

Uvod u softversko inženjerstvo

Čiste klase

Nikola Luburić

nikola.luburic@uns.ac.rs

```
        message =  
        if not hasattr(self, '_headers_buffer'):  
            self._headers_buffer = []  
        self._headers_buffer.append((" %s %d %s\r\n" %  
            (self.protocol_version, code, message)).en  
            'latin-1', 'strict'))  
  
    def end_header(self, keyword, value):  
        """Send a MIME header to the headers buffer."""  
        if self.request_version != 'HTTP/0.9':  
            if not hasattr(self, '_headers_buffer'):  
                self._headers_buffer = []  
            self._headers_buffer.append(  
                ("%s: %s\r\n" % (keyword, value)).encode('lati  
            keyword.lower() == 'connection':  
            if value.lower() == 'close':  
                self.close_connection = True  
            if value.lower() == 'keep-alive':  
                self.close_connection = False
```

Šta donose klase?

Koje su kategorije klasa?

Kako nastaju klase?

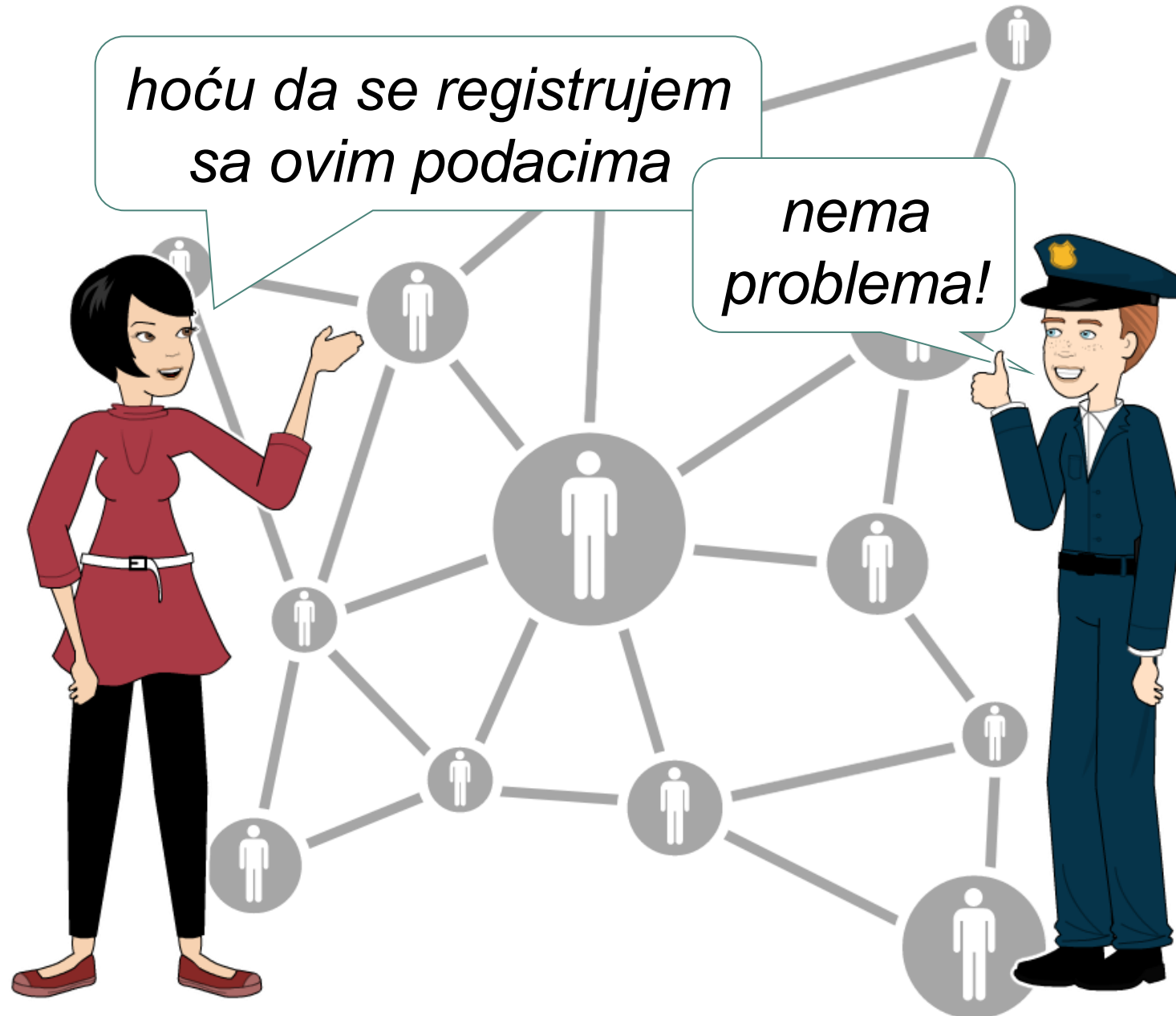
Čiste klase

Šta su svojstva dobrih klasa?

Šta donose klase inženjerstvu softvera?



Šta donose klase inženjerstvu softvera?



Šta donose klase inženjerstvu softvera?

Apstrakcija

Enkapsulacija

Nasleđivanje

Polimorfizam

Kako kategorizujemo klase spram ponašanja koje podržavaju?

Klase struktura podataka

Klase pametnih objekata

Klase koordinatora

Šta je struktura podataka, a šta objekat?

```
public class Play {  
    public String name;  
    public String type;  
}
```

*šta je formalno
instanca ove klase?*

```
public class Play {  
    private String name;  
    private String type;  
    public String getType() { return type; }  
    public void setType(String t) { this.type = t; }  
    public String getName() { return name; }  
    public void setName(String n) { this.name = n; }  
}
```

*šta je suštinski
instanca ove klase?*

Šta je struktura podataka, a šta objekat?

- ❖ Struktura podataka izlaže podatke,
nema ponašanje

```
public class FuelTank {  
    double GetTankCapacityInLiters() {...}  
    double GetLitersOfGasoline() {...}  
}
```

- ❖ Objekat sakriva podatke, izlaže ponašanje

```
public class FuelTank {  
    double GetPercentFuelRemaining() {...}  
}
```



Šta je struktura podataka, a šta objekat?

Square
+ topLeft : Point
+ side : double

Rectangle
+ topLeft : Point
+ height : double
+ width : double

Circle
+ center : Point
+ radius : double

Geometry
+ area (Object shape) : double

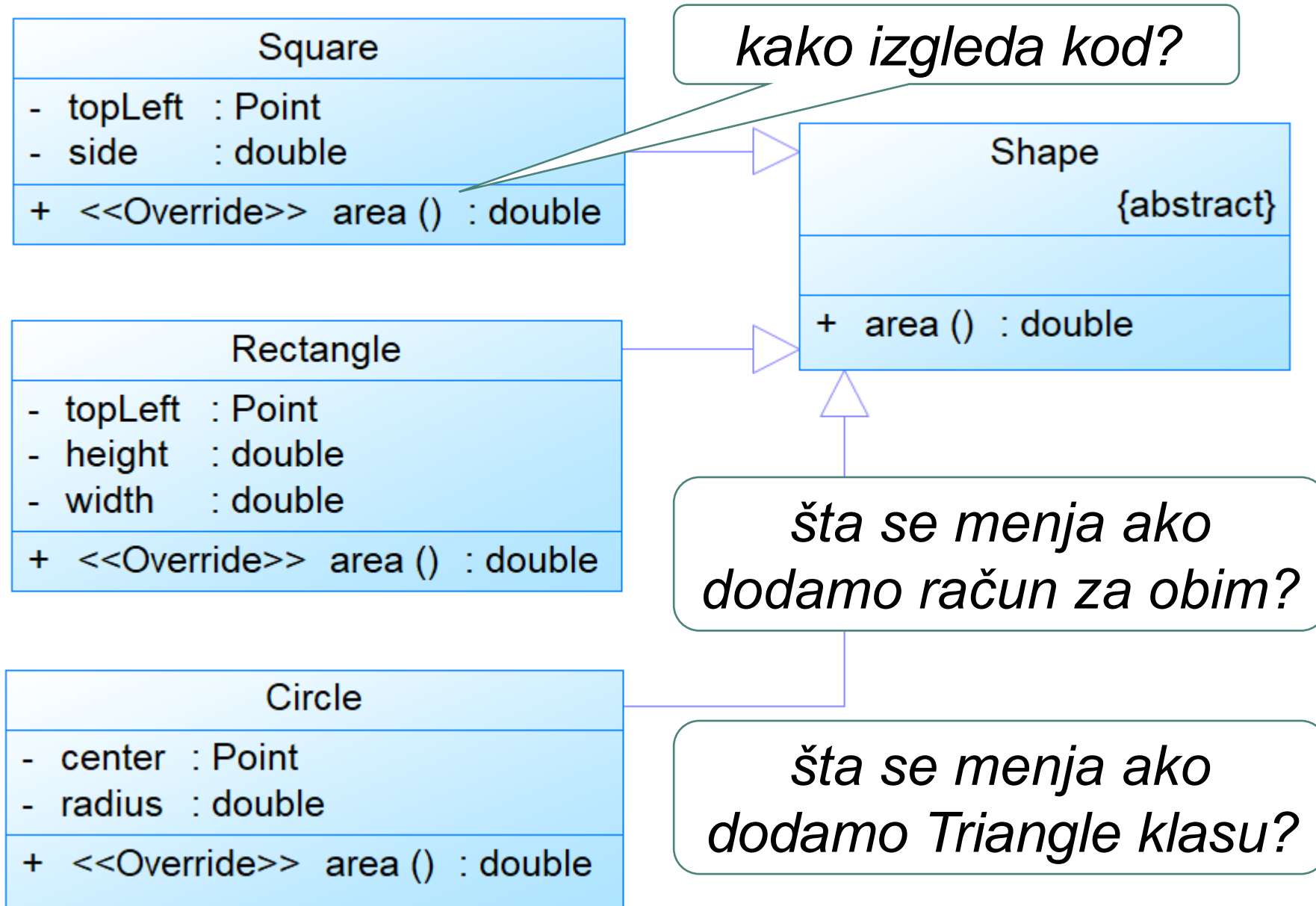


kako izgleda kod?

*šta se menja ako
dodamo račun za obim?*

*šta se menja ako
dodamo Triangle klasu?*

Šta je struktura podataka, a šta objekat?



Šta je struktura podataka, a šta objekat?

- ❖ Strukture podataka su stanje
Glupava grupa podataka (*DTO*)
- ❖ Objekti su ponašanje
Pamet i dinamika
- ❖ Koordinacija je takođe (specijalna) vrsta ponašanja



Identifikuj ulogu koju klasa ima u sistemu

Kako nastaju klase?

Diktira problem

Poslovni podaci
i operacije

Diktira tehnologija

Šabloni odabranih
tehnologija (*framework...*)

Čist dizajn koda

Iz primitiva
Iz funkcija
Iz klasa

Ekstrakcija klase iz primitiva

*String vs Enum
vs Class?*

```
int countHigherPriority(Order[] orders) {  
    int count = 0;  
    for(Order o: orders)  
        if(o.prio.equals("high") || o.prio.equals("rush"))  
            count++;  
    return count;  
}
```

*izbroj sve sa većim
prioritetom od low?*

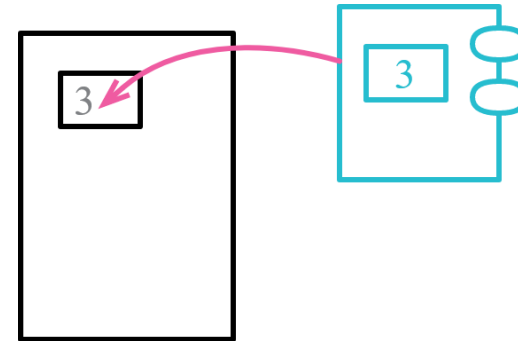
```
int countHigherPriority(Order[] orders, Priority p) {  
    int count = 0;  
    for(Order o: orders)  
        if(o.prio.higherThan(p))  
            count++;  
    return count;  
}
```

*šta sadrži
Priority klasa?*

Ekstrakcija klase iz primitiva

- ❖ Osnovni tipovi domena
- ❖ Grupa primitiva koje često „idu zajedno“

```
class Coordinates  
class Range  
class Money
```

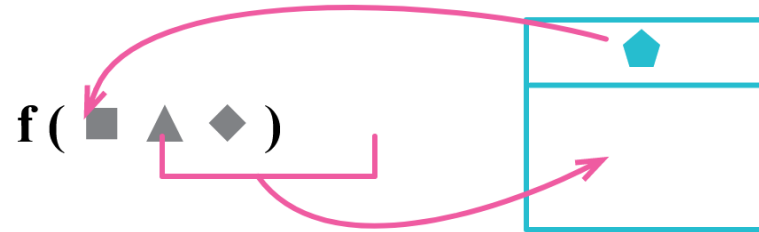
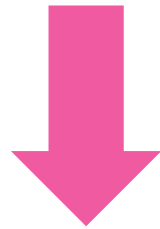


- ❖ Posebno obratiti pažnju na *String*

Ekstrakcija klase iz parametra funkcije

❖ Grupa podataka koji često „idu zajedno“

```
double amountInvoiced(Date start, Date end) {...}  
double amountReceived(Date start, Date end) {...}  
double amountOverdue(Date start, Date end) {...}
```



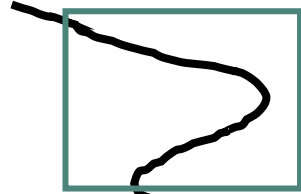
```
double amountInvoiced(DateRange range) {...}  
double amountReceived(DateRange range) {...}  
double amountOverdue(DateRange range) {...}
```

Ekstrakcija klase iz funkcije

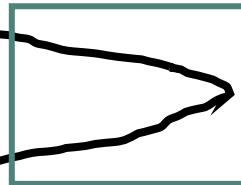
```
importModel(...) {
```

```
  Object A;
```

