UNIVERZITET U BANJOJ LUCI
ELEKTROTEHNIČKI FAKULTET

Prof. dr Dražen Brđanin

PROJEKTOVANJE SOFTVERA
/ reuse /

Banja Luka
2024.

Reuse

Skup aktivnosti i tehnika kojima se postojeće komponente ili obrasci prilagođavaju za novu primjenu.

Glavni ciljevi:

- **Ponovna upotreba znanja stečenih u prethodnim projektima**
- **Ponovna upotreba postojećih funkcionalnosti & artefakata**

Vidovi ponovne upotrebe: *Black-box reuse* / *White-box reuse*

Black-box reuse: **KOMPOZICIJA**

- Nova funkcionalnost realizuje se agregacijom/kompozicijom
- Novi objekat sa više funkcionalnosti realizuje se kao kompozicija koja sadrži postojeći objekat

White-box reuse: **NASLJEĐIVANJE**

- Nova funkcionalnost realizuje se nasljeđivanjem/specijalizacijom
- Novi objekat sa više funkcionalnosti je specijalizovani postojeći objekat

Reuse

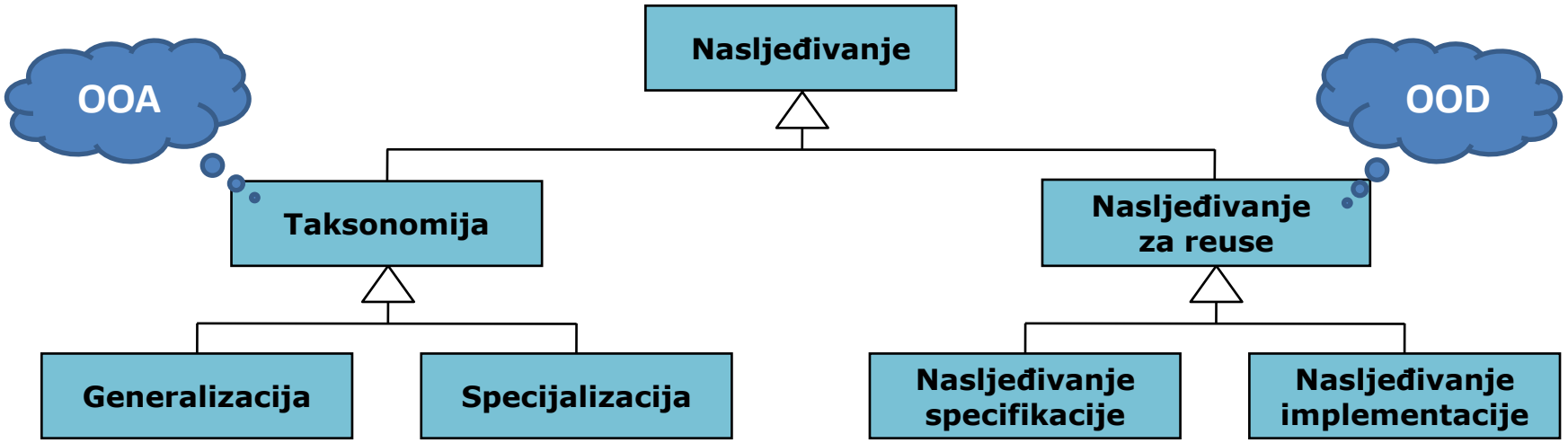
NASLJEĐIVANJE u kontekstu ponovne upotrebe

- **Koncept nasljeđivanja koristi se na četiri različita načina:**
  - **generalizacija** (*generalization*)
  - **specijalizacija** (*specialization*)
  - **nasljeđivanje specifikacije** (*specification inheritance*)
  - **nasljeđivanje implementacije** (*implementation inheritance*)

OOA (generalizacija, specijalizacija)

OOD (nasljeđivanje specifikacije, nasljeđivanje implementacije)

Meta-model nasljeđivanja



Reuse

“Otkrivanje” veza nasljeđivanja u objektnom modelu

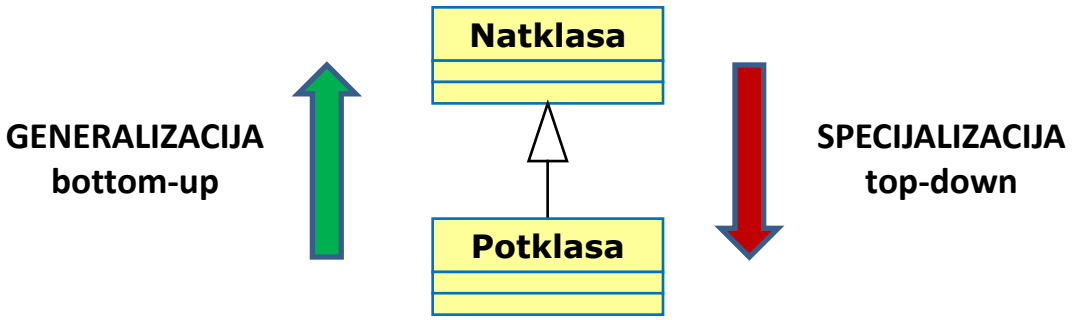
- nasljeđivanje se “otkriva” postupkom **generalizacije** ili **specijalizacije**

generalizacija

- “**bottom-up**” princip identifikacije nasljeđivanja
- na osnovu postojeće klase (potklasa) vrši se generalizacija (uopštavanje) i identifikuje natklasa

