Projektni obrasci

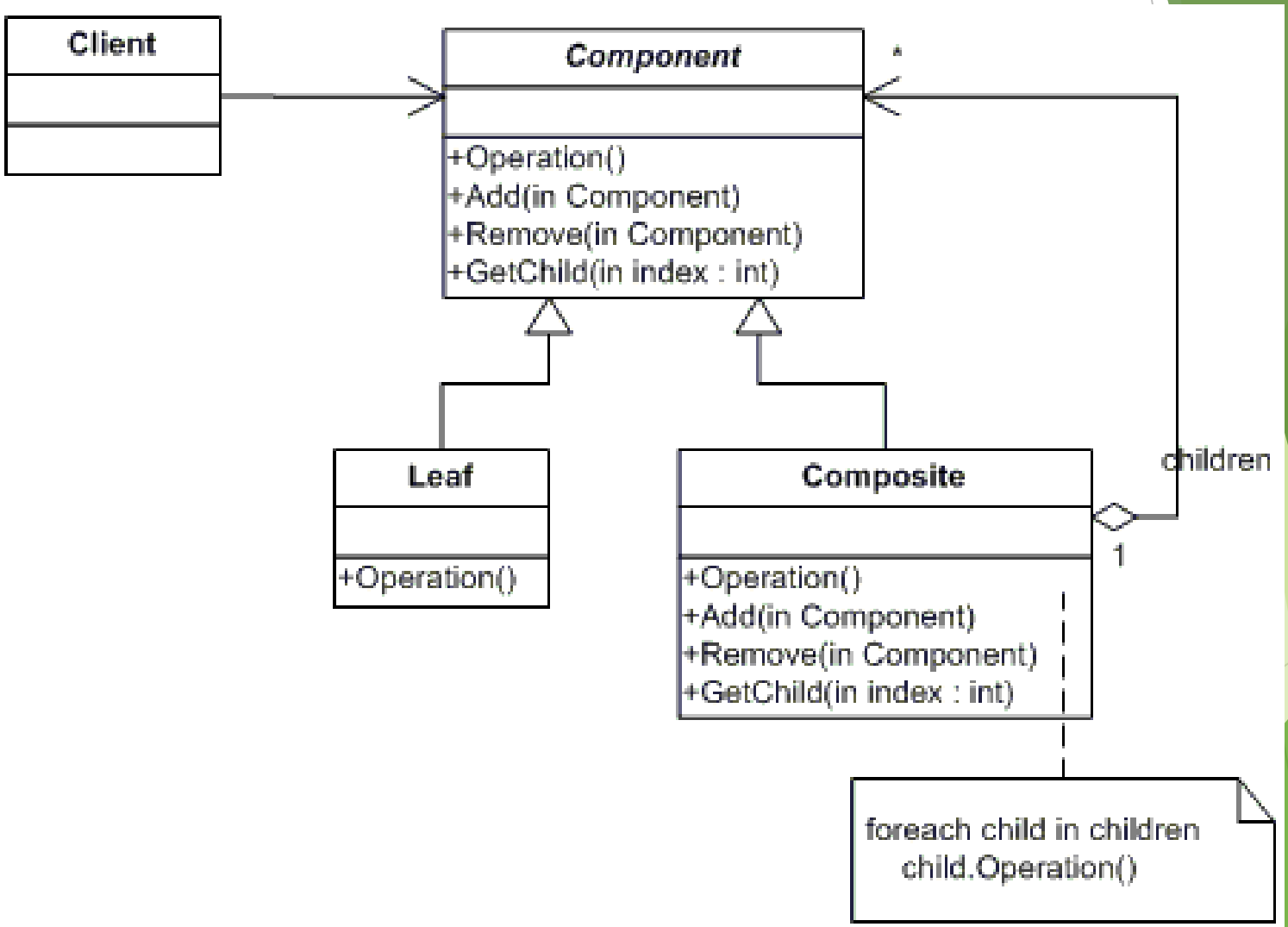
Composite – Decorator – Abstract Factory

Composite

Definicija

- Uređuje objekte u hijerarhijsku strukturu stabla. Obrazac Composite dozvoljava klijentu da i posebne objekte i kompozicije tretira na isti način.
- composite.cs