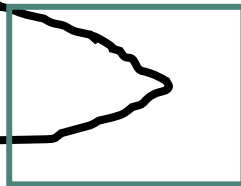
```
  Object B;
```



A

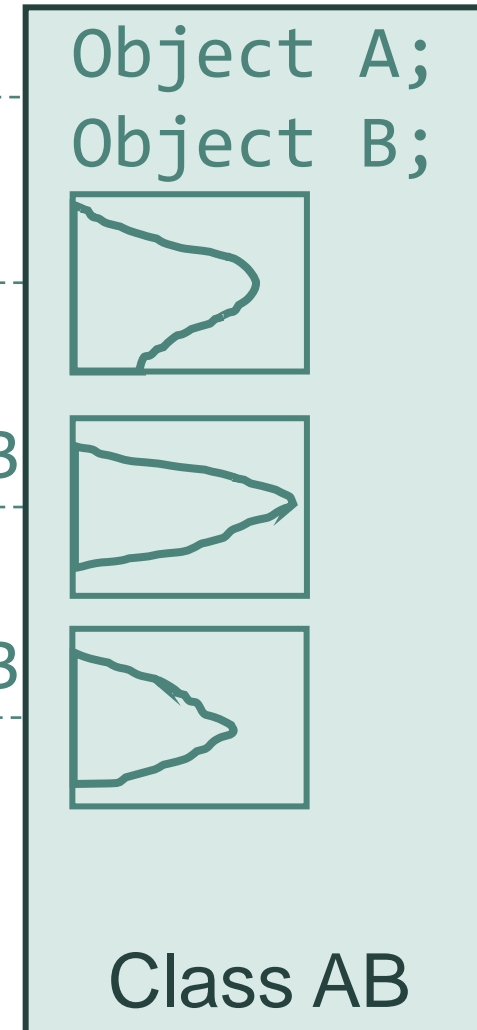


A, B



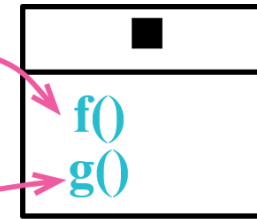
A, B

```
}
```



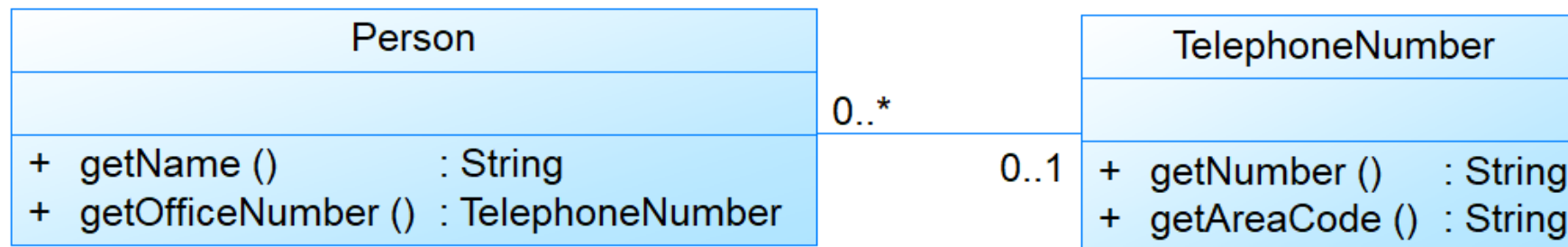
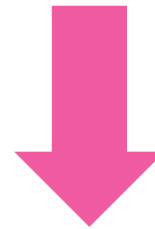
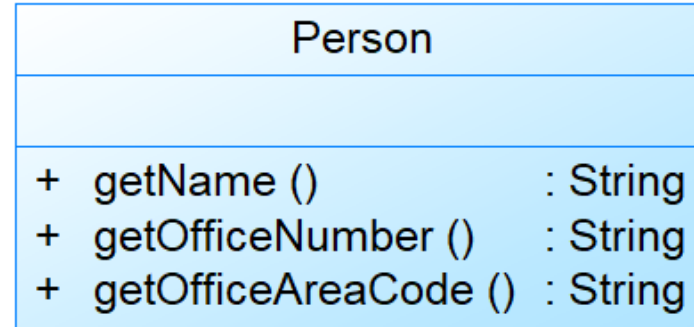
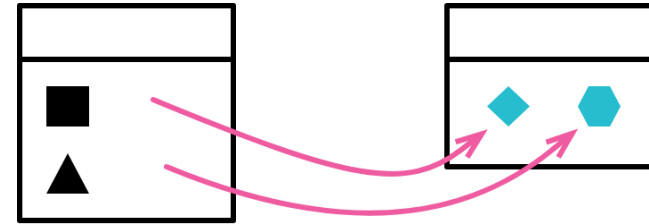
$f(\blacksquare)$

$g(\blacksquare)$



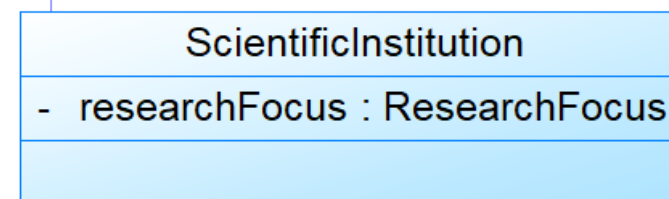
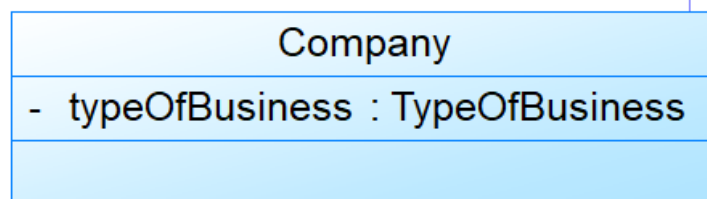
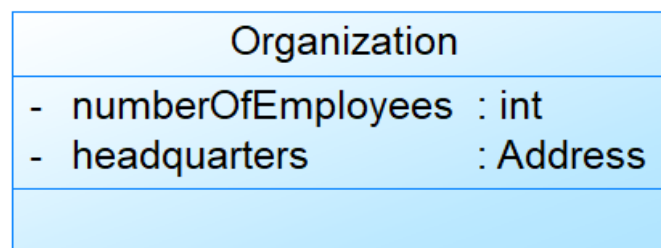
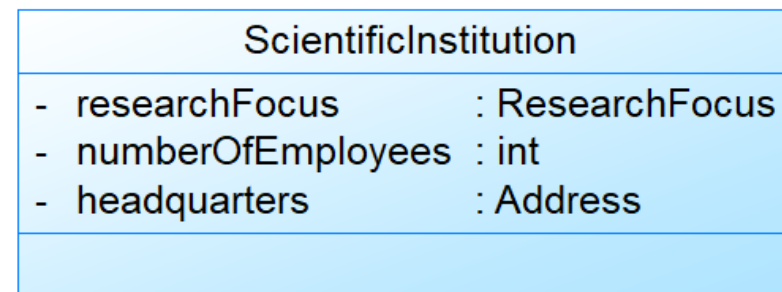
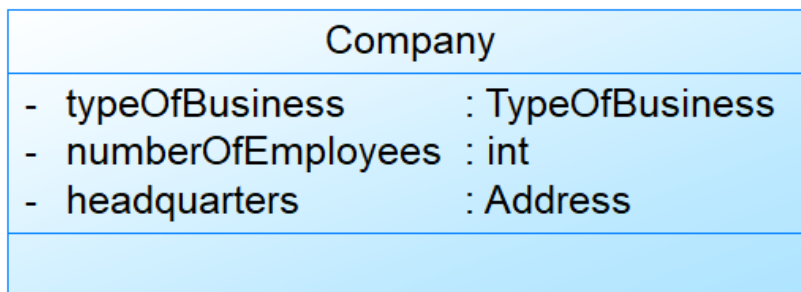
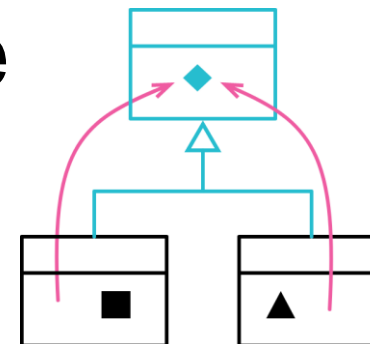
Ekstrakcija klasa iz klase

❖ Podela odgovornosti



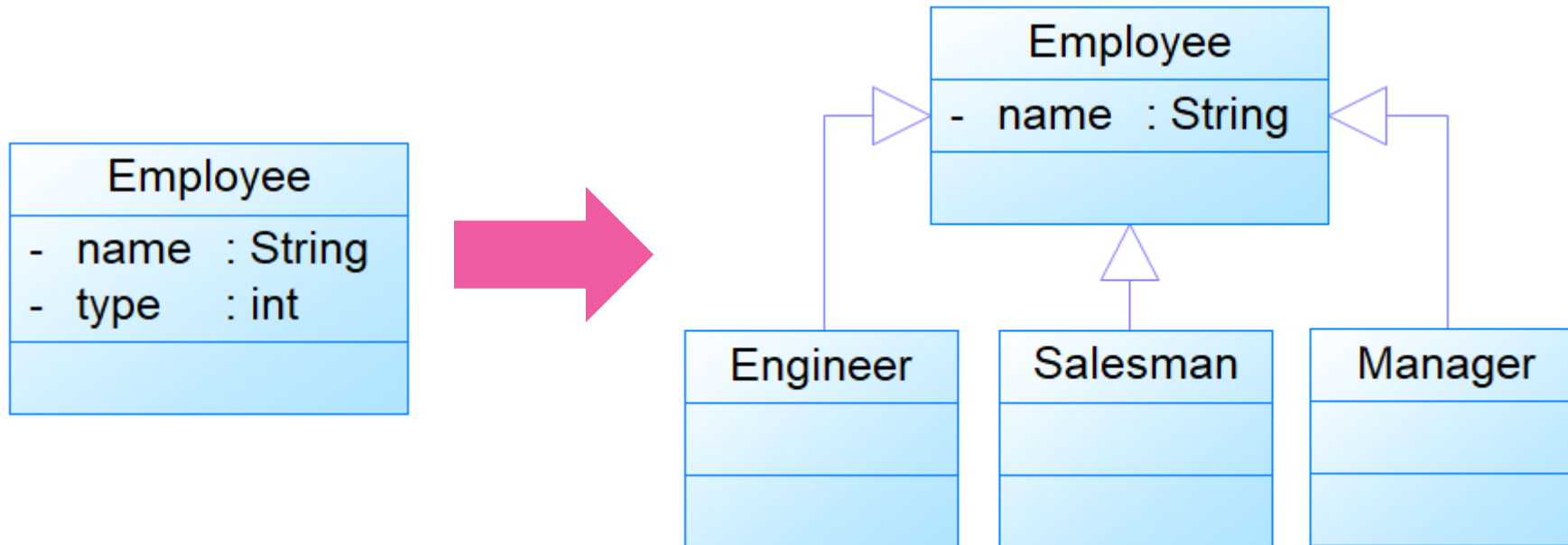
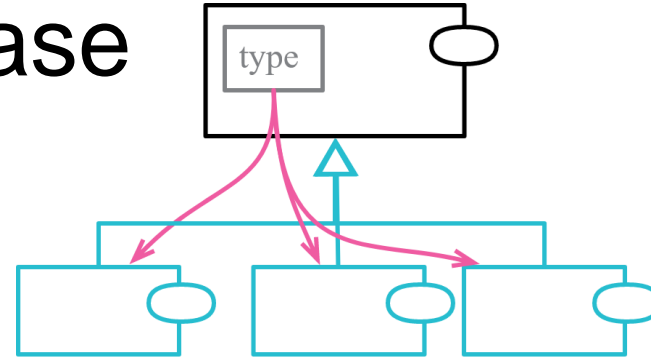
Ekstrakcija klasa iz klase

- ❖ Podela odgovornosti
- ❖ Razrada hijerarhije



Ekstrakcija klasa iz klase

- ❖ Podela odgovornosti
- ❖ Razrada hijerarhije



- ❖ Polimorfizam umesto *switch* bloka

Šta su svojstva dobro napisane klase?



Konstruiši klase koje imaju jednu odgovornost

Jedna
odgovornost

Značajan naziv

Visoka kohezija

Niska sprega ka
ostatku sistema



Analiziraj spregnutost klase sa okolnim klasama



Analiziraj strukturalnu koheziju klase



Analiziraj semantičku koheziju klase

Klasa treba da ima visoku koheziju

- ❖ Stepen upotrebe polja od strane metoda
- ❖ Max kad sve metode koriste sva polja

```
class Stack {  
    int topOfStack = 0;  
    List<Object> elements = new LinkedList<Object>();  
  
    int size() { return topOfStack; }  
    void push(int element) {  
        topOfStack++;  
        elements.add(element);  
    }  
}
```

*visoka ili niska
kohezija?*

Klasa treba da ima visoku koheziju

- ❖ Stepen upotrebe polja od strane metoda
- ❖ Max kad sve metode koriste sva polja

```
class Example {
```

```
    Object A;
```

```
    Object B;
```

```
    Object C;
```

```
    void methodA()    // uses A
```

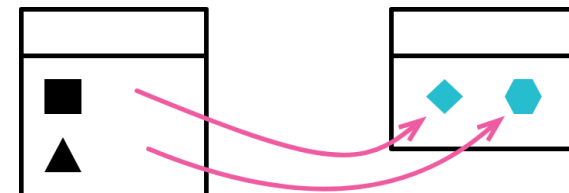
```
    void methodBA()  // uses B and A
```

```
    void methodC1()  // uses C
```