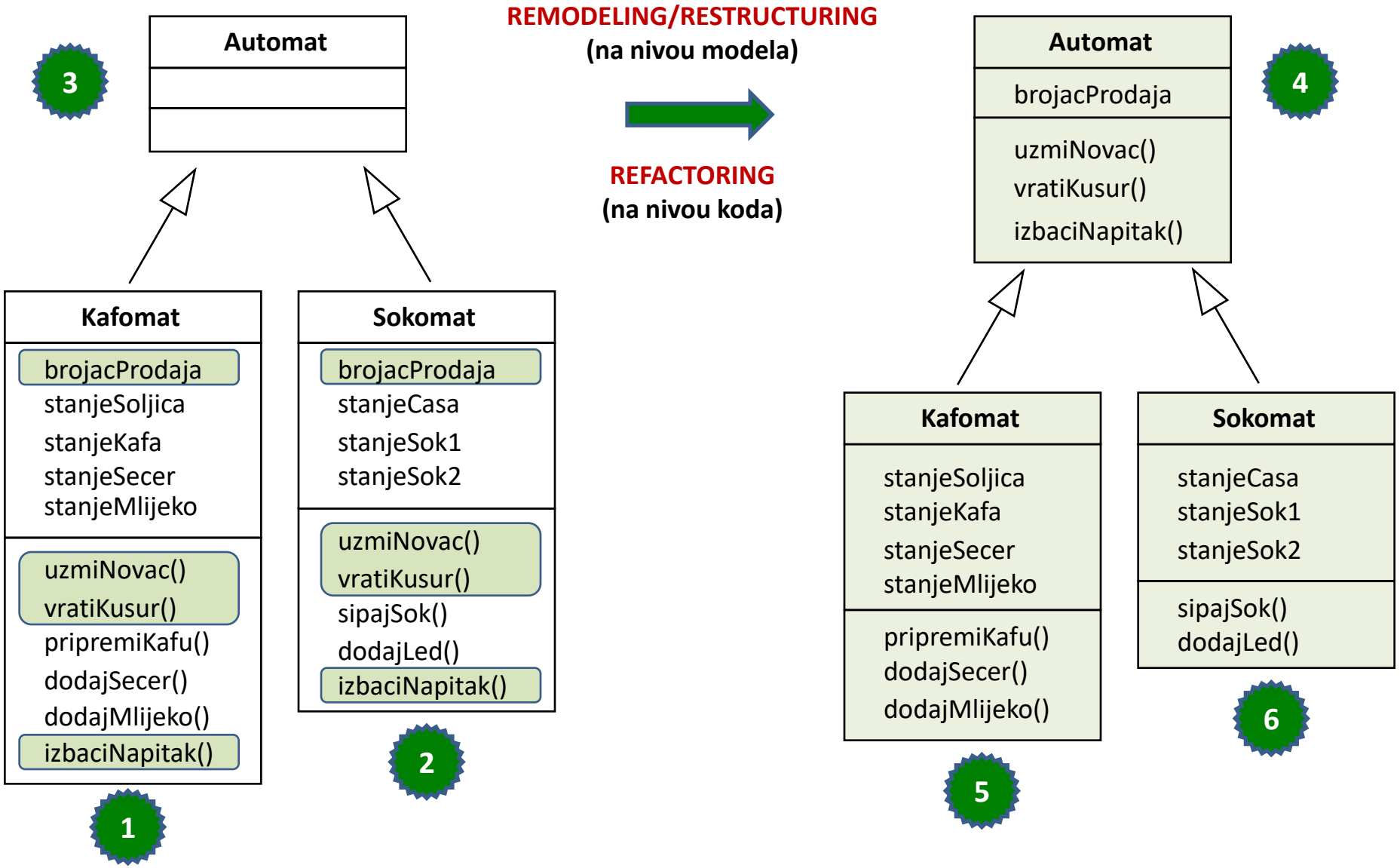
specijalizacija

- “**top-down**” princip identifikacije nasljeđivanja
- na osnovu postojeće klase (natklasa) vrši se specijalizacija i identifikuje potklasa



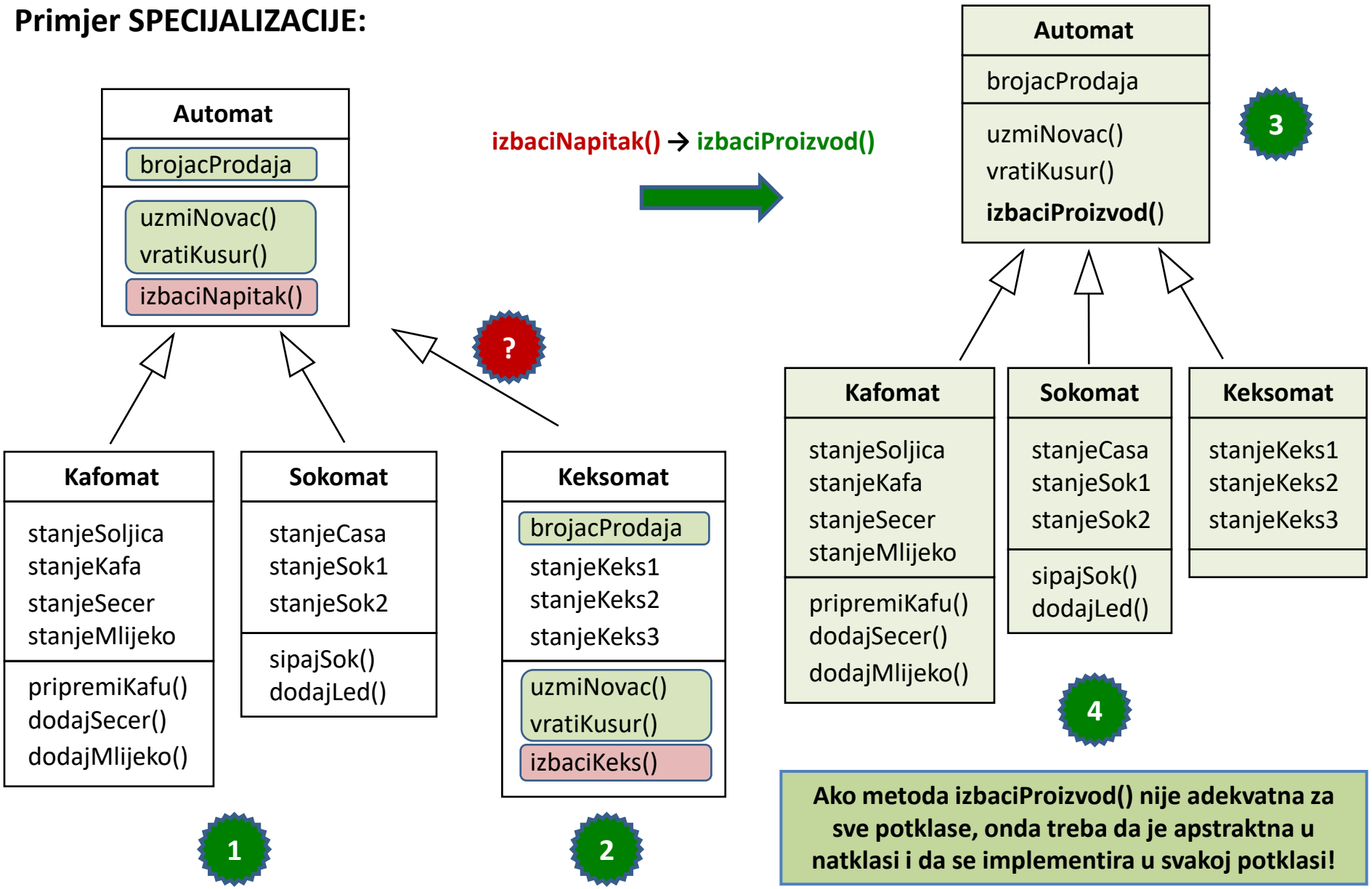
Reuse

Primjer GENERALIZACIJE:



Reuse

Primjer SPECIJALIZACIJE:



Reuse

Nasljeđivanje SPECIFIKACIJE / Nasljeđivanje IMPLEMENTACIJE

- **Koristi se u OOD u cilju smanjenja redundantnosti i omogućavanja proširljivosti dizajna**

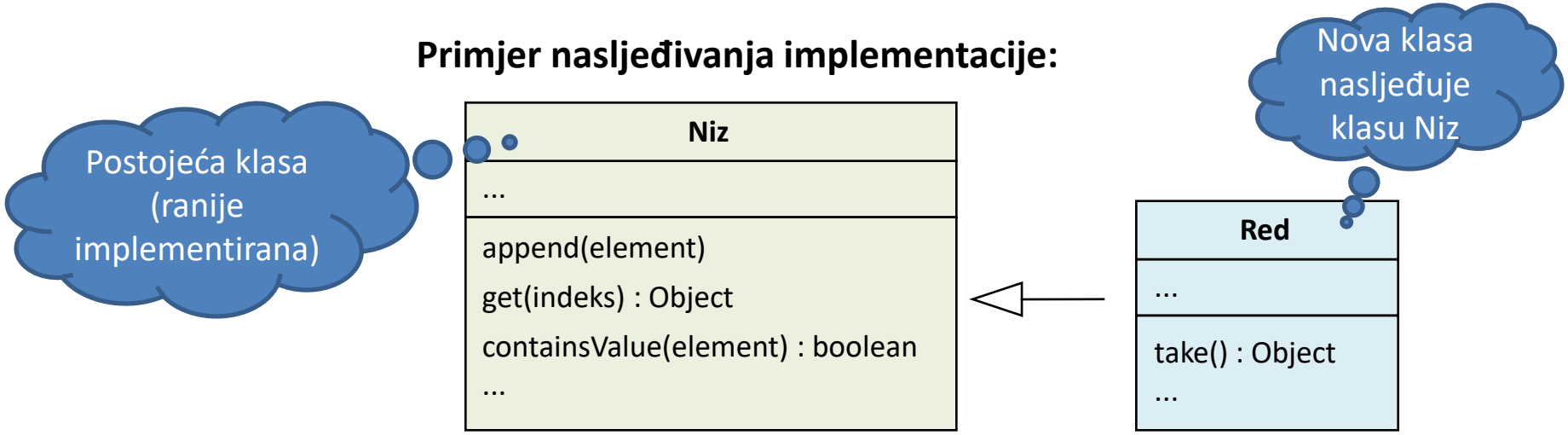
nasljeđivanje specifikacije
(*specification inheritance*)

- **Specifikacija je apstraktna klasa** (klasa sa operacijama bez implementacije)
- Često se naziva i **subtyping (specijalizacija tipa/interfejsa)**
- **Koristi se za klasifikaciju tipova (hijerarhija tipova)**

nasljeđivanje implementacije
(*implementation inheritance*)

- Često se naziva i **nasljeđivanje klasa = nasljeđivanje konkretne klase** (implementirane operacije)
- **Cilj: proširivanje skupa funkcionalnosti korišćenjem funkcionalnosti neke postojeće klase**

Primjer nasljeđivanja implementacije:

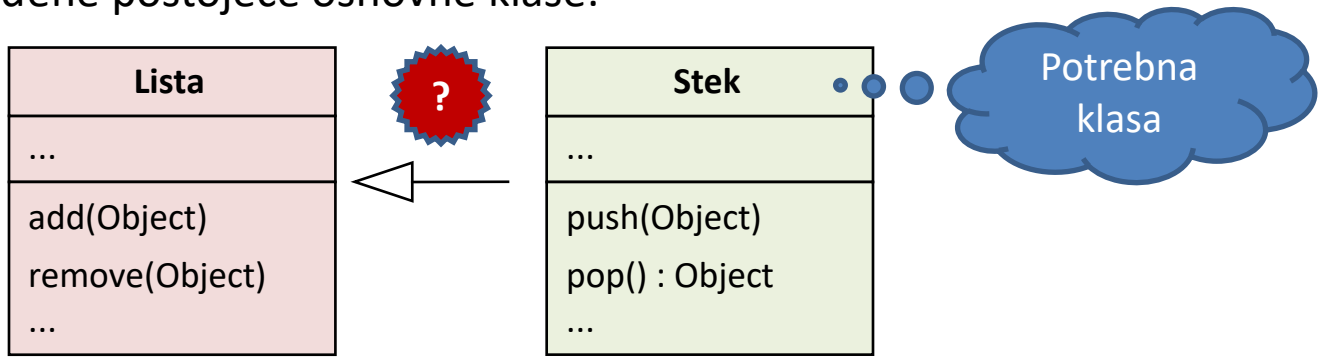


Reuse

Problemi kod nasljeđivanja IMPLEMENTACIJE

- **Često postojeća klasa nije u potpunosti pogodna za dobijanje željene potklase**
  - Npr. postojeća klasa ima suvišne operacije ili operacije koje ne daju poželjan rezultat

Primjer neprilagođene postojeće osnovne klase:



Prevazilaženje problema kod nasljeđivanja IMPLEMENTACIJE

- **Problem suvišne operacije ili operacije koja ne daje poželjan rezultat, može da se riješi:**
  - **Nasljeđivanjem**
    - Dodavanjem nove operacije u osnovnu klasu (ako je moguće)
    - Redefinisanjem operacije u izvedenoj klasi
  - **Delegacijom**

Reuse

Nasljeđivanje SPECIFIKACIJE

- Često se naziva i **subtyping**
- **Koristi se za klasifikaciju tipova (hijerarhija tipova)**

Princip SUPSTITUCIJE (Liskov, 1988)

- Formalna definicija nasljeđivanja specifikacije:

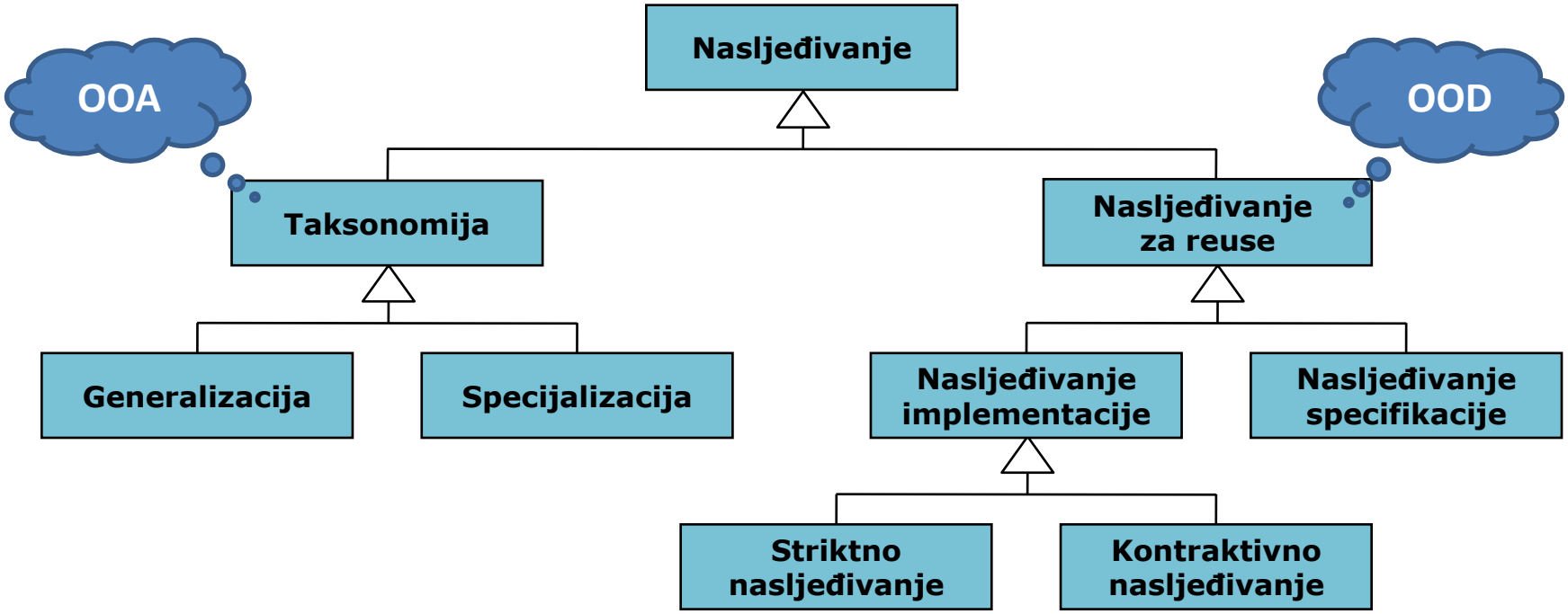
Ako objekat tipa S može bilo gdje da zamijeni očekivani objekat tipa T, tada je S podtip od T.

- **Interpretacija:**
  - Za sve klase koje su podtipovi neke superklase vrijedi da **sve veze nasljeđivanja predstavljaju nasljeđivanje specifikacije**, odnosno da neka **metoda kako je definisana u superklasi mora biti dostupna i u bilo kojoj potklasi, i klijent može da je koristi kao da pristupa superklasi.**
  - Postojećoj hijerarhiji klasa (tipova) **može da se dodaje nova potklasa bez modifikacije metoda iz superklase**, čime se omogućava proširljivost.

- **Nasljeđivanje u skladu sa principom supstitucije naziva se striktno nasljeđivanje, a ako nije u skladu sa principom supstitucije naziva se kontraktivno nasljeđivanje.**

Reuse

Revidirani meta-model nasljeđivanja



Reuse

Delegacija

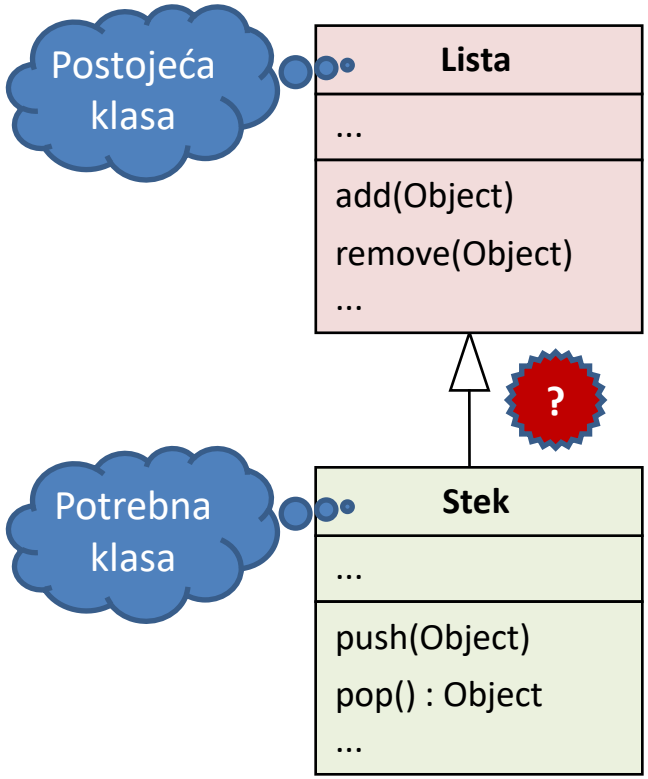
Delegacija je tehnika kojom klasa implementira operaciju prosljeđujući poruku drugoj klasi.

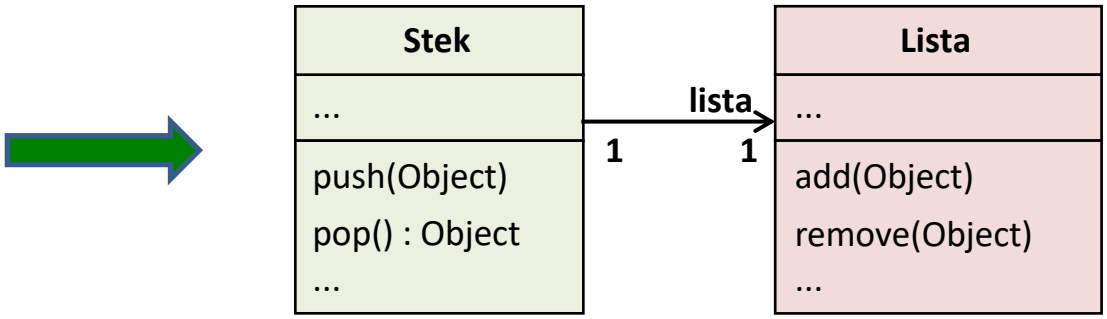
- **U rješavanju zahtjeva nekog klijenta učestvuju dva objekta:**
  - **delegator** (proxy) prima zahtjev klijenta i prosljeđuje ga **delegatu**
  - delegator provjerava/štiti delegata od "zloupotrebe" od strane klijenta

