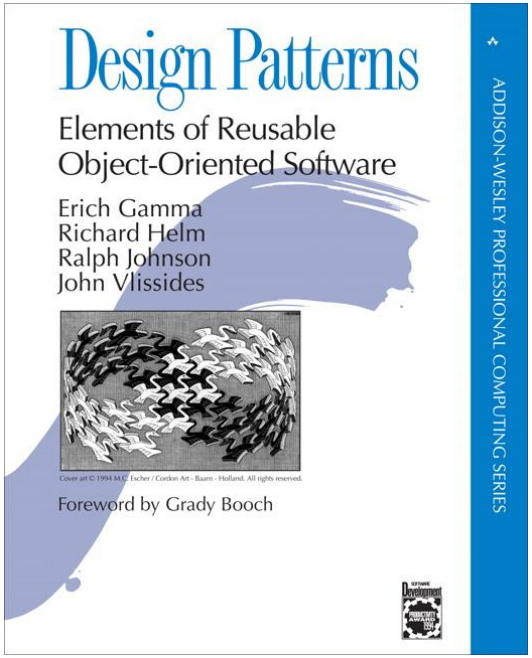
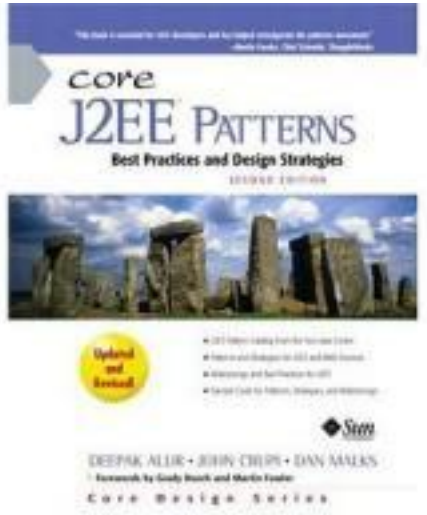
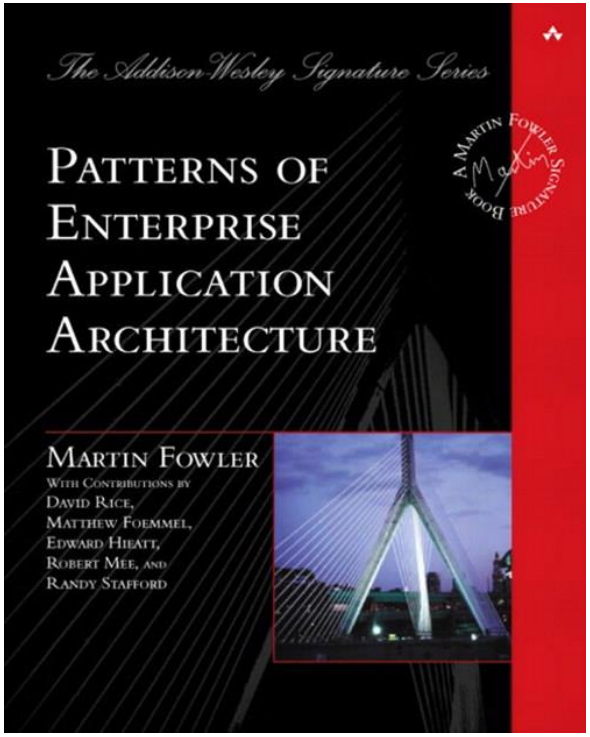
DESIGN PATTERNS, PRESENTATION

DESIGN PATTERNS,
PRESENTATION LAYER OF EAA,

DESIGN PATTERNS







Gang of Four (GOF)

Java EE Core

EAA

DESIGN PATTERNS

Design and implementation without flaws!

We specifically look into Façade, Observer & Model-View-Controller

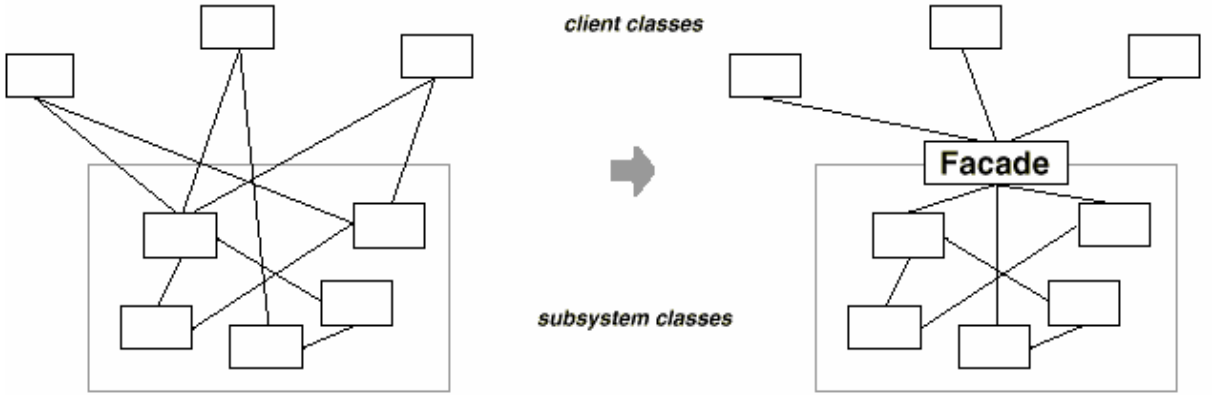
More in optional course A7B36ASS

Summer http://bilakniha.cvut.cz/cs/predmet1469406.html

FAÇADE

CZECH

FACADE



Známý jako

- Facade, Fasáda

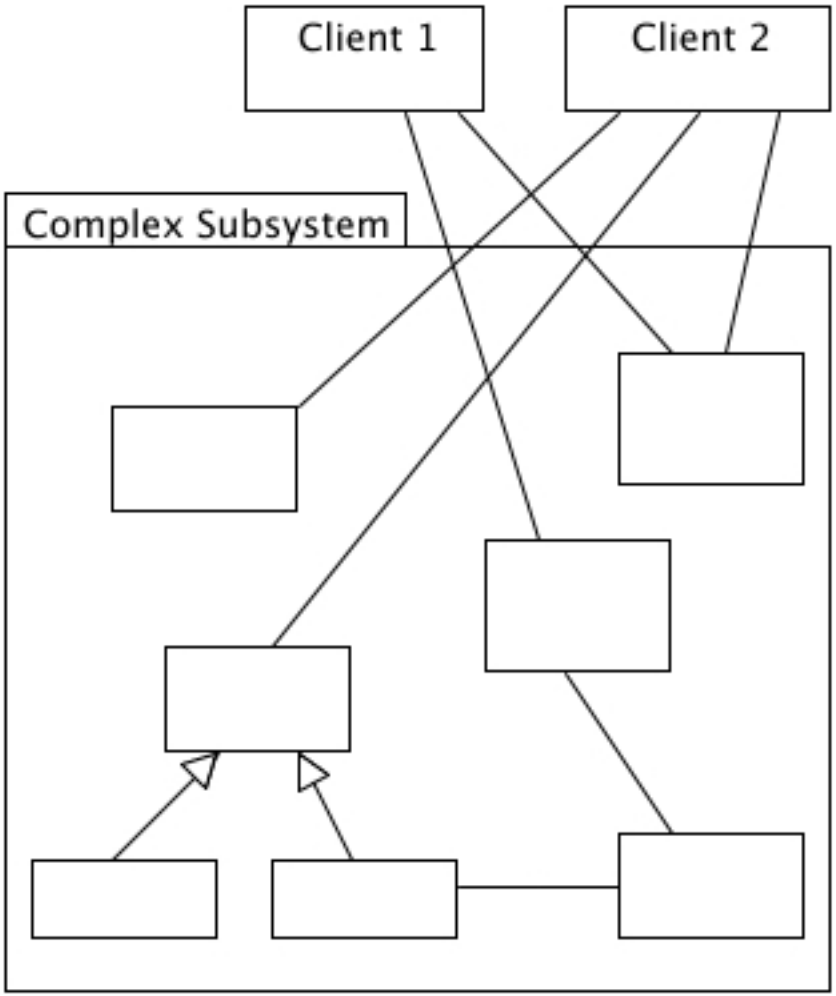
Účel

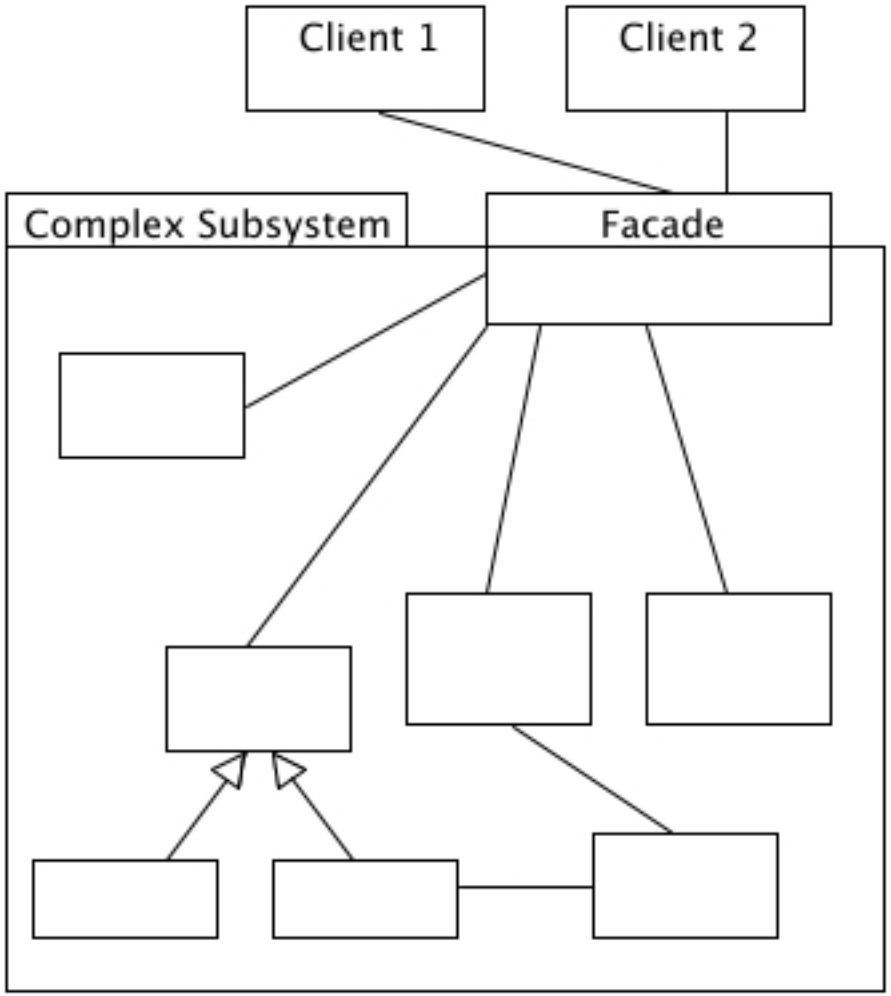
- sjednocené high-level rozhraní pro subsystém
- zjednodušuje použití subsystému
- zapouzdření – skrytí návrhu před uživateli

Motivace

- zjednodušit komunikaci mezi subsystémy
- zredukovat závislosti mezi subsystémy
- neznemožnit používání low-level interfaců
    - pro použití „na míru“

FACADE - MOTIVACE





FACADE – MOTIVACE

Puštění filmu:

- zatáhnutí žaluzií
- zapnutí projektoru
- zapnutí DVD přehrávače
- zapnutí ozvučení

Ukončení přehrávání:

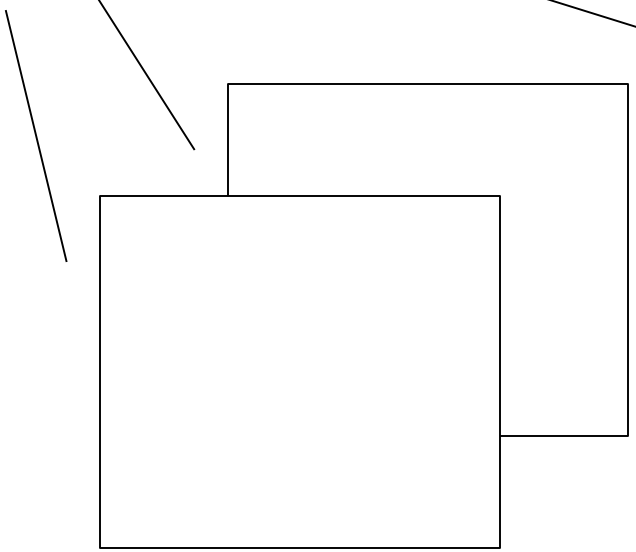
- vypnout DVD přehrávač
- vypnout ozvučení
- vypnout projektor
- vytáhnout žaluzie

FACADE – MOTIVACE

Řešení:

Univerzální ovladač s funkcemi:

- Přehrát film
- Ukončit film
- Přehrát hudbu
- Vypnout hudbu
- …



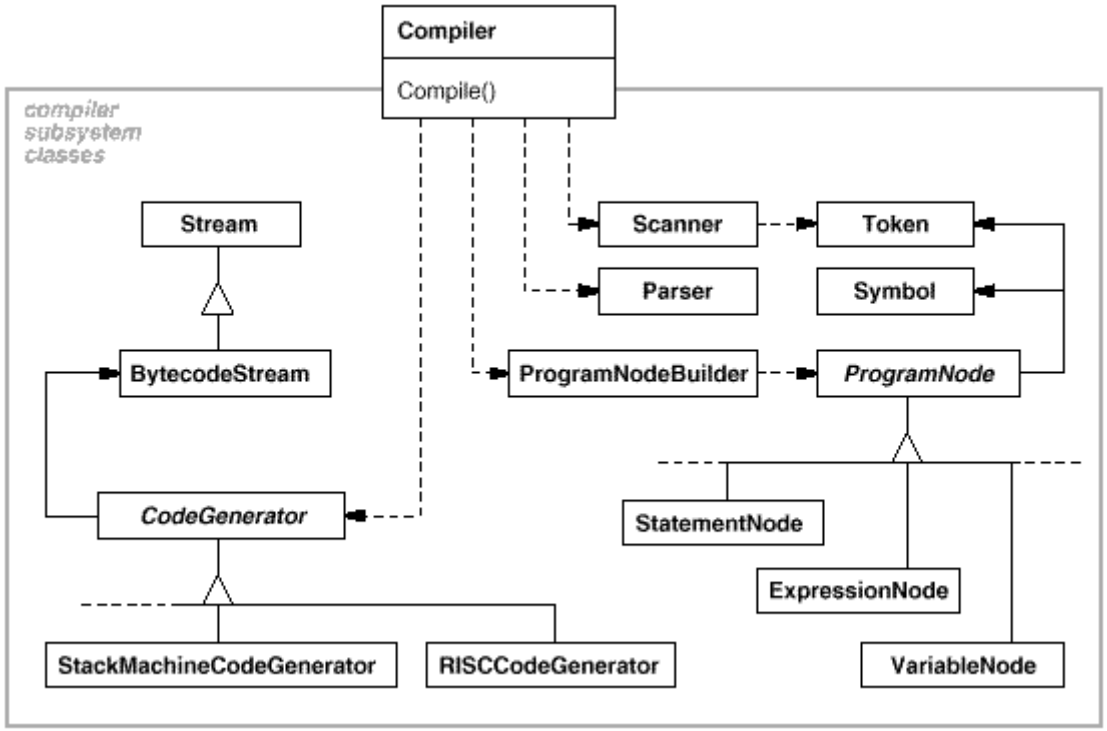
- zatáhnout žaluzie
- zapnout projektor
- zapnout DVD přehrávač
- zapnout ozvučení

- vypnout DVD přehrávač
- vypnout ozvučení
- vypnout projektor
- vytáhnout žaluzie

FACADE - MOTIVACE

Příklad

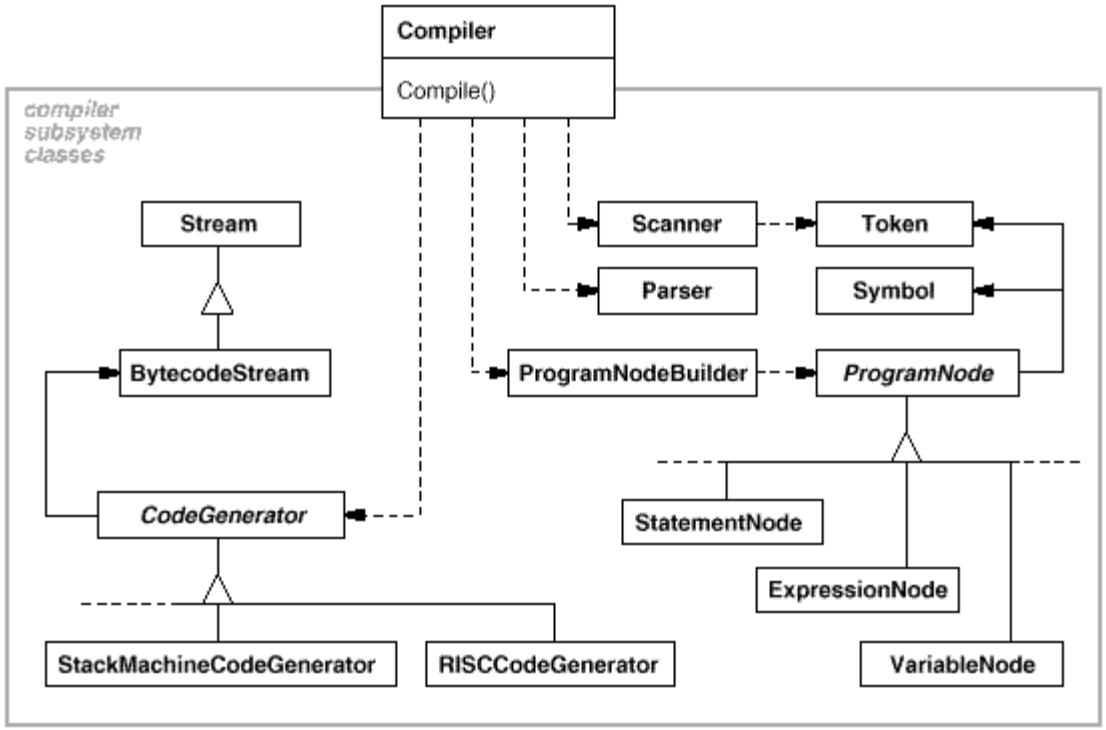
- *Compiler*
- třídy *Scanner*, *Parser*, *ProgramNodeBuilder*, ..
- Facade poskytuje rozhraní pro práci s kompilátorem na high-level úrovni



FACADE – MOTIVACE

Příklad

- *transparentnost* – třídy subsystému o Facade nevědí
- *svoboda volby*- neskrývá třídy subsystému (Parser, Scanner)

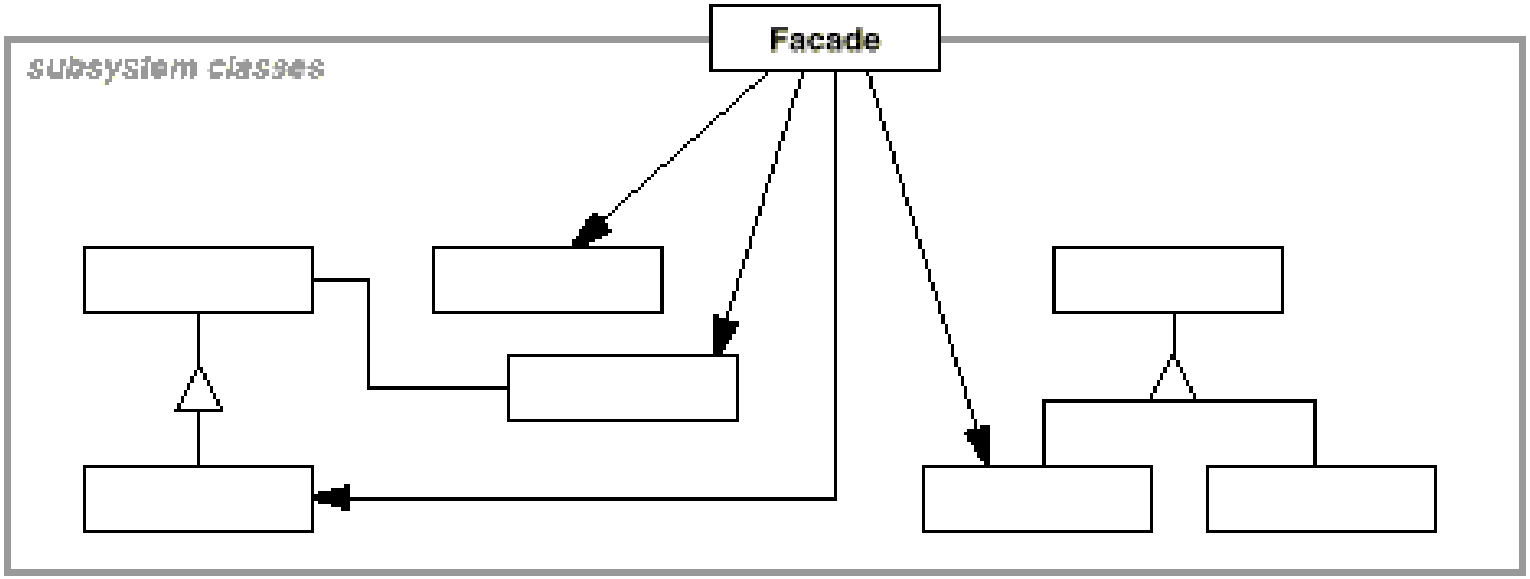


FACADE - POUŽITÍ

Použití

- když je subsystém složitý na běžné použití
- když existuje mnoho závislostí vně subsystému
- při vytváření vstupních bodů vrstveného systému

Struktura



FACADE – POUŽITÍ

Účastníci

- Facade (kompilátor)
  - zná třídy uvnitř subsystému
  - deleguje požadavky
- třídy subsystému (Scanner, Parser,...)
  - nevědí o existenci Facade
  - implementují funkčnost

Souvislosti

- klient se subsystémem komunikuje přes Facade
  - Facade předává požadavky dál
  - může obsahovat vlastní logiku – překlad požadavků na třídy subsystému
- klienti nemusí používat subsystém přímo