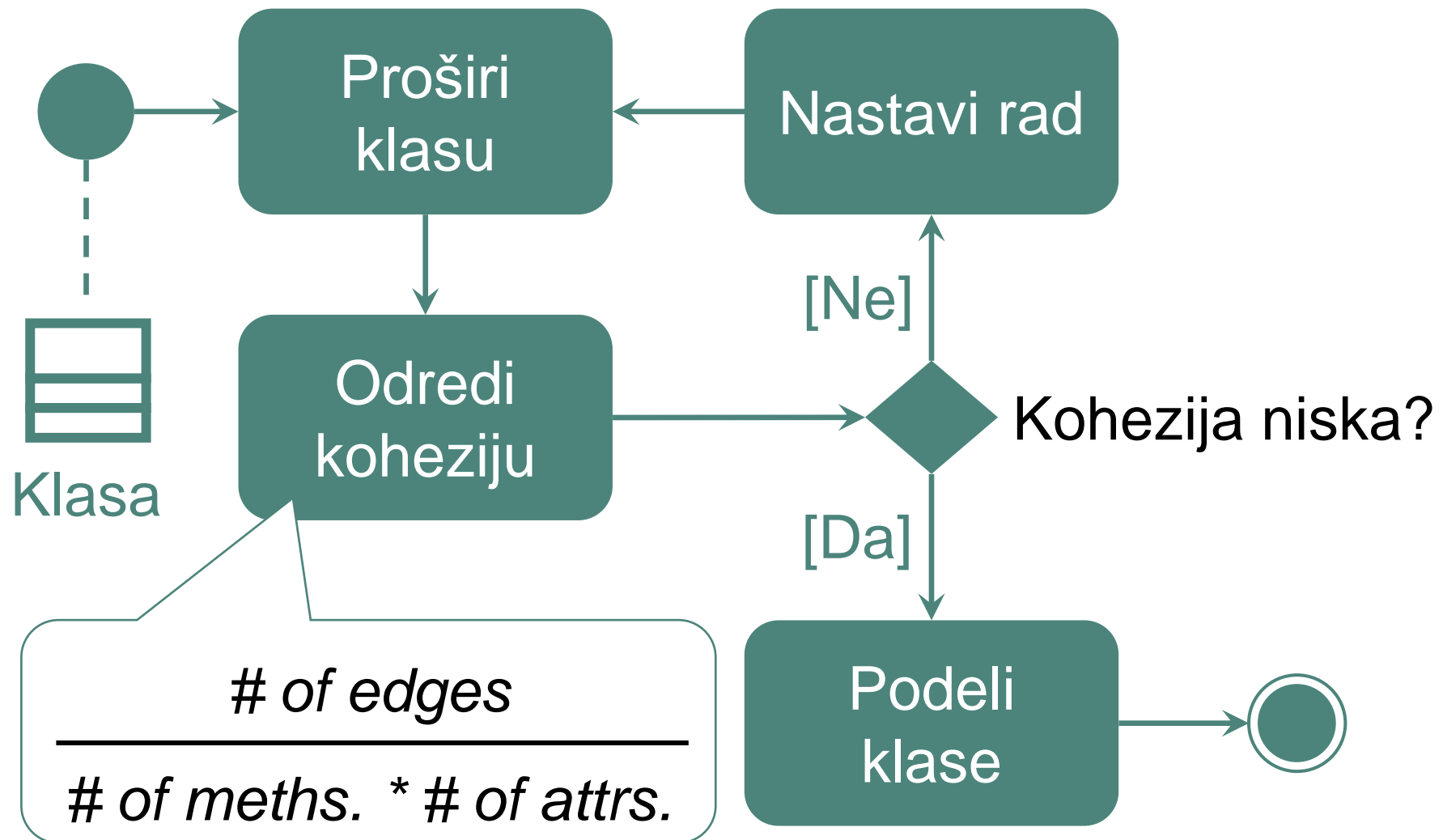
```
    void methodC2()  // uses C
```

```
}
```

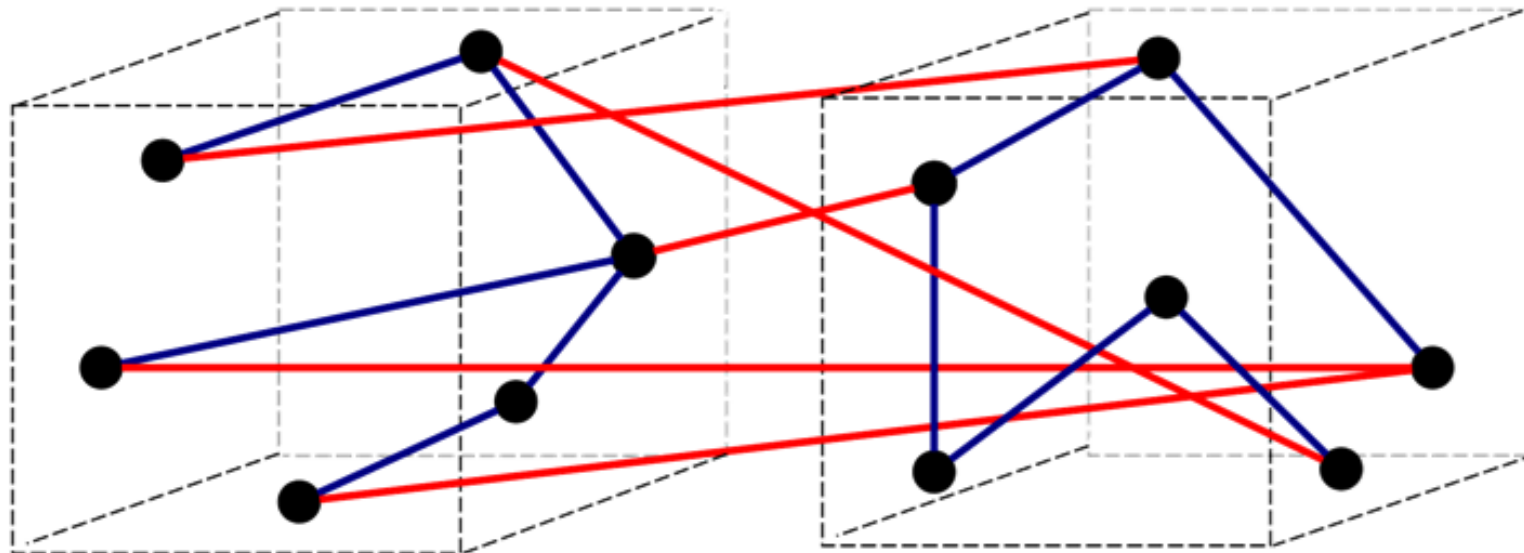
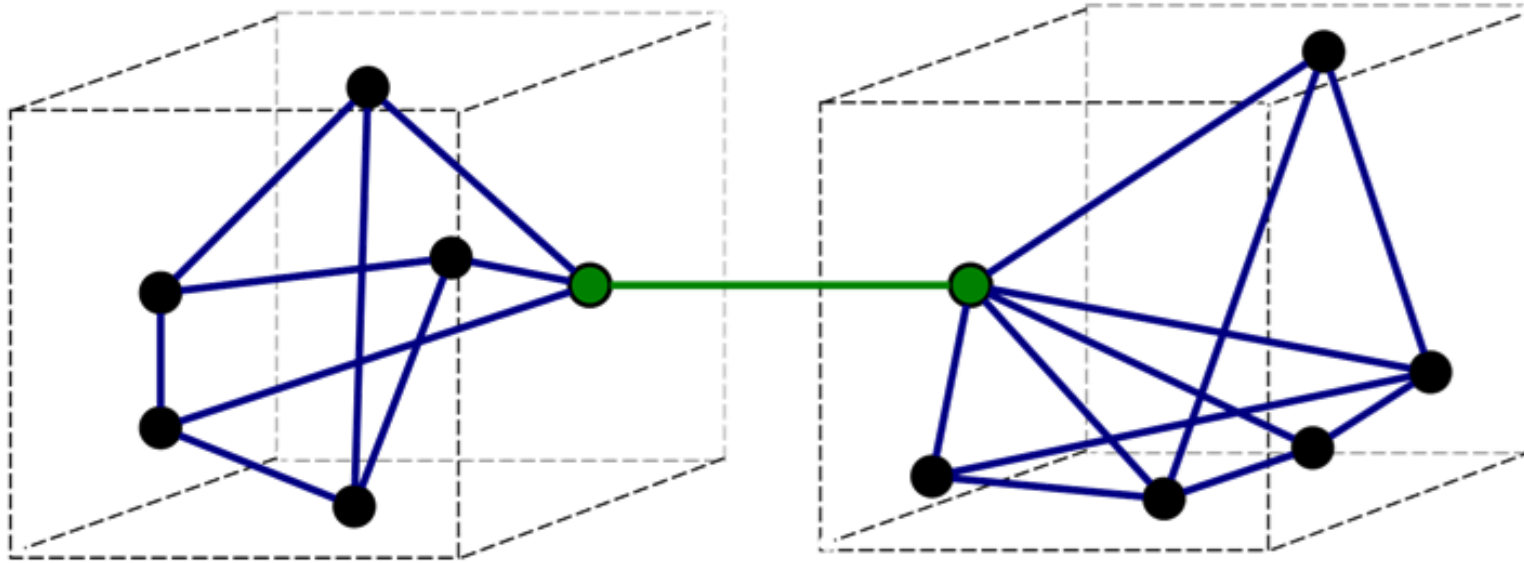
*visoka ili niska
kohezija?*



Kako održavati visoku koheziju klase?



Klasa treba da je slabo spregnuta sa ostatkom sistema



Klasa treba ima jednu odgovornost

Odgovornost = briga o detaljima

```
public class SuperDashboard extends JFrame {  
    Class[] getDataBaseClasses()  
    MetadataFeeder getMetadataFeeder()  
    void addProject(Project project)  
    boolean setCurrentProject(Project project)  
    boolean removeProject(Project project)  
    Project loadProject(String projectName)  
    MetaProjectHeader getProgramMetadata()  
    void resetDashboard()  
}
```

Klasa treba imati jednu odgovornost

Odgovornost = briga o detaljima

```
public class SuperDashboard extends JFrame {  
    Component getLastFocusedComponent()  
    void setLastFocused(Component lastFocused)  
    int getMajorVersionNumber()  
    int getMinorVersionNumber()  
}
```

Repo?

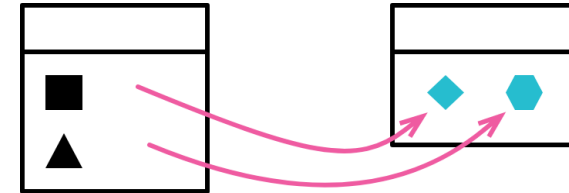
Opiši klasu u do 25 reči bez „i“, „ili“, „ako“

- ❖ *SuperDashboard* vodi računa o poslednje pristupljenoj komponenti i verziji aplikacije

Klasa treba imati jednu odgovornost

Odgovornost = briga o detaljima

```
public class SuperDashboard extends JFrame {  
    Component getLastFocusedComponent()  
    void setLastFocused(Component lastFocused)  
}  
  
public class Version  
    int getMajorVersionNumber()  
    int getMinorVersionNumber()  
}
```



Skupi stvari koje se menjaju iz istih razloga
Razdvoj one koje se menjaju iz različitih

Employee

```
+ calculatePay()      : double  
+ save()              : void  
+ reportEmployee()   : String  
+ findById(int id)   : Employee
```

broj odgovornosti?

što je problem?

Skupi stvari koje se menjaju iz istih razloga
Razdvoj one koje se menjaju iz različitih

Employee

*kako raspakovati
da prati SRP?*

```
+ calculatePay()      : double  
+ save()              : void  
+ reportEmployee()   : String  
+ findById(int id)   : Employee
```

Šta bi uzrokovalo promenu ove klase?

❖ Izmena skladišta podataka

❖ Izmena izveštavanja

❖ Izmena računice plate

*ko naručuje
ove promene?*

Skupi stvari koje se menjaju iz istih razloga

Razdvoj one koje se menjaju iz različitih

*šta menja
nova baza?*

Employee

EmployeeRepository

```
+ save(Employee e) : void  
+ findById(int id) : Employee
```

PaycheckCalculator

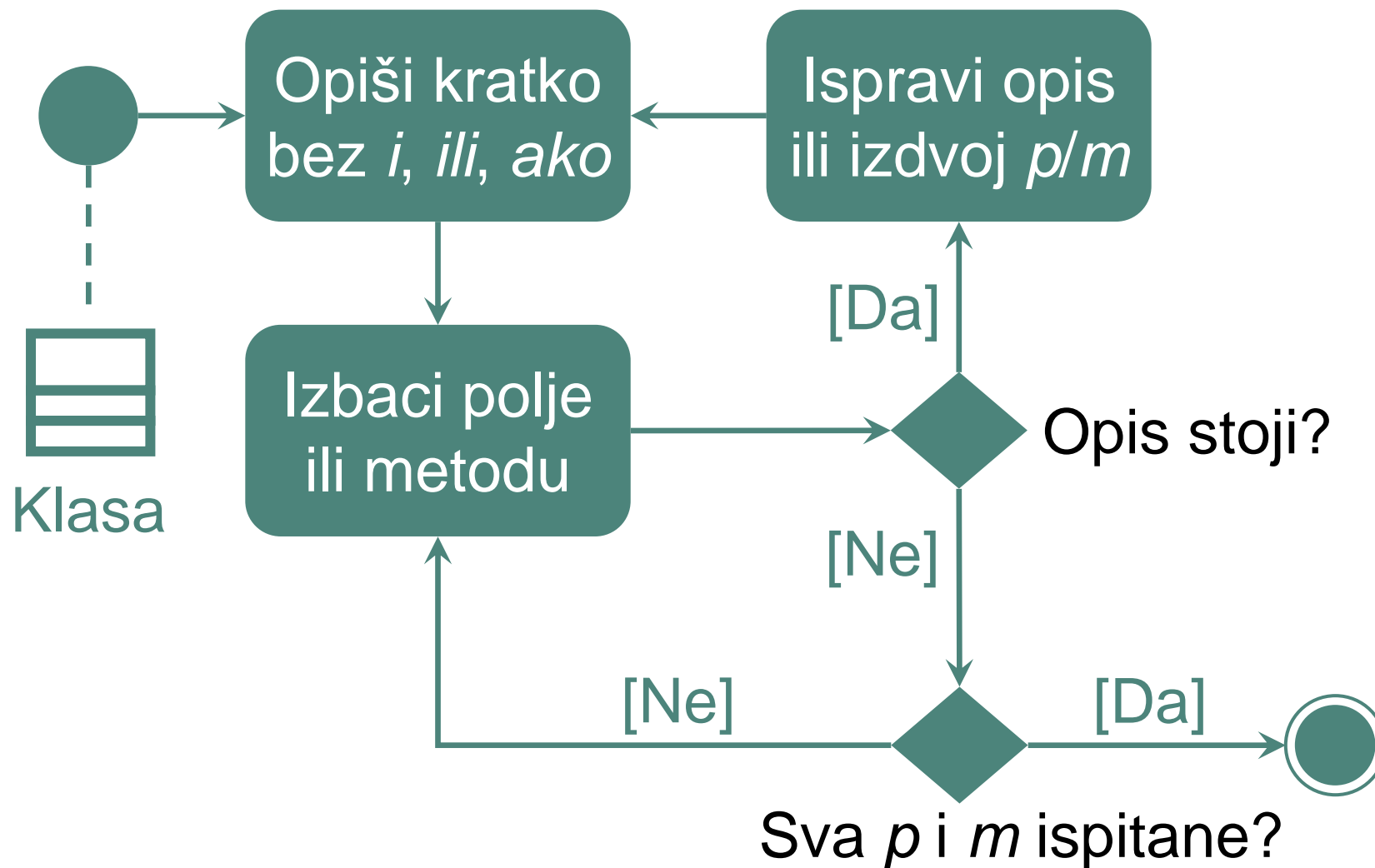
```
+ calculatePay(Employee e) : double
```

*šta menja
nov izveštaj?*

EmployeeReporter

```
+ report(Employee e) : String
```

Skupi stvari koje se menjaju iz istih razloga
Razdvoj one koje se menjaju iz različitih



Skupi stvari koje se menjaju iz istih razloga
Razdvoj one koje se menjaju iz različitih

SRP je primenljiv na razne apstrakcije

Klase

Paketi

Mikroservisi

SRP donosi više prednosti

Povećana
čitljivost

Olakšane
izmene

Manja šansa
za runtime
greške

Šta su svojstva dobro napisane klase?