- **Primjena delegacije za reuse postojećih klasa:**
  - **Delegacija je ekvivalentna kompoziciji**
  - **Delegacija predstavlja bolji mehanizam za reuse postojeće klase**:
    - delegacija obezbjeđuje novu klasu od neželjenog ponašanja iz postojeće klase
    - nova klasa ne nasljeđuje ništa iz postojeće klasa, pa se ne ponaša kao postojeća klasa i nema straha da će doći do pogrešne upotrebe nove klase umjesto postojeće
    - delegacija ne zahtijeva nikakve izmjene na postojećoj klasi

Reuse

Primjer DELEGACIJE:





Rezultantni kod:

```
// Implementacija Steka delegacijom preko Liste
class Stek
{
   private Lista lista;
   Stek() { lista = new Lista(); }
   void push(Object element)
   {
      ...
      lista.add(element);
      ...
   }
   ...
}
```

Reuse

Najvažnije prednosti višestruke upotrebe dizajna i implementacije

- **efikasnost razvoja (*efficiency*)**
  - ponovna upotreba postojećeg dizajna i implementacije **značajno skraćuje vrijeme** i **ubrzava razvoj** **i** **smanjuje cijenu razvoja** ciljnog sistema
- **robusnost i pouzdanost** (***robustness / reliability***)
  - provjerene komponente i obrasci rezultat su dugogodišnjeg rada velikog broja domenskih eksperata, pa njihova primjena **smanjuje rizik** od pogrešnog dizajna i **povećava pouzdanost** ciljnog sistema
- **proširljivost i prilagodljivost (*extensibility / adaptability*)**
  - AF omogućava **proširivanje postojeće funkcionalnosti** i **prilagođavanje za konkretnu primjenu**
- **modularnost (*modularity*)**
  - **AF poboljšavaju modularnost**, jer AF **uvode komponente sa stabilnim implementacijama** i **dobro definisanim interfejsima**
- **olakšana komunikacija**
  - **projektni obrasci imaju jedinstvene nazive**, što **uniformiše korišćenu terminologiju** i **olakšava komunikaciju** između pojedinaca / projektnih tim(ov)a

Reuse – Aplikativni radni okviri

Application framework – AF

- **Parcijalna aplikacija koja se koristi kao osnov za produkciju prilagođene i specijalizovane aplikacije za konkretan domen** (Johnson & Foote, 1988)
- **Prije pojave AF**, u razvoju softvera *reuse* funkcionalnosti ostvarivao se pomoću:
  - **biblioteka funkcija** (razvoj softvera primjenom proceduralnih prog. jezika)
  - **biblioteka klasa** (razvoj softvera primjenom O-O prog. jezika), npr. STL
- AF su u softversko inženjerstvo donijeli **reuse dizajna i reuse koda**
- AF omogućavaju **visok nivo višestruke upotrebe koda** (***large scale reuse***)
- AF **definišu dizajn i implementaciju specifičnih softverskih sistema/komponenata**
- **AF definiše**:
  - **dizajn ciljnog softverskog sistema**
  - **osnovne klase, odgovornosti i interakcije objekata**
  - **kontrolu toka** (izvršavanje uvijek započinje unutar AF koda)
- **Mehanizmi za proširenje** AF funkcionalnosti (***extension points***)
  - **nasljeđivanje** (*hook* metode) – “white box”
  - **delegacija** – “black box”

Reuse – AF

Klasifikacija AF prema poziciji u softverskom procesu

Infrastrukturni AF

- **namijenjeni za pojednostavljenje softverskog procesa** i **brži razvoj pojedinih softverskih podsistema** (arhitektura, GUI, perzistencija, web aplikacije, ...)
- infrastrukturni AF koriste se u okviru softverskog projekta i tipično se ne isporučuju klijentu
- **tipični primjeri infrastrukturnih AF:**
  - **za razvoj GUI** – AF koji realizuju GUI
    - npr. Swing, JavaFX, ...
  - **za perzistenciju** – AF koji realizuju ORM
    - npr. Hibernate, Django, ...
  - **za razvoj softverskih alata vezanih za modelovanje**
    - npr. EMF (Eclipse Modeling Framework)
  - **web AF** – AF za razvoj web aplikacija (arhitektura, web servisi, API)
    - npr. Angular, Django, ASP.NET, Meteor, Spring, ...

iAF isu ključni za brz i kvalitetan razvoj softvera!

Reuse – AF

Klasifikacija AF prema poziciji u softverskom procesu

Middleware AF

- **koriste se za integraciju distribuiranih aplikacija i komponenata**
- tipični primjeri: RMI, JMS, CORBA, DCOM, ...

mAF isu ključni za brz i kvalitetan razvoj softvera!

Enterprise AF

- **aplikativno specifični i fokusirani na konkretne domene**
- neki primjeri:
  - **ERP (*Enterprise Resource Planning*)** – poslovni informacioni sistemi (upravljanje dokumentima, magacinsko poslovanje, kupci-dobavljači, ...)
    - **MiTOS**, **Apache OFBiz, ...**
  - **eZdravstvo (*eHealth*)**
    - OpenEHR, ...
  - **javne nabavke (*public procurement*)**
    - EllectraWEB

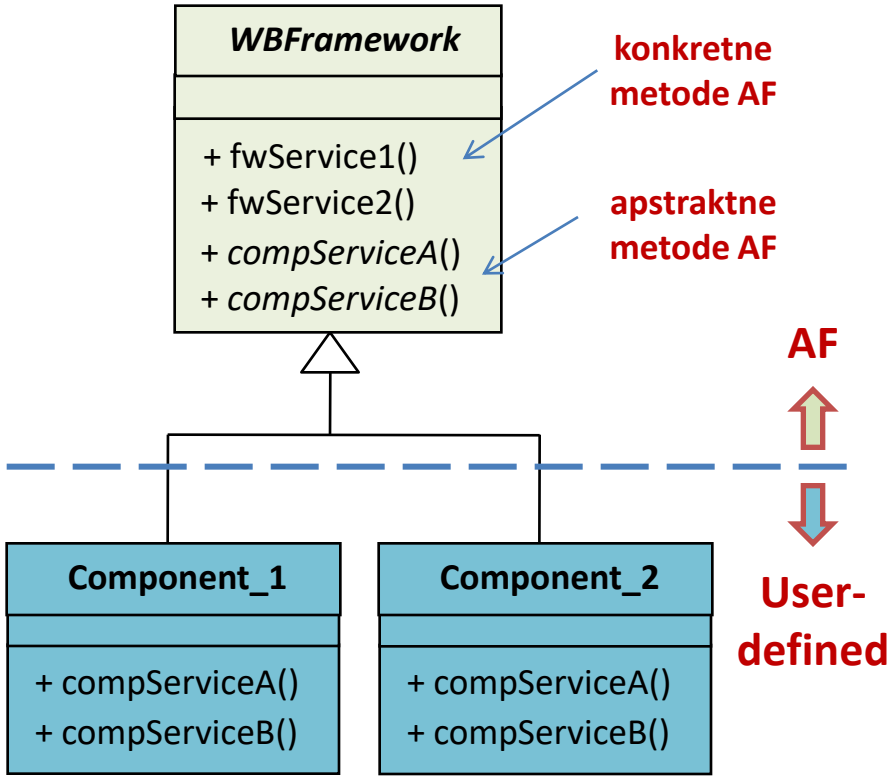
eAF pogodni za razvoj end-user aplikacija!

