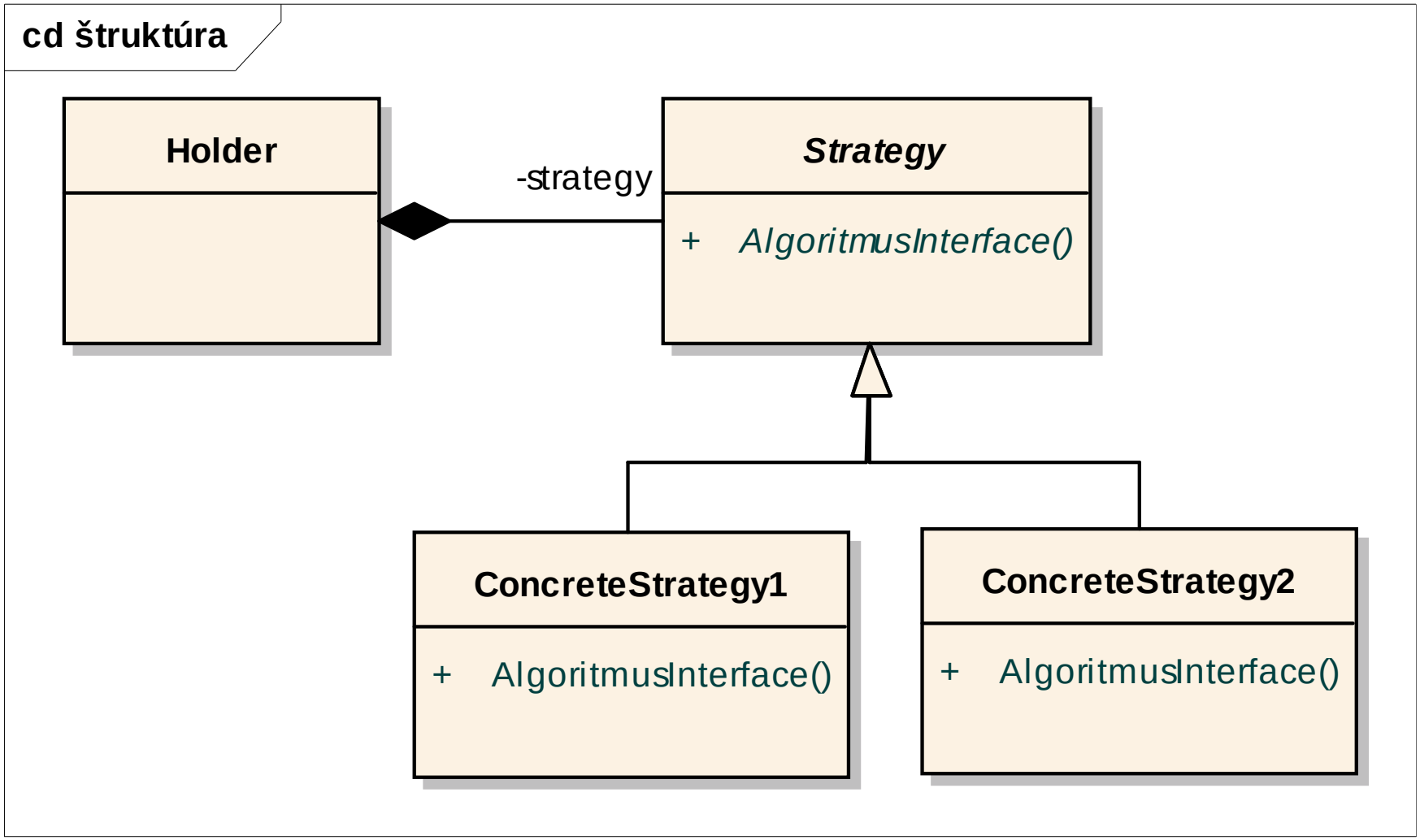
Použitie

- Prehľadnosť a flexibilita
- Vyššia konzistencia
- Riešenie prechodov
- Stavové veličiny, atribúty
- Vytváranie a rušenie objektov stavu