Konstruiši klase koje imaju jednu odgovornost

Jednu
odgovornost

Značajan naziv

Visoku koheziju

Nisku spregu ka
ostatku sistema



Analiziraj spregnutost klase sa okolnim klasama



Analiziraj strukturalnu koheziju klase

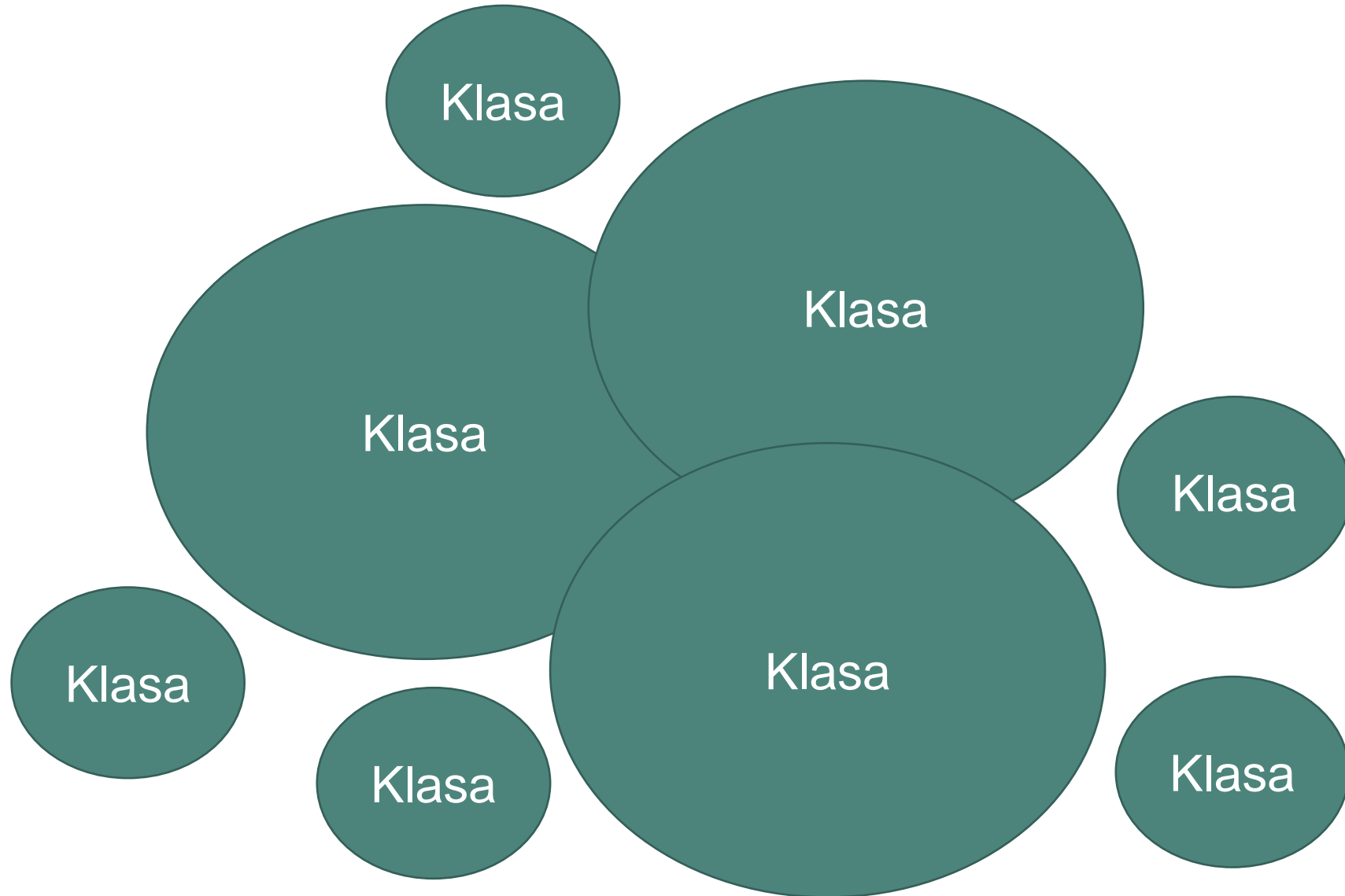


Analiziraj semantičku koheziju klase

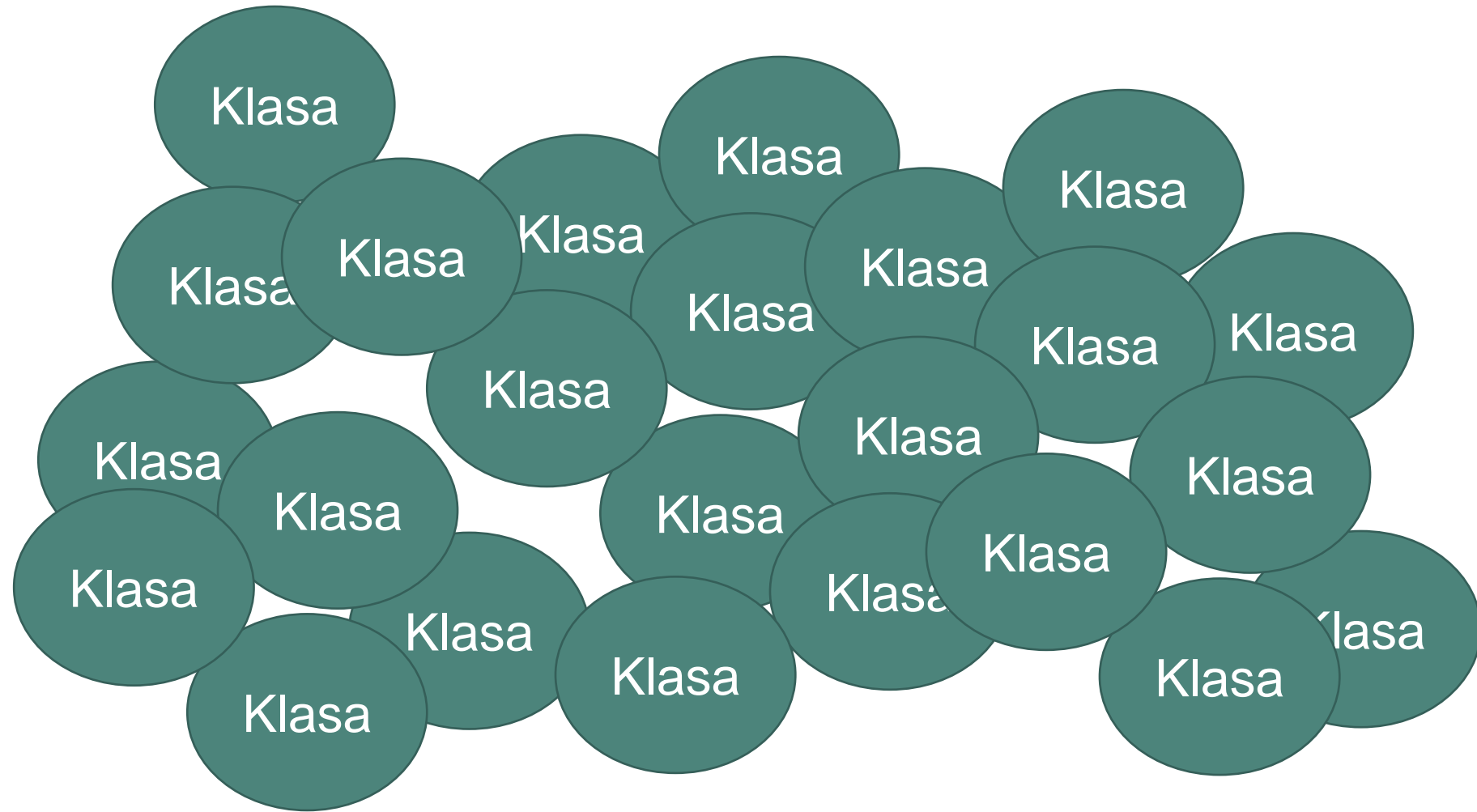
Šta su posledice dobro napisane klase?



Šta su posledice dobro napisane klase?

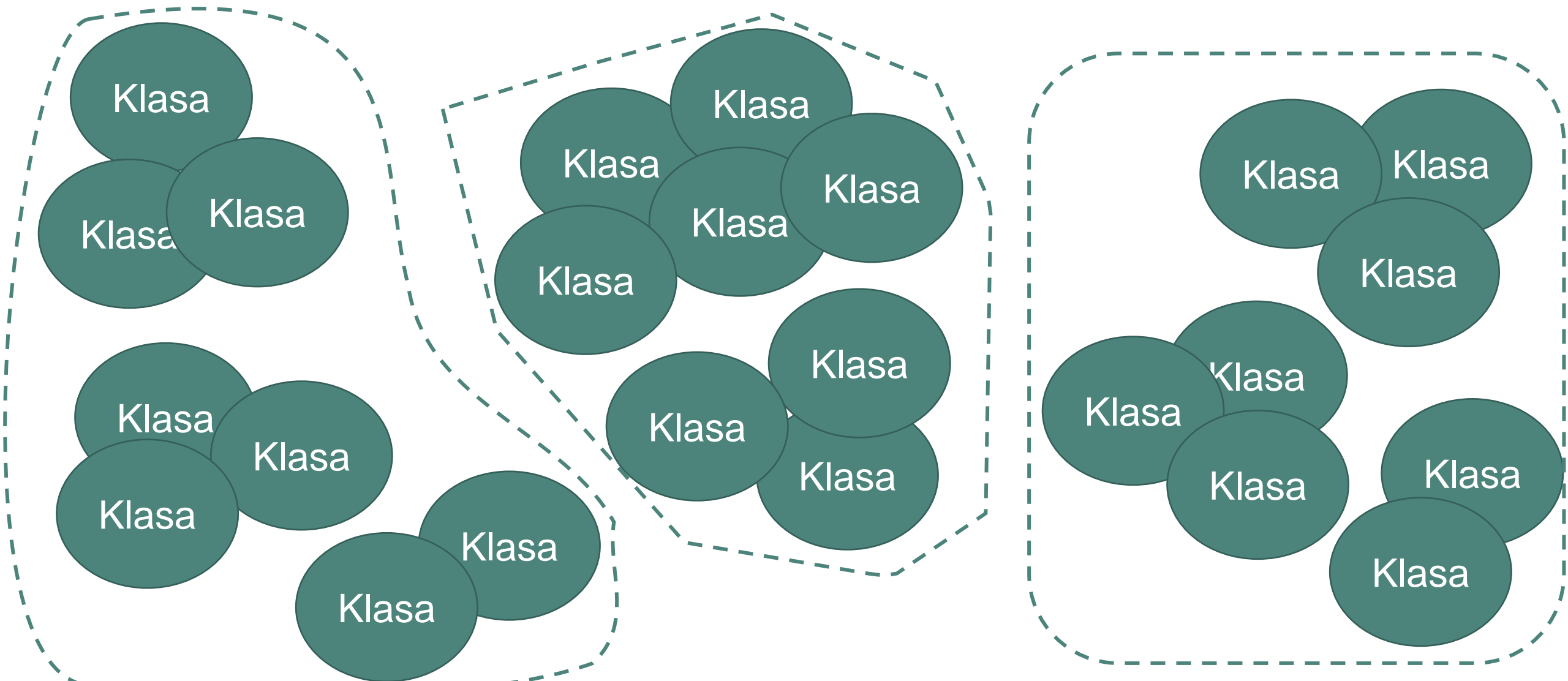


Šta su posledice dobro napisane klase?



Kompleksnost je neizbežna ako smo ozbiljni

Šta su posledice dobro napisane klase?



Kompleksnost je neizbežna ako smo ozbiljni

SRP

OCP

LSP

ISP

DIP

Šta je SOLID?

- **S**ingle Responsibility Principle
- **O**pen-Closed Principle
- **L**iskov Substitution Principle
- **I**nterface Segregation Principle
- **D**ependency Inversion Principle

Šta nije SOLID?