FACADE - DŮSLEDKY

Výhody použití

- redukuje počet objektů, se kterými klienti komunikují
  - snadnější použití subsystému
- zmenšuje počet závislostí mezi klienty a subsystémem
  - odstraňuje komplexní a kruhové závislosti
  - méně kompilačních závislostí
- neskrývá třídy subsytému
  - klient si může vybrat jednoduchost nebo použití na „míru“
- umožňuje rozšířit stávající funkcionalitu
  - kombinací subsystémů a jejich metod
  - monitorování systému – přístupy, využívání jednotlivých metod

Kdy se vyplatí facade do systému implementovat

- cena za vytvoření fasády menší než je nastudování podsystému uživateli

FACADE – PŘÍKLAD

```
class Scanner {
public:
    …
    virtual Token& Scan();
    …
};
```

Třída subsystému

```
class Parser {
public:
  …
  virtual void Parse(Scanner&,ProgramNodeBuilder&);
  …
};
```

Třída subsystému

```
class ProgramNodeBuilder {
public:
  ProgramNodeBuilder();
  virtual ProgramNode* NewVariable(…);
  virtual ProgramNode* NewAssignment(…);
  virtual ProgramNode* NewReturnStatement(…);
  virtual ProgramNode* NewCondition(…) const;
  …
  ProgramNode* GetRootNode();
  …
};
```

Třída subsystému

FACADE – PŘÍKLAD

Třída subsystému

```
class ProgramNode {
public:
  // program node manipulation
  virtual void GetSourcePosition(int& line, int& index);
  …
  // child manipulation
  virtual void Add(ProgramNode*);
  virtual void Remove(ProgramNode*);
  …

  virtual void Traverse(CodeGenerator&);
  protected: ProgramNode();
};
```

Třída subsystému

```
class CodeGenerator {
public:
  virtual void Visit(StatementNode*);
  virtual void Visit(ExpressionNode*);
  …
protected:
  CodeGenerator(BytecodeStream&);
protected:
  BytecodeStream& _output;
};
```

FACADE – PŘÍKLAD

```
void ExpressionNode::Traverse (CodeGenerator& cg) {
  cg.Visit(this);
  ListIterator i(_children);
  for (i.First(); !i.IsDone(); i.Next()) {
        i.CurrentItem()->Traverse(cg);
  }
}
```

Facade

```
class Compiler {
public:
  Compiler();
  virtual void Compile(istream&,BytecodeStream&);
};

void Compiler::Compile ( istream& input, BytecodeStream& output )
{
  Scanner scanner(input);
  ProgramNodeBuilder builder;                    // Návrhový vzor Builder
  Parser parser;
  parser.Parse(scanner, builder);
  RISCCodeGenerator generator(output);           // potomek CodeGenerator
  ProgramNode* parseTree = builder.GetRootNode();
  parseTree->Traverse(generator);
}
```

FACADE – IMPLEMENTACE

Konfigurace fasády

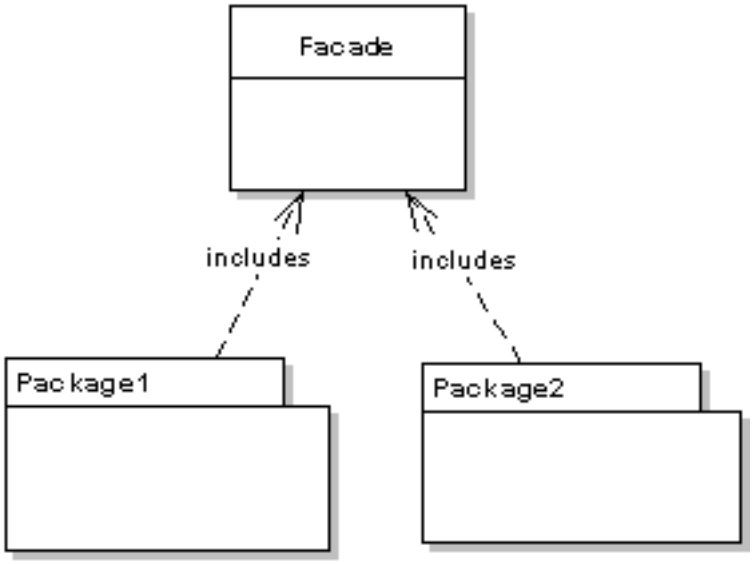
- Facade jako abstraktní třída
  - konkrétní implementace subsystému je jejím potomkem
  - klienti komunikují se subsystémem přes rozhraní abstraktní třídy
    - Ideálně propojit s Dependency Injection

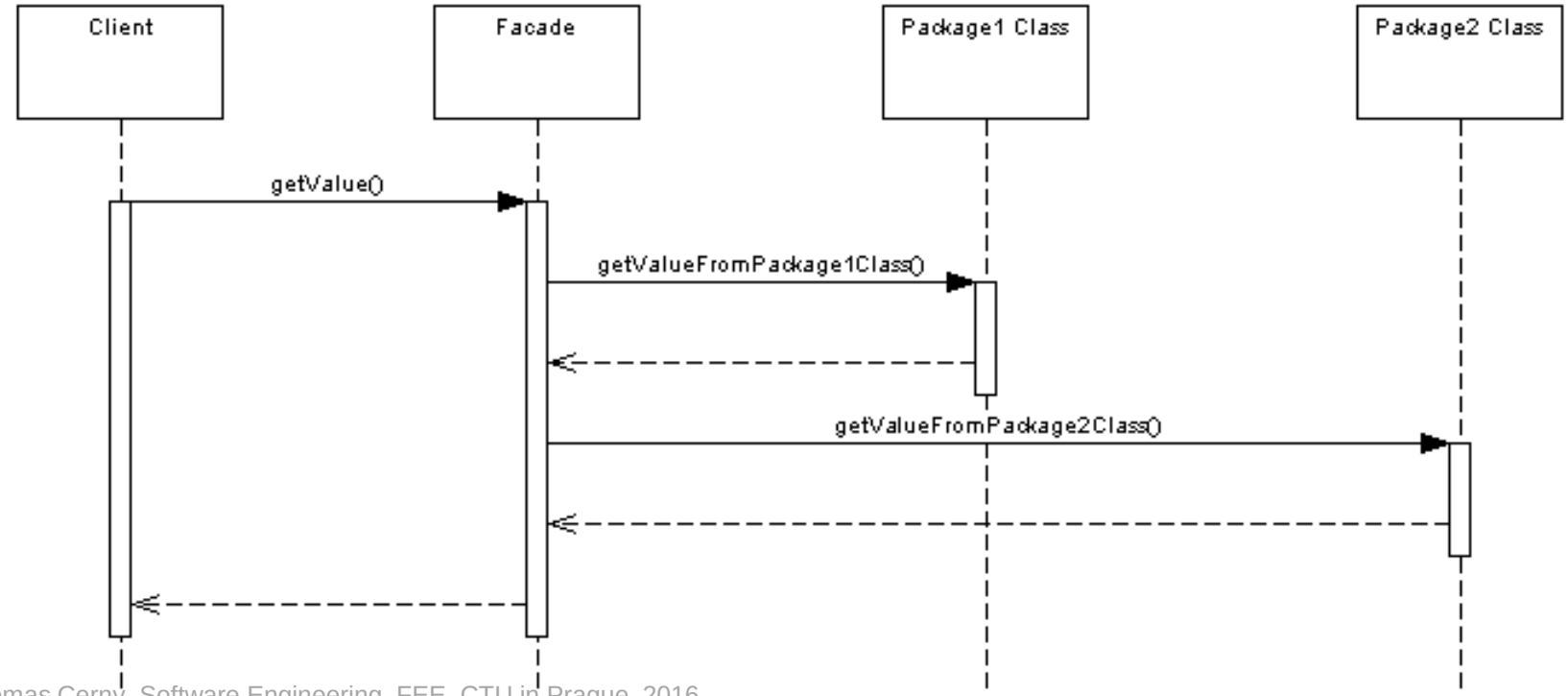
- Facade jako jedna konfigurovatelná třída
  - výměna komponent subsystému

Viditelnost komponent subsystému

- Zakomponování kontroly přístupu

FACADE – REÁLNÉ POUŽITÍ





FACADE – PŘÍKLAD

Services

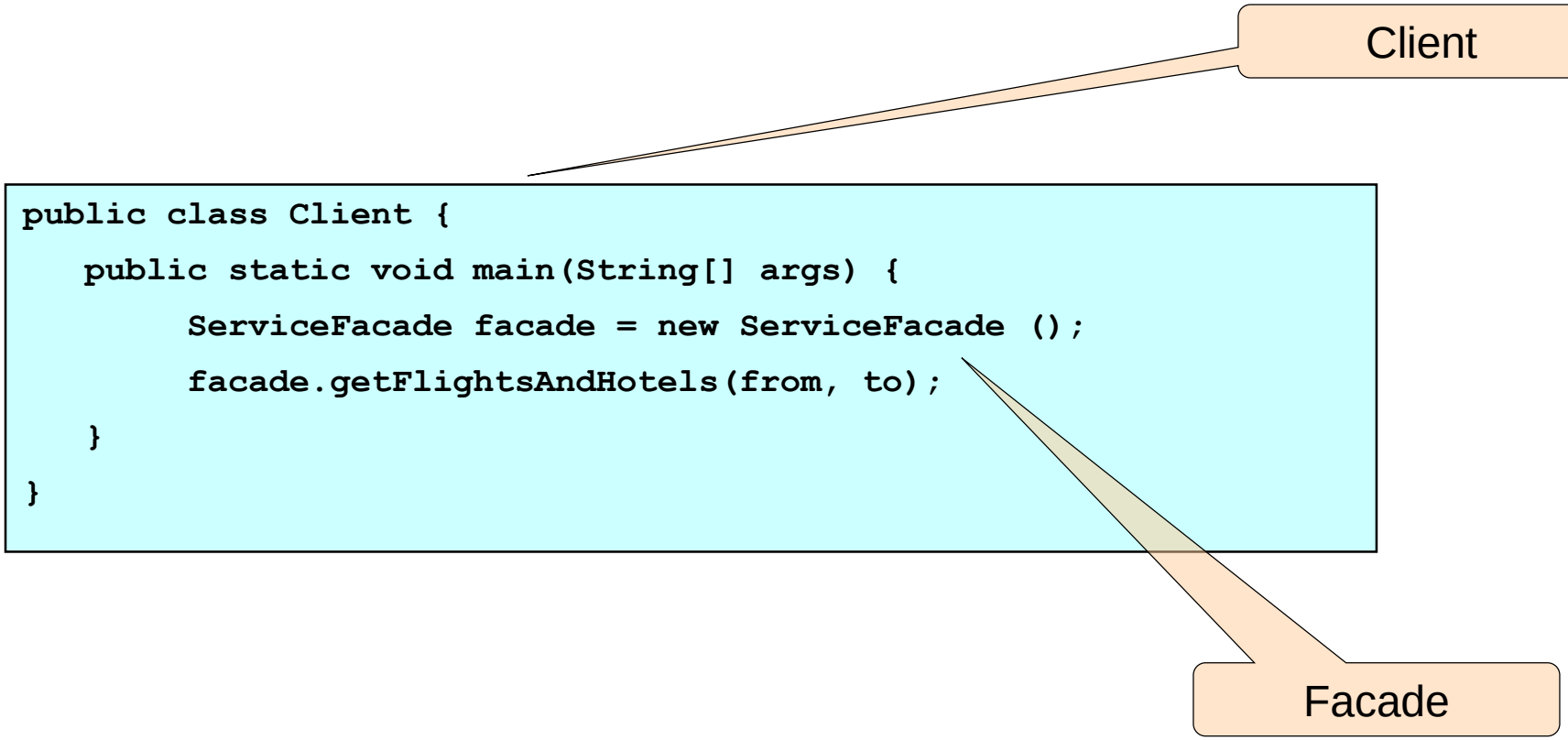
```
public class HotelBookerImpl implements HotelBooker {
  public List<Hotel> getHotelNamesFor(Date from, Date to) {
      //returns hotels available in the particular date range
  }
}
```

```
public class FlightBookerImpl implements FlightBooker {
  public List<Flight> getFlightsFor(Date from, Date to) {
      //returns flights available in the particular date range
  }
}
```

Facade

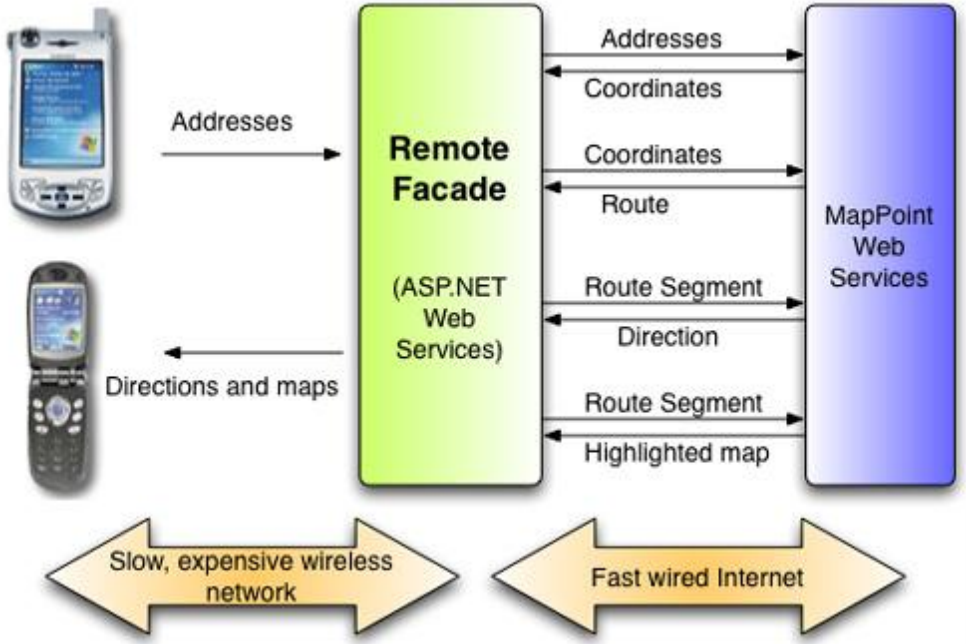
```
public class ServiceFacade {
   private HotelBooker hotelBooker;          // Dependency Injection
   private FlightBooker flightBooker;

   public void getFlightsAndHotels(Date from, Data to) {
         List<Flight> flights = flightBooker.getFlightsFor(from, to);
         List<Hotel> hotels = hotelBooker.getHotelsFor(from, to);
         //process and return
   }
}
```

FACADE – PŘÍKLAD



FACADE – REÁLNÉ POUŽITÍ

V mobilních aplikacích



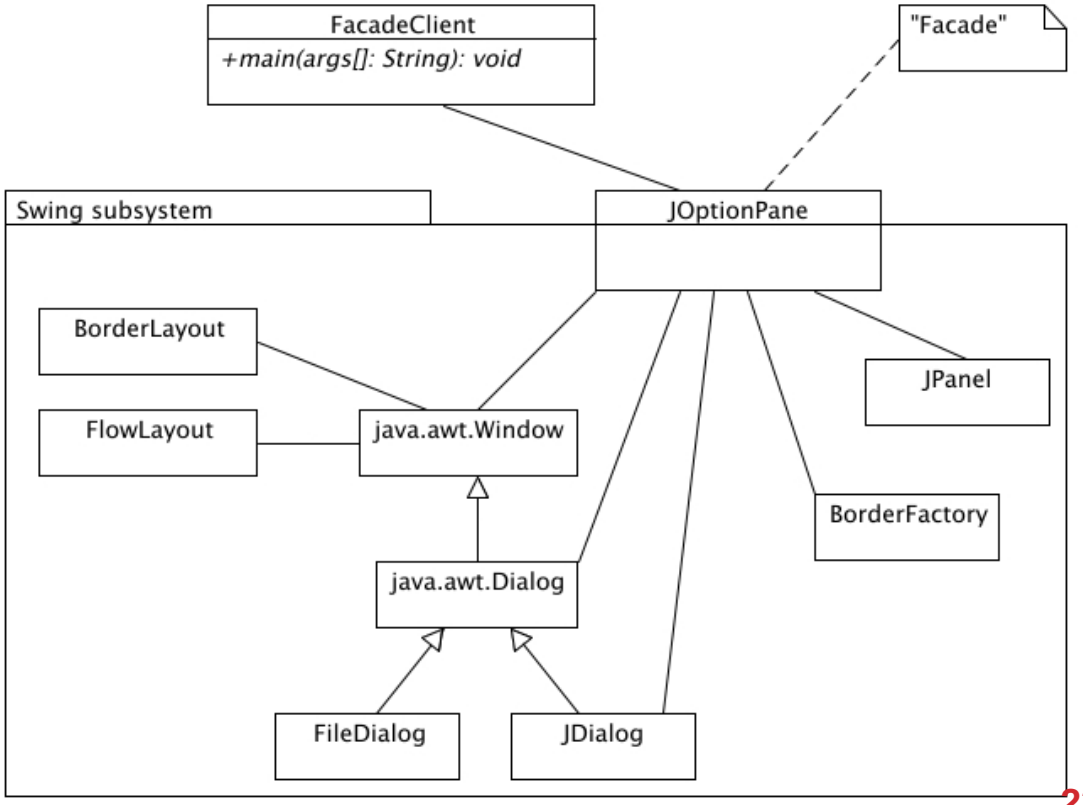
Session Facade v Java EE

- Fasáda pro webové služby

FACADE – REÁLNÉ POUŽITÍ

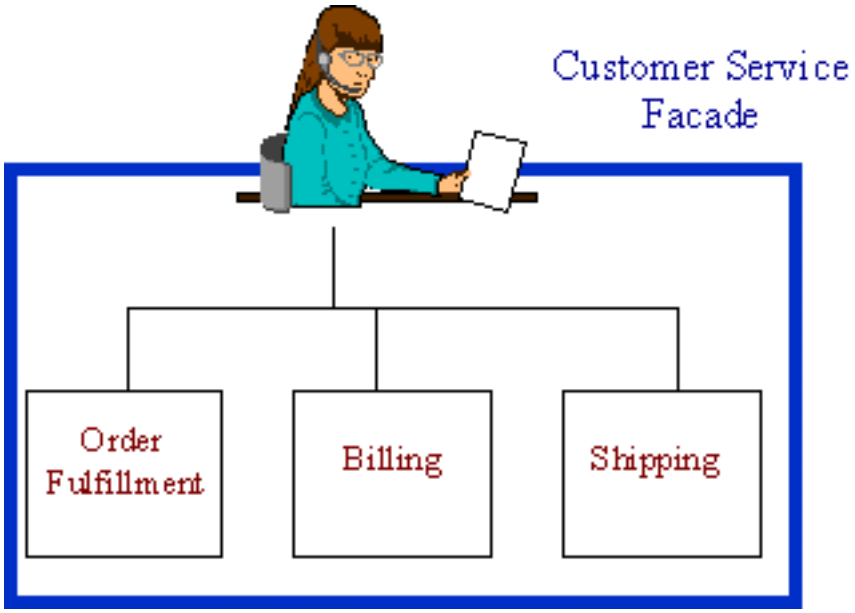
JOptionPane ve Swingu

- vytváří různé typy základních dialogových oken a zobrazuje je
- zjednodušuje používání této rozsáhlé knihovny

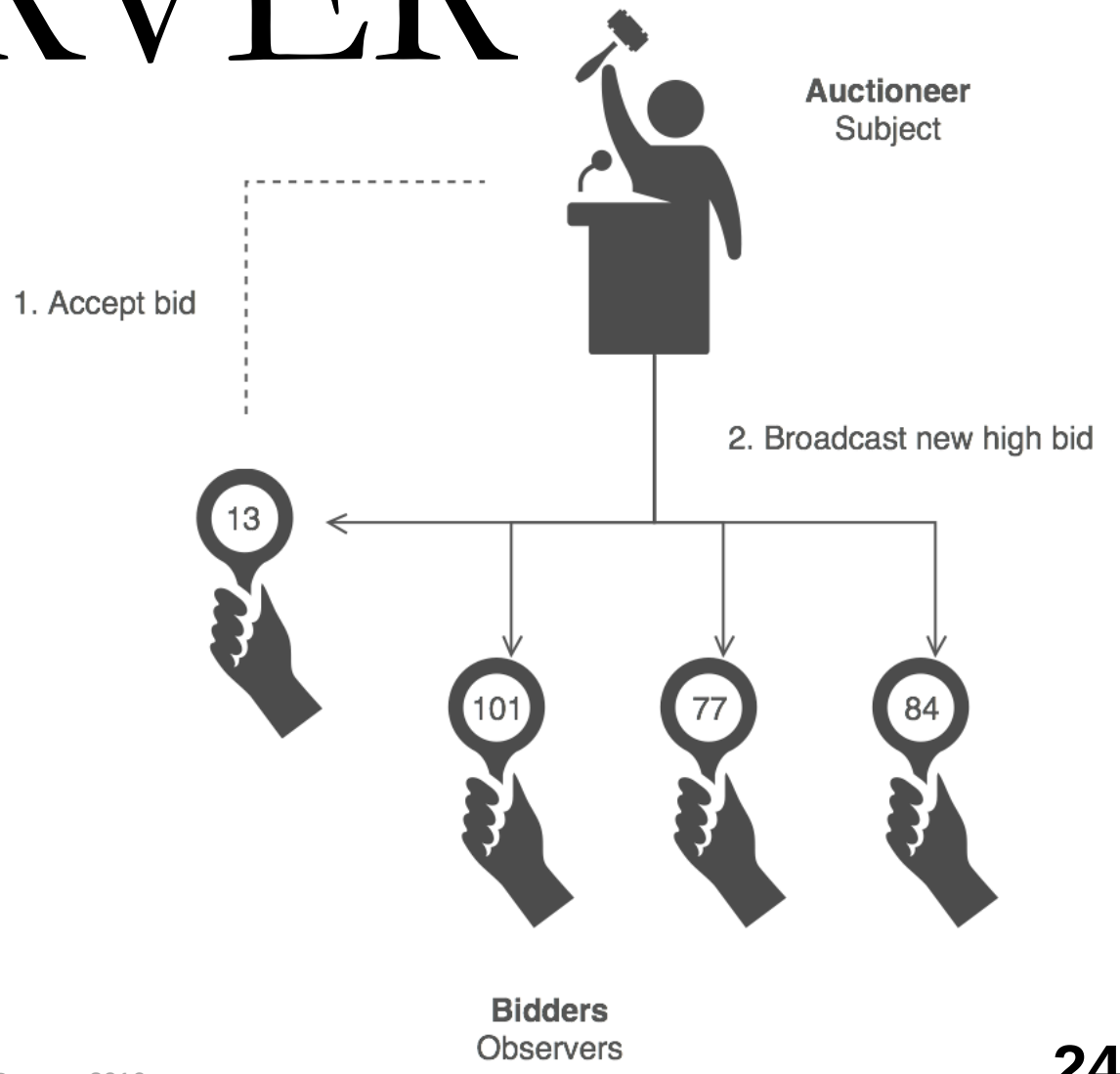


FACADE – REÁLNÉ POUŽITÍ

V byznys světě



OBSERVER


Auctioneer
Subject
1. Accept bid
2. Broadcast new high bid
13
101
77
84
Bidders
Observers

OBSERVER

Smysl

- Zavádí možnost sledování změn u objektu tak, že když objekt změní stav, ostatní objekty na tuto změnu zareagují, přičemž nedojde k přímé vazbě (coupling) od sledovaného objektu k těmto objektům.