Reuse – AF

Klasifikacija AF prema načinu proširivanja funkcionalnosti

"white box" AF

- mehanizmi za proširivanje funkcionalnosti: **nasljeđivanje** i **dinamičko vezivanje** (*dynamic binding*),
- postojeće AF funkcionalnosti proširuju se **nasljeđivanjem osnovnih klasa iz AF** i **redefinisanjem *hook* metoda primjenom odgovarajućih obrazaca** (npr. *template method*)
- da bi se proširila postojeća funkcionalnost, **neophodno je poznavanje interne strukture AF**
- zbog velike sprege sa hijerarhijom klasa u AF, izvedeni softverski sistemi moraju da se rekompajliraju u slučaju izmjena u AF
- primjenjuje se princip "**inverzija kontrole**" (*inversion of control*) – metoda iz osnovne klase iz AF poziva redefinisanu metodu proširenja



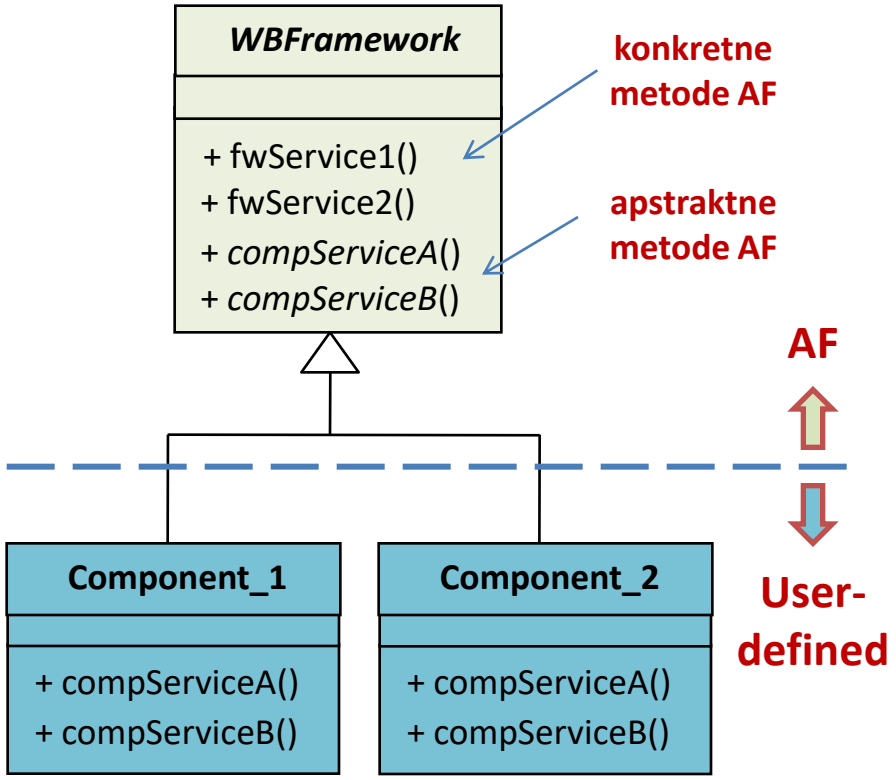
Reuse – AF

"white box" AF

```
package whiteboxAF;
public abstract class KlasaAF
{
   public void f() { metoda(); }
   protected abstract void metoda();
}
```

```
package myAppWhitebox;
import whiteboxAF.KlasaAF;
public class MyApp extends KlasaAF
{
  @Override
  protected void metoda() { // spec.impl }
}
```

```
package myAppWhitebox;
public class App
{
  public static void main(String args[])
  {
     MyApp m = new MyApp();
     m.f();
  }
}
```

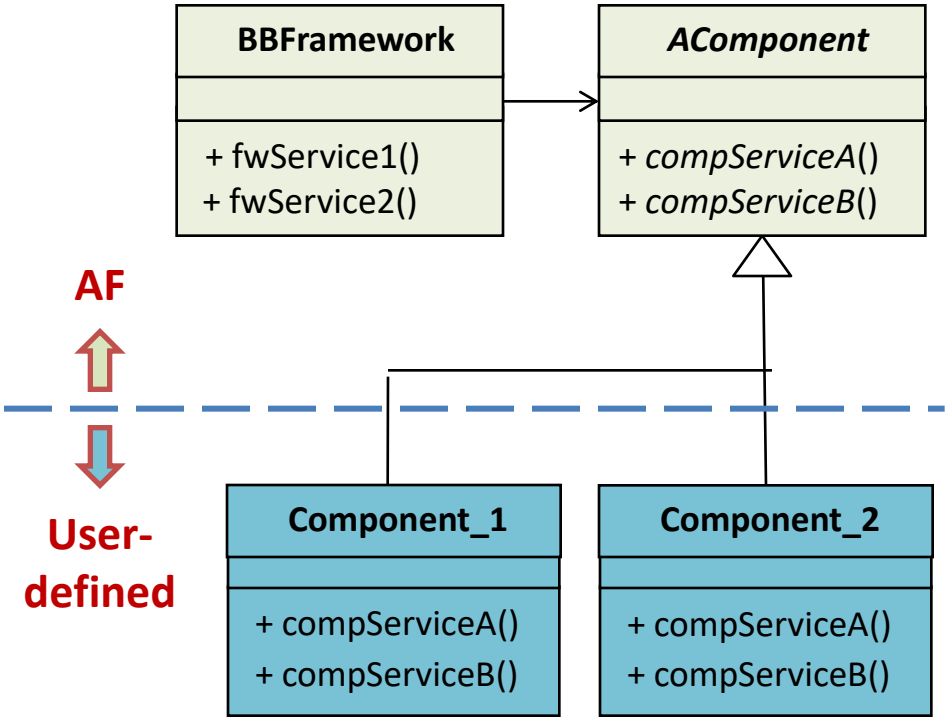


Reuse – AF

Klasifikacija AF prema načinu proširivanja funkcionalnosti

"black box " AF

- proširivanje funkcionalnosti zasniva se na definisanju odgovarajuće sprege sa AF komponentama
- postojeće funkcionalnosti se višestruko koriste definisanjem komponenata u skladu sa specifičnim interfejsom i integracijom tih komponenata sa AF **primjenom delegacije**
- **korišćenje *black box* AF je jednostavnije** od *white box* AF, jer se zasniva na primjeni delegacije, a ne na nasljeđivanju
- **razvoj *black box* AF je složeniji** nego razvoj *white box* AF, jer neophodno definisanje interfejsa i *hook* metoda za širok spektar slučajeva upotrebe