- Gospodar
- *Framework*, biblioteka, šablon
- *Technology-specific*

Apstrakcije ne zavise od implementacije, implementacije zavise od apstrakcija

```
class EmployeeReporter {  
    void reportEmployees() {  
        List<Employee> ems = new ArrayList<>();  
        String query = "select * from Employees";  
        try {  
            Statement stmt = con.createStatement();  
            ResultSet rs = stmt.executeQuery(query);  
            while (rs.next())  
                ems.push(compileEmployee(rs));  
            printEmployees(ems);  
        } catch (Exception e) {...}  
    }  
}
```

*odakle da
počnemo?*

Apstrakcije ne zavise od implementacije, implementacije zavise od apstrakcija

```
class EmployeeReporter {  
    void reportEmployees() {  
        EmployeeDB edb = new EmployeeDB();  
        printEmployees(edb.getEmployees());  
    }  
}
```

*Znak da se
DIP ne prati*

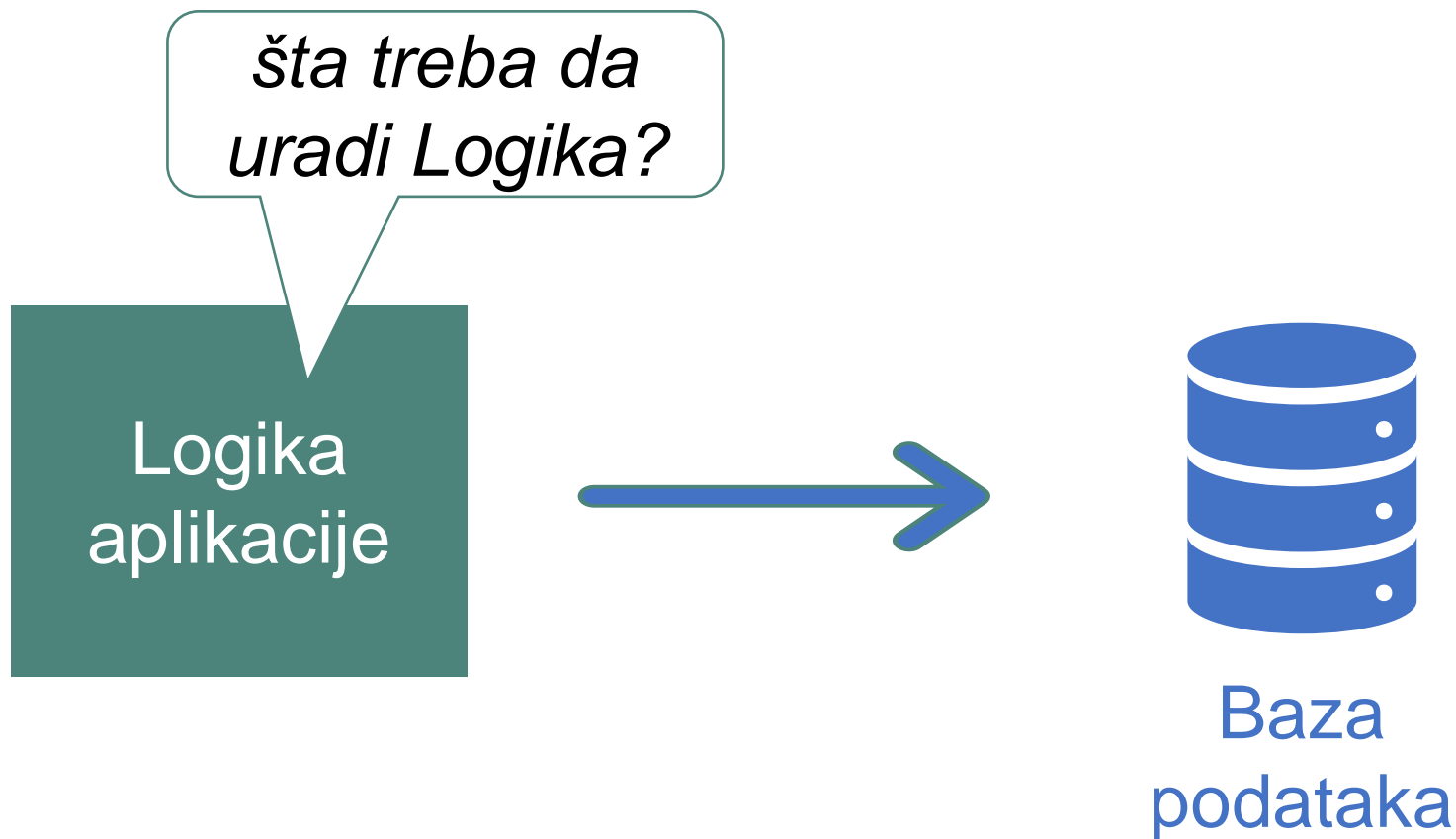
*da li Reporter
zavisi od DB?*

```
class EmployeeReporter {  
    void reportEmployees(IEmployeeRepo er) {  
        printEmployees(er.getEmployees());  
    }  
}
```

*šta su
prednosti?*

SRP

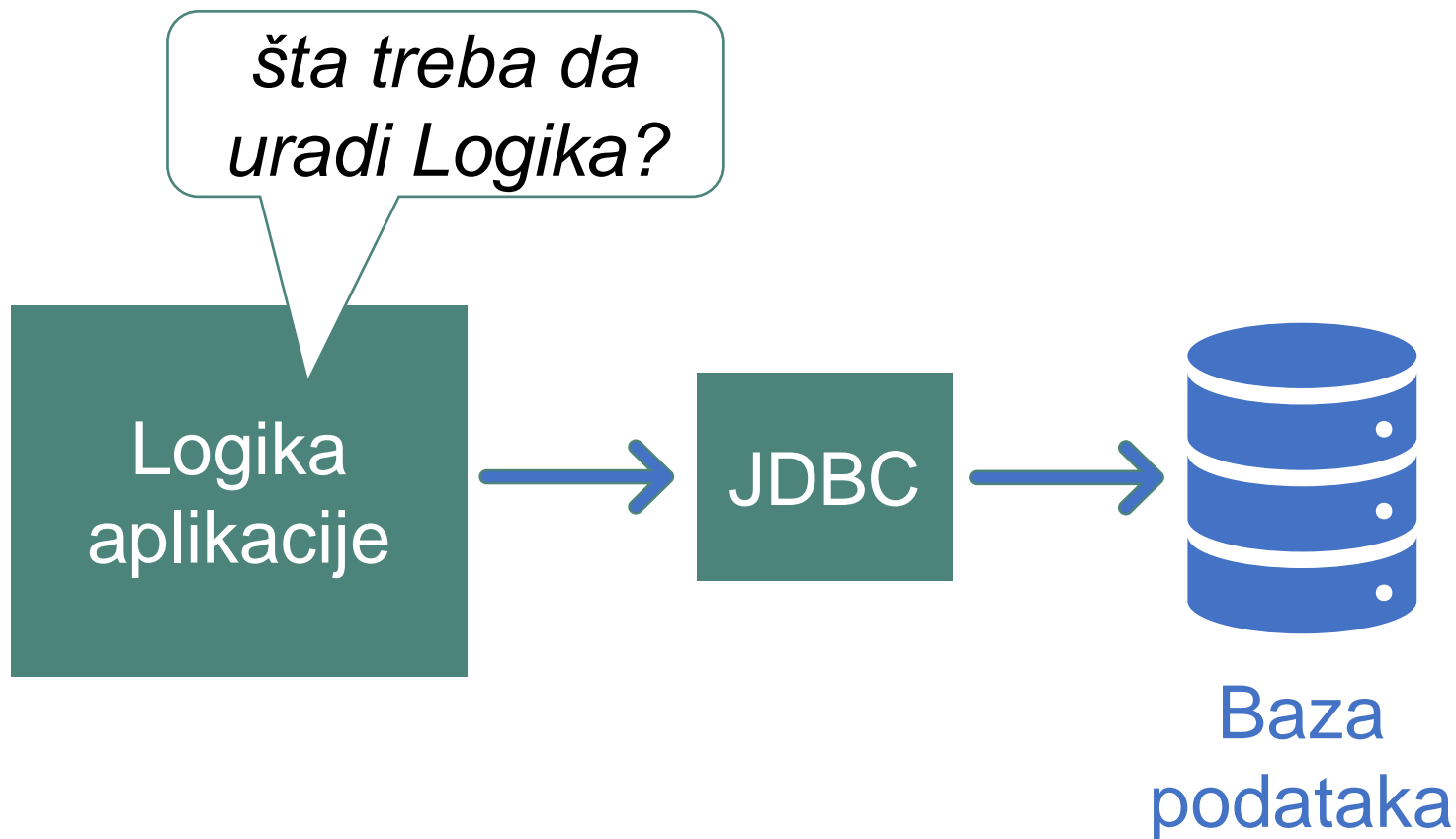
Apstrakcije ne zavise od implementacije,
implementacije zavise od apstrakcija



DIP

SRP

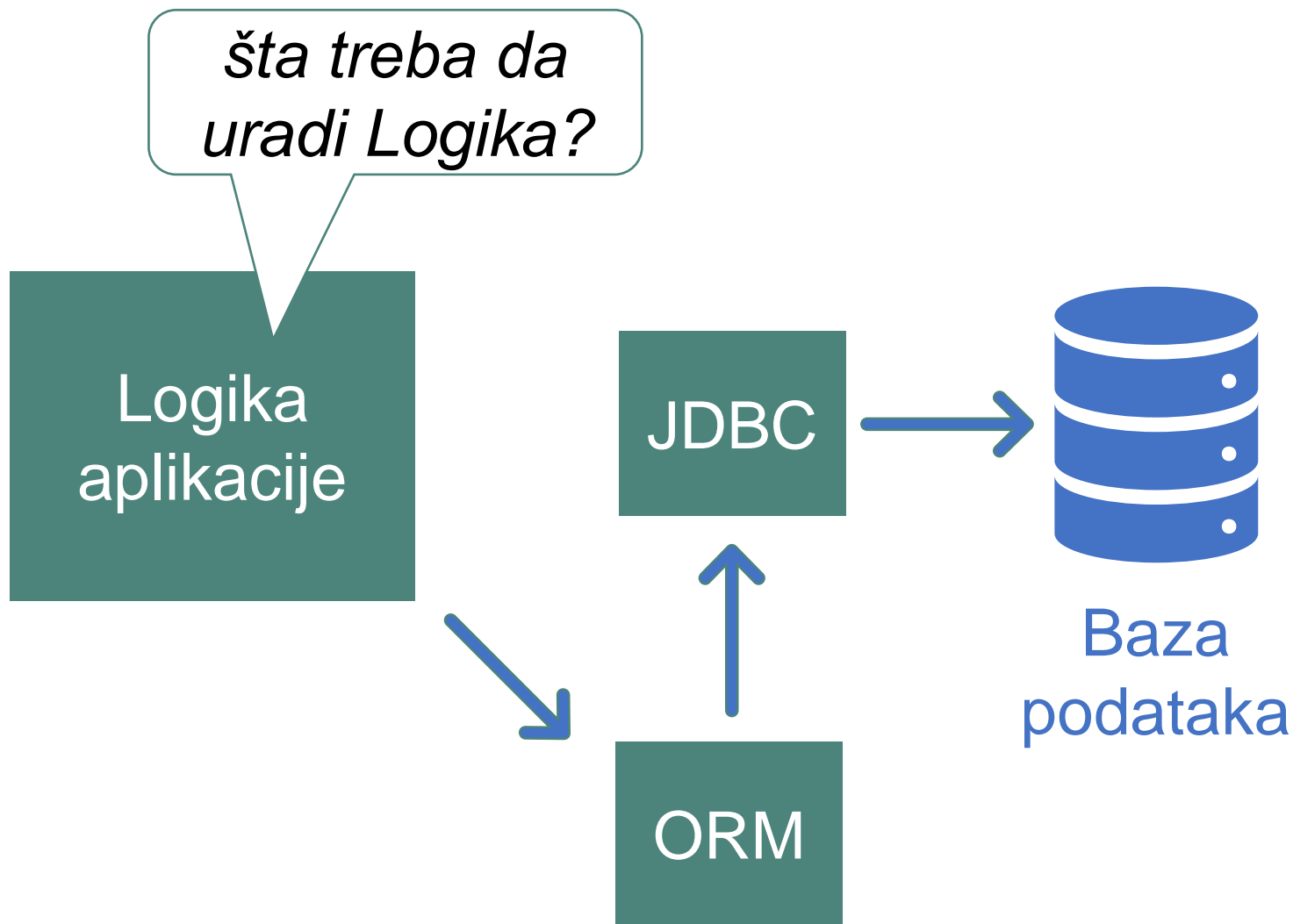
Apstrakcije ne zavise od implementacije,
implementacije zavise od apstrakcija