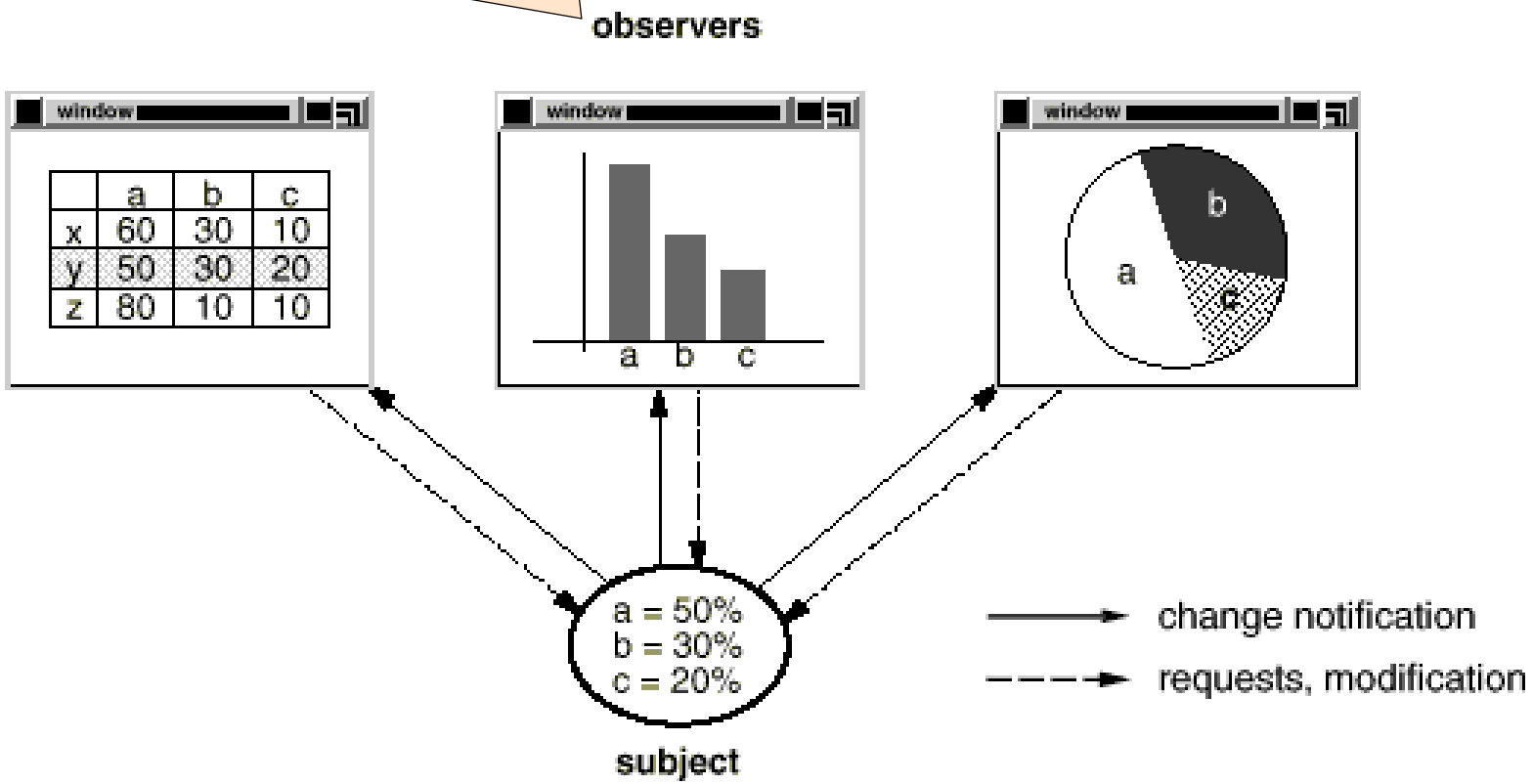
Potřeba sledování změn objektu a notifikace

- Událostní řízení – řeší observer

Obdoba systému událostí (Java) vlastními prostředky

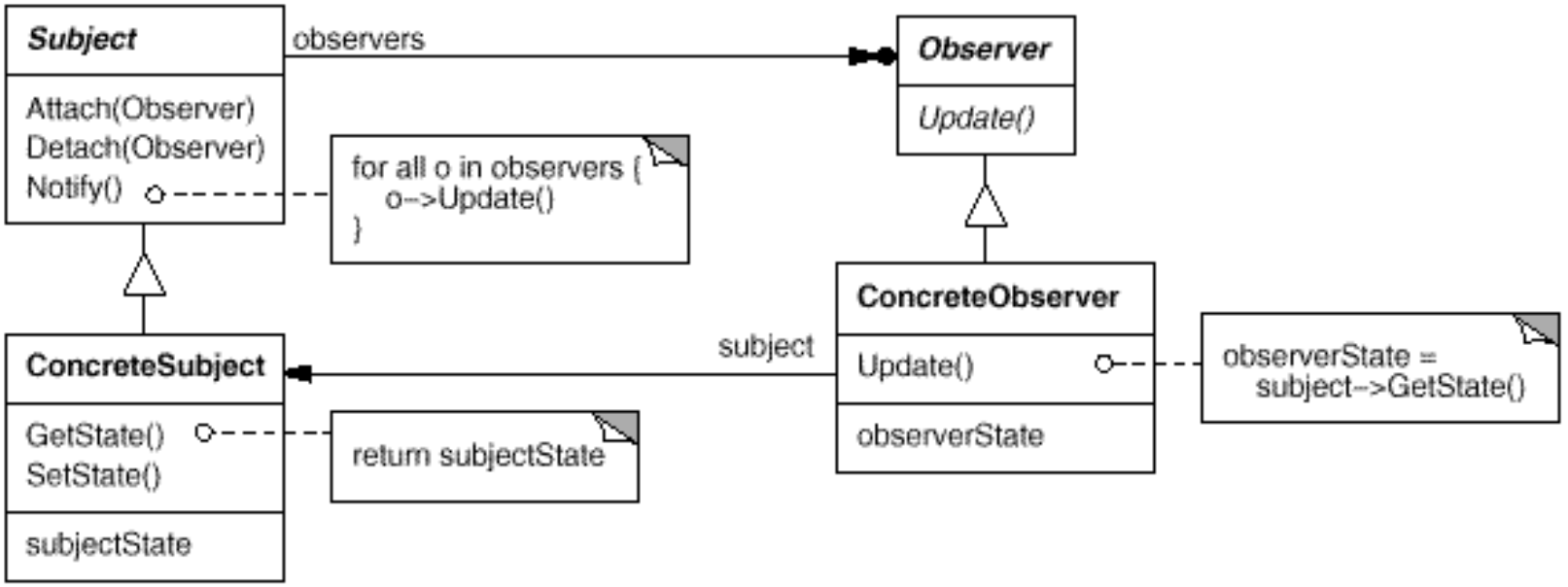
OBSERVER - MOTIVACE

Při změně dat očekávají oznámení od subjektu, že se mají aktualizovat



Udržuje svůj stav a informuje všechny zájemce o jeho změnách

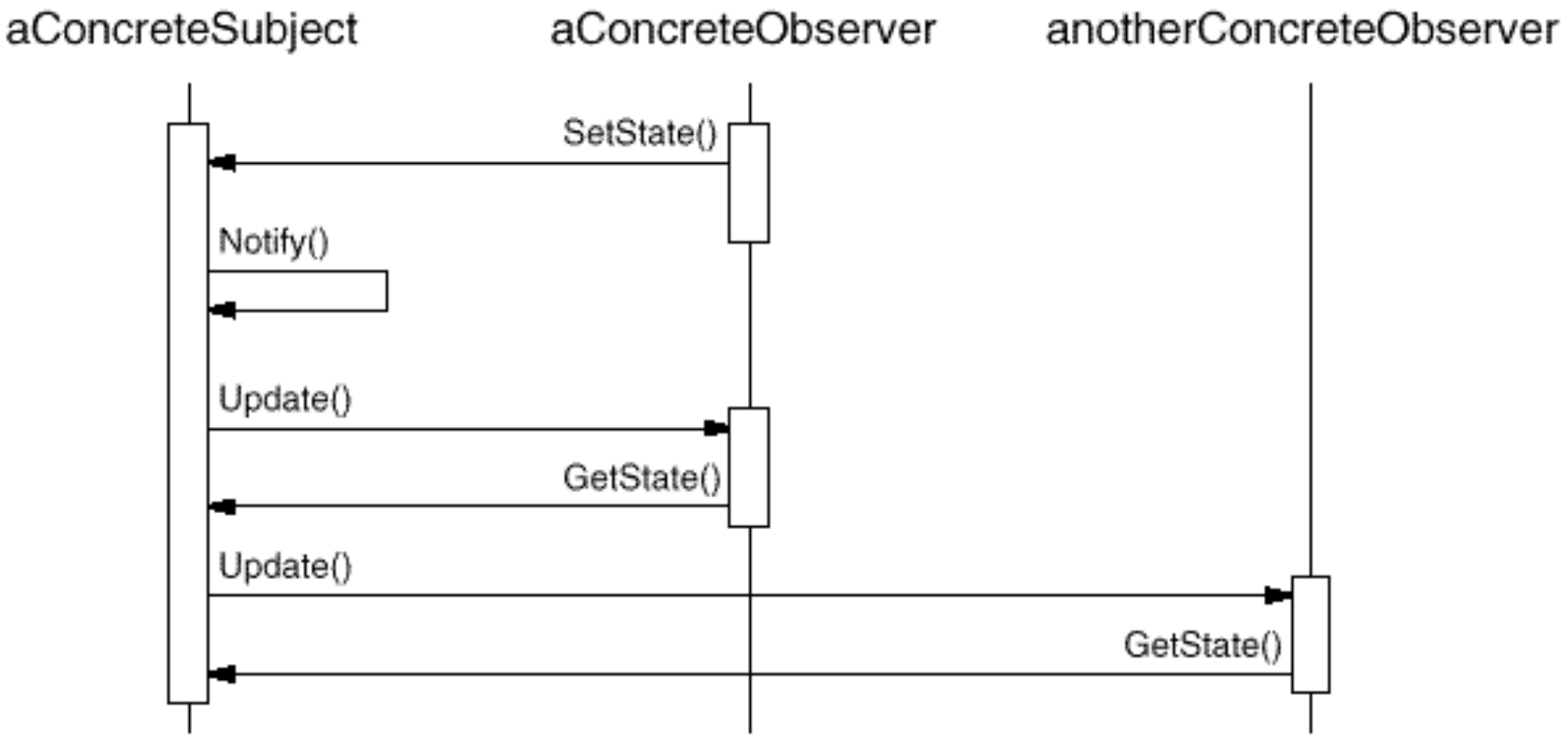
OBSERVER - STRUKTURA

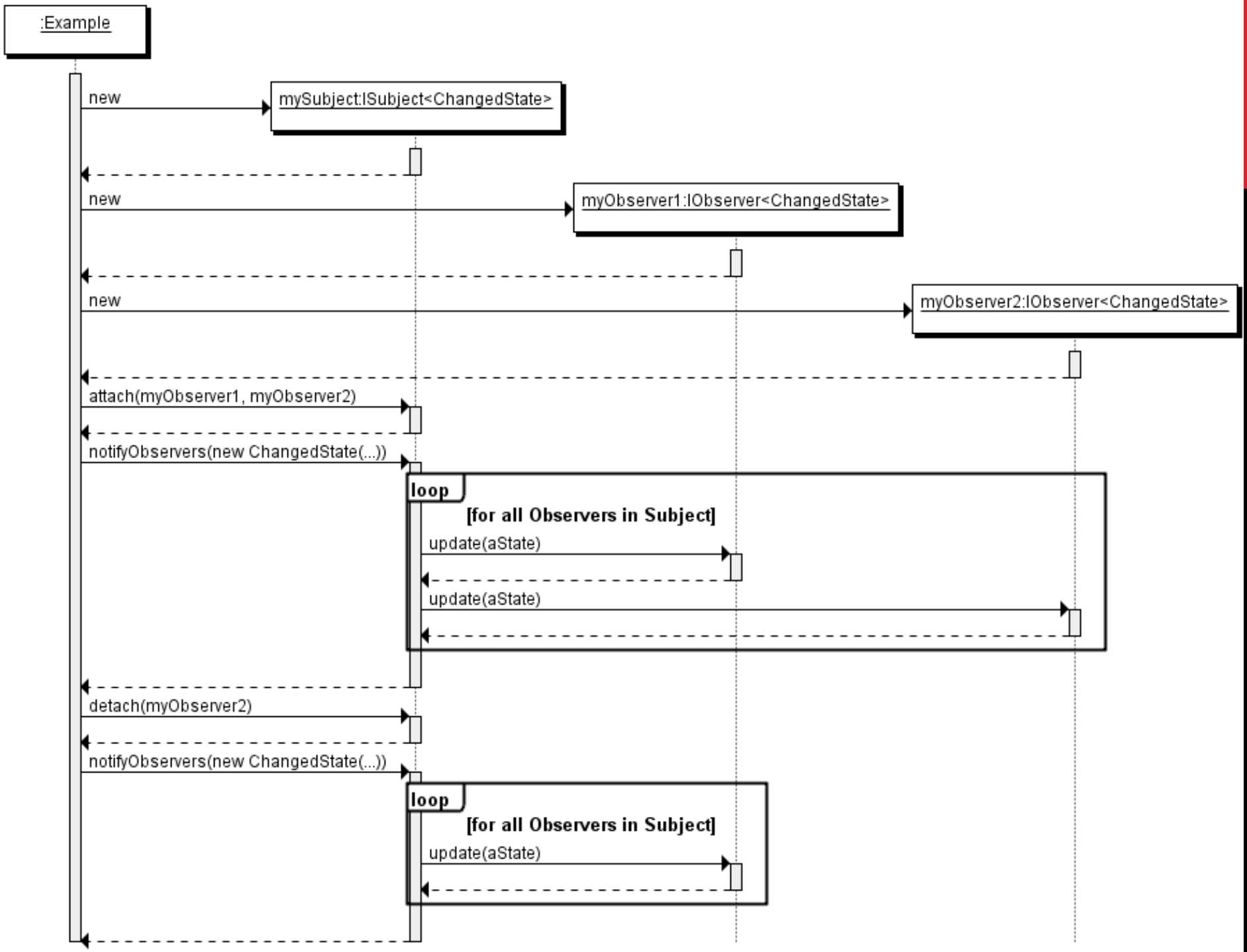


Účastníci

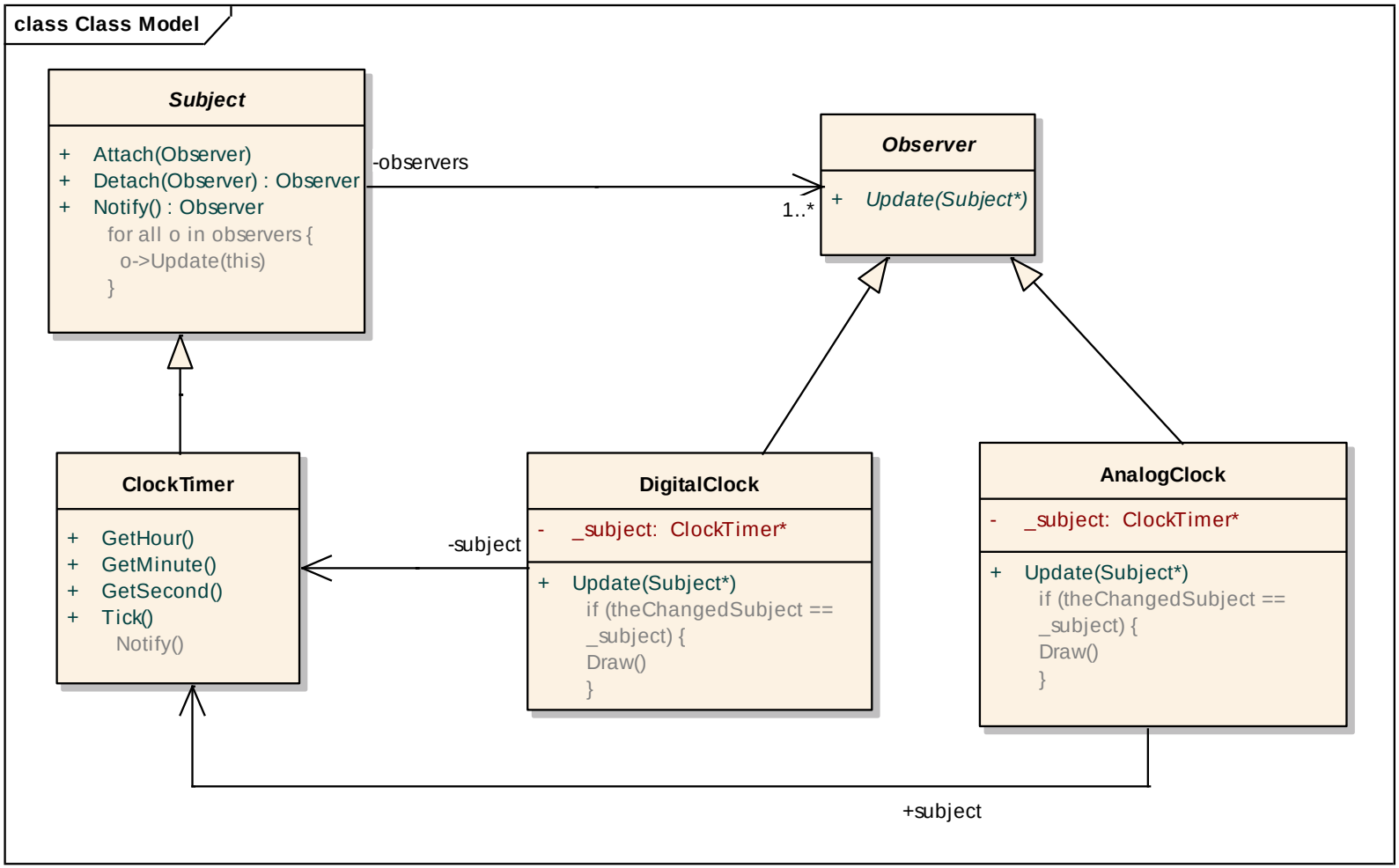
- Subject
  - Interface pro udržování seznamu zájemců o upozornění a posílání oznámení
- Observer
  - Jednoduchý interface pro přijímání upozornění
- ConcreteSubject
  - Má vlastní funkčnost a stav, rozesílá oznámení o jeho změnách
- ConcreteObserver
  - Implementuje Observer interface a tak udržuje svůj stav konzistentní se stavem ConcreteSubject

OBSERVER - INTERAKCE




:Example
new
mySubject:ISubject<ChangedState>
new
myObserver1:IObserver<ChangedState>
new
myObserver2:IObserver<ChangedState>
attach(myObserver1, myObserver2)
notifyObservers(new ChangedState(...))
loop
[for all Observers in Subject]
update(aState)
update(aState)
detach(myObserver2)
notifyObservers(new ChangedState(...))
loop
[for all Observers in Subject]
update(aState)

OBSERVER - PŘÍKLAD



PŘÍKLAD - ABSTRAKTNÍ TŘÍDY

```
public abstract class Observer {
    public update();
};
```

```
public abstract class Subject {

    private List<Observer> observers;

    public void attach(Observer o) {
        observers.add(o);
    }
    public void detach(Observer o) {
        observers.remove(o);
    }
    public void notify();
        for (Observer o : observers) {
            o.update()
        }
    };
```

abstraktní třídy, definují rozhraní

PŘÍKLAD

konkrétní třída, uchovává čas, upozorňuje každou sekundu

```
public class ClockTimer extends Subject {
    public ClockTimer() {..};
    public int getHour();
    public int getMinute();
    public int getSecond();
    public void tick() {
        // aktualizovat vnitřní stav udržující čas
        // ...
      super.notify();
    }
};
```

```
Public class DigitalClock extends Observer {
  private ClockTimer subject;

  public DigitalClock(ClockTimer t);
  public void update();
    // překrývá operaci třídy Observer
  public void draw();
    // překrývá operaci třídy Widget
    // definuje, jak vykreslit digitální hodiny
  public void close();
};
```

konkrétní třída, zobrazuje čas jako digitální hodiny

PŘÍKLAD

```
Public class DigitalClock extends
Observer {
  private ClockTimer subject;

  public DigitalClock(ClockTimer t);
  public void update();
  public void draw();
  public void close();
};
```

```
DigitalClock(ClockTimer s) {
     this.subject = s;
     this.subject.attach(this);
}
```

```
public void close() {
     subject.detach(this);
}
```

```
public void update () {
     draw();
}
```

```
public void draw() {
       // načíst hodnoty ze subjektu

     int hour = subject.getHour();
     int minute = subject.getMinute();
       // etc.

       // nakreslit digitální hodiny
}
```

PŘÍKLAD

```
Public class AnalogClock extends Observer {
    public AnalogClock(ClockTimer);
    public void update();
      // překrývá operaci třídy Observer
    public void draw();
      // překrývá operaci třídy Widget
      // definuje, jak vykreslit digitální hodiny
    public void close();
};
```

jako DigitalClock, ale zobrazuje analogové hodiny

```
public void draw() {
      // načíst hodnoty ze subjektu

    int hour = subject.getHour();
    int minute = subject.getMinute();
    int seconds = subject.getSeconds()
      // etc.

      // nakreslit digitální hodiny
}
```