Strategy - Policy



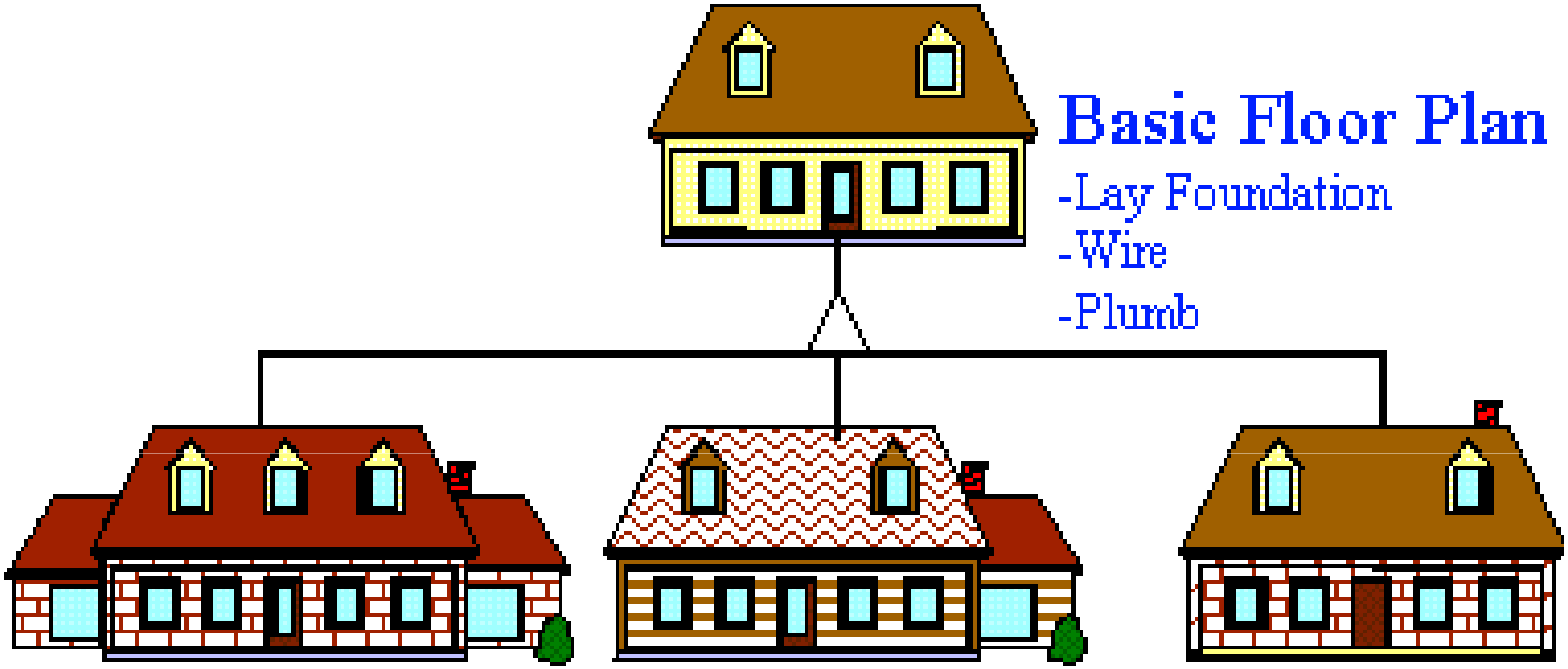
Štruktúra


cd štruktúra
Holder
-strategy
Strategy
+ AlgoritmusInterface()
ConcreteStrategy1
+ AlgoritmusInterface()
ConcreteStrategy2
+ AlgoritmusInterface()

Použitie

- Rodiny súvisiacich algoritmov
- Dediči Holdra?
- Spätná väzba
- Klient by mal poznať rozdiely v Strategies