DIP

SRP

Apstrakcije ne zavise od implementacije,
implementacije zavise od apstrakcija

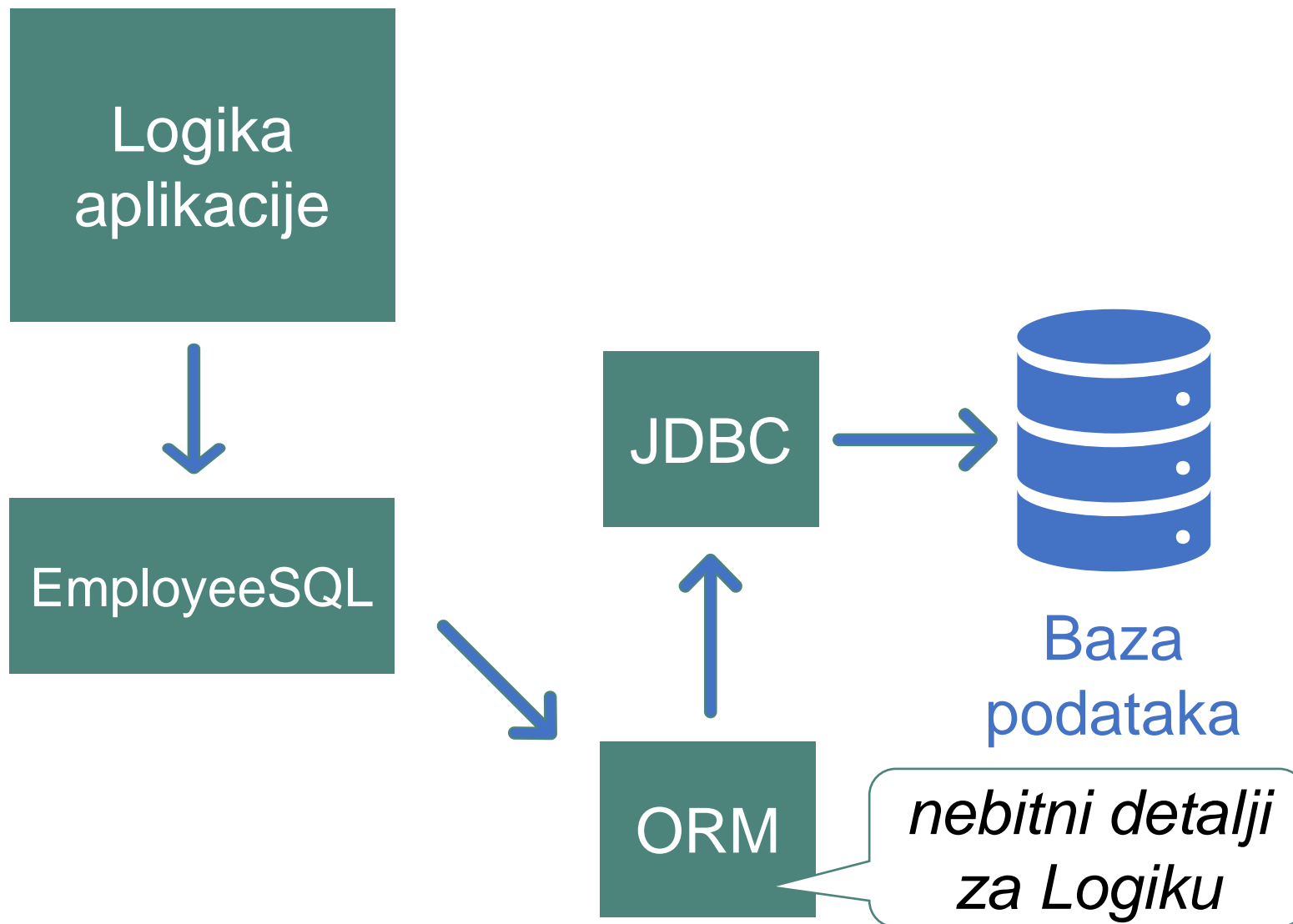


DIP

SRP

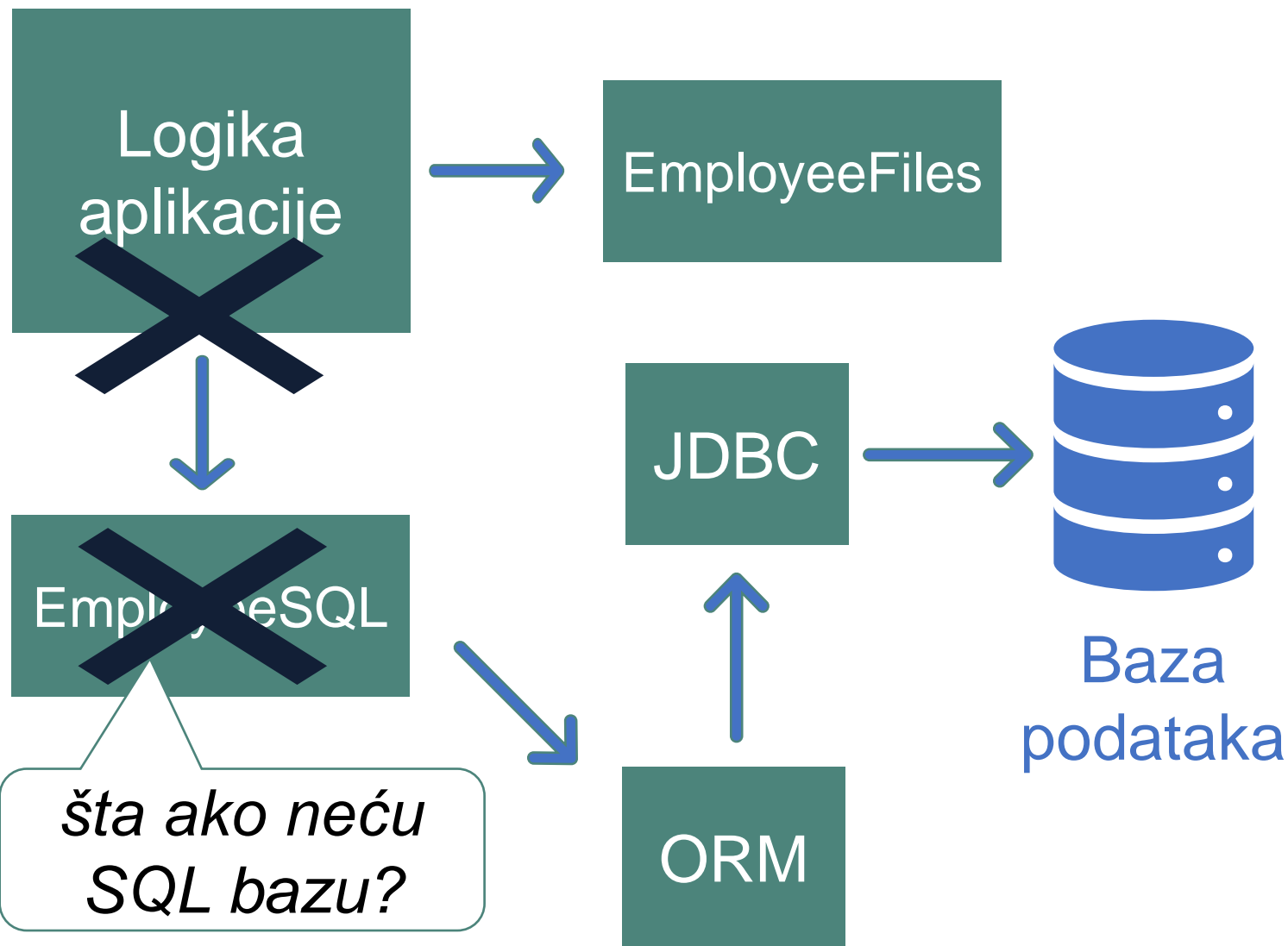
Apstrakcije ne zavise od implementacije,
implementacije zavise od apstrakcija

DIP



SRP

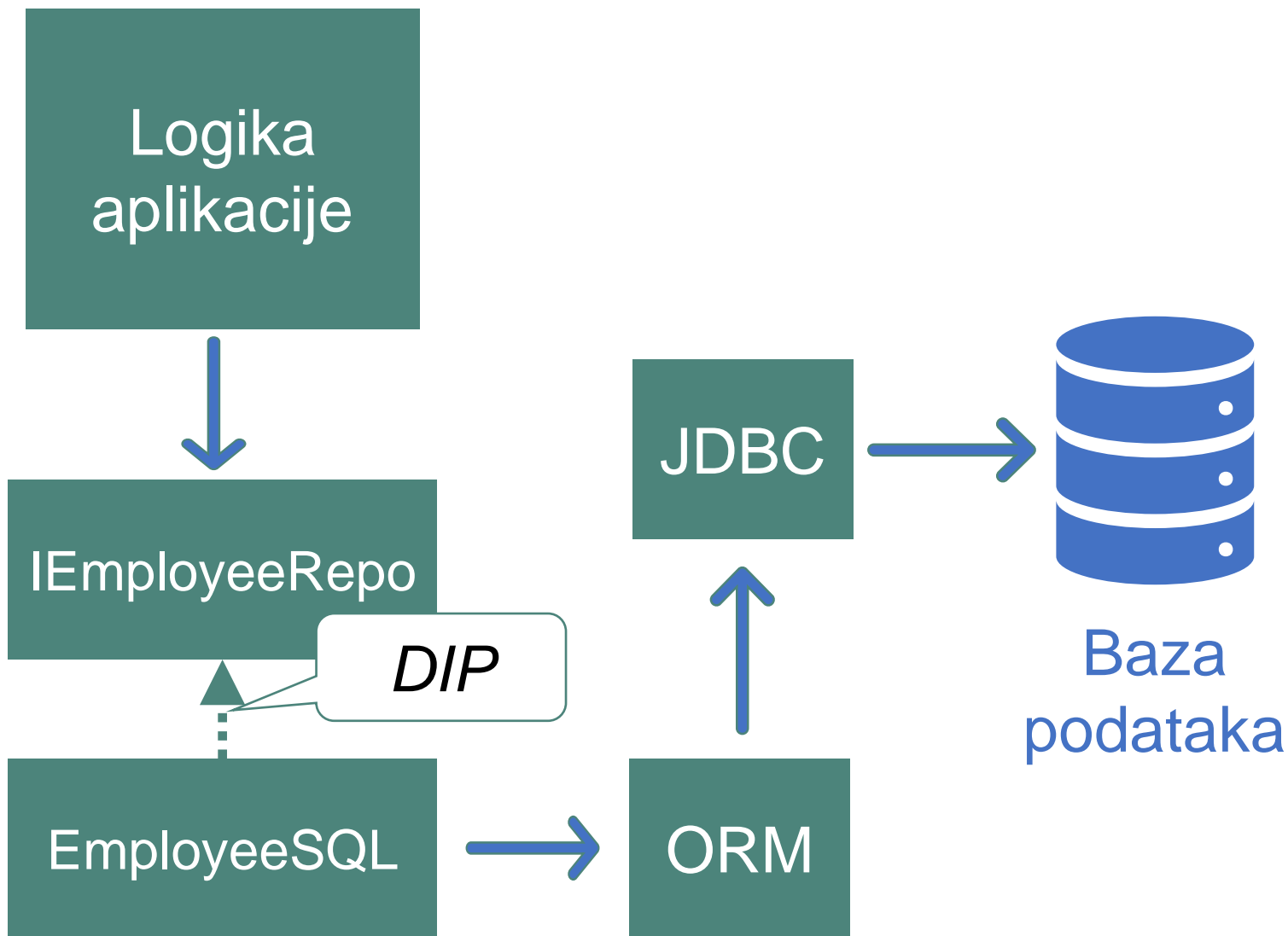
Apstrakcije ne zavise od implementacije,
implementacije zavise od apstrakcija



DIP

SRP

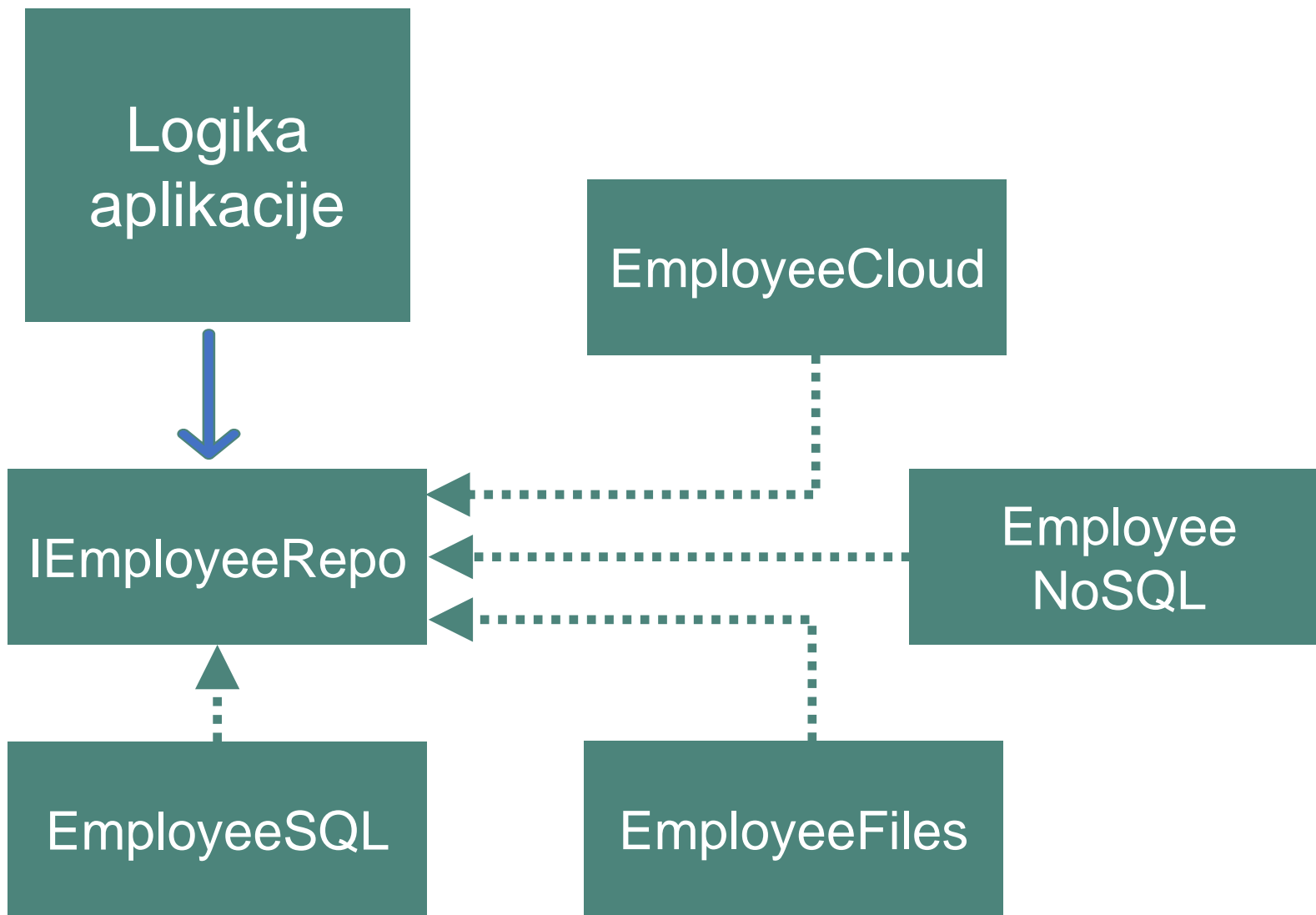
Apstrakcije ne zavise od implementacije,
implementacije zavise od apstrakcija



DIP

SRP

Apstrakcije ne zavise od implementacije,
implementacije zavise od apstrakcija



DIP

Apstrakcije ne zavise od implementacije,
implementacije zavise od apstrakcija

Dependency injection

- ❖ Šablon koji pomaže da se ostvari *DIP*
- ❖ Opskrbi modul sa onim što mu je potrebno
- ❖ Nivo konstruktora, polja (*setter*) i funkcije

```
class EmployeeReporter {  
    void reportEmployees(IEmployeeRepo er) {  
        printEmployees(er.getEmployees());  
    }  
}
```


Apstrakcije ne zavise od implementacije,
implementacije zavise od apstrakcija

Dependency injection

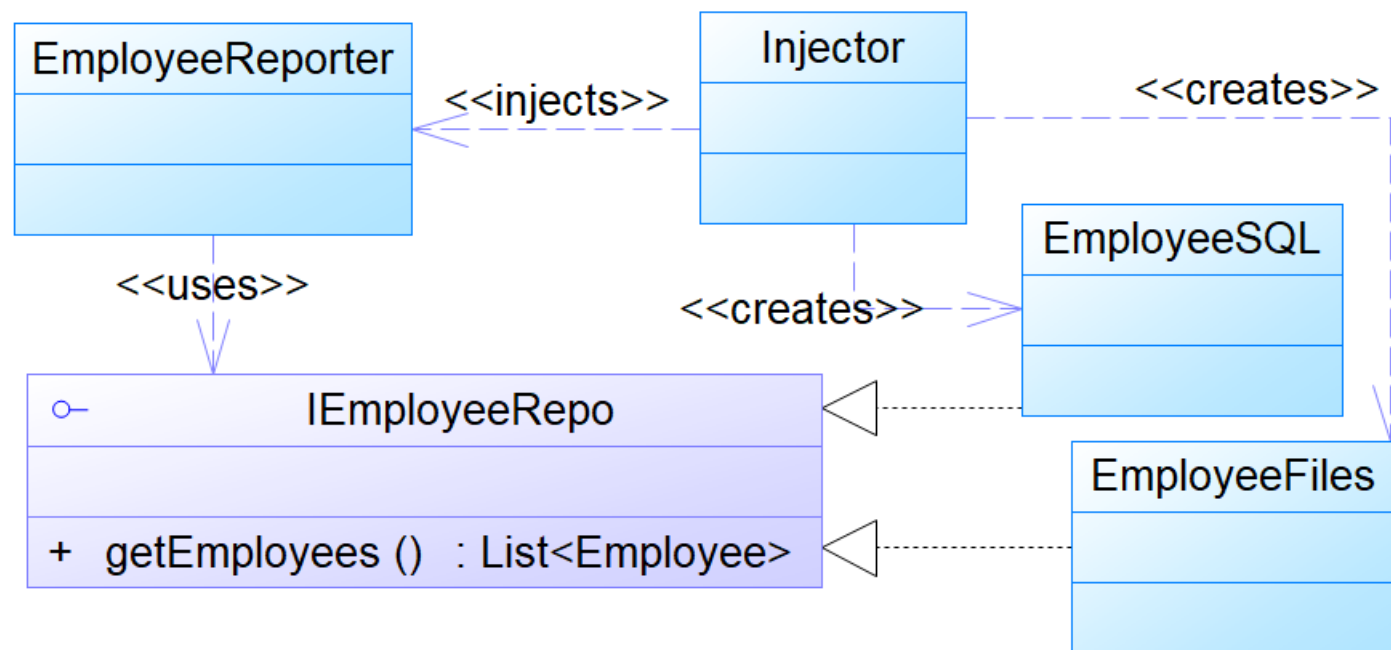
- ❖ Šablon koji pomaže da se ostvari *DIP*
- ❖ Opskrbi modul sa onim što mu je potrebno
- ❖ Nivo konstruktora, polja (*setter*) i funkcije

```
class EmployeeReporter {  
    IEmployeeRepo er;  
    public EmployeeReporter(IEmployeeRepo e) {  
        this.er = e;  
    }  
}
```