UML Dijagram klasa

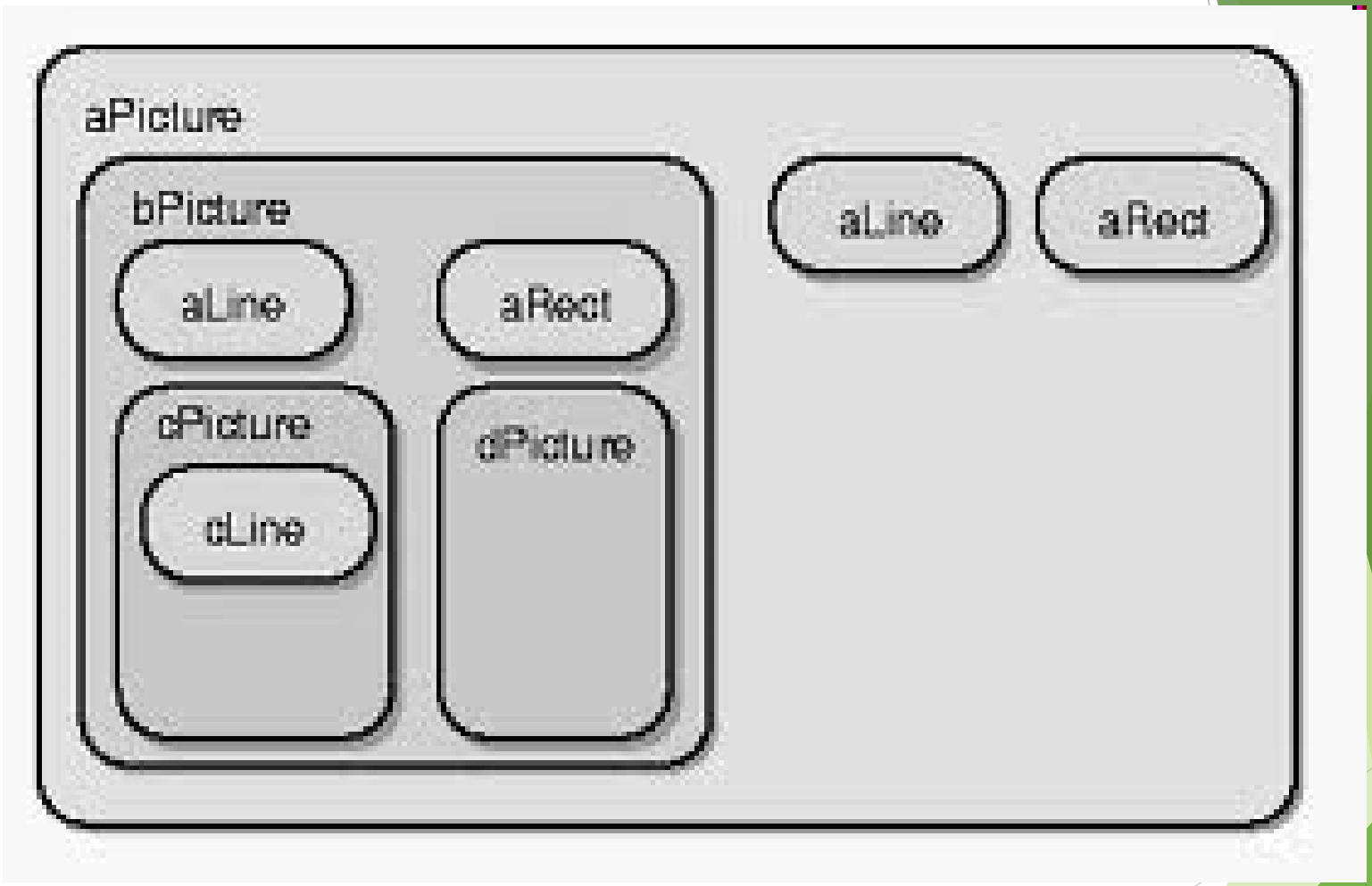


Elementi

- **Component (DrawingElement)**
  - deklariše interfejs za objekte koji mogu da čine kompoziciju
  - implementira osnovno ponašanje za interfejs zajednički svim klasama
  - deklariše interfejs za pristup i manipulaciju njegovim podelementima (child components)
  - (opciono) definiše i implementira interfejs za pristup roditeljskom elementu u rekurzivnim strukturama
- **Leaf (PrimitiveElement)**
  - predstavlja objekat koji učestvuje u kompoziciji. Ovako se predstavljaju tzv. listovi (elementi koji nemaju podelemente)
  - definiše ponašanje za primitivne objekte u kompoziciji
- **Composite (CompositeElement)**
  - definiše ponašanje elemenata koji mogu da imaju podelemente
  - čuva podelemente
  - implementira operacije koje se odnose na podelemente u interfejsu klase Component
- **Client (CompositeApp)**
  - manipuliše objektima kompozicije kroz interfejs koji obezbeđuje klasa Component

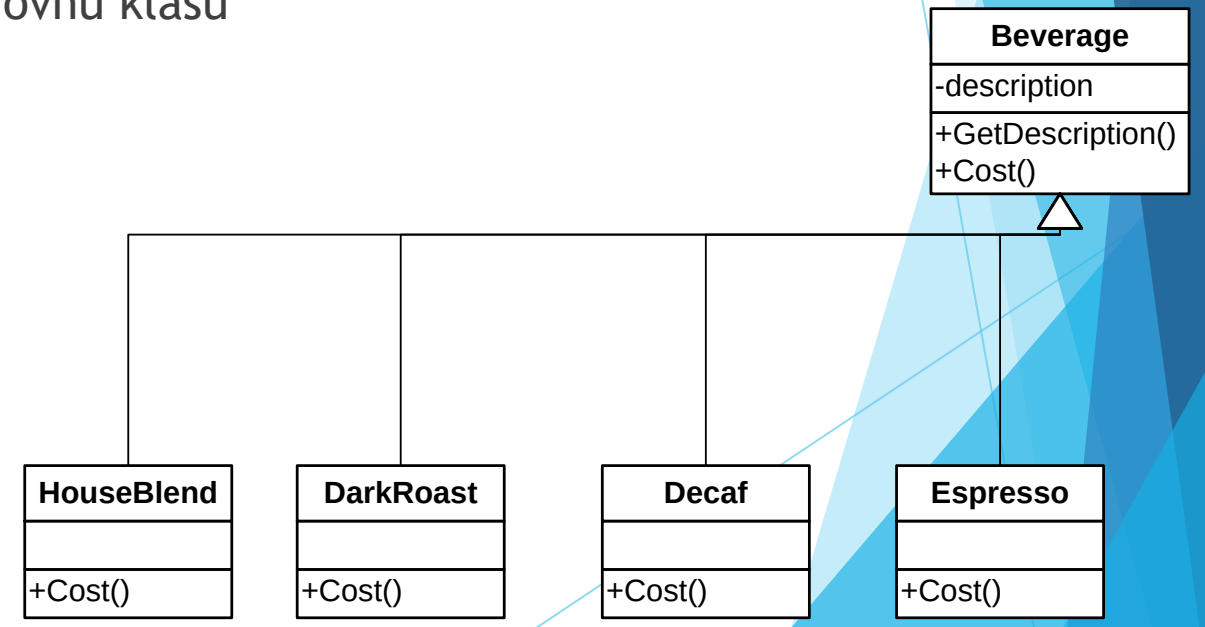
Primer iz stvarnog sveta

- Ovaj primer iz stvarnog sveta prikazuje ulogu obrasca Composite u kreiranju grafičke strukture stabla koja je sačinjena od primitivnih elemenata (linija, krugova) i složenih čvorova (grupe elemenata za crtanje od kojih se prave složeniji elementi).
- compositeRW.cs


aPicture
bPicture
aLine
aRect
cPicture
cLine
dPicture
aLine
aRect

Coffee shop

- Kafedžinica prodaje samo kafe
- Svaka kafa je definisana svojim opisom i cenom
- Kad hoće da dodaju novu kafu dodaju jednu klasu koja nasleđuje njihovu osnovnu klasu
- I to radi