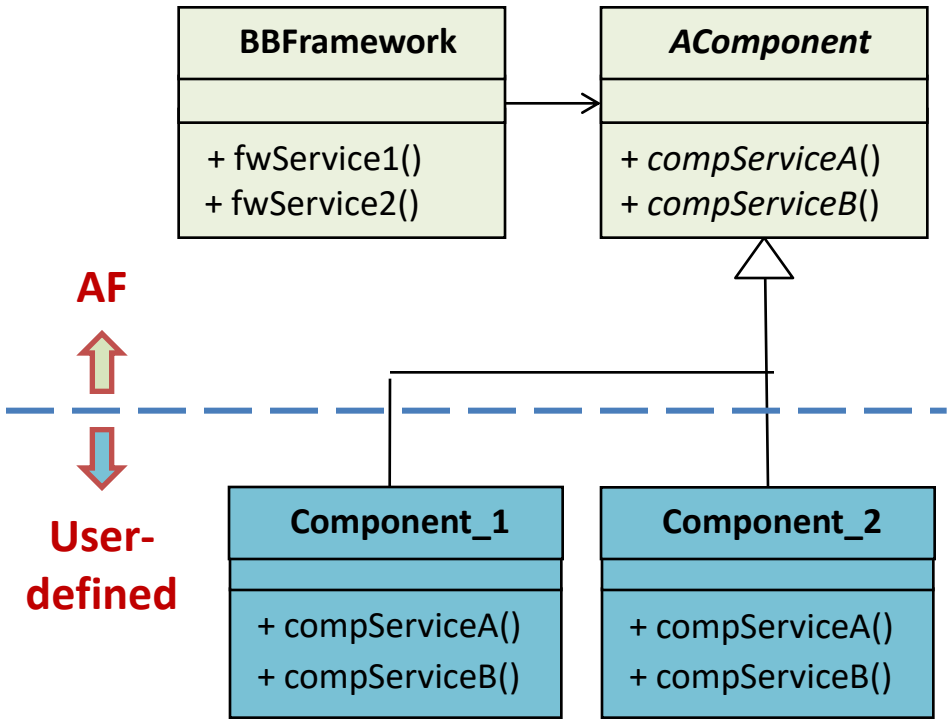
Reuse – AF

"black box" AF

```
package blackboxAF;
public final class KlasaAF {
   public void fun(Funkcionalnost f)
   { f.metoda(); }
}
```

```
package blackboxAF;
public interface Funkcionalnost {
   public void metoda();
}
```

```
package myAppBlackbox;
public class myF implements Funkcionalnost
{
   public void metoda() { // impl. }
}
public class App
{
  public static void main(String args[])
  {
     myK mk = new KlasaAF();
     mk.fun(new myF());
  }
}
```