KDY POUŽÍT

Správné rozdělení systému do vrstev

Odstínění přímého volání sledujícího objektu od sledovaného

Více možných pohledů na jeden model

- chceme je udržovat konzistentní

Objekt, který má informovat nezávislé objekty

- publish-subscribe

NEZÁVISLOST MEZI SUBJECT A OBSERVERS UMOŽŇUJE

Volně zaměňovat jednotlivé observery nebo subjecty

Přidávat observers bez změny subject nebo ostatních observers

Subject a observers můžou patřit do různých vrstev abstrakce

Komunikaci low-level subject s high-level observers

Broadcast komunikaci

- Přidávání a odebírání observers za běhu

JAVA EE 7

```
@Stateless
public class MemberRegistration {

    @Inject
    private Logger log;

    @Inject
    private EntityManager em;

    @Inject
    private Event<Member> memberEventSrc;

    public void register(Member member) throws Exception {
        log.info("Registering " + member.getName());
        em.persist(member);
        memberEventSrc.fire(member);
    }
}
```

Facade

Event

Fire
Notify

```
@RequestScoped
public class MemberListProducer {

    @Inject
    private MemberRepository memberRepository;

    private List<Member> members;

    // @Named provides access the return value via the EL variable name
    // "members" in the UI (e.g. Facelets or JSP view)
    @Produces
    @Named
    public List<Member> getMembers() {
        return members;
    }

    public void onMemberListChanged(@Observes(notifyObserver =
                Reception.IF_EXISTS) final Member member) {
        retrieveAllMembersOrderedByName();
    }

    @PostConstruct
    public void retrieveAllMembersOrderedByName() {
        members = memberRepository.findAllOrderedByName();
    }
}
```

Controller

User Interface binding

Observe

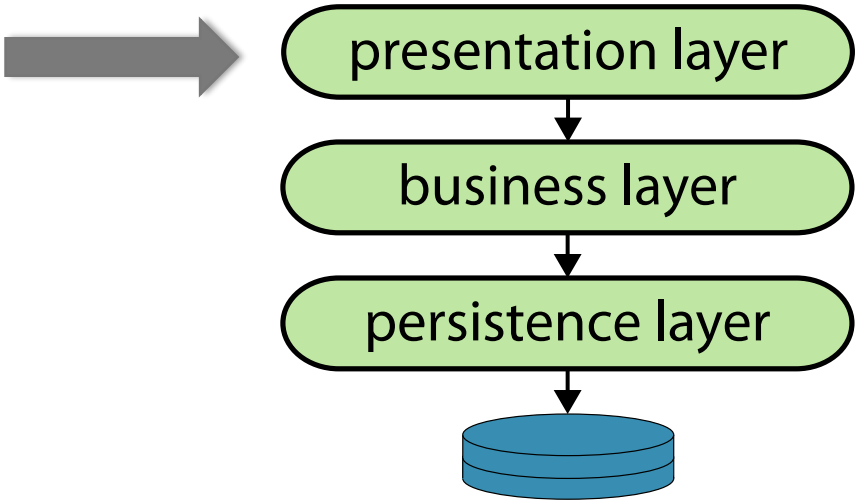
JAVA EE 7

```
@RequestScoped
public class MemberListProducer {

    @Inject
    private MemberRepository memberRepository;

    private List<Member> members;

    // @Named provides access the return value via the EL variable name
    // "members" in the UI (e.g. Facelets or JSP view)
    @Produces
    @Named
    public List<Member> getMembers() {
        return members;
    }

    public void onMemberListChanged(@Observes(notifyObserver =
                Reception.IF_EXISTS) final Member member) {
        members.add(member);
    }

    @PostConstruct
    public void retrieveAllMembersOrderedByName() {
        members = memberRepository.findAllOrderedByName();
    }
}
```

Controller

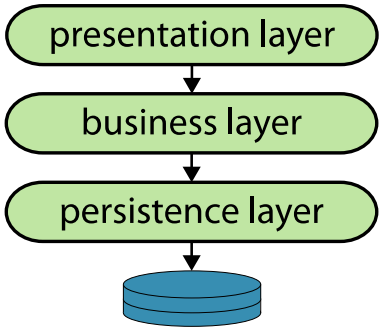
User Interface binding

Observe



PRESENTATION LAYER

PRESENTATION LAYER



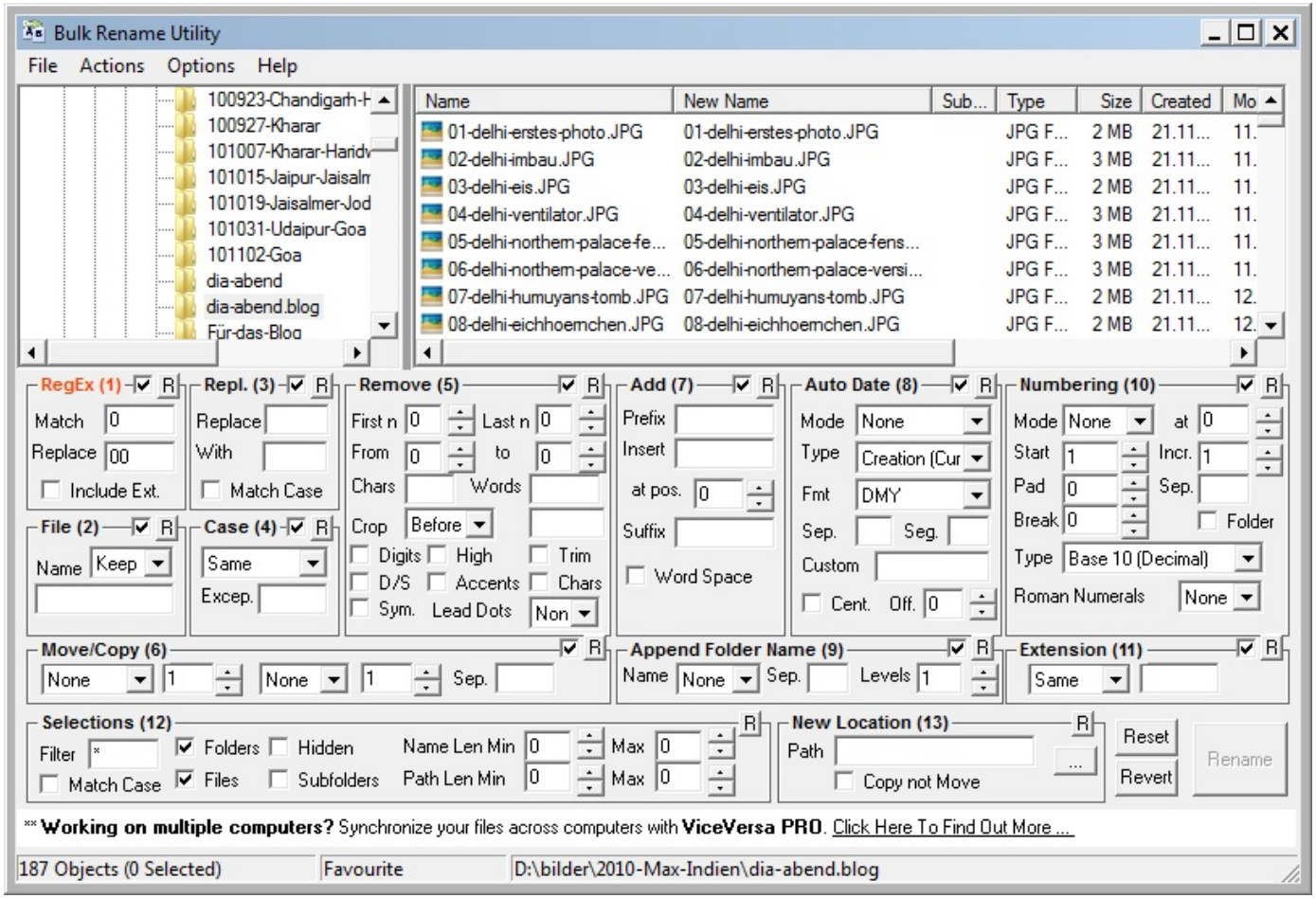
Presents knowledge captured by below layers

Definition:

User interface (UI) provides the visual part of a computer application or a system, which can be seen by human user, it provides mechanisms to control the system through commands, to submit data to the system or receive them and influences how information is displayed on the screen. The UI graphical design aims to make user interaction simple and efficient regards a particular task.

PRESENTATION LAYER

Usability is the ease of use and learnability of a software application

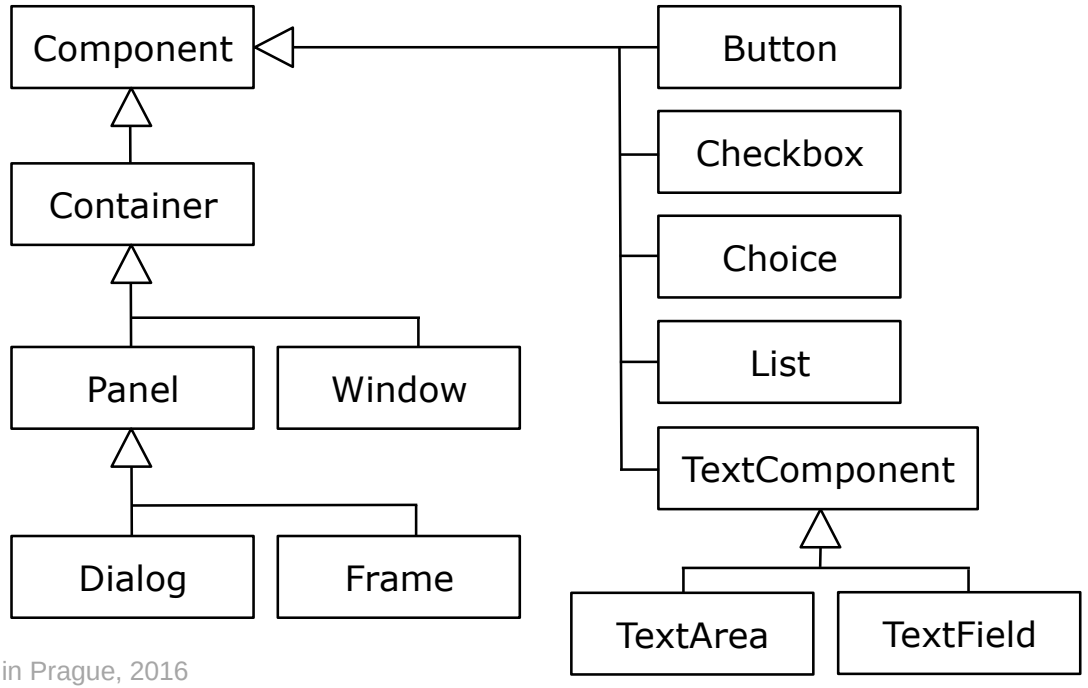


PRESENTATION LAYER

Examples:

- JavaServer Faces (JSF) – web-based
- Swing – standalone apps

- **Component tree**
  - Composite Design Pattern



PRESENTATION LAYER

Examples:

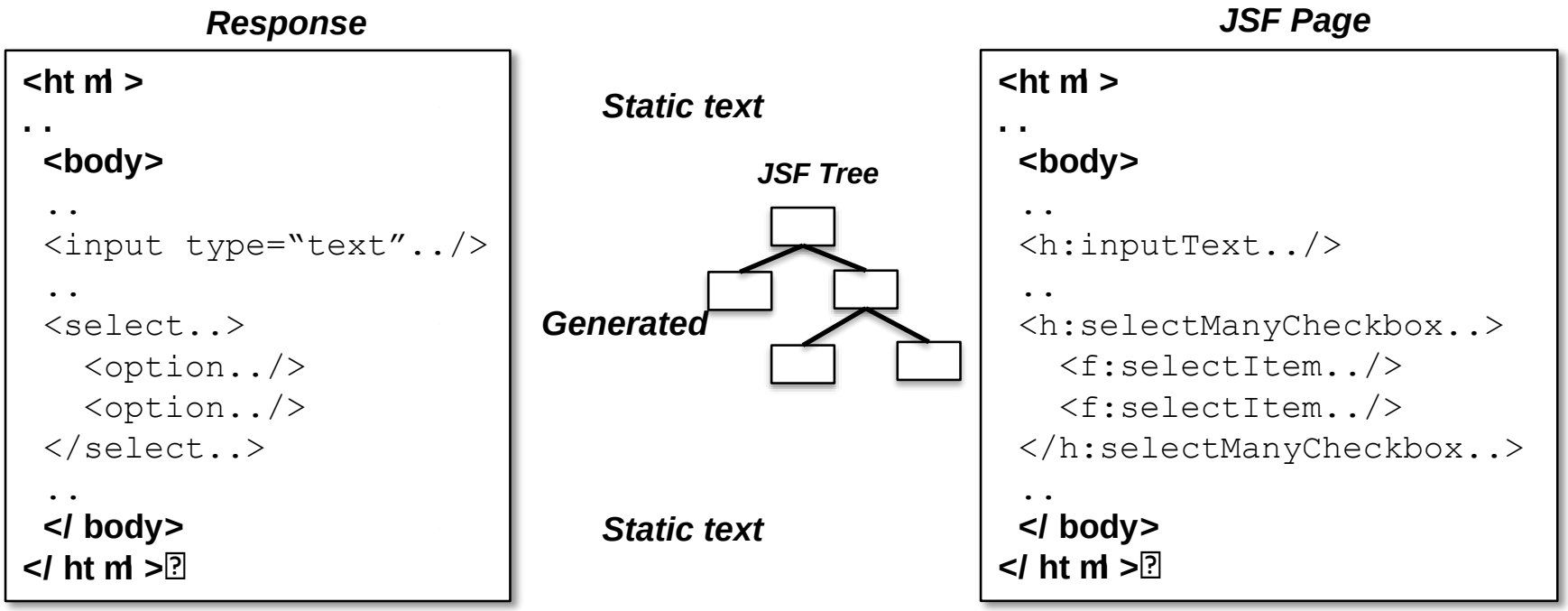
- JavaServer Faces (JSF) – web-based
  - Domain specific language to describe UI (DSL)
  - Transformation to HTML
  - Interpretation by Web Browser
  - Component-based
  - Weak type safety

- Swing – standalone apps
  - General Purpose Language (GPL)
  - Interpretation by JVM
  - Component-based

PRESENTATION LAYER

Examples:

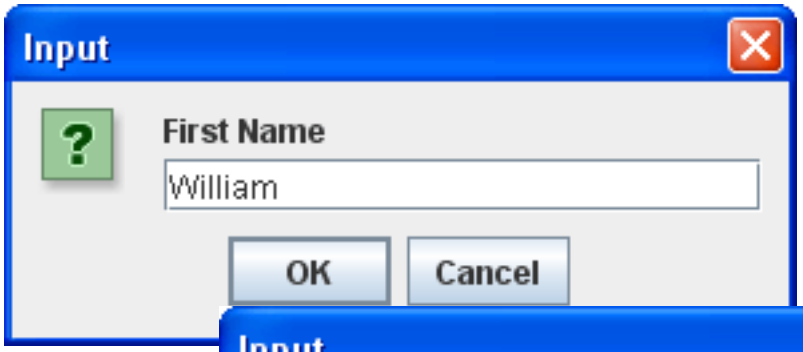
- JavaServer Faces – web-based

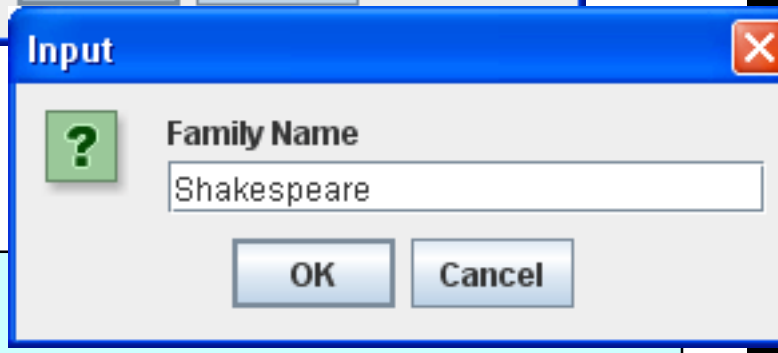


PRESENTATION LAYER

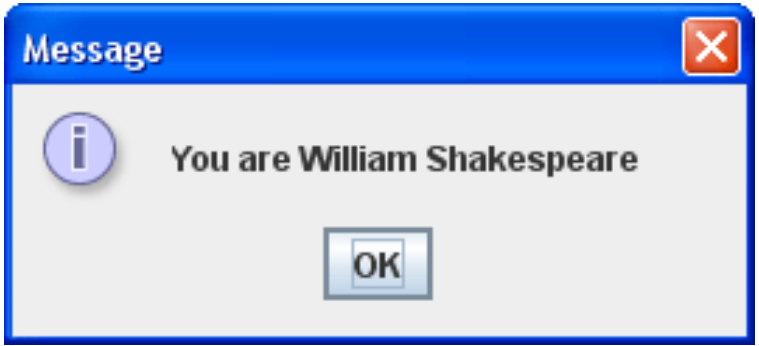
Examples:

- Swing






```
public static void main(String[] args) {
 String firstName = JOptionPane.showInputDialog("First Name");
 String lastName = JOptionPane.showInputDialog("Family Name");
 fullMame = "You are " + first_name + " " + family_name;
 JOptionPane.showMessageDialog( null, fullMame);
```



JAVASERVER FACES

JavaServer Faces (JSF) is a MVC web framework that simplifies the construction of user interfaces (UI) for server-based applications by using reusable UI components in a page.

JSF provides facility to connect UI widgets with data sources and to server-side event handlers.

The JSF specification defines a set of standard UI components and provides an Application Programming Interface (API) for developing components.

JSF enables the reuse and extension of the existing standard UI components.

JAVASERVER FACES

2 core elements

- **Managed Bean**
  - Controller
- **JSF Page**

```
@ManagedBean(name = "helloWorld", eager = true)
public class HelloWorld {
   public HelloWorld() {
      System.out.println("HelloWorld started!");
   }
   public String getMessage() {
      return "Hello World!";
   }
}
```

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
Transitional//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
   <title>JSF Tutorial!</title>
</head>
<body>
   #{helloWorld.message}
</body>
</html>
```

CONTROLLER - BEAN

```
@RequestScoped
public class MemberListProducer {

    @Produces
    @Named
    public List<Member> getMembers() {
        return members;
    }
```

```
@Model //same as @RequestScoped and @Named
public class MemberController {

    @Inject
    private FacesContext facesContext;

    @Inject
    private MemberRegistration memberRegistration;

    @Produces
    @Named
    private Member newMember;

    public void register() throws Exception {
      try {
       memberRegistration.register(newMember);
       FacesMessage m = new FacesMessage(FacesMessage.SEVERITY_INFO,
                            "Registered", "Registration successful");
       facesContext.addMessage(null, m);
       initNewMember();
      } catch (Exception e) {
         String errorMessage = getRootErrorMessage(e);
         FacesMessage m = new FacesMessage(FacesMessage.SEVERITY_ERROR,
                          errorMessage, "Registration unsuccessful");
         facesContext.addMessage(null, m);
      }
    }
```

JSF PAGE

Header

Body

```
<ui:composition xmlns="http://www.w3.org/1999/xhtml"
    xmlns:ui="http://java.sun.com/jsf/facelets"
    xmlns:f="http://java.sun.com/jsf/core"
    xmlns:h="http://java.sun.com/jsf/html"
    template="/WEB-INF/templates/default.xhtml">
    <ui:define name="content">
        ..
    </ui:define>
</ui:composition>
```

```
<h:dataTable var="_member"
value="#{members}”
  rendered="#{not empty members}”>
  <h:column>
    <f:facet name="header">Id</f:facet>
     #{_member.id}
  </h:column> ...
  <h:column>
   <a href="#{request.contextPath}
       /rest/members/#{_member.id}">
     /rest/members/#{_member.id}</a>
  </h:column>
</h:dataTable>
</ui:define>
```

```
<ui:define name="content">
<h:form id="reg">
  <h:outputLabel for="name" value="Name:”/>
  <h:inputText id="name”
            value="#{newMember.name}" />
  <h:message for="name”/>
...
  <h:commandButton id="register”
        action="#{memberController.register}”
   value="Register" styleClass="register" />
  <h:messages globalOnly="true" />
</h:form>
```

JSF PAGE

Header

Body

```
<ui:composition xmlns="http://www.w3.org/1999/xhtml"
    xmlns:ui="http://java.sun.com/jsf/facelets"
    xmlns:f="http://java.sun.com/jsf/core"
    xmlns:h="http://java.sun.com/jsf/html"
    template="/WEB-INF/templates/default.xhtml">
    <ui:define name="content">
        ..
    </ui:define>
</ui:composition>
```

```
<h:dataTable var="_member"
value="#{members}”
  rendered="#{not empty members}”>
  <h:column>
    <f:facet name="header">Id</f:facet>
     #{_member.id}
  </h:column> ...
  <h:column>
   <a href="#{request.contextPath}
       /rest/members/#{_member.id}">
     /rest/members/#{_member.id}</a>
  </h:column>
</h:dataTable>
</ui:define>
```

```
<ui:define name="content">
<h:form id="reg">
  <h:outputLabel for="name" value="Name:”/>
  <h:inputText id="name”
            value="#{newMember.name}" />
  <h:message for="name”/>
...
  <h:commandButton id="register”
        action="#{memberController.register}”
   value="Register" styleClass="register" />
  <h:messages globalOnly="true" />
</h:form>
```

JSF PAGE

Header

Body - components

```
<ui:composition xmlns="http://www.w3.org/1999/xhtml"
    xmlns:ui="http://java.sun.com/jsf/facelets"
    xmlns:f="http://java.sun.com/jsf/core"
    xmlns:h="http://java.sun.com/jsf/html"
    template="/WEB-INF/templates/default.xhtml">
    <ui:define name="content">
        ..
    </ui:define>
</ui:composition>
```

```
<h:dataTable var="_member"
value="#{members}”
  rendered="#{not empty members}”>
  <h:column>
    <f:facet name="header">Id</f:facet>
     #{_member.id}
  </h:column> ...
  <h:column>
   <a href="#{request.contextPath}
       /rest/members/#{_member.id}">
     /rest/members/#{_member.id}</a>
  </h:column>
</h:dataTable>
</ui:define>
```