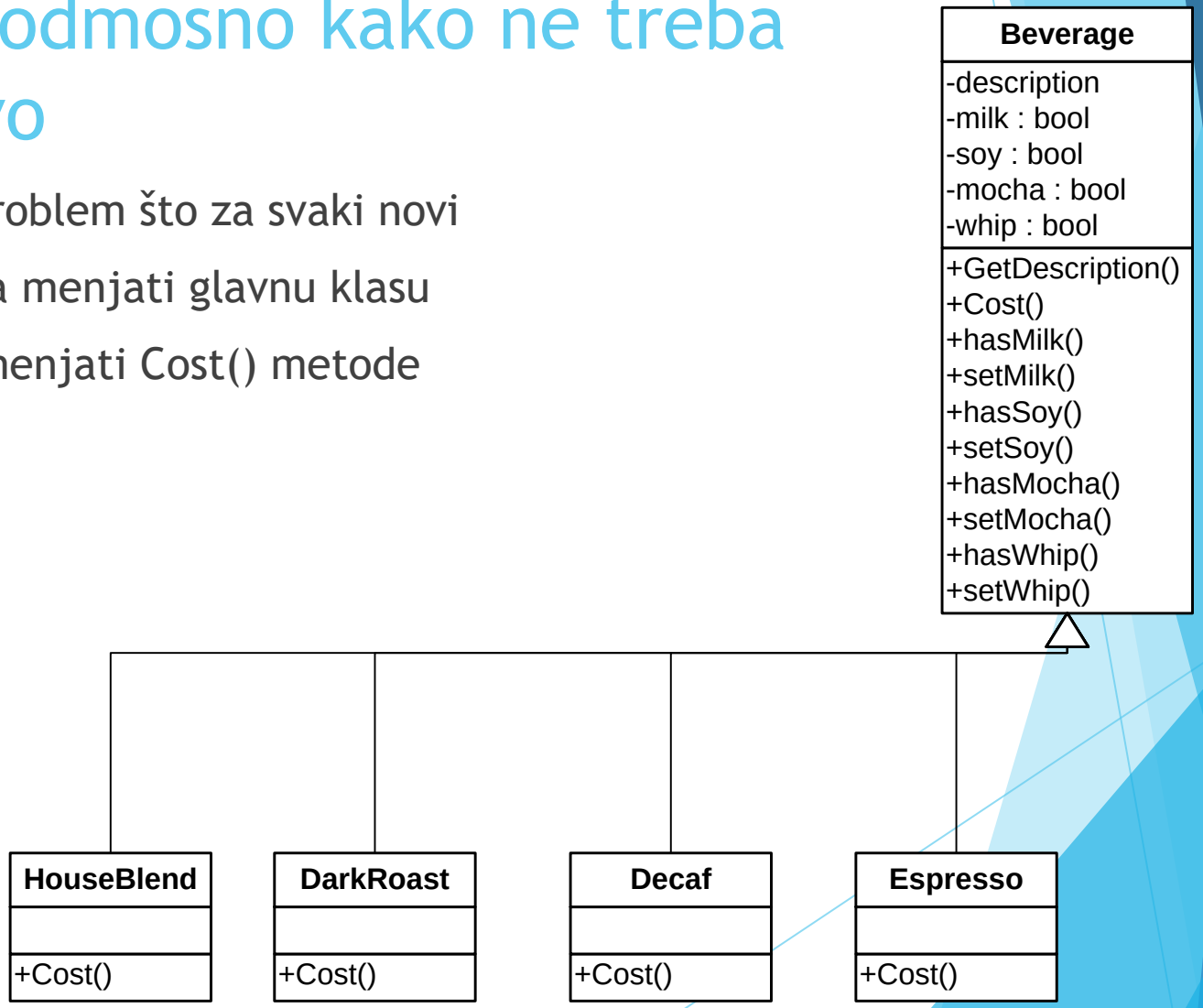


Coffe shop unapređuje posao

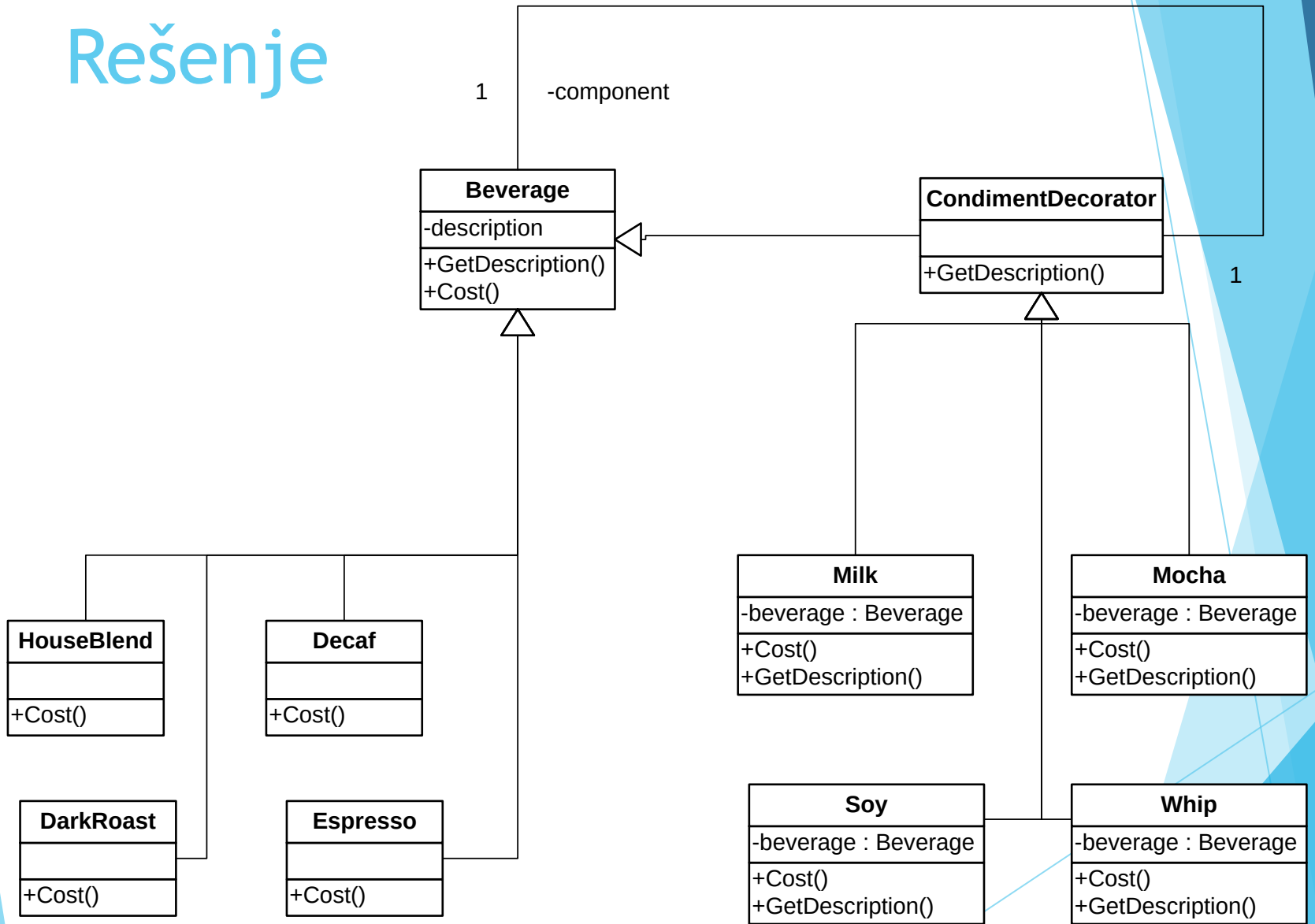
- Žele da uvedu dodatke na kafu koji ne mogu pojedinačno da se prodaju
- Dodaci povećavaju cenu kafi na koju se dodaju
- Na početku imaju 4 dodatka: mleko, sojino mleko, čokoladu i krem; a biće ih i više
- Šta raditi?
- Ako će da prave klasu za svaku moguću kombinaciju kafe i dodataka trebaće 64 klase
- Previše, a šta kad uvedu sledeći dodatak