Apstrakcije ne zavise od implementacije,
implementacije zavise od apstrakcija

Dependency injection

- ❖ Šablon koji pomaže da se ostvari *DIP*
- ❖ Opskrbi modul sa onim što mu je potrebno
- ❖ Nivo konstruktora, polja (*setter*) i funkcije



Apstrakcije ne zavise od implementacije,
implementacije zavise od apstrakcija

DIP omogućuje *loose coupling*

Među klasama

Među
komponentama

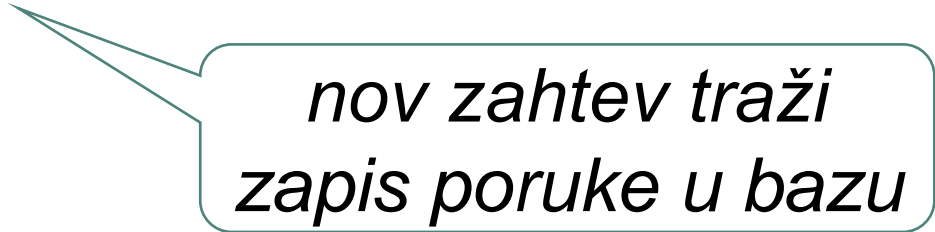
DIP i DI su osnovni mehanizmi za ono što
zovemo proširivost u OO jezicima

SRP

OCP

Ponašanje modula je moguće menjati bez da se menja njegov izvorni kod

```
public class Logger {  
    void Log(String message, String logType) {  
        switch(logType) {  
            case "Console":  
                System.out.println(message);  
                break;  
            case "File":  
                // Code to write message to file  
                break;  
        }  
    }  
}
```



*nov zahtev traži
zapis poruke u bazu*

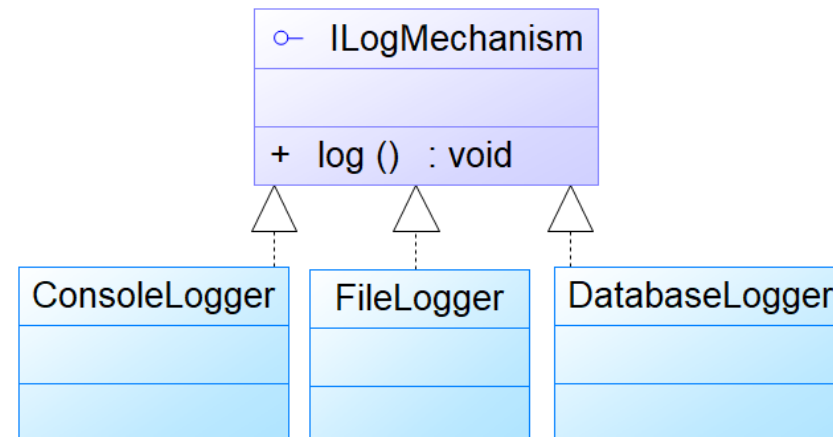
DIP

SRP

OCP

Ponašanje modula je moguće menjati bez da se menja njegov izvorni kod

```
public class Logger {  
    private ILogMechanism logMechanism;  
    public Logger(ILogMechanism lm) {  
        this.logMechanism = lm;  
    }  
    public void Log(String message) {  
        this.logMechanism.log(message);  
    }  
}
```



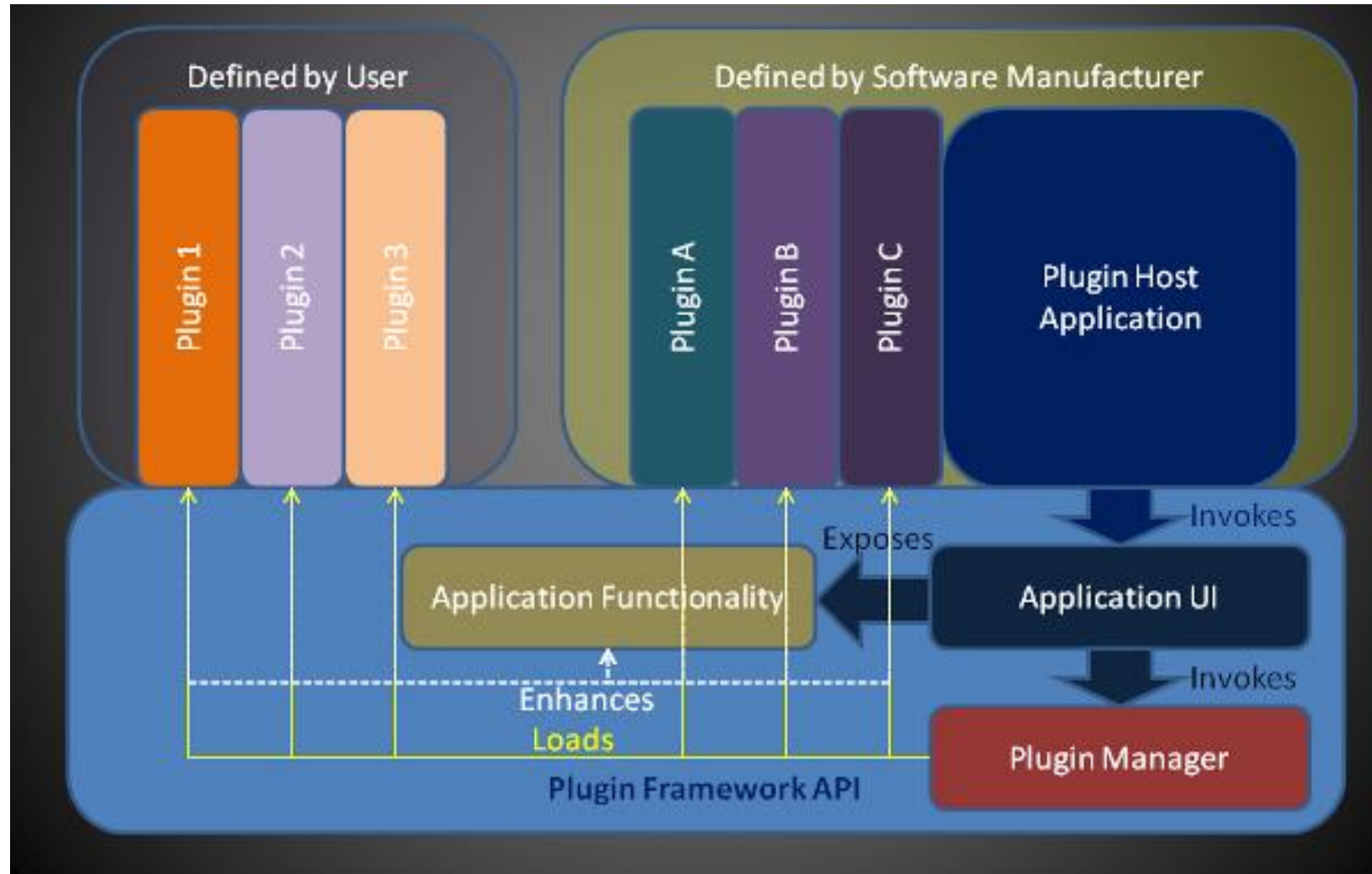
DIP

SRP

OCP

Ponašanje modula je moguće menjati
bez da se menja njegov izvorni kod

DIP



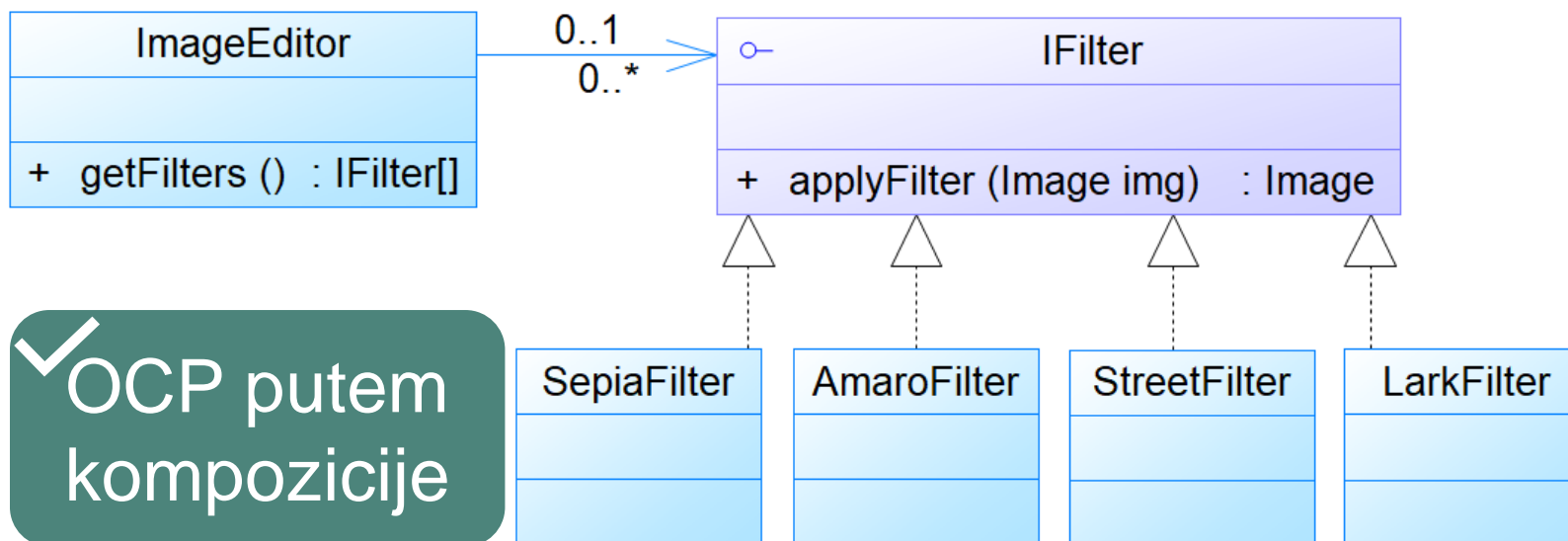
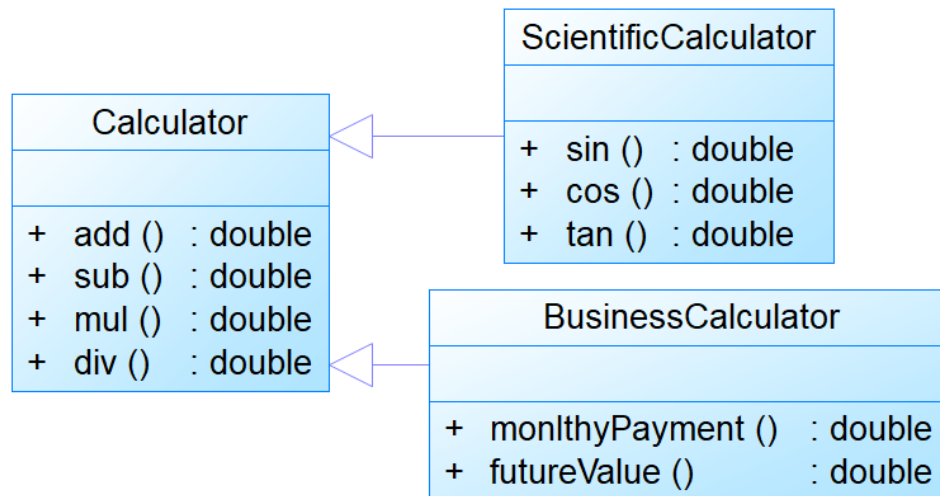
SRP

OCP

Ponašanje modula je moguće menjati bez da se menja njegov izvorni kod

seal by default

OCP putem nasleđivanja



✓ OCP putem kompozicije

DIP

SRP

OCP

Ponašanje modula je moguće menjati
bez da se menja njegov izvorni kod

OCP zahteva procenu

Šta konkretizovati

Šta apstrahovati

OCP donosi više prednosti

Fleksibilnost

Ponovnu
iskoristivost

Olakšano
održavanje

DIP

SRP

OCP

LSP

ISP

DIP

Šta je SOLID?

- **S**ingle Responsibility Principle
- **O**pen-Closed Principle
- **L**iskov Substitution Principle
- **I**nterface Segregation Principle
- **D**ependency Inversion Principle

Šta nije SOLID?