```
<ui:define name="content">
<h:form id="reg">
  <h:outputLabel for="name" value="Name:”/>
  <h:inputText id="name”
            value="#{newMember.name}" />
  <h:message for="name”/>
...
  <h:commandButton id="register”
        action="#{memberController.register}”
   value="Register" styleClass="register" />
  <h:messages globalOnly="true" />
</h:form>
```

JSF PAGE

Header

```
<ui:composition xmlns="http://www.w3.org/1999/xhtml"
    xmlns:ui="http://java.sun.com/jsf/facelets"
    xmlns:f="http://java.sun.com/jsf/core"
    xmlns:h="http://java.sun.com/jsf/html"
    template="/WEB-INF/templates/default.xhtml">
    <ui:define name="content">
        ..
    </ui:define>
</ui:composition>
```

Body - components

http://www.jsftoolbox.com/documentation/help/12-TagReference/core/index.jsf

http://www.jsftoolbox.com/documentation/help/12-TagReference/html/index.jsf

```
<ui:define name="content">
<h:form id="reg">
  <h:outputLabel for="name" value="Name:”/>
  <h:inputText id="name”
            value="#{newMember.name}" />
  <h:message for="name”/>
...
  <h:commandButton id="register”
        action="#{memberController.register}”
   value="Register" styleClass="register" />
  <h:messages globalOnly="true" />
</h:form>
```

JSF PAGE

Header

Body - binding

```
<ui:composition xmlns="http://www.w3.org/1999/xhtml"
    xmlns:ui="http://java.sun.com/jsf/facelets"
    xmlns:f="http://java.sun.com/jsf/core"
    xmlns:h="http://java.sun.com/jsf/html"
    template="/WEB-INF/templates/default.xhtml">
    <ui:define name="content">
        ..
    </ui:define>
</ui:composition>
```

```
<h:dataTable var="_member"
value="#{members}"
  rendered="#{not empty members}">
  <h:column>
    <f:facet name="header">Id</f:facet>
     #{_member.id}
  </h:column> ...
  <h:column>
   <a href="#{request.contextPath}
       /rest/members/#{_member.id}">
     /rest/members/#{_member.id}</a>
  </h:column>
</h:dataTable>
</ui:define>
```

```
<ui:define name="content">
<h:form id="reg">
  <h:outputLabel for="name" value="Name:"/>
  <h:inputText id="name"
            value="#{newMember.name}" />
  <h:message for="name"/>
...
  <h:commandButton id="register"
        action="#{memberController.register}"
   value="Register" styleClass="register" />
  <h:messages globalOnly="true" />
</h:form>
```

CONTROLLER - BEAN

```
@RequestScoped
public class MemberListProducer {

    @Produces
    @Named
    public List<Member> getMembers() {
        return members;
    }
```

```
@Model // same as @RequestScoped and @Na
public class MemberController {

    @Inject
    private FacesContext facesContext;

    @Inject
    private MemberRegistration memberRegistration;

    @Produces
    @Named
    private Member newMember;