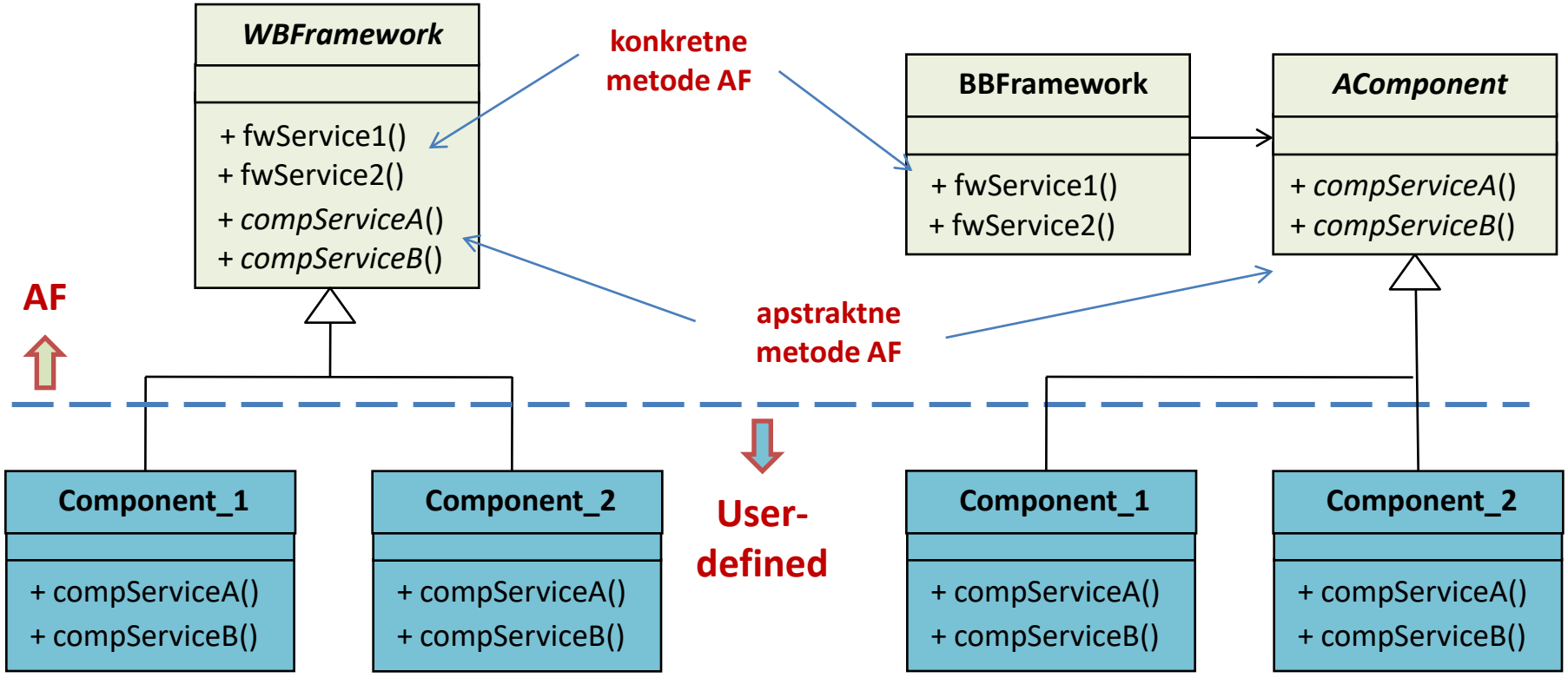


Reuse – AF

Klasifikacija AF prema načinu proširivanja funkcionalnosti

"white box" AF

"black box " AF



Reuse – AF

AF ↔ biblioteke klasa

- Klase u **AF** su u međusobnim vezama koje obezbjeđuju višestruko upotrebljiv **arhitekturni *skeleton*** za familiju sličnih aplikacija.
- **Biblioteke klasa** (npr. kontejnerske klase) su manje domenski specifične i omogućavaju ***reuse* manjeg obima**, ali **u različitim domenima**.
- **Biblioteke klasa su tipično pasivne** – ne implementiraju niti ograničavaju kontrolu toka.
- **AF su aktivni** – kontrolišu tok u aplikaciji.
  - **princip inverzije kontrole** (metoda iz osnovne klase iz AF poziva redefinisanu metodu proširenja)
- U praksi se **AF i biblioteke klasa zajedno** koriste u realizaciji sistema – AF tipično koriste biblioteke funkcija za razvoj i proširivanje funkcionalnosti AF, implementacija specifičnog aplikativnog koda takođe se zasniva na bibliotekama klasa (obrada stringova, manipulacija fajlovima, ...)

Reuse – AF

AF ↔ projektni obrasci

- **AF je parcijalna aplikacija koja se koristi kao osnov za produkciju prilagođene i specijalizovane aplikacije za konkretan domen.**
- **AF se fokusira na višestruku upotrebu konkretnog dizajna**, algoritama i **implementacije u konkretnom programskom jeziku** i **u konkretnom aplikativnom domenu**.
- **Projektni obrasci fokusiraju se na apstraktan dizajn** i **male kolekcije povezanih klasa** za **primjene u različitim domenima**.
- **Projektni obrazac je uopšteno (šablonsko) rješenje za tipične projektne probleme** (strukturni obrasci / obrasci ponašanja / kreacioni obrasci), koje omogućava višestruku upotrebu u rješavanju konkretnih projektnih situacija u različitim domenima.
- **Projektni obrasci nisu fokusirani na implementaciju** (nego na projektovanje), a **AF predstavljaju parcijalno implementiranu aplikaciju.**

Reuse – AF

Ključne koristi / prednosti primjene AF

- **višestrukost upotrebe (*reusability*)**:
  - veliko domensko znanje i iskustvo projektanata ugrađeno u razvoj AF značajno skraćuje vrijeme implementacije
  - nema potrebe za ponovnim razvojem i validacijom
- **proširljivost (*extensibility*)**:
  - AF tipično raspolaže **hook metodama**, koje se redefinišu u aplikaciji, čime se AF proširuje i prilagođava za konkretnu primjenu
- **modularnost (*modularity*)**:
  - AF poboljšavaju modularnost, jer AF uvode komponente sa stabilnim implementacijama i dobro definisanim interfejsima
- **efikasnost razvoja (*efficiency*)**:
  - primjena AF značajno skrajuće vrijeme i ubrzava razvoj ciljnog sistema

Projektni obrasci (*Design patterns*)

- Projektni obrazac predstavlja **uopšteno (šablonsko) rješenje za tipične projektne probleme**, koje omogućava **višestruku upotrebu u rješavanju konkretnih projektnih situacija**.
- Projektni obrazac predstavlja **jezgro za rješavanje konkretnih problema iste/slične prirode – svaki put isto jezgro uz prilagođavanje konkretnoj situaciji**.
- Mnogi **projektni obrasci su sistematično dokumentovani** i mogu **slobodno da se koriste** u projektovanju.
- Projektni obrasci su dobar mehanizam za **učenje projektovanja na osnovu tuđih iskustava** (višegodišnje, višestruko rafinirano iskustvo u rješavanju klasa sličnih/srodnih problema)!
- Projektni obrasci imaju dugu istoriju:
  - 1960-70: **Christopher Alexander** uveo **pojam projektnih obrazaca u arhitekturi** i urbanizmu,
  - 1980-90: **OOPSLA** (*Object-Oriented Programming, Systems, Languages and Applications*)
    - projektni obrasci u O-O softverskom inženjerstvu
    - ***Gang of Four*** (**GoF**) obrasci:

      E. Gamma, R. Helm, R. Johnson, J. Vlissides:
      "Design Patterns: Elements of Reusable Object-Oriented Software",
      Addison-Wesley, 1995.

GoF projektni obrasci

- **GoF (*Gang of Four*) projektni obrasci – katalog sa 23 fundamentalna obrasca**
- Karakteristike svakog projektnog obrasca:
  - **naziv**: jedinstveno ime (kratko, 1-2 riječi), koje dobro reprezentuje namjenu obrasca
  - **klasifikacija** :
    - **kreacioni obrazac** (ako je fokusiran na kreiranje objekata)
    - **strukturni obrazac** (ako je fokusiran na strukturu)
    - **obrazac ponašanja** (ako je fokusiran na kolaboraciju objekata)
  - **namjena**: kratak opis (jedna-dvije rečenice) o namjeni obrasca
    npr. "*Prilagođenje interfejsa jedne klase u interfejs kakav očekuje druga klasa*"
  - **aliasi**: alternativni nazivi koji se ponekad koriste
  - **motivacija**: opis projektne situacije/problema koji se rješava primjenom datog obrasca
  - **primjenljivost**: oblasti u kojima se može primijeniti dati obrazac i kako ih prepoznati
  - **struktura**: skup povezanih klasa i interfejsa za rješavanje projektnog problema
  - **učesnici**: kratak opis uključenih objekata i njihovih uloga
  - **kolaboracije**: opis kolaboracija između učesnika (tekstualno i/ili grafički)
  - **posljedice**: prednosti i nedostaci datog obrasca, uključujući preporuke za nedostatke
  - **implementacija**: preporuke za implementaciju obrasca, uključujući korisne "trikove"

Klasifikacija GoF projektnih obrazaca

strukturni obrasci	kreacioni obrasci	obrasci ponašanja
Adapter (Pregovarač, Prilagođenje)	**Abstract Factory** (Apstraktna fabrika)	**Command** (Komanda)
Bridge (Most)	**Builder** (Graditelj)	**Chain of Responsibility** (Komandni lanac)
Composite (Kompozicija)	**Factory Method** (Fabrički metod)	**Interpreter** (Tumač)
Decorator (Dekorater)	**Prototype** (Prototip)	**Memento** (Podsjetnik, Podsjećanje)
Facade (Fasada)	**Singleton** (Usamljenik)	**Iterator** (Brojač)
Flyweight (Superlaki)		**Mediator** (Posrednik)
Proxy (Poslanik)		**State** (Stanje)
		Observer (Posmatrač, Nadzornik)
		Visitor (Posjetilac)
		Strategy (Strategija)
		Template Method (Šablonski metod)