Template method



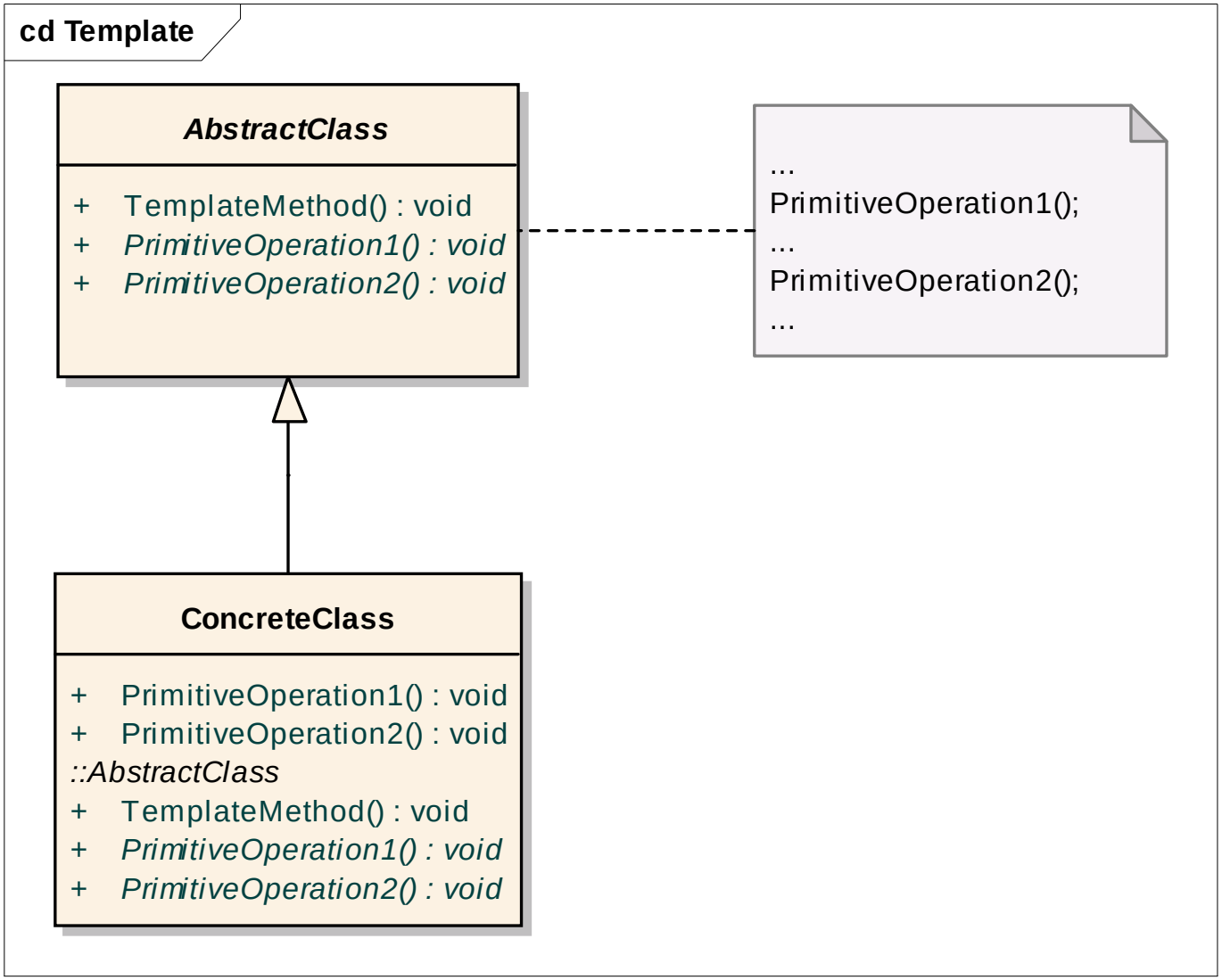
Variations added to Template Floor Plan

Klasifikácia a zmysel

- Class
- Behavioral

- Zavádza scenár na abstraktnej hornej úrovni predka zložený z niekoľkých polymorfných metód.