Rešenje koje prvo padne na pamet, odmosno kako ne treba rešiti ovo

- Ovde je problem što za svaki novi dodatak treba menjati glavnu klasu u sistemu, i menjati Cost() metode



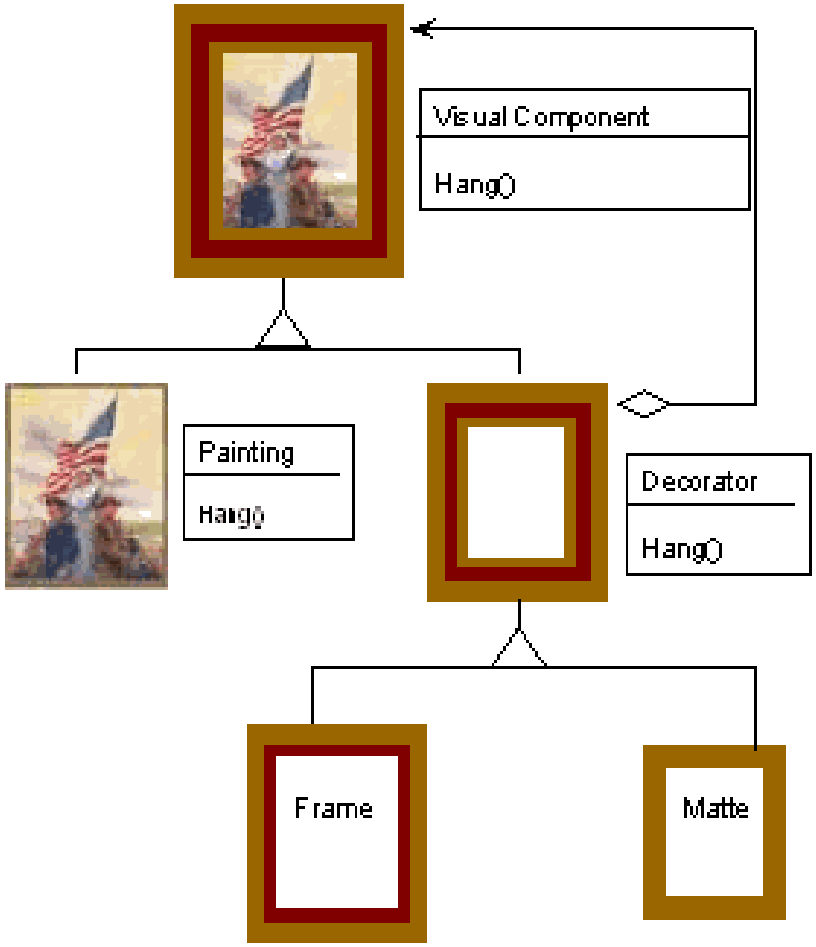
Rešenje


1
-component
Beverage
-description
+GetDescription()
+Cost()
CondimentDecorator
+GetDescription()
1
HouseBlend
+Cost()
Decaf
+Cost()
DarkRoast
+Cost()
Espresso
+Cost()
Milk
-beverage : Beverage
+Cost()
+GetDescription()
Mocha
-beverage : Beverage
+Cost()
+GetDescription()
Soy
-beverage : Beverage
+Cost()
+GetDescription()
Whip
-beverage : Beverage
+Cost()
+GetDescription()