    public void register() throws Exception {
      try {
       memberRegistration.register(newMember);
       FacesMessage m = new FacesMessage(FacesMessage.SEVERITY_INFO,
                            "Registered", "Registration successful");
       facesContext.addMessage(null, m);
       initNewMember();
      } catch (Exception e) {
         String errorMessage = getRootErrorMessage(e);
         FacesMessage m = new FacesMessage(FacesMessage.SEVERITY_ERROR,
                          errorMessage, "Registration unsuccessful");
         facesContext.addMessage(null, m);
      }
    }
```

MODEL-VIEW CONTROLLER (MVC)

INTENT

Separate out User Interfaces its logic and from data

3 main components

- **Model**
- **View**
- **Controller**

INTENT

Separates out User Interfaces its logic and from data/knowledge

3 main components

- **Model**
  - Represents knowledge
- **View**
  - Visual representation of model
- **Controller**
  - Link between User and the system
  - Provides User with input – relevant views, menu, commands
  - Takes input from user

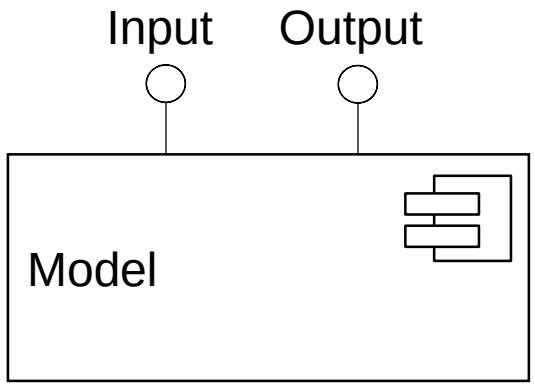
- **Extended version hierarchical MVC**

COMPONENTS

Model

- **Captures system data**
- **Knowledge**

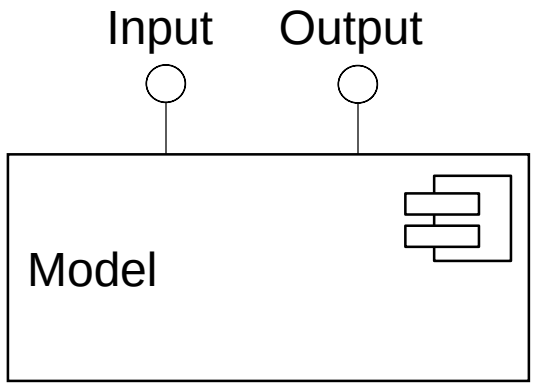
Passive / Active model



COMPONENTS

Model

- **Captures system data**
- **Knowledge**



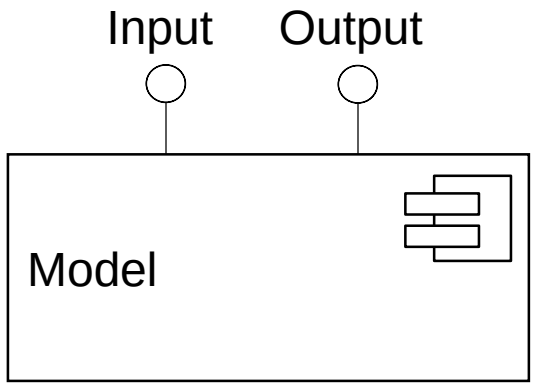
Where are business rules?

Where are services?

COMPONENTS

Model

- **Captures system data**
- **Knowledge**



Where are business rules?

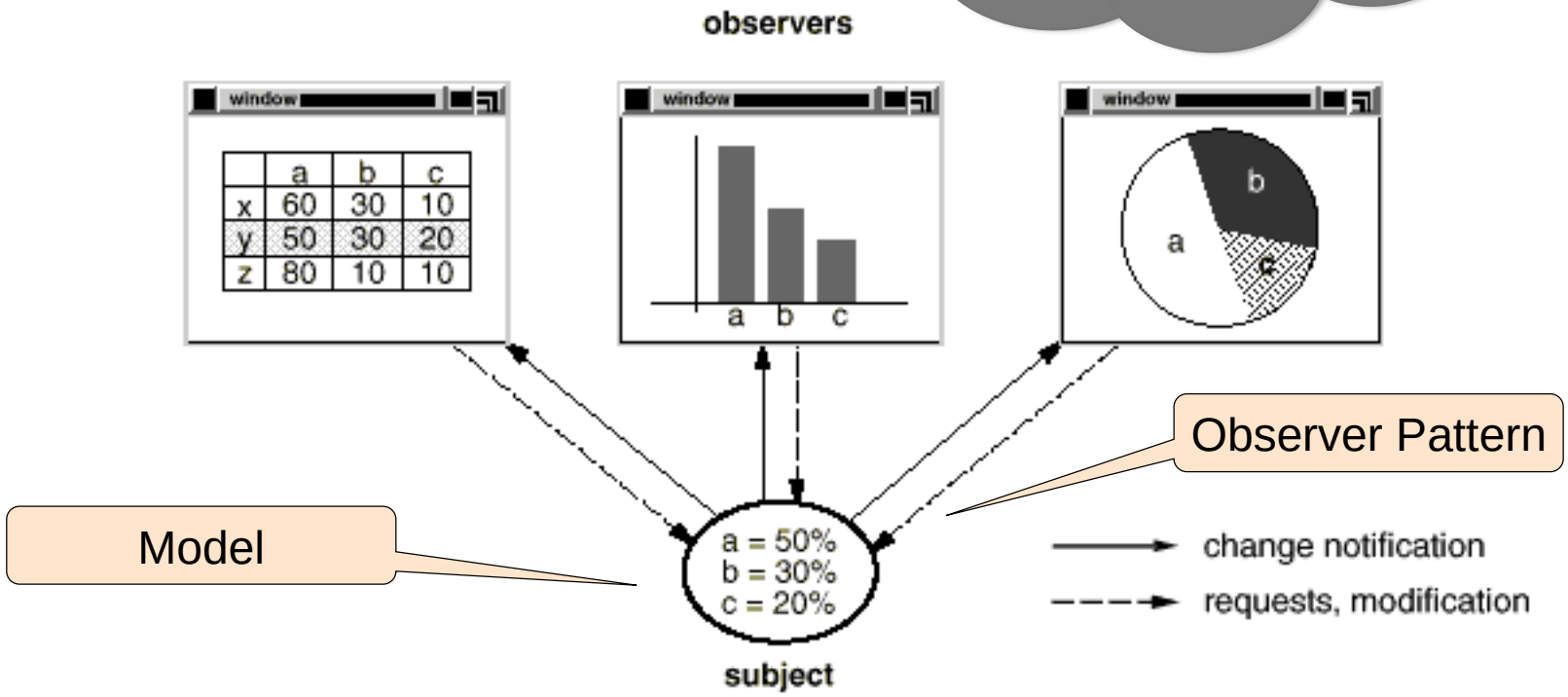
Multiple views can exist on the same model

COMPONENTS

Model

- **Captures system data**
- **Knowledge**

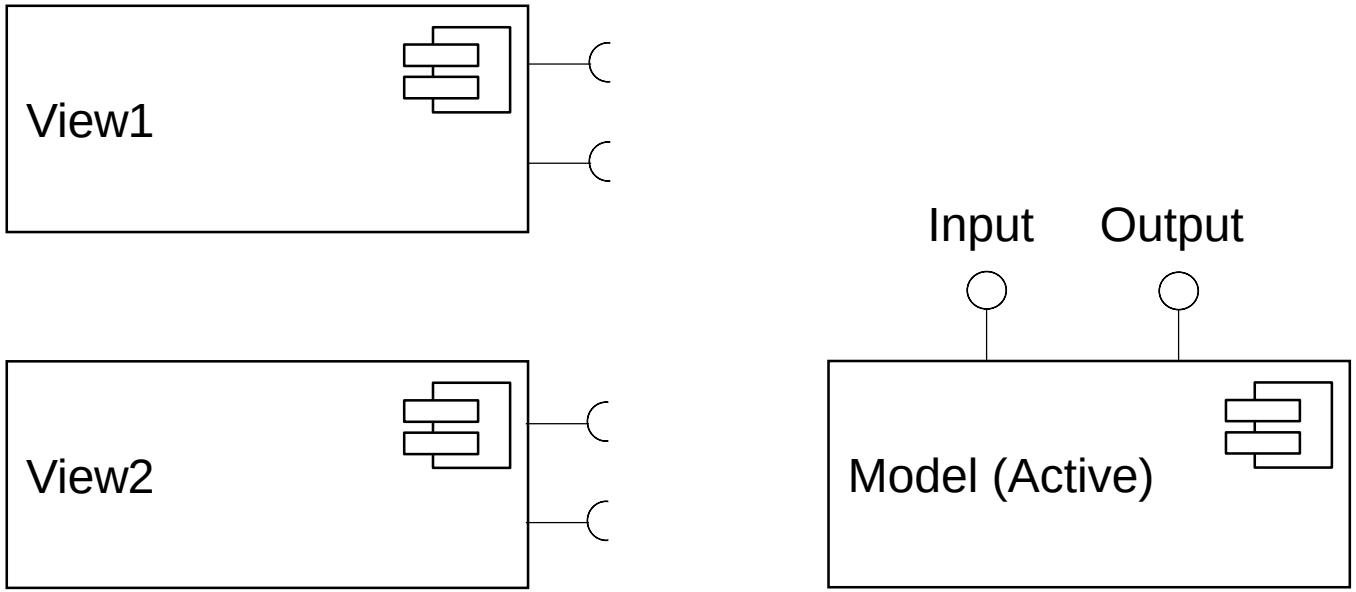
Where are business rules?



COMPONENTS

View

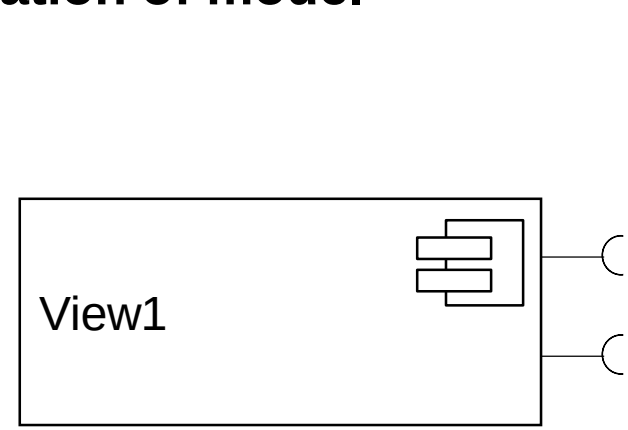
- **Visualization of model**

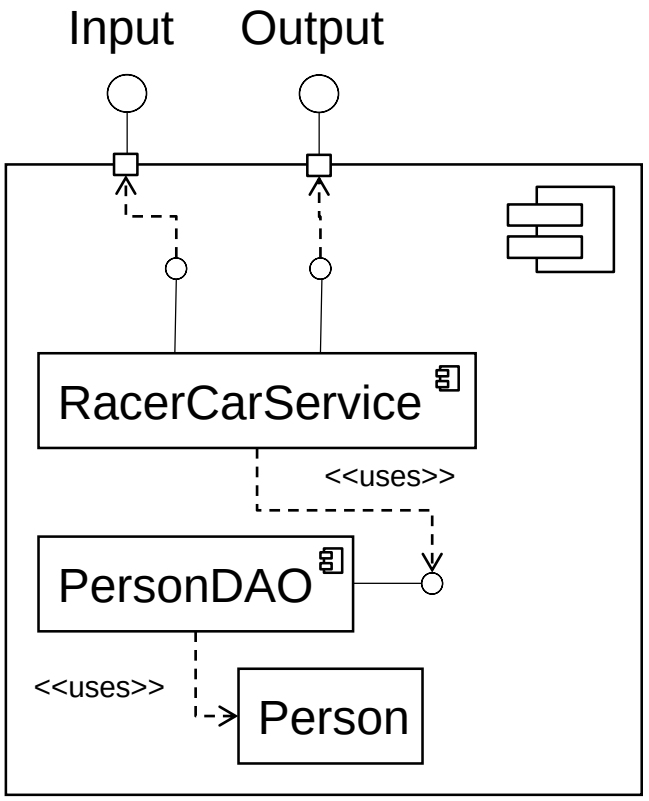


COMPONENTS

View

- **Visualization of model**

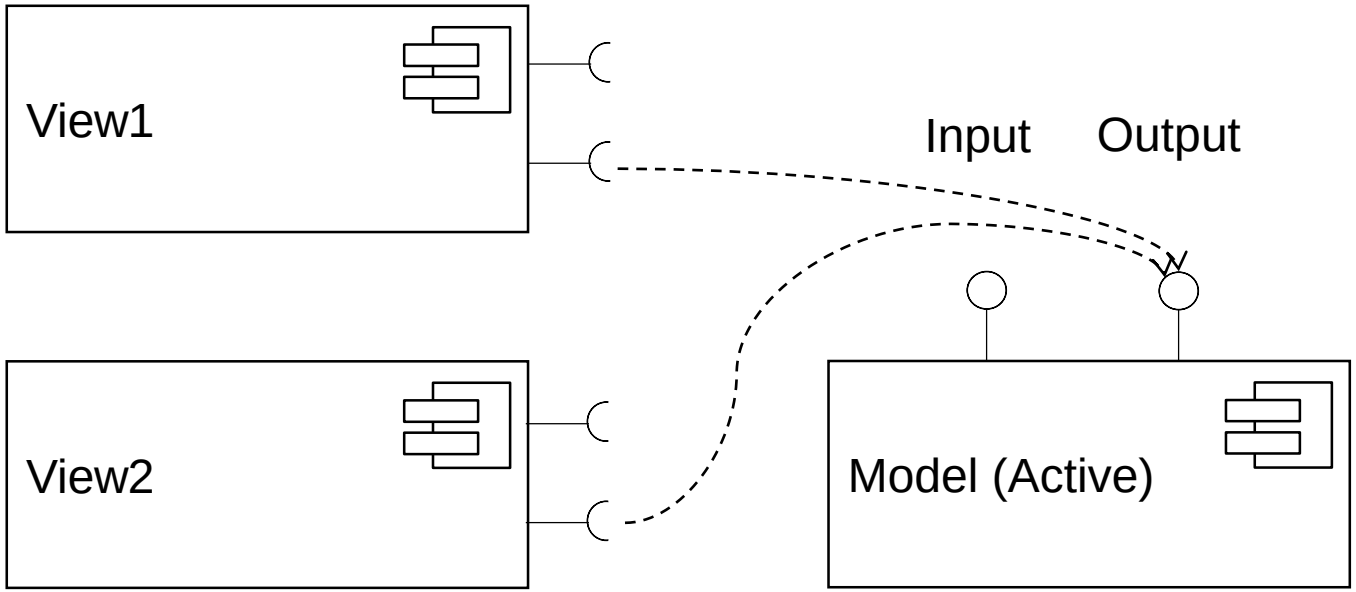




COMPONENTS

View

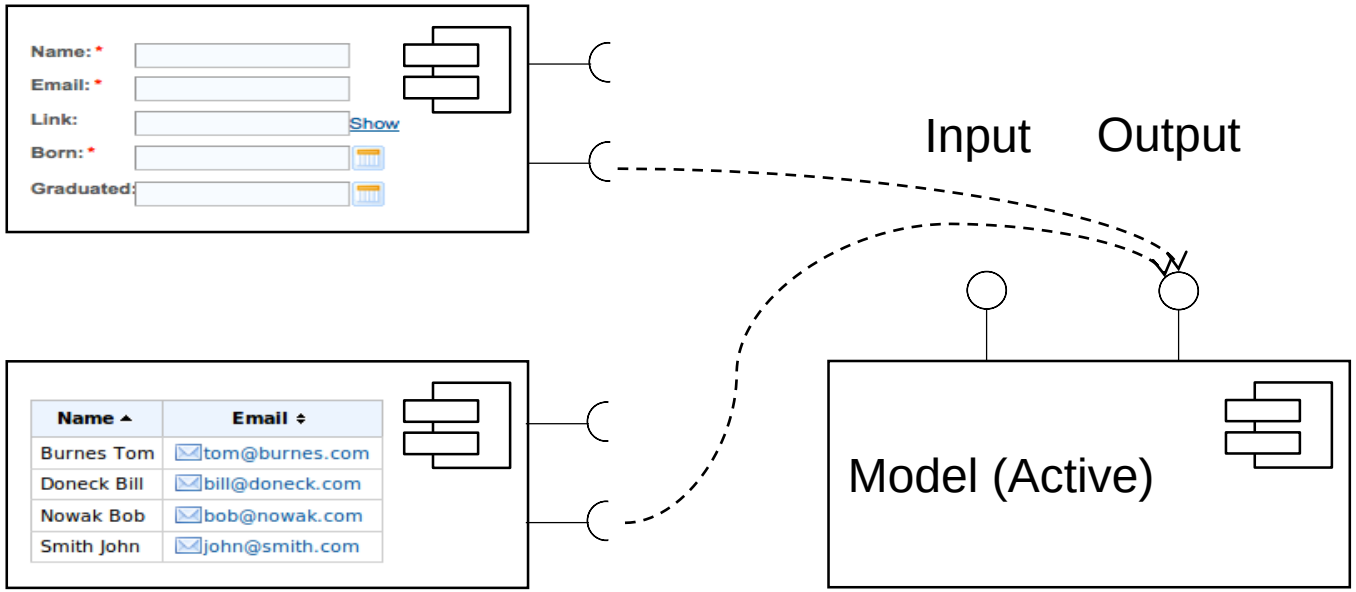
- **Visualization of model**



COMPONENTS

View

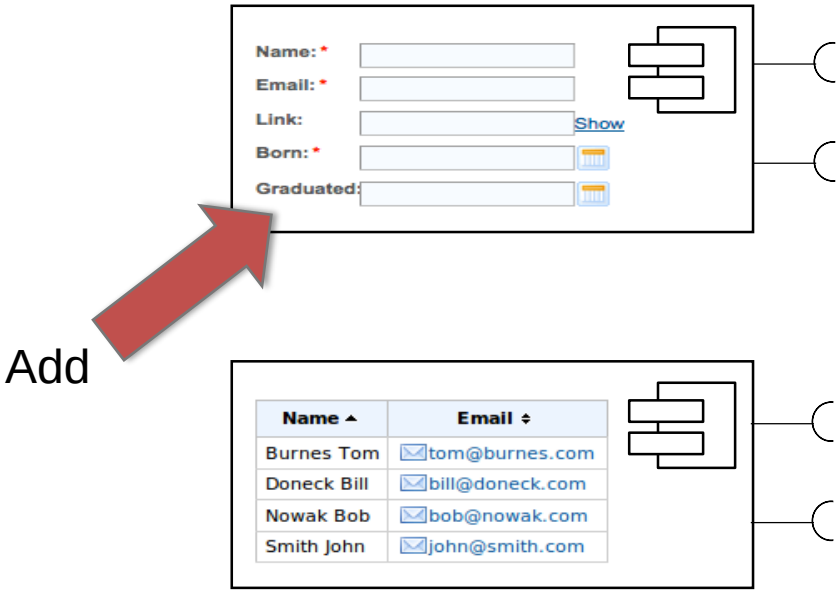
- **Visualization of model**

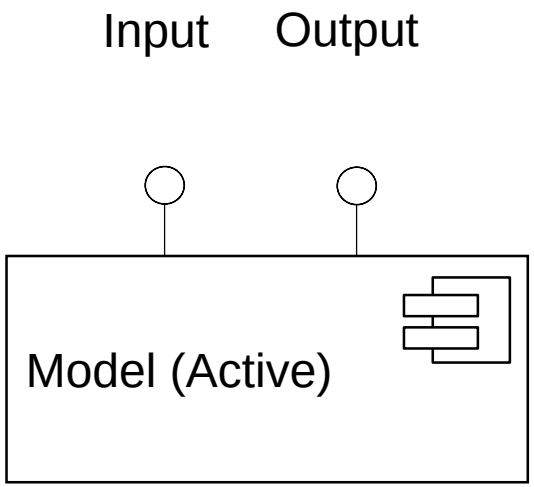


COMPONENTS

View