- Gospodar
- *Framework*, biblioteka, šablon
- *Technology-specific*

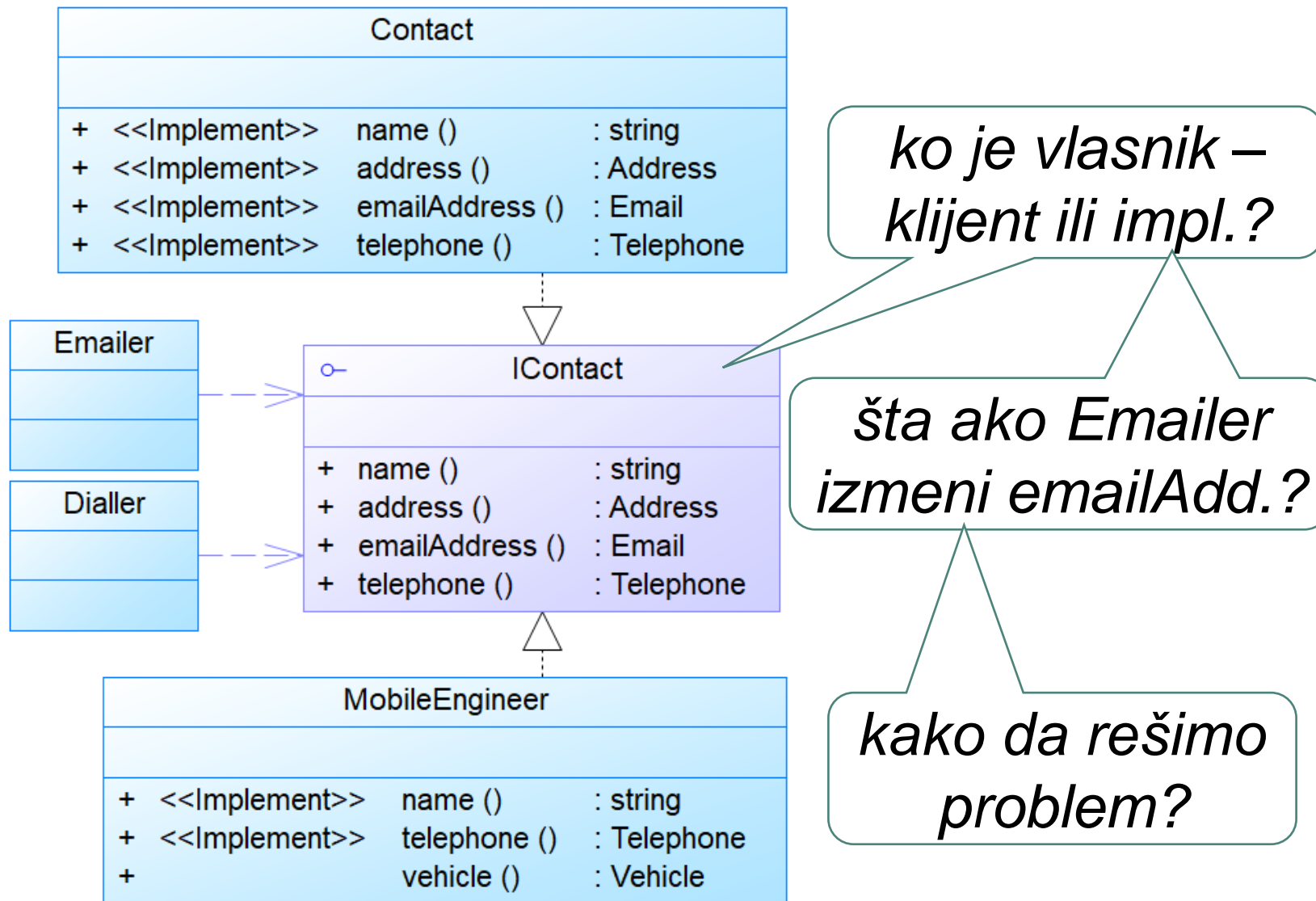
SRP

OCP

ISP

DIP

Interfejsi su tanki, da njihovi klijenti ne
zavise od metoda koje ne koriste



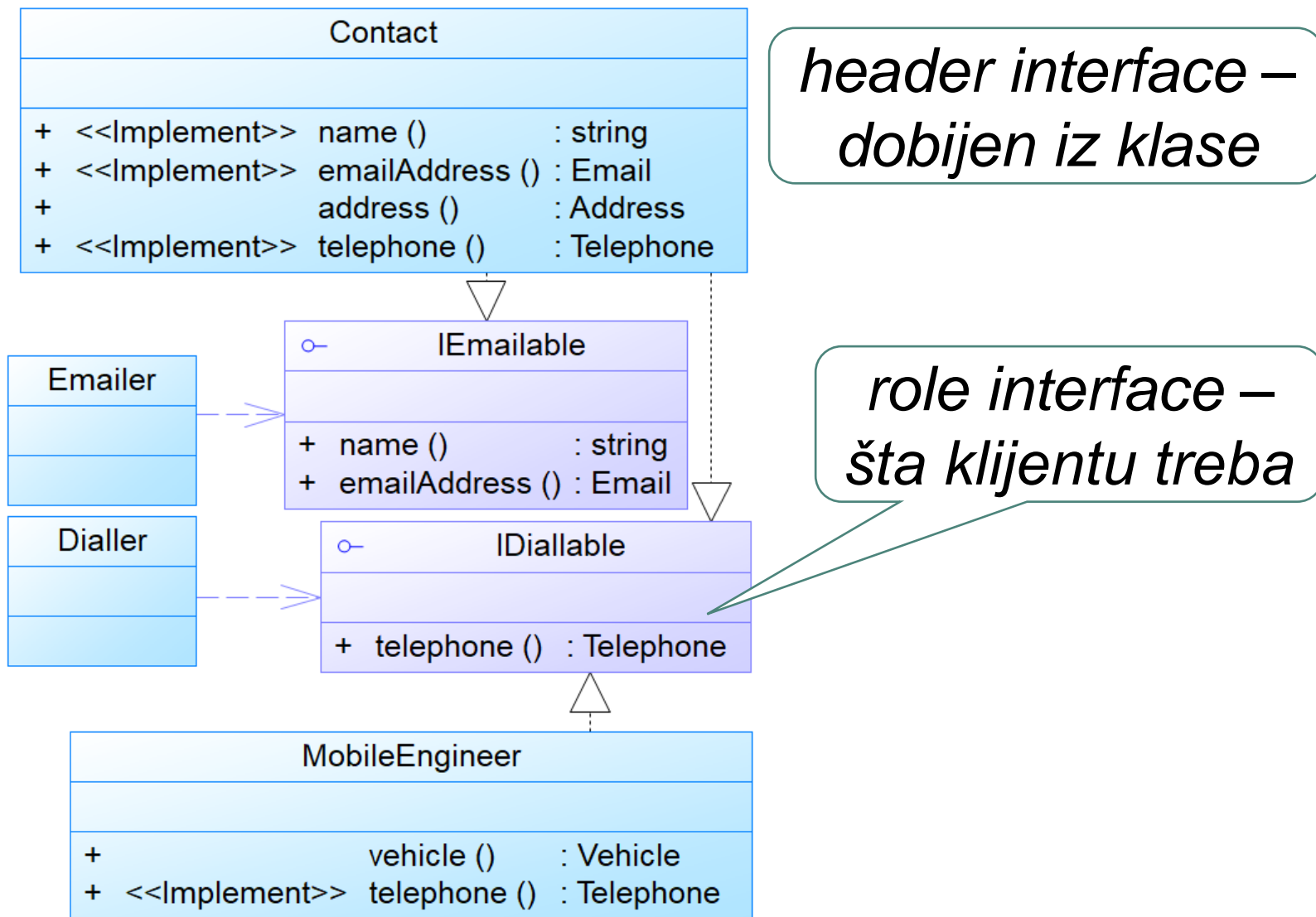
SRP

OCP

ISP

DIP

Interfejsi su tanki, da njihovi klijenti ne
zavise od metoda koje ne koriste



SRP

Interfejsi su tanki, da njihovi klijenti ne
zavise od metoda koje ne koriste

OCP

ISP podržava druge principe

Implementacija
„debelog“ interfejsa
narušava SRP

Implementacija dela
„debelog“ interfejsa
narušava LSP

ISP

ISP podstiče dobar dizajn

Implementacije

Klijenta

DIP

SRP

OCP

LSP

ISP

DIP

Podklase se koriste umesto nadklase bez da klijent nadklase uočava razliku

```
class Rectangle {  
    int height;  
    int width;  
  
    void setW(int w) {  
        this.width = w;  
    }  
    void setH(int h) {  
        this.height = h;  
    }  
    // Kod za getere  
}
```

```
class Square  
    extends Rectangle {  
    @Override  
    void setW(int w) {  
        this.width = w;  
        this.height = w;  
    }  
    @Override  
    void setH(int h) {  
        this.height = h;  
        this.width = h;  
    }  
}
```

SRP

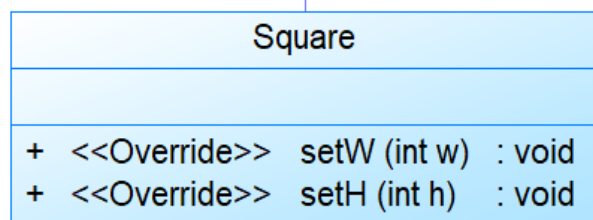
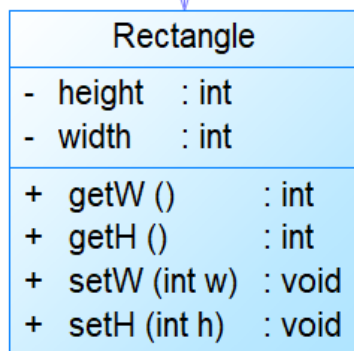
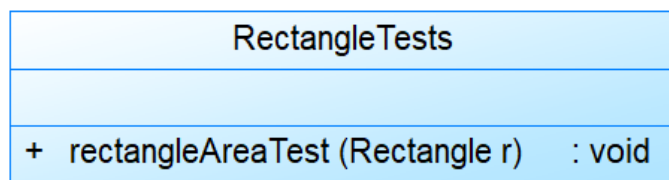
OCP

LSP

ISP

DIP

Podklase se koriste umesto nadklase bez da klijent nadklase uočava razliku



```
class RectangleTests {  
    void rectangleAreaTest(  
        Rectangle r) {  
        r.setW(5);  
        r.setH(2);  
        assert  
            r.getW()*r.getH() == 10;  
    }  
}
```

*šta dodati da
radi za oba?*

SRP

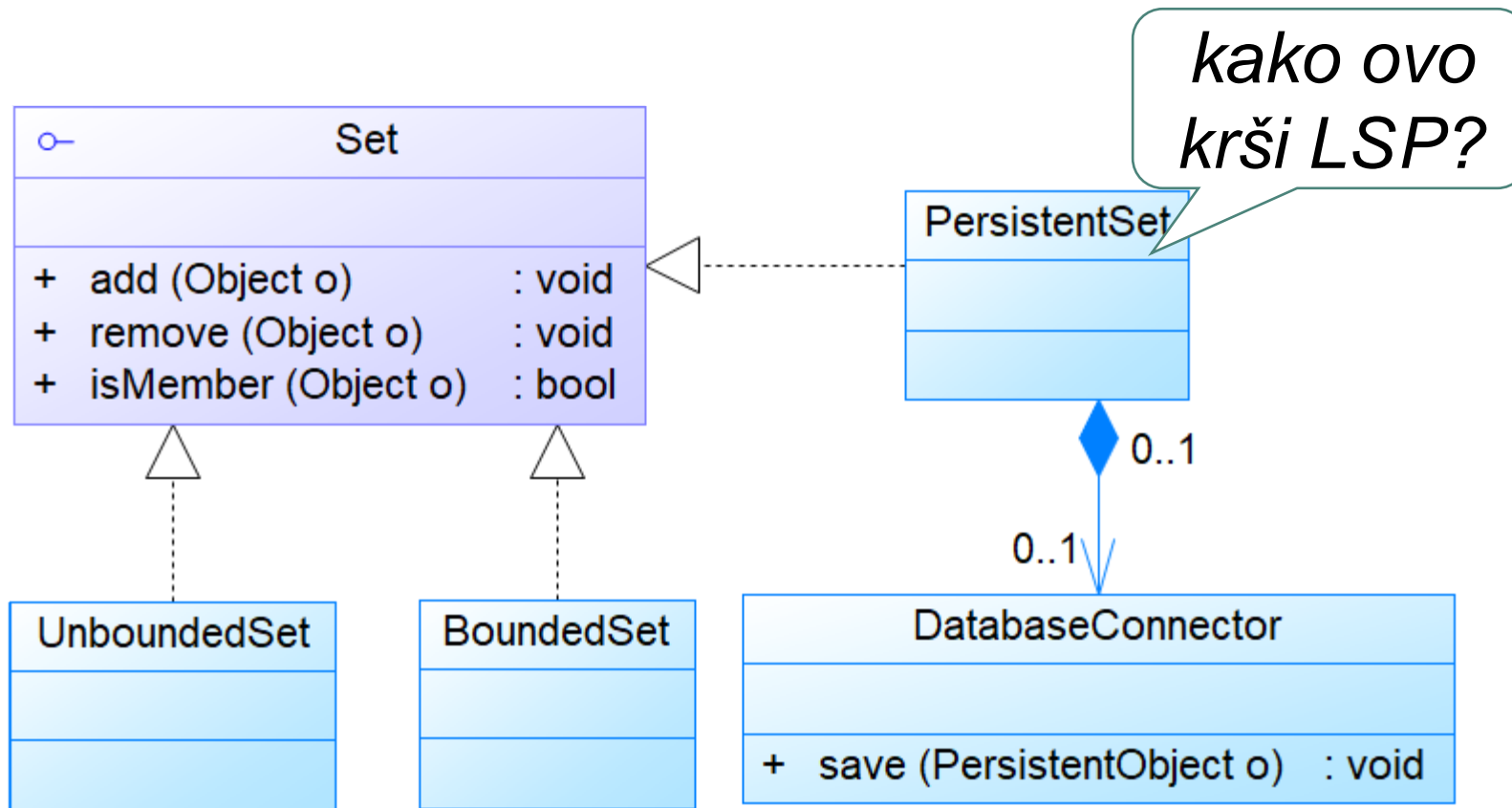
OCP

LSP

ISP

DIP

Podklase se koriste umesto nadklase bez
da klijent nadklase uočava razliku



SRP

OCP

LSP

ISP

DIP

Podklase se koriste umesto nadklase bez
da klijent nadklase uočava razliku

Kršenje LSP-a

- (Uglavnom) *Cast* u kodu klijenta nadklase (eksplicitan ili *instanceof*)
- Kada podklasa ima zahtevnije preduslove za izvršavanje ponašanja koje nasleđuje
- Kada podklasa ima slabije postuslove kao rezultat ponašanja koje nasleđuje
- Kada podklasi nije potreban deo metoda i polja nadklase (*refused bequest*)

SRP

Podklase se koriste umesto nadklase bez
da klijent nadklase uočava razliku

OCP

LSP zahteva brigu o apstrakciji

Da li *override* menja
pred i post uslove
koje očekuju klijenti

Da li je data klasa
potomak nadklase

LSP

LSP donosi više prednosti

Ispravnost
polimorfizma

Smislene
hijerarhije

ISP

DIP

SRP

OCP

LSP

ISP

DIP

Šta je SOLID?

- **S**ingle Responsibility Principle
- **O**pen-Closed Principle
- **L**iskov Substitution Principle
- **I**nterface Segregation Principle
- **D**ependency Inversion Principle

Šta nije SOLID?

- Gospodar
- *Framework*, biblioteka, šablon
- *Technology-specific*