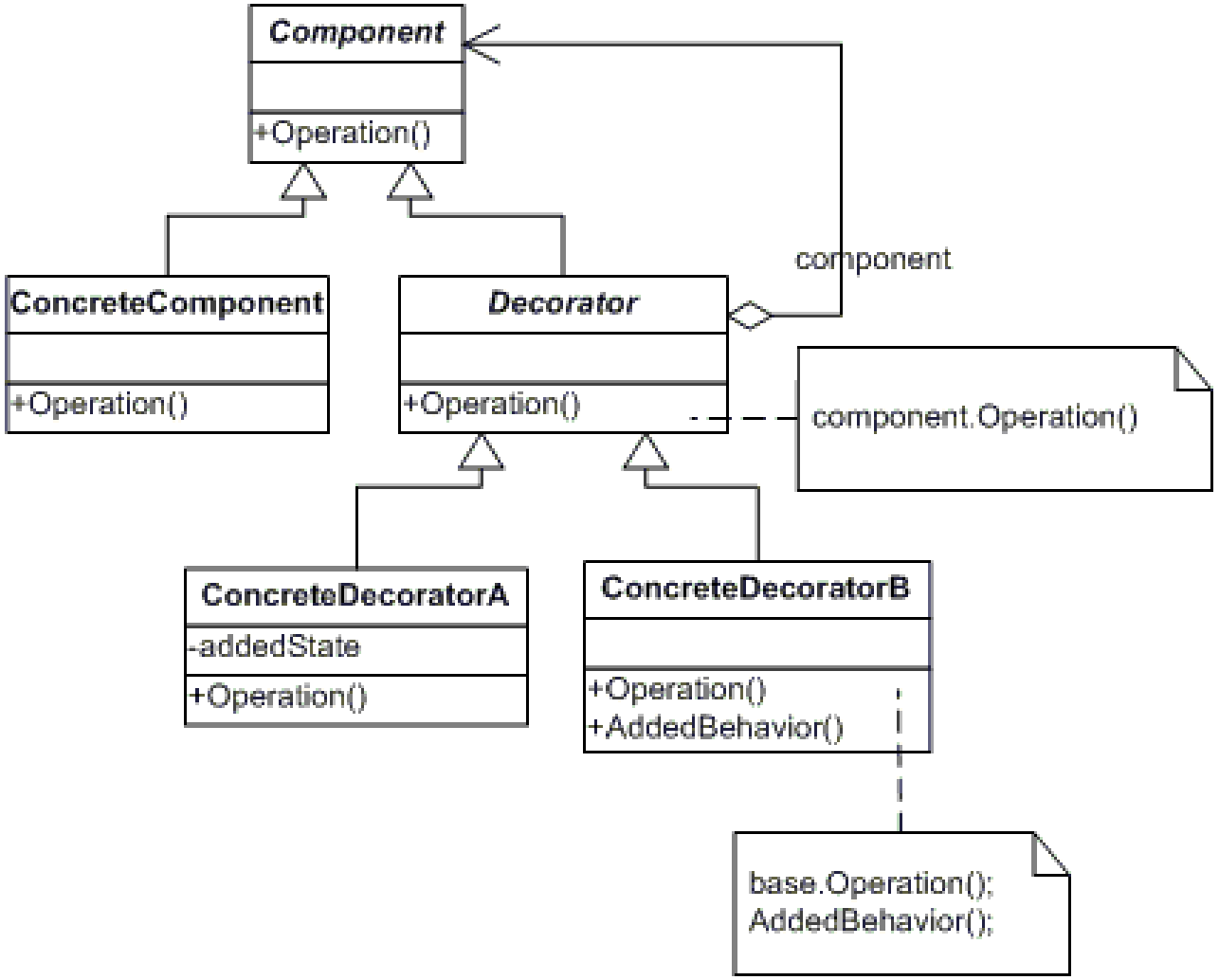
Obrazac Decorator

- Služi za dinamičko dodavanje novih funkcija nekom objektu. Obrazac Decorator također omogućava fleksibilan način za unapređenje funkcionalnosti već dodatih potklasa.
- decorator.cs