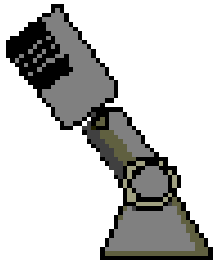
Štruktúra


cd Template
AbstractClass
+ TemplateMethod() : void
+ PrimitiveOperation1() : void
+ PrimitiveOperation2() : void
...
PrimitiveOperation1();
...
PrimitiveOperation2();
...
ConcreteClass
+ PrimitiveOperation1() : void
+ PrimitiveOperation2() : void
::AbstractClass
+ TemplateMethod() : void
+ PrimitiveOperation1() : void
+ PrimitiveOperation2() : void

Použitie

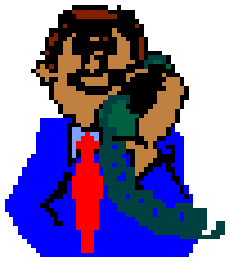
- Výhoda presne definovaných postupov
- Všetko obsahujúce scenáre
- Volanie zdedenej metódy
- Minimalizácia primitívnych operácii

Visitor



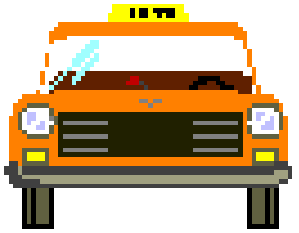
Cab Company Dispatcher

(Object Structure is List of Customers)



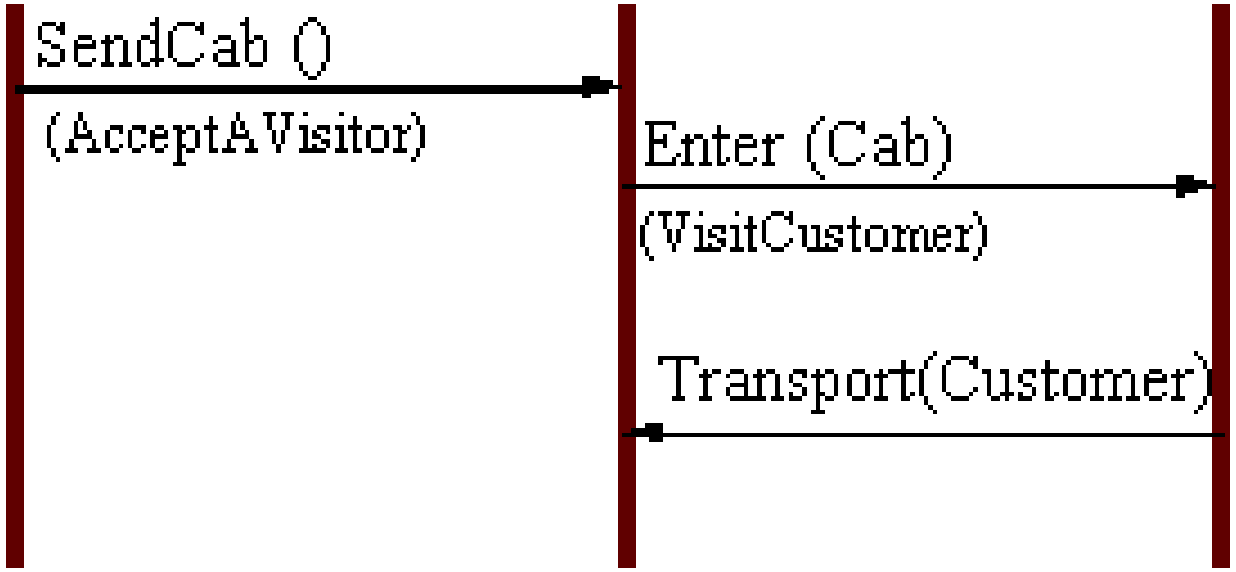
Customer

(Concrete Element of Customer List)



Taxi

(Visitor)



Dvojitý polymorfizmus

- Visit:
  - VisitKruh(Kruh kr)
  - VisitObdlznik(Obdlznik ob)
  - ...

```
void Kruh::Accept(Visitor aVis) {
      aVis->VisitKruh(this);
}

void Obrazec::Accept(Visitor aVis) {
      // pre kazdy podobrazec zavolaj
      item->Accept(aVis);
}
```

Použitie

- Flexibilita operácií, ne-flexibilita elementov
- Zberné Visitory