- **Visualization of model**

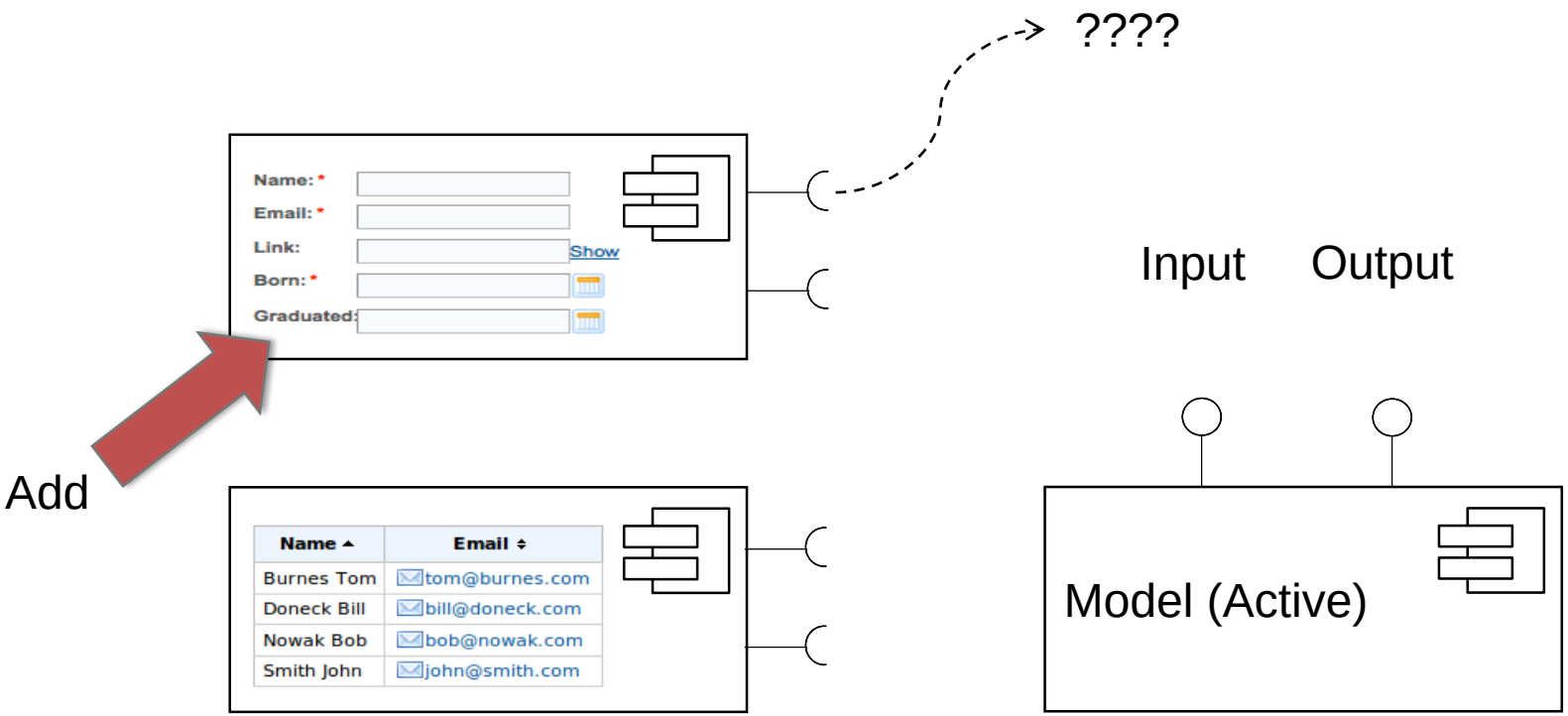




COMPONENTS

View

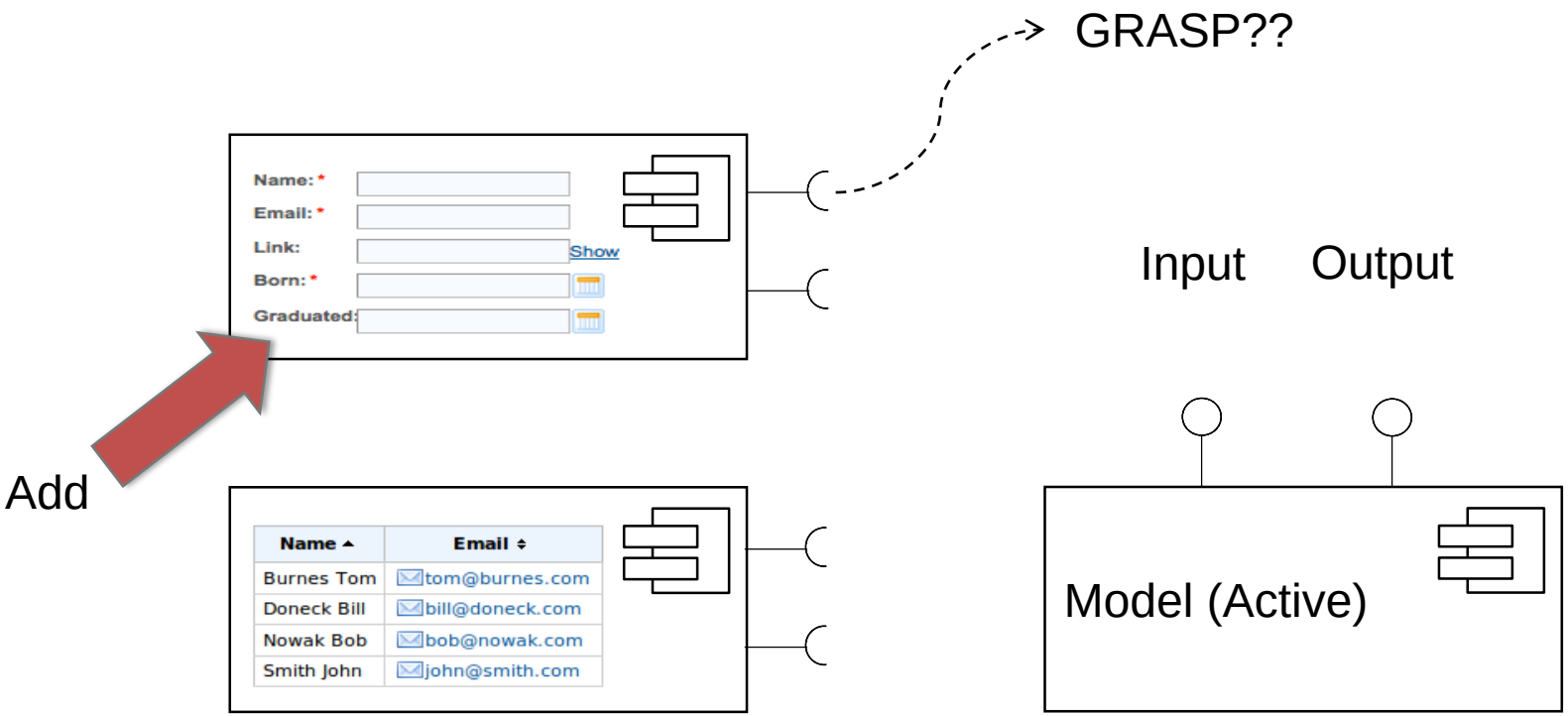
- **Visualization of model**



COMPONENTS

View

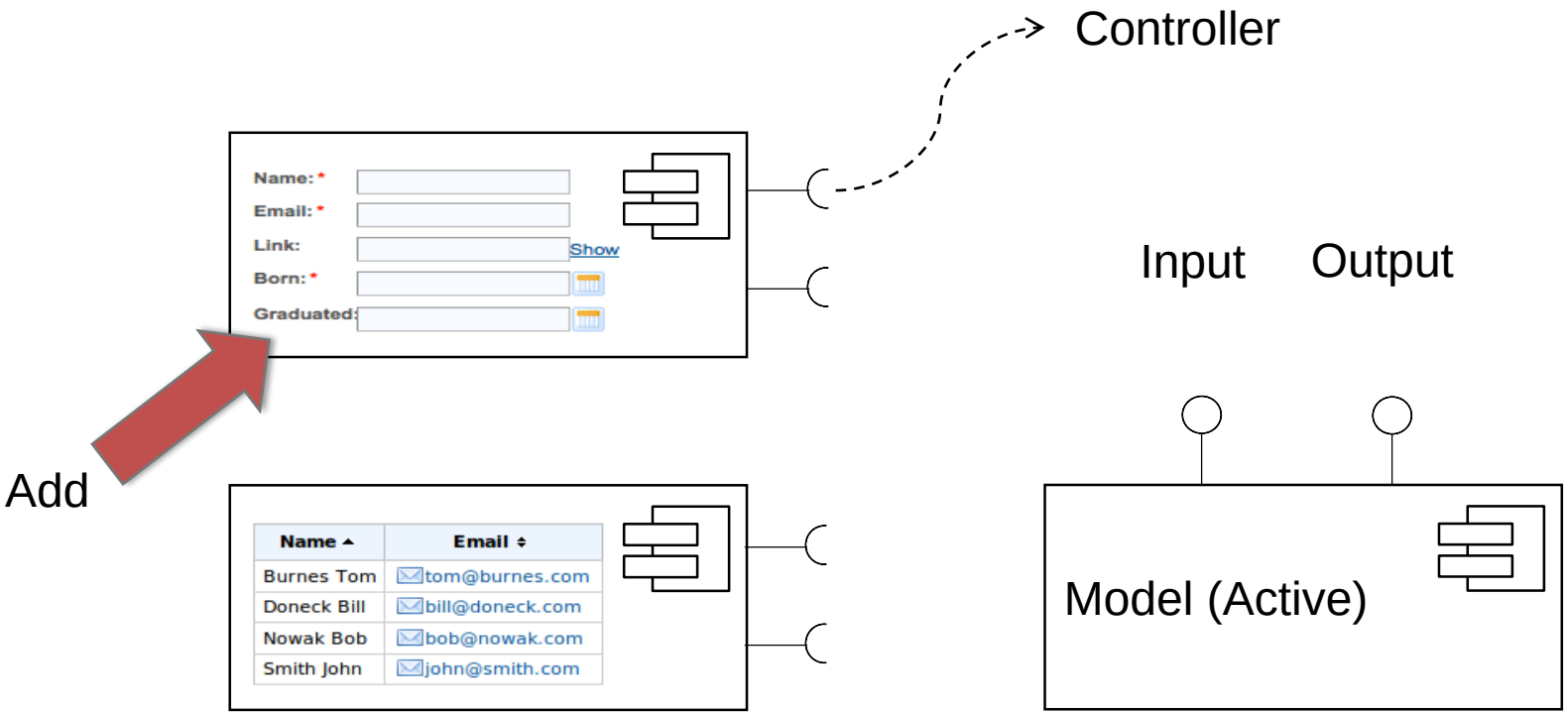
- **Visualization of model**



COMPONENTS

View

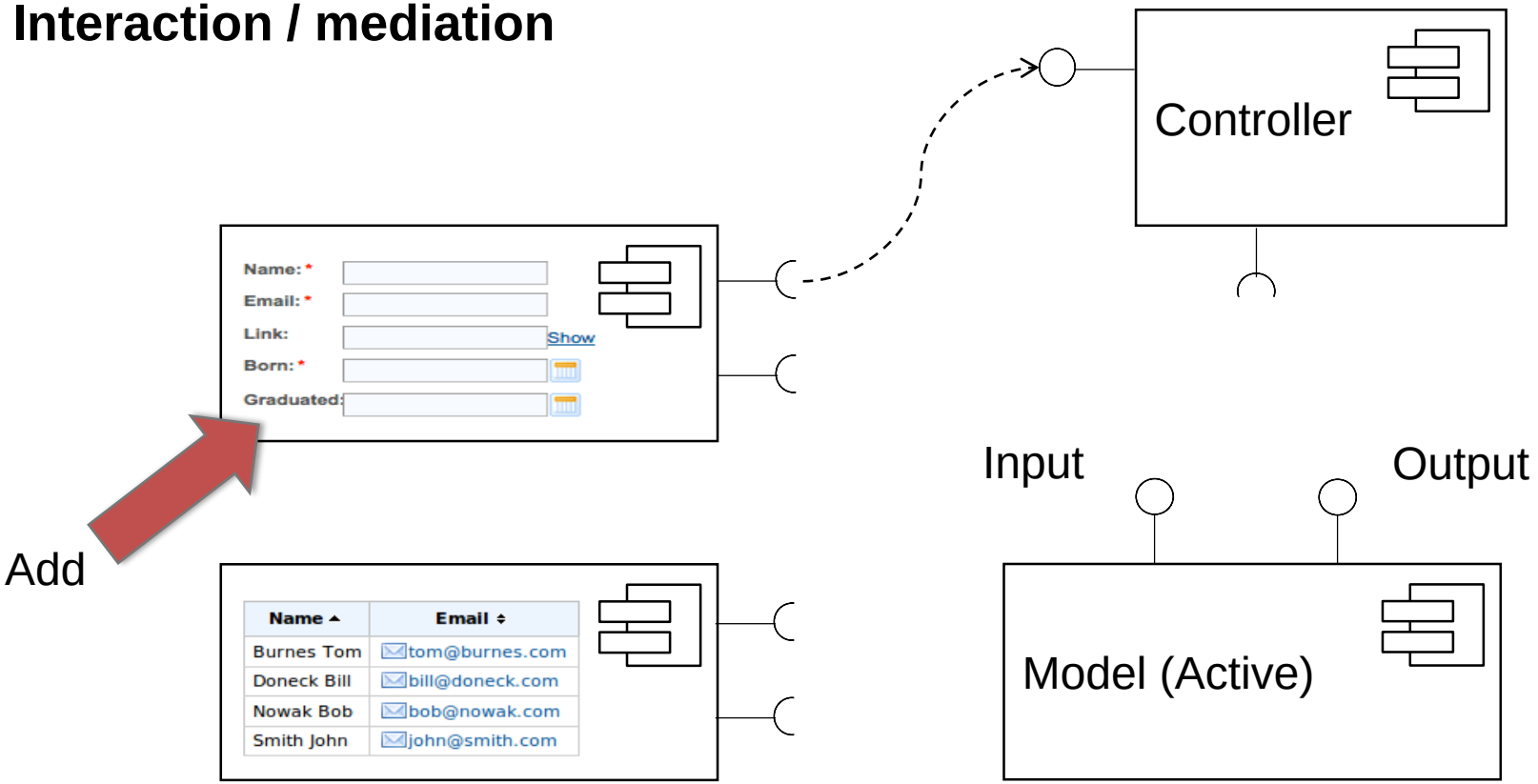
- **Visualization of model**



COMPONENTS

Controller

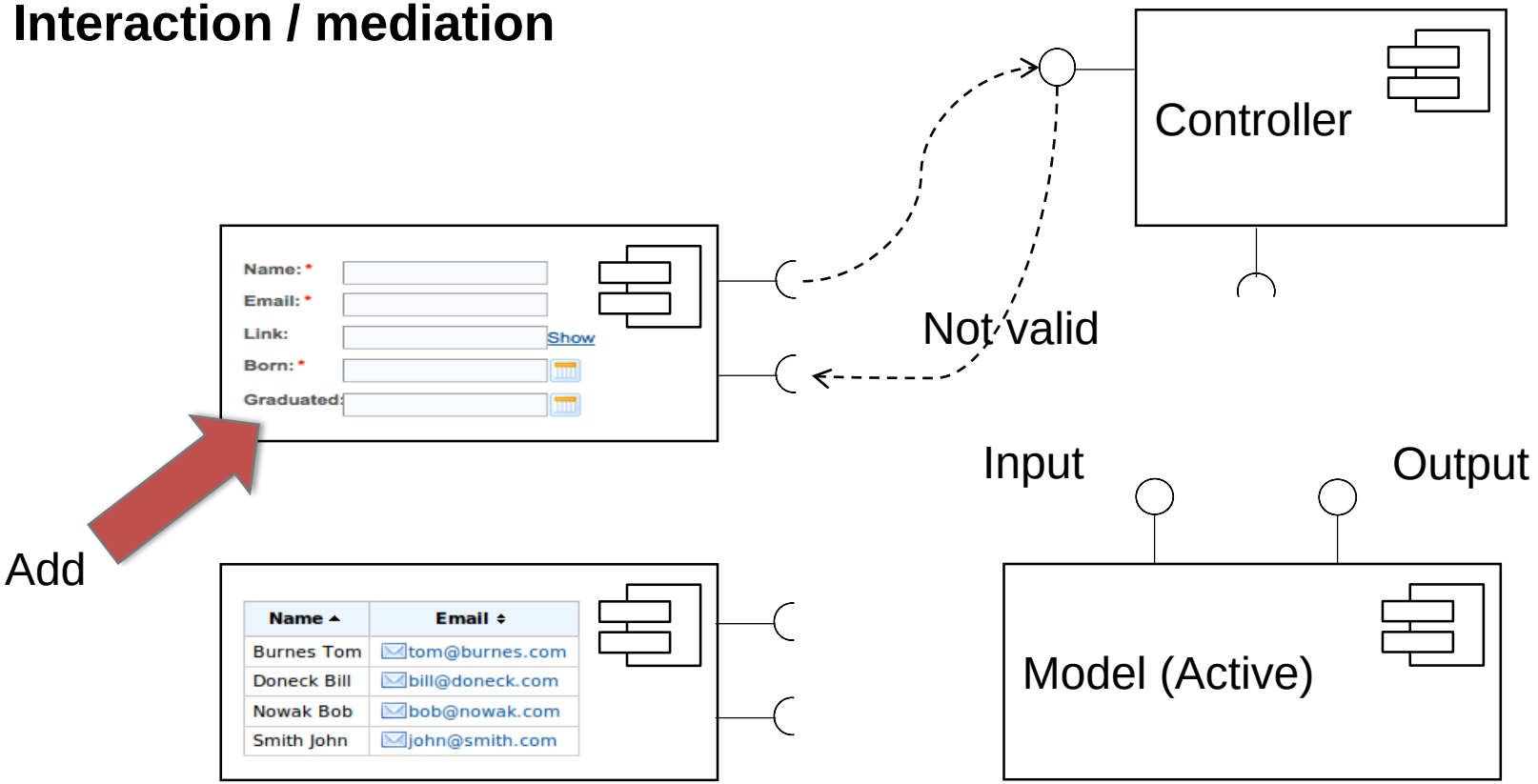
- **Interaction / mediation**



COMPONENTS

Controller

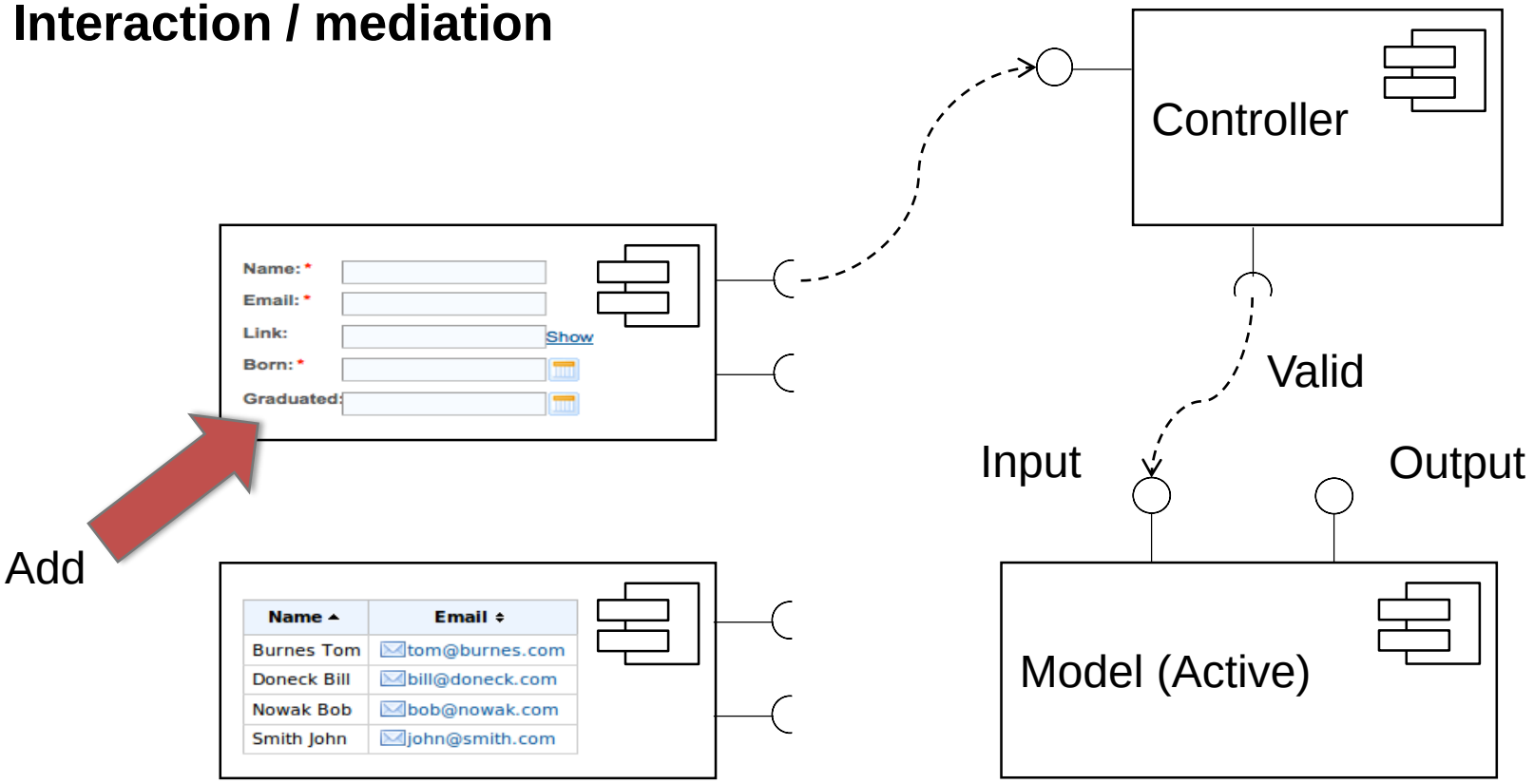
- **Interaction / mediation**



COMPONENTS

Controller

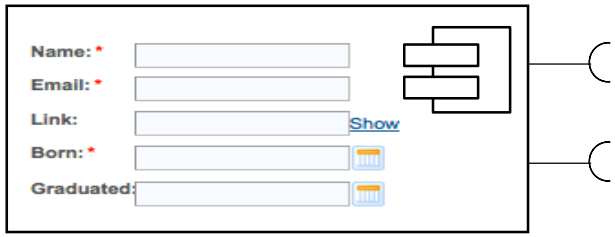
- **Interaction / mediation**



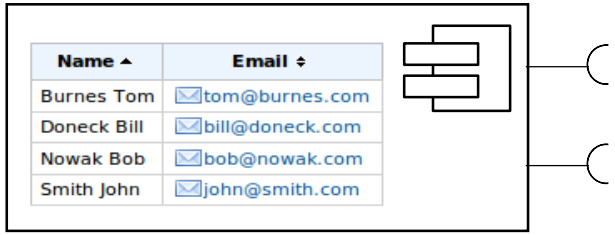
COMPONENTS

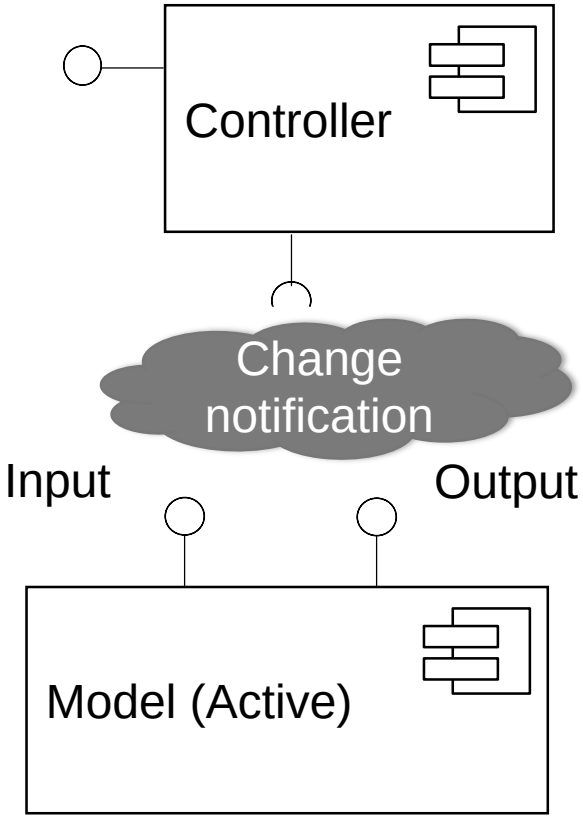
Model

- **Notification mechanism**
- **Observer Design Pattern**



Add

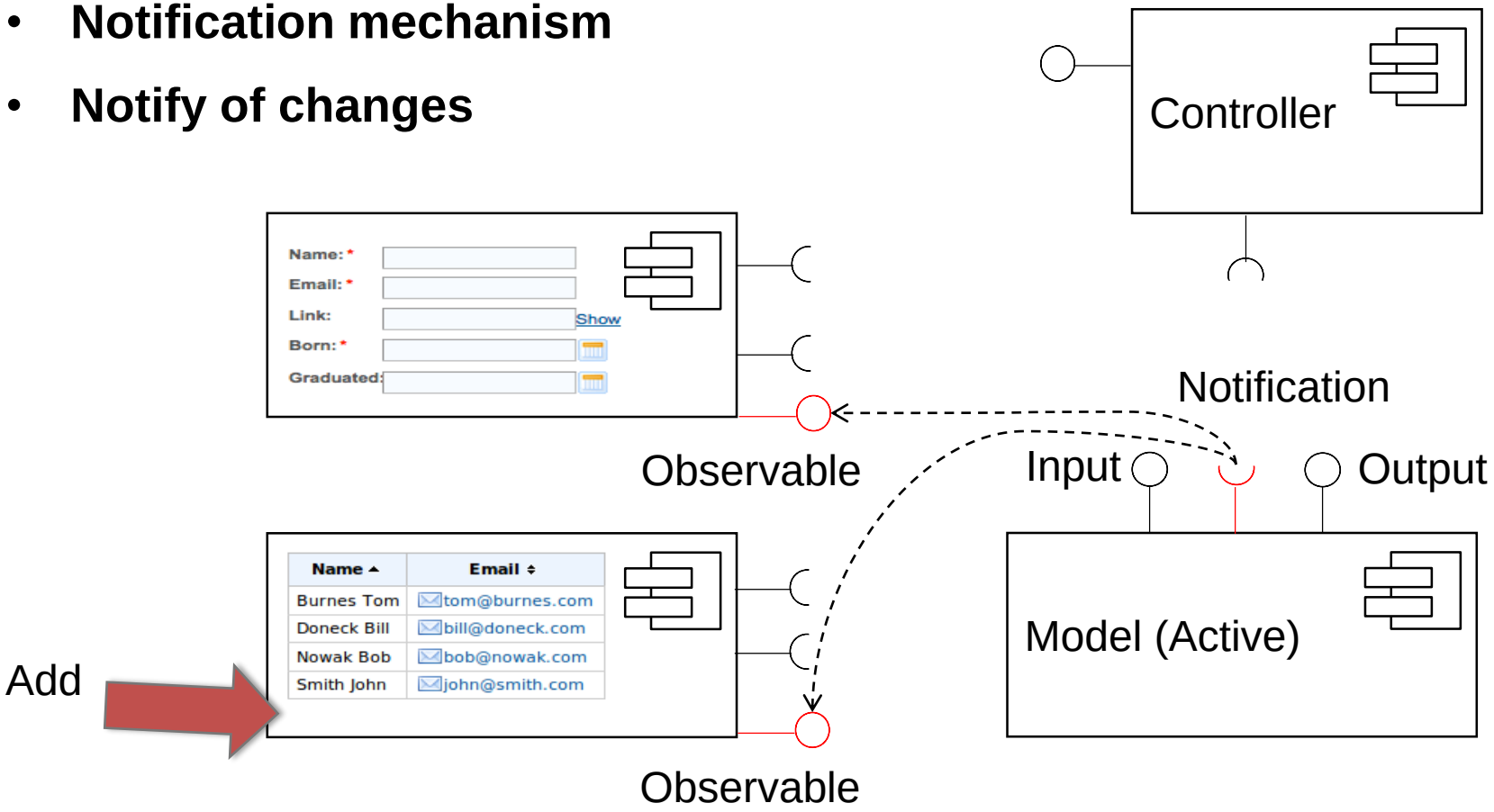




COMPONENTS

Model

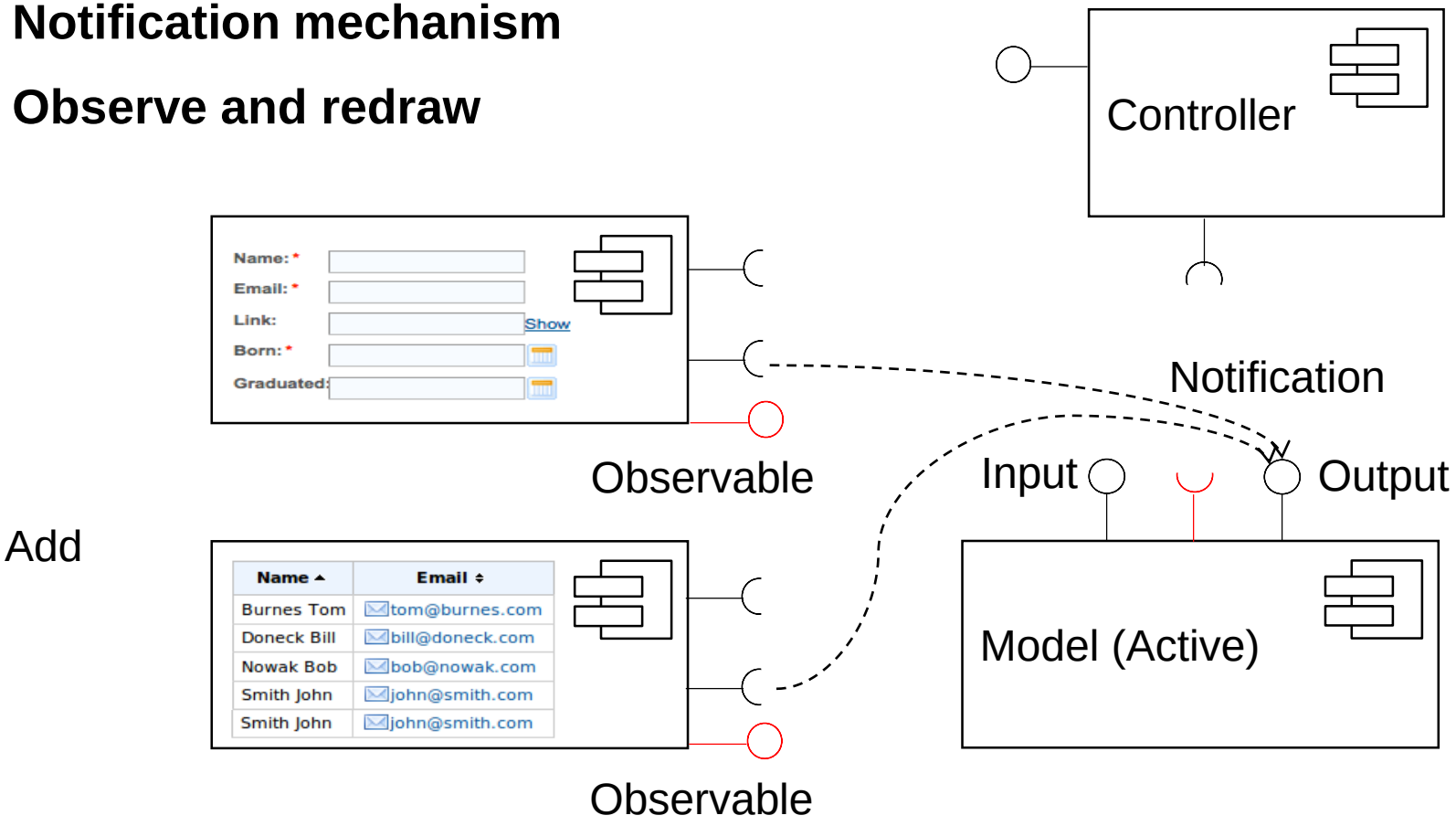
- **Notification mechanism**
- **Notify of changes**



COMPONENTS

Model

- **Notification mechanism**
- **Observe and redraw**



CONTROLLER

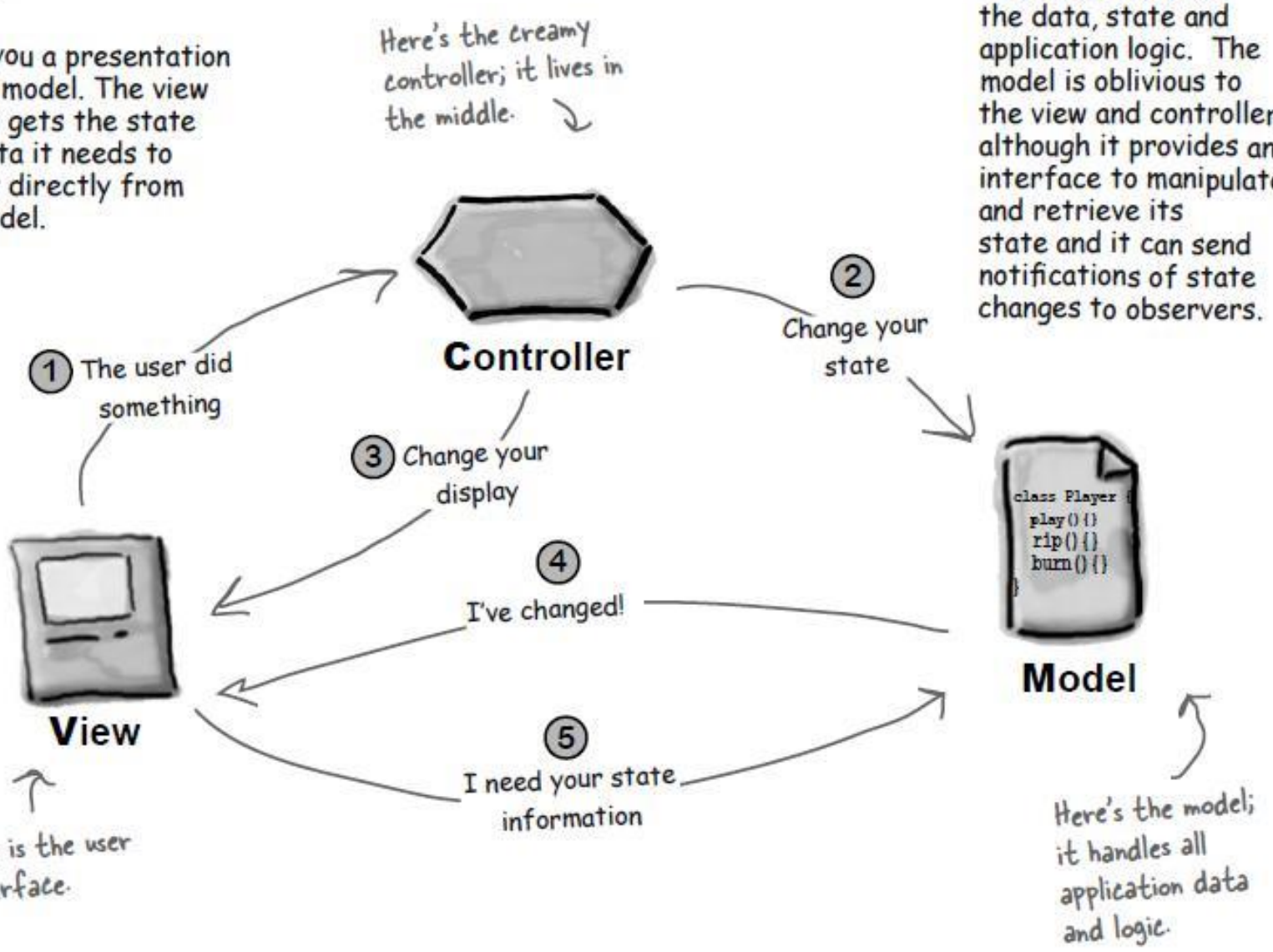
Takes user input and figures out what it means to the model.

MODEL

The model holds all the data, state and application logic. The model is oblivious to the view and controller, although it provides an interface to manipulate and retrieve its state and it can send notifications of state changes to observers.

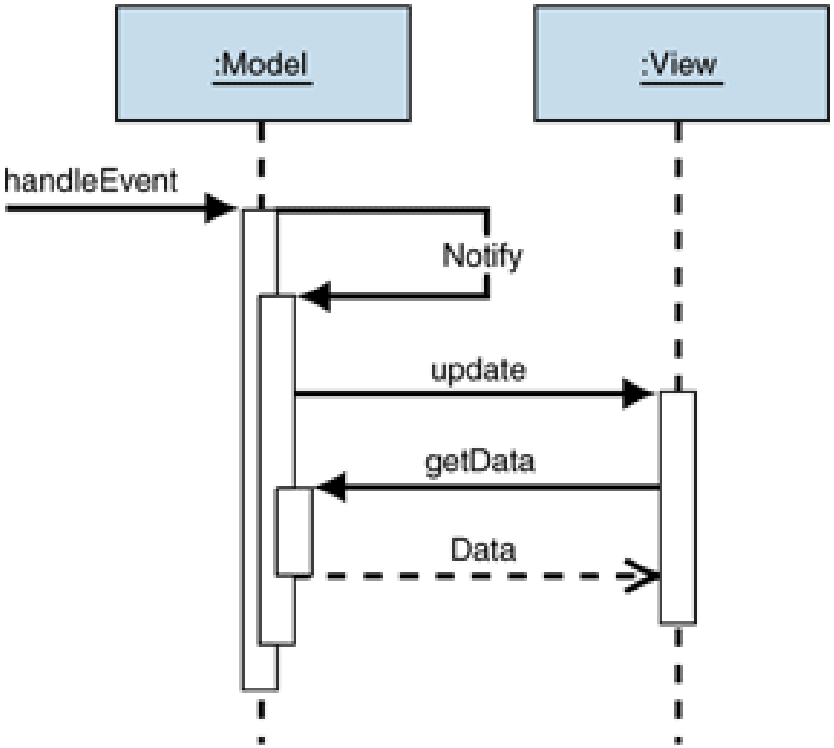
VIEW

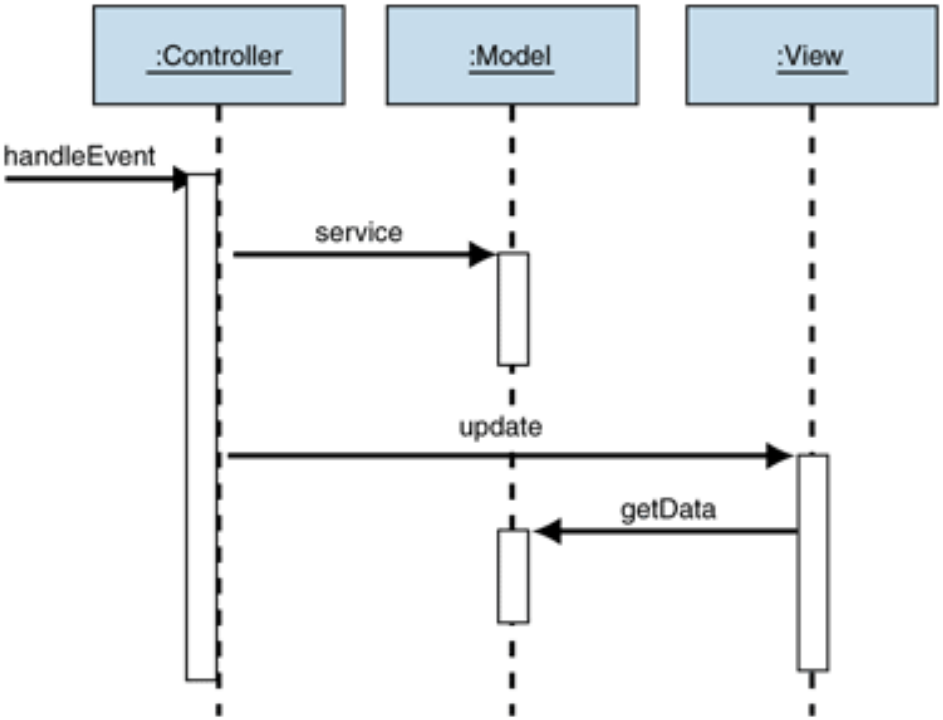
Gives you a presentation of the model. The view usually gets the state and data it needs to display directly from the model.



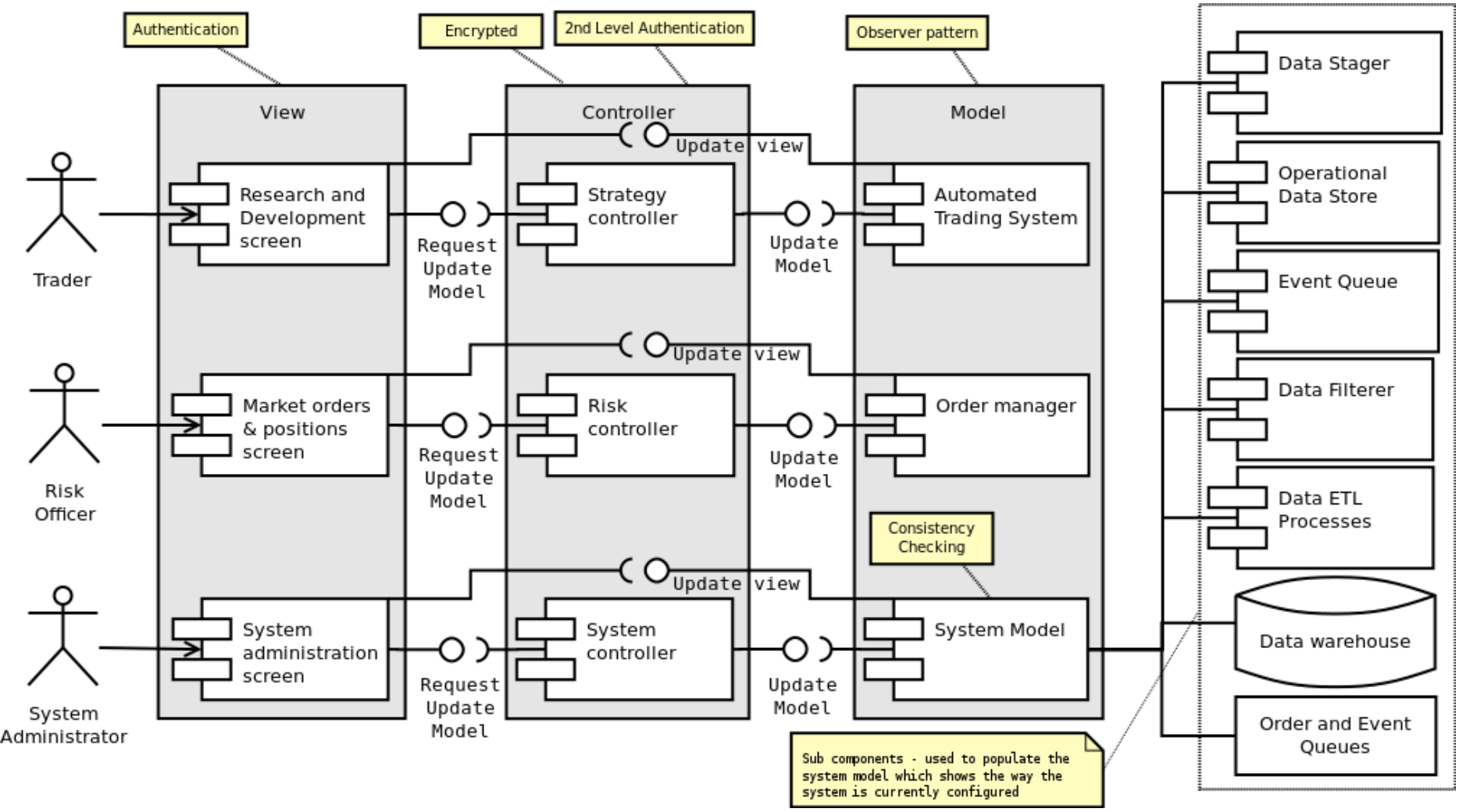
COMPONENTS

Active model **vs** **Passive model**





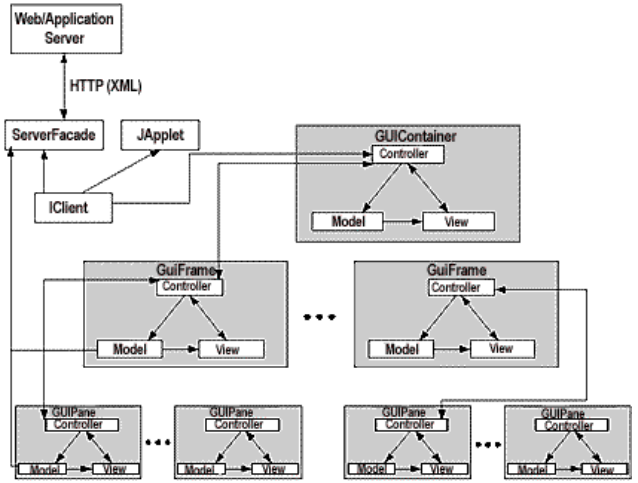
LARGE SYSTEM



ALTERNATIVES FOR VIEW

Many other design patterns for view handling

- **Hierarchical MVC**
- **Presentation Abstraction Control (PAC)**
- **Model View Presenter**
- **Model View ViewModel**



- **Small variants**
  - Page Controller
  - Front Controller
  - Template View

STATIC ANALYSIS OF THE CODE

STATIC CODE ANALYSIS

Java, C++, C#, etc. are GPL with strong type safety!

- Finding errors by the type safety
- Easy to refactor