Decorator


Visual Component
Hang()
Painting
Hang()
Decorator
Hang()
Frame
Matte

UML dijagram klasa

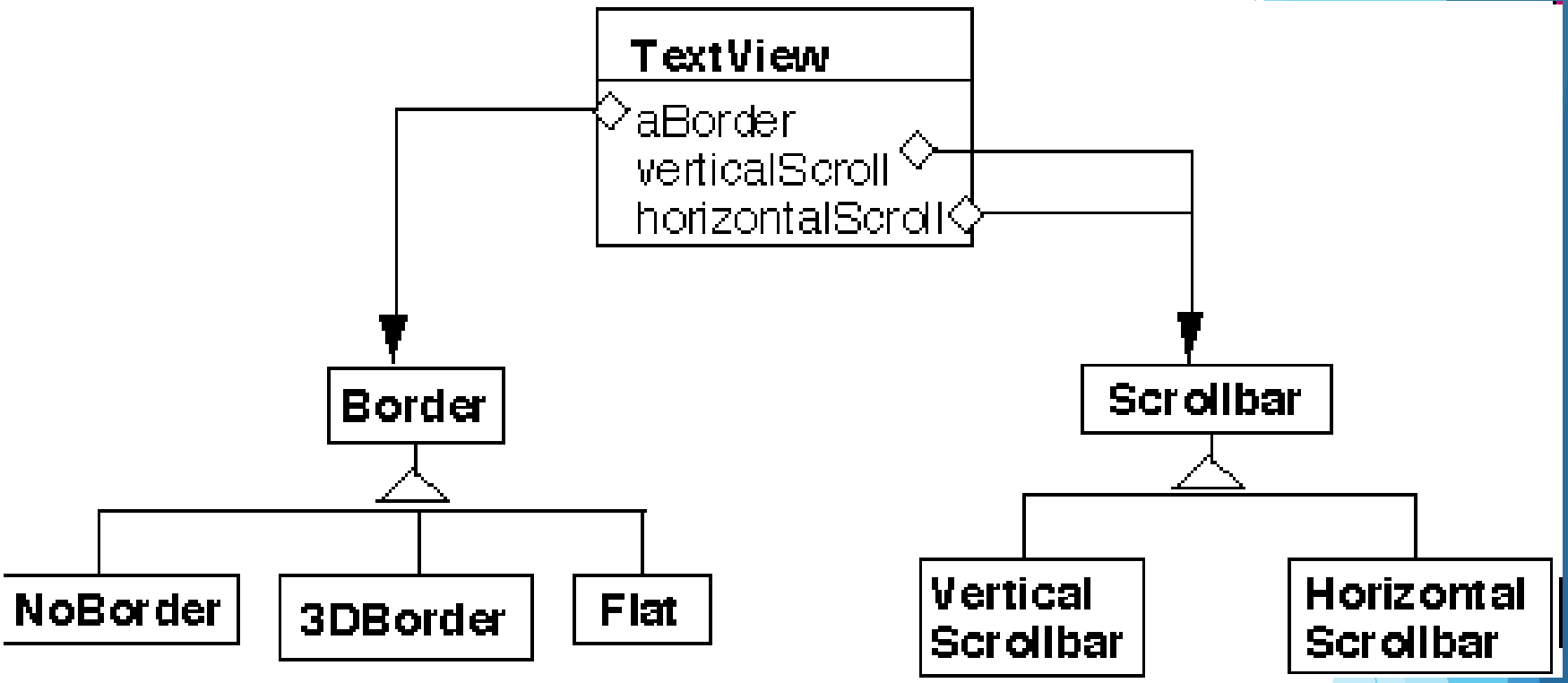


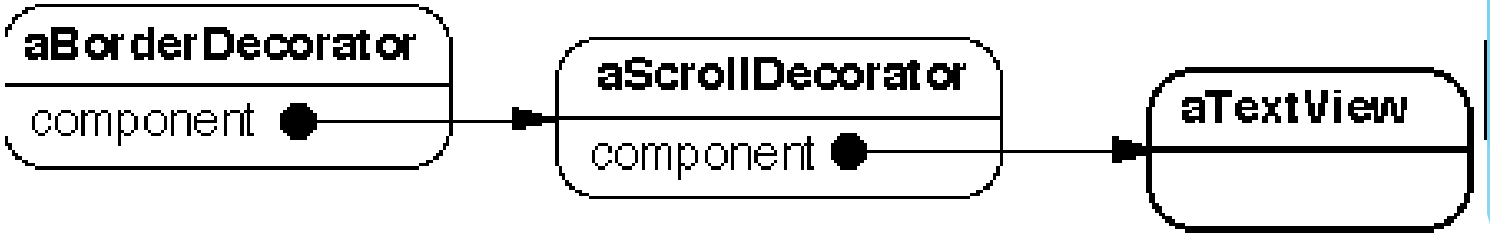
Elementi

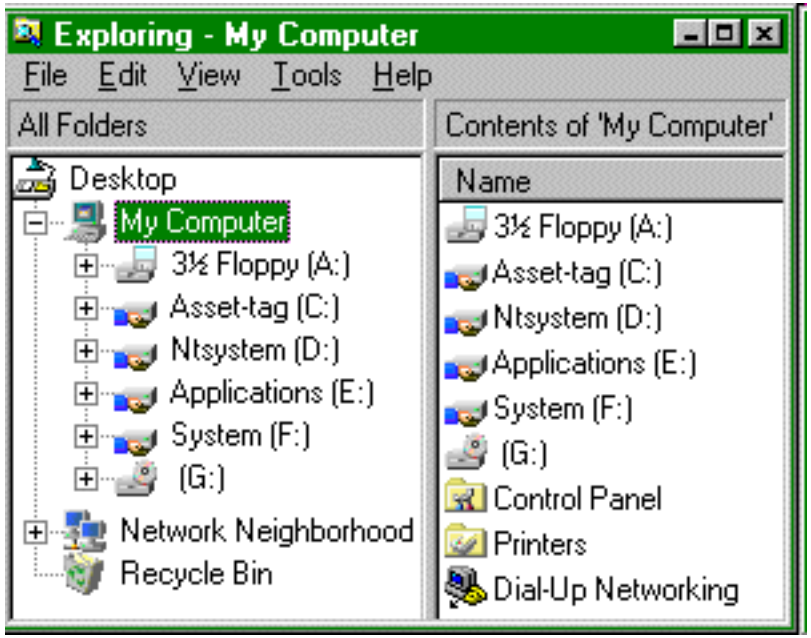
- **Component (LibraryItem)**
  - definiše interfejs za objekte kojima se dinamički mogu dodavati nove funkcije
- **ConcreteComponent (Book, Video)**
  - predstavlja klasu čijim se objektima mogu dinamički dodavati funkcije
- **Decorator (Decorator)**
  - čuva referencu na objekat klase Component i definiše interfejs koji se slaže sa interfejsom klase Component
- **ConcreteDecorator (Borrowable)**
  - dodaje nove funkcije komponenti

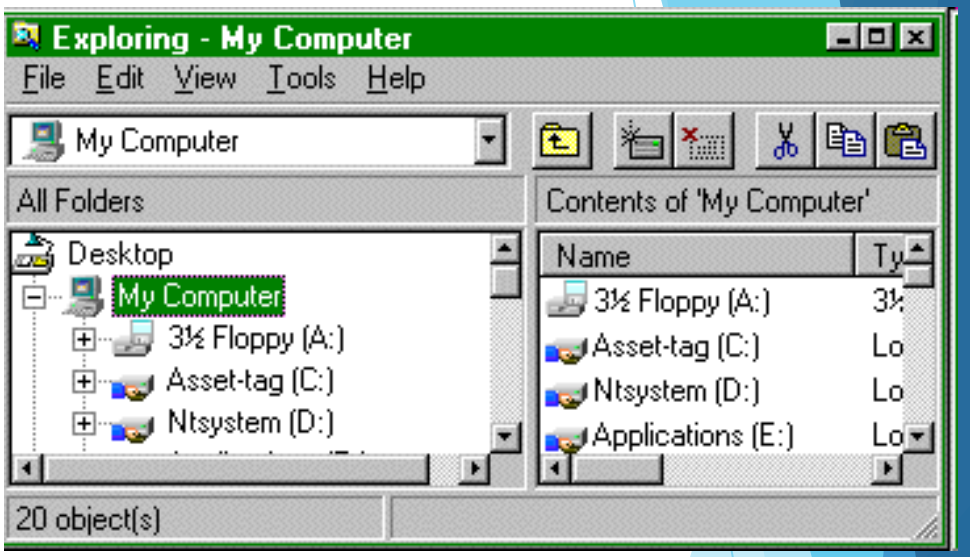
Primer iz stvarnog sveta

- Ovaj primer pokazuje kako obrazac Decorator pozajmljuje“ funkcionalnost postojećim elementima biblioteke.
- decoratorRW.cs


TextView
aBorder
verticalScroll
horizontalScroll
Border
Scrollbar
NoBorder
3DBorder
Flat
Vertical Scrollbar
Horizontal Scrollbar


aBorderDecorator
component
aScrollDecorator
component
aTextView


Exploring - My Computer
File Edit View Tools Help
All Folders
Contents of 'My Computer'
Desktop
My Computer
3½ Floppy (A:)
Asset-tag (C:)
Ntsystem (D:)
Applications (E:)
System (F:)
(G:)
Network Neighborhood
Recycle Bin
Name
3½ Floppy (A:)
Asset-tag (C:)
Ntsystem (D:)
Applications (E:)
System (F:)
(G:)
Control Panel
Printers
Dial-Up Networking

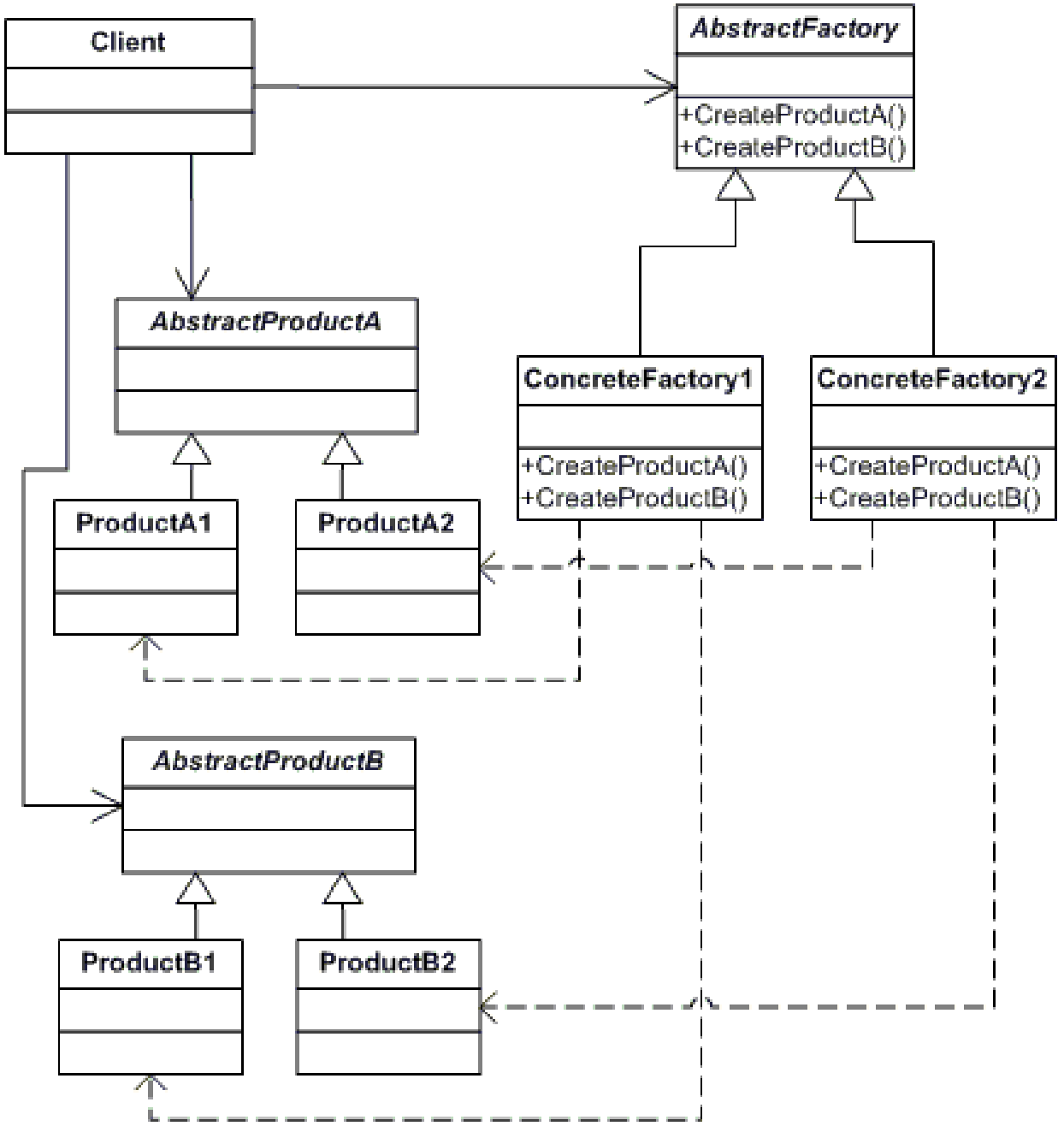

Exploring - My Computer
File Edit View Tools Help
My Computer
All Folders
Contents of 'My Computer'
Desktop
My Computer
3½ Floppy (A:)
Asset-tag (C:)
Ntsystem (D:)
Name
3½ Floppy (A:)
Asset-tag (C:)
Ntsystem (D:)
Applications (E:)
20 object(s)

Abstract Factory

Definicija

- Obezbeđuje interfejs za kreiranje familije povezanih i međusobno zavisnih objekata bez specificiranja njihovih konkretnih klasa
- abstractFactory.cs