```
int x = 5;
x = x + 1;
String b = “Java rocks!“;
println(x + b);
```

PHP, Ruby, Python, etc. “

- Weak type safety
- Error-prone
- Tests needed for refactoring!

```
$x = 5;
$x = $x + 1;
$b = “PHP rocks!“;
echo $x + $b;
```

STATIC CODE ANALYSIS

Java, C++, C#, etc. are GPL with strong type safety!

- Finding errors by the type safety
- Easy to refactor

```
int x = “Five“;
x = x + 1;
String b = “Java rocks!“;
println(x + b);
```

Compile Error

PHP, Ruby, Python, etc. “

- Weak type safety
- Error-prone
- Tests needed for refactoring!

```
$x = “Five“;
$x = $x + 1;
$b = “PHP rocks!“;
echo $x + $b;
```

Runtime Error

STATIC CODE ANALYSIS

- **Analysis of computer software**
  - Performed without actually executing programs
  - Highlighting possible coding errors
  - Formal methods to mathematically prove properties of a program
  - Software metrics and reverse engineering can be described as forms of static analysis.

  - Verification of properties of software used in safety-critical computer systems and locating potentially vulnerable code
    - Nuclear power plant, Aviation, Medical software

  - Code audit

STATIC CODE ANALYSIS

- **PMD**
  - PMD is a source code analyzer.
  - It finds common programming flaws like unused variables, empty catch blocks, unnecessary object creation, and so forth.
  - It supports Java, JavaScript, XML, XSL.
  - Additionally it includes the copy-paste-detector.
    - Finds duplicated code in Java, C, C++, C#, PHP, Ruby, Fortran, JavaScript.

STATIC CODE ANALYSIS

- **FindBugs**
  - Uses static analysis to look for bugs in Java code.
  - Based on the concept of bug patterns.
    - A bug pattern is a code idiom that is often an error.
      - Difficult language features
      - Misunderstood API methods
      - Misunderstood invariants when code is modified during maintenance
      - Garden variety mistakes: typos, use of the wrong boolean operator

SOFTWARE TESTING

investigation conducted to provide investor with information about the quality of the product

Classification:

- Static vs Dynamic
  - Code audit vs Program Execution
- White vs Black box
  - We know vs We do not know what is inside
- Alpha vs Beta
  - SW Constructions vs Transition to deployment
- Functionality vs Perfomance vs Security vs Reliability
- Validation vs Verification

SOFTWARE TESTING

Rule 40/20/40 Planning/Implementation/Testing

Scale

- **Unit Testing** – independent
- **Component testing**
- **Integration tests**
- **System test**

SOFTWARE TESTING

Other categories

- **Regressive testing**
- Alpha
  - Development testing
- Beta
  - It scales and provides all advertised functionality
- Acceptance
  - Verifies it meets customers expectations

SOFTWARE TESTING

Other categories

- **Stress test**
  - JMeter, Selenium
- **User Interface testing**
  - Selenuim, TestComplete
- **Performance, Usability, Compatibility, Error handling, load, volume, security, installation, maintenance, accessibility..**

JUNIT - HTTP://JUNIT.SOURCEFORGE.NET/

- **JUnit is a simple framework to write repeatable tests.**
- **For graphical feedback, most IDE's support JUnit 4.**

- **Implementace**
  - Annotate a method with @org.junit.Test
  - To check a value, import org.junit.Assert.* statically
    - call assertTrue() and pass true if the test succeeds

```
@Test
public void simpleAdd() {
     Money m12CHF= new Money(12, "CHF");
     Money m14CHF= new Money(14, "CHF");
     Money expected= new Money(26, "CHF");
     Money result= m12CHF.add(m14CHF);
     assertTrue(expected.equals(result));
}
```

JUNIT

- **What if you have two or more tests that operate on the same or similar sets of objects?**
- **Test Fixture**
  - Before or After

```
public class MoneyTest {
    private Money f12CHF;
    private Money f14CHF;
    private Money f28USD;

 @Before
 public void setUp() {
    f12CHF= new Money(12, "CHF");
    f14CHF= new Money(14, "CHF");
    f28USD= new Money(28, "USD");
 }
}
```

JUNIT

- **Run test**
  - `Console`

```
org.junit.runner.JUnitCore.runClasses(TestClass1.class, ...);
```

  - `Command line`

```
java org.junit.runner.JUnitCore TestClass1 [...other test classes...]
```

  - `TestRunner`

```
public static junit.framework.Test suite() {
                    return new JUnit4TestAdapter(Example.class);
}
```

- **Expected Exceptions**

```
@Test(expected= IndexOutOfBoundsException.class)
public void empty() {
                    new ArrayList<Object>().get(0);
}
```

JUNIT

- **Run test**
  - Do not expect test order execution
  - If you must then use

```
@FixMethodOrder(MethodSorters.NAME_ASCENDING)
```

JUNIT

- **Feed with Parameters**

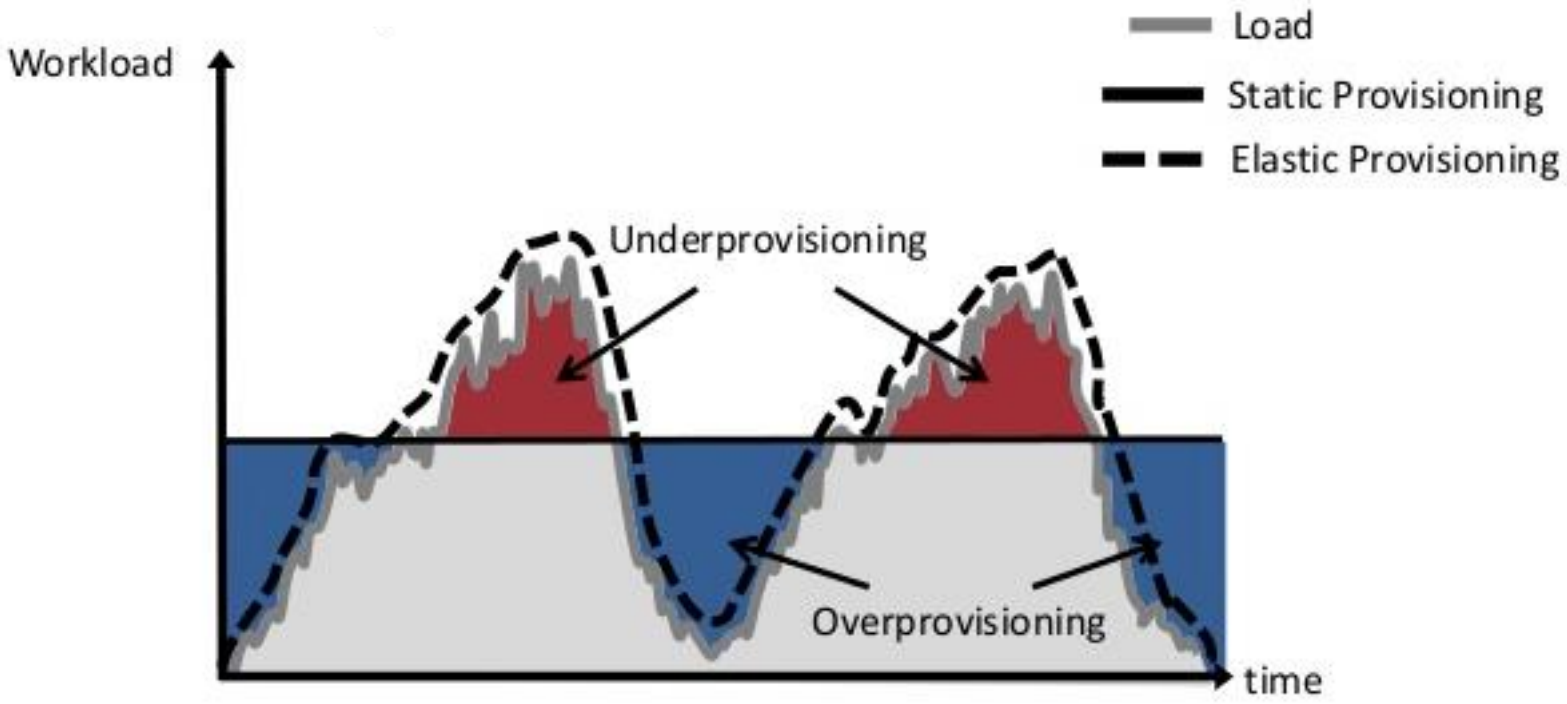
```
@RunWith(Parameterized.class)
public class FibonacciTest {

  @Parameters(name = "{index}: fib({0})={1}")
  public static Iterable<Object[]> data() {
    return Arrays.asList(new Object[][] {
          { 0, 0 }, { 1, 1 }, { 2, 1 }, { 3, 2 },
          { 4, 3 }, { 5, 5 }, { 6, 8 }
    });
  }

  private int input;
  private int expected;

  public FibonacciTest(int input, int expected) {
    this.input = input;
    this.expected = expected;
  }
  @Test
  public void test() {
    assertEquals(expected, Fibonacci.compute(input));
  }
}
```

Many loops
Different
data

DEPLOYMENT, MAINTENANCE AND REPORTS

The games just begin

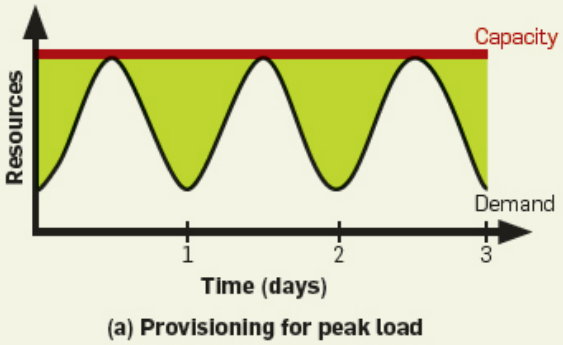
- Get ready to be available next 48 hours in the row!



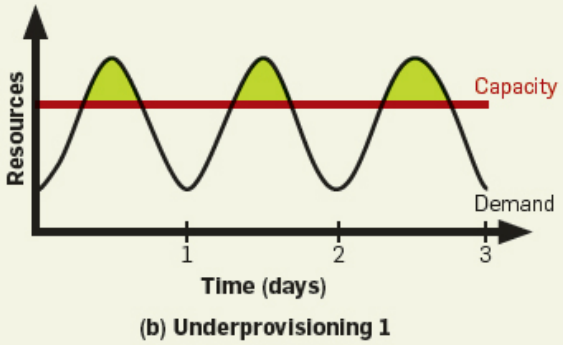
DEPLOYMENT, MAINTENANCE AND REPORTS

The games just begin

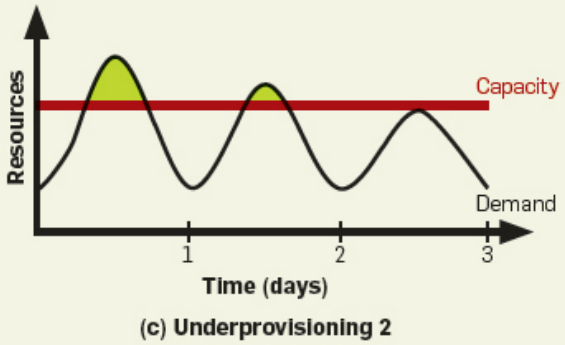
- Get ready to be available next 48 hours in the row!



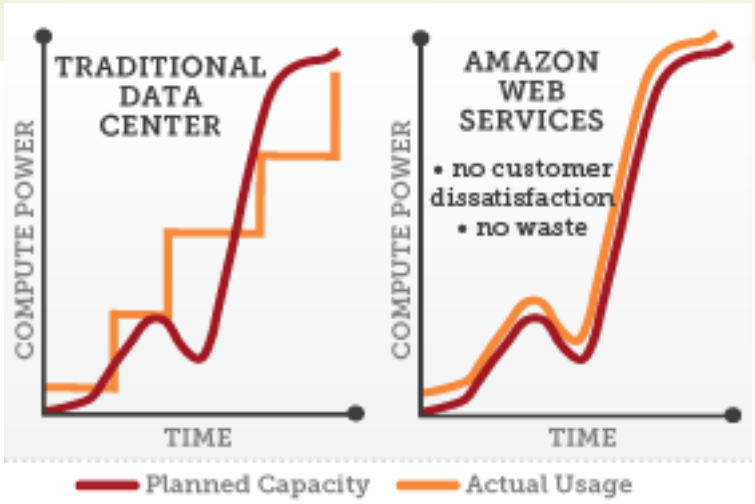
(a) Provisioning for peak load



(b) Underprovisioning 1



(c) Underprovisioning 2



DEPLOYMENT, MAINTENANCE AND REPORTS

At any time you must know what is happening

- Program Log
  - Logger
- Access log
  - Who is there doing what
- System monitoring
  - CPU load, Memory, Harddrive use, Backups
- Error reporting
  - Where? JIRA? Email?

DEPLOYMENT, MAINTENANCE AND REPORTS

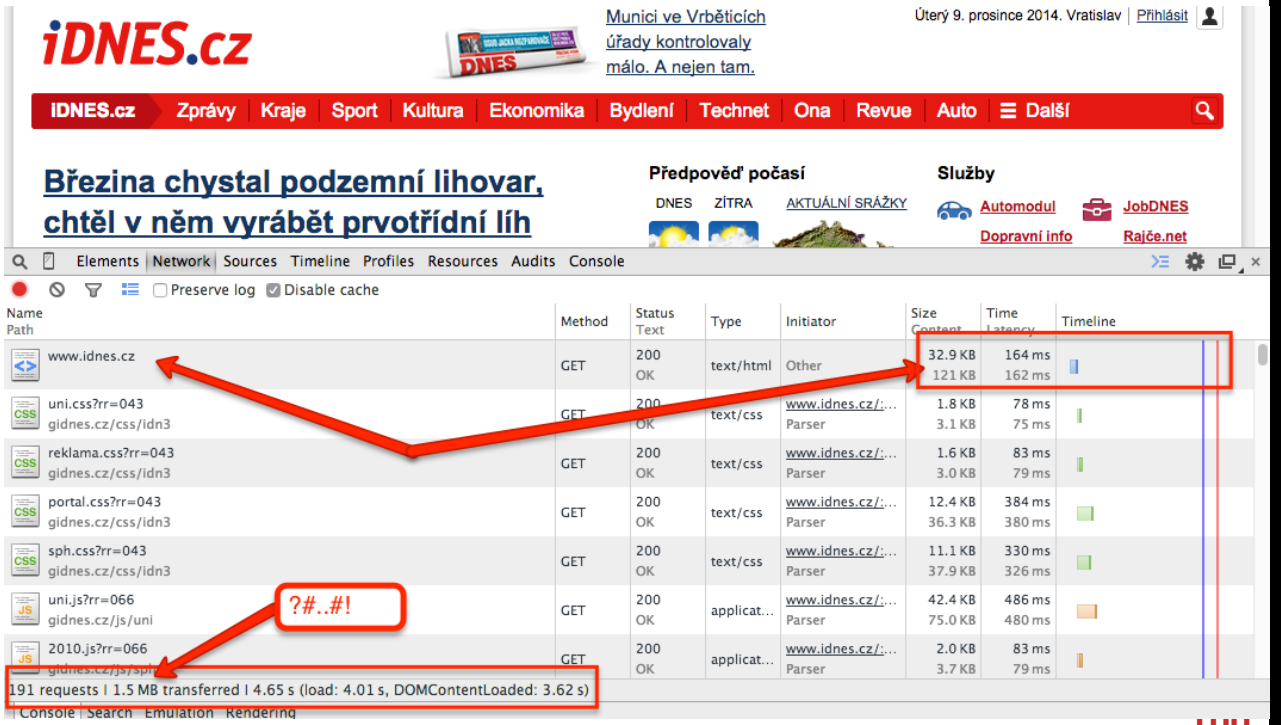
Automated monitoring systems

- Verify CPU/Load/Mem/Disk and issues
- Notify by email or SMS
  - Zabbix
  - Nagios
  - … many others.

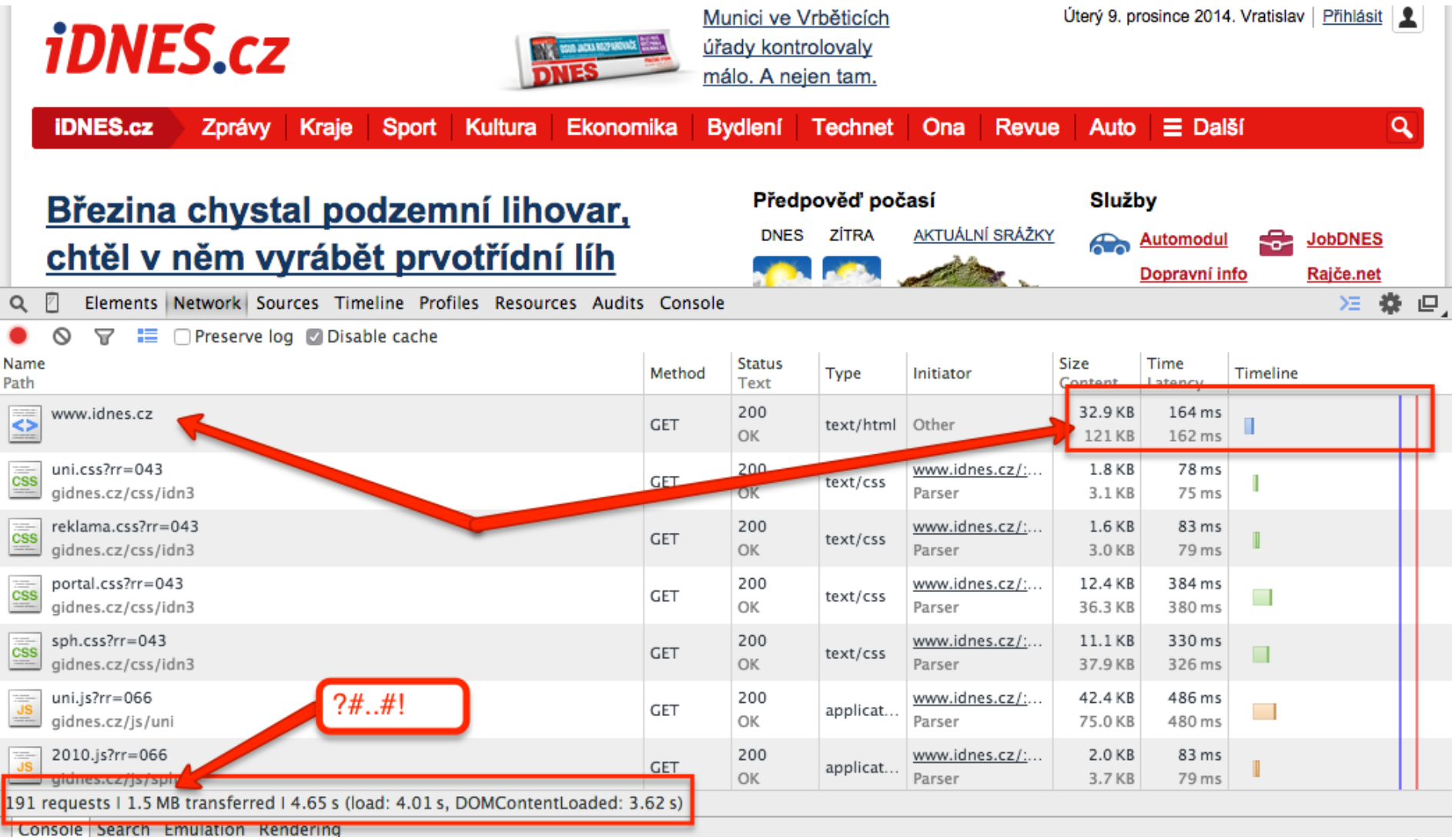
DEPLOYMENT, MAINTENANCE AND REPORTS

Get user experience

- Register your app at Google Analytics to see how it performs
- HTML5 allows you to see the statistics



DEPLOYMENT, MAINTENANCE AND REPORTS



EVIL THINGS TO HAPPEN

System crash!

- Memory leak
- Resource leak
- New closing loop
- Cannot survive the load

- Heap analysis
- Profiler
- Reports from web server

MAINTENANCE

The king is dead, long live the king!

Success of the long live depends on the design, scalability, ability to fix bugs and many many other factors!!

Prevent bugs, leaks, flaw design

Use Design Patterns!

A7B36ASS – STM Architektura SW systémů

http://www.fel.cvut.cz/education/bk/predmety/14/69/p1469406.html