UML Dijagram

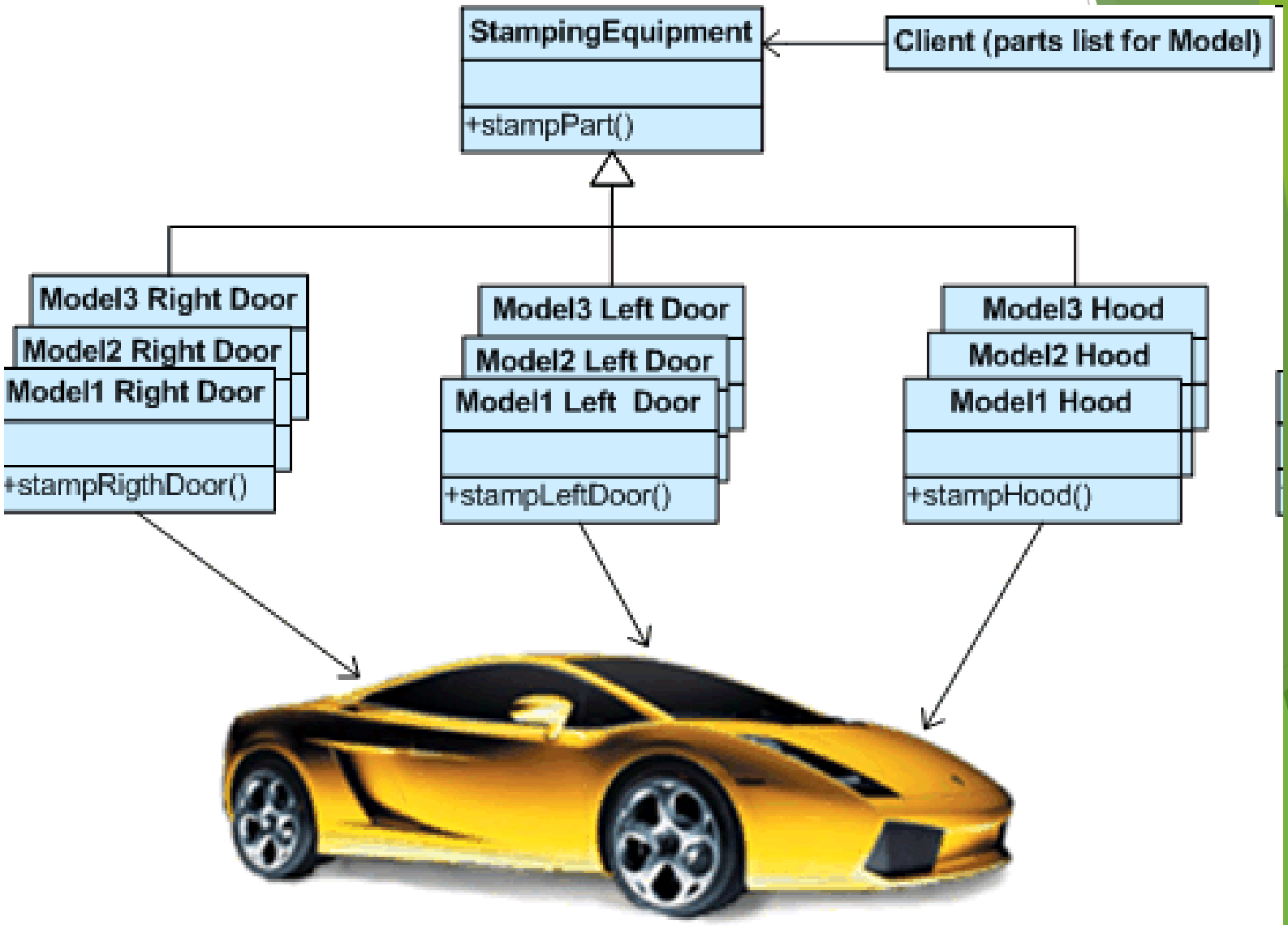


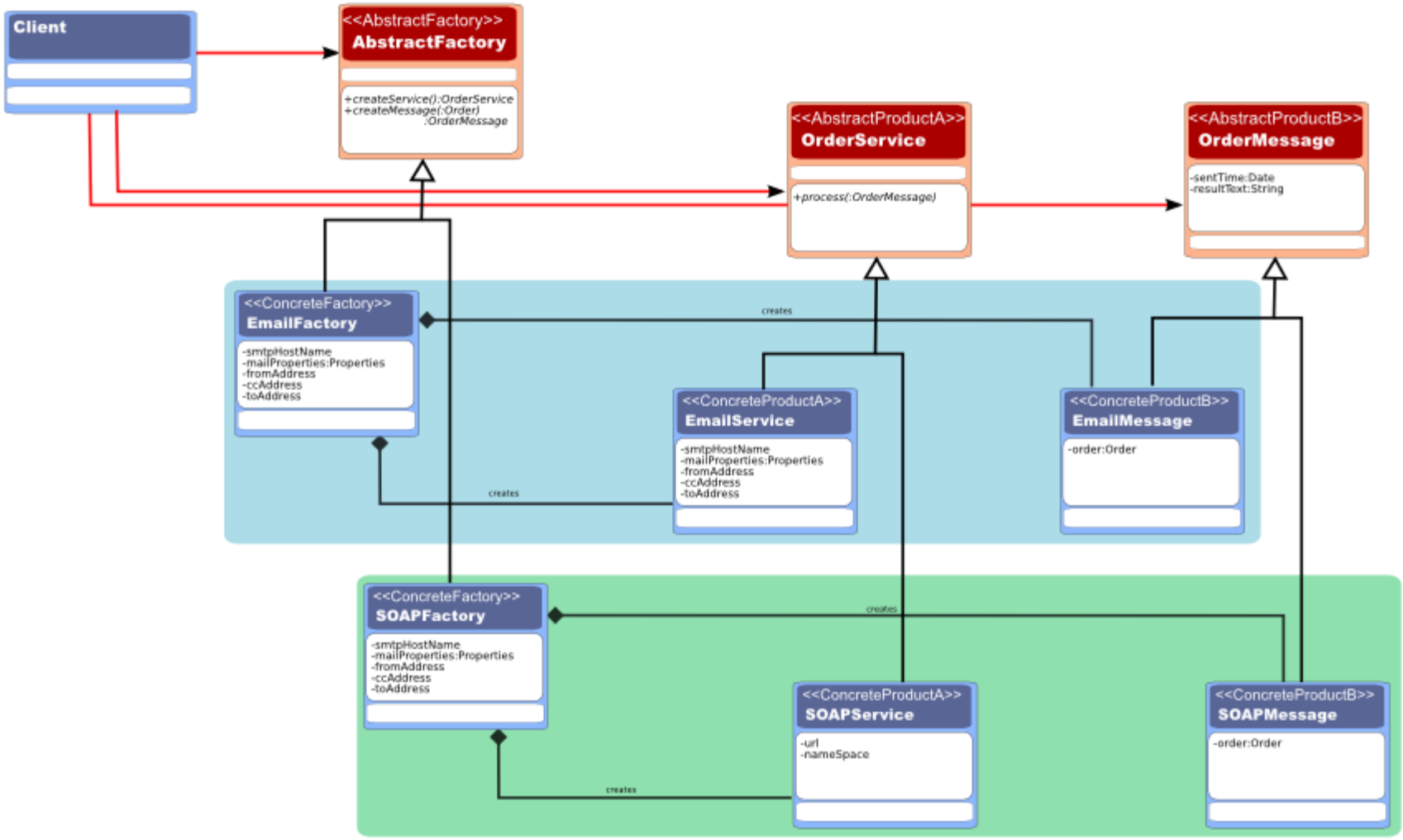
Elementi

- **AbstractFactory (ContinentFactory)**
  - deklariše interfejs za operacije koje kreiraju određene apstraktne objekte
- **ConcreteFactory (AfricaFactory, AmericaFactory)**
  - implemetira operacije za kreiranje određene vrste konkretnih objekata
- **AbstractProduct (Herbivore, Carnivore)**
  - deklariše interfejs za određenu vrstu objekata
- **Product (Wildebeest, Lion, Bison, Wolf)**
  - definiše objekat koji će biti kreiran u skladu sa odgovarajućom ConcreteFactory klasom
  - implementira interfejs koji definiše klasa AbstractProduct
- **Client (AnimalWorld)**
  - Korisnički interfejs ka klasama AbstractFactory i AbstractProduct

Primer iz stvarnog sveta

- Ovaj primer iz stvarnog sveta ili *real-world code* opisuje stvaranje različitih životinjskih svetova za npr. neku igricu korišćenjem različitih fabrika. Bez obzira što su kreirane životinje različite u zavisnosti od kontinenta, interakcija među njima ostaje ista.
- abstractFactoryRW.cs


StampingEquipment
+stampPart()
Client (parts list for Model)
Model3 Right Door
Model2 Right Door
Model1 Right Door
+stampRigthDoor()
Model3 Left Door
Model2 Left Door
Model1 Left Door
+stampLeftDoor()
Model3 Hood
Model2 Hood
Model1 Hood
+stampHood()


Client
<<AbstractFactory>>
AbstractFactory
+createService():OrderService
+createMessage(:Order)
:OrderMessage
<<AbstractProductA>>
OrderService
+process(:OrderMessage)
<<AbstractProductB>>
OrderMessage
-sentTime:Date
-resultText:String
<<ConcreteFactory>>
EmailFactory
-smtpHostName
-mailProperties:Properties
-fromAddress
-ccAddress
-toAddress
creates
<<ConcreteProductA>>
EmailService
-smtpHostName
-mailProperties:Properties
-fromAddress
-ccAddress
-toAddress
creates
<<ConcreteProductB>>
EmailMessage
-order:Order
<<ConcreteFactory>>
SOAPFactory
-smtpHostName
-mailProperties:Properties
-fromAddress
-ccAddress
-toAddress
creates
<<ConcreteProductA>>
SOAPService
-url
-nameSpace
creates
<<ConcreteProductB>>
SOAPMessage
-order:Order