INTRODUCTION
TO
SOFTWARE DESIGN

FROM ANALYSIS TO DESIGN,
ENTERPRISE APPLICATION ARCHITECTURE (EAA),
PERSISTENT LAYER OF EAA

WHAT TO EXPECT FROM SOFTWARE DESIGN

Problem analysis (Komarek)

- Analysis gave us a real-world view/perspective to the problem
- No software constraints were considered
- Objects/concepts of domain

Software design (Cerny)

- Finding an appropriate software structure to capture the problem
- Determining architecture
- Design objects reflect software constraints
  - Data types, structural decomposition, databases, networks
  - Artificial objects that do not have anything to do with real world

WHAT TO EXPECT FROM SOFTWARE DESIGN

Problem analysis

- System is understood as a black box (BB)
- Analysis described input and output expected from the BB
- Black box uses Domain terminology

Software design

- Opens the BB and aims to design what is inside
- Multiple approaches to do that influenced by many factors

WHAT TO EXPECT FROM SOFTWARE DESIGN

Problem analysis

- We know what customer wants
- Vision
- Functional requirements
- Domain model
- Scenarios (use cases)
- Business models

Software design

- **Motivation**
    - Do not reinvent the wheel
    - Reuse what possible
    - Design something easy to comprehend, easy to read
    - Provide the expected functionality

WHAT TO EXPECT FROM SOFTWARE DESIGN

Problem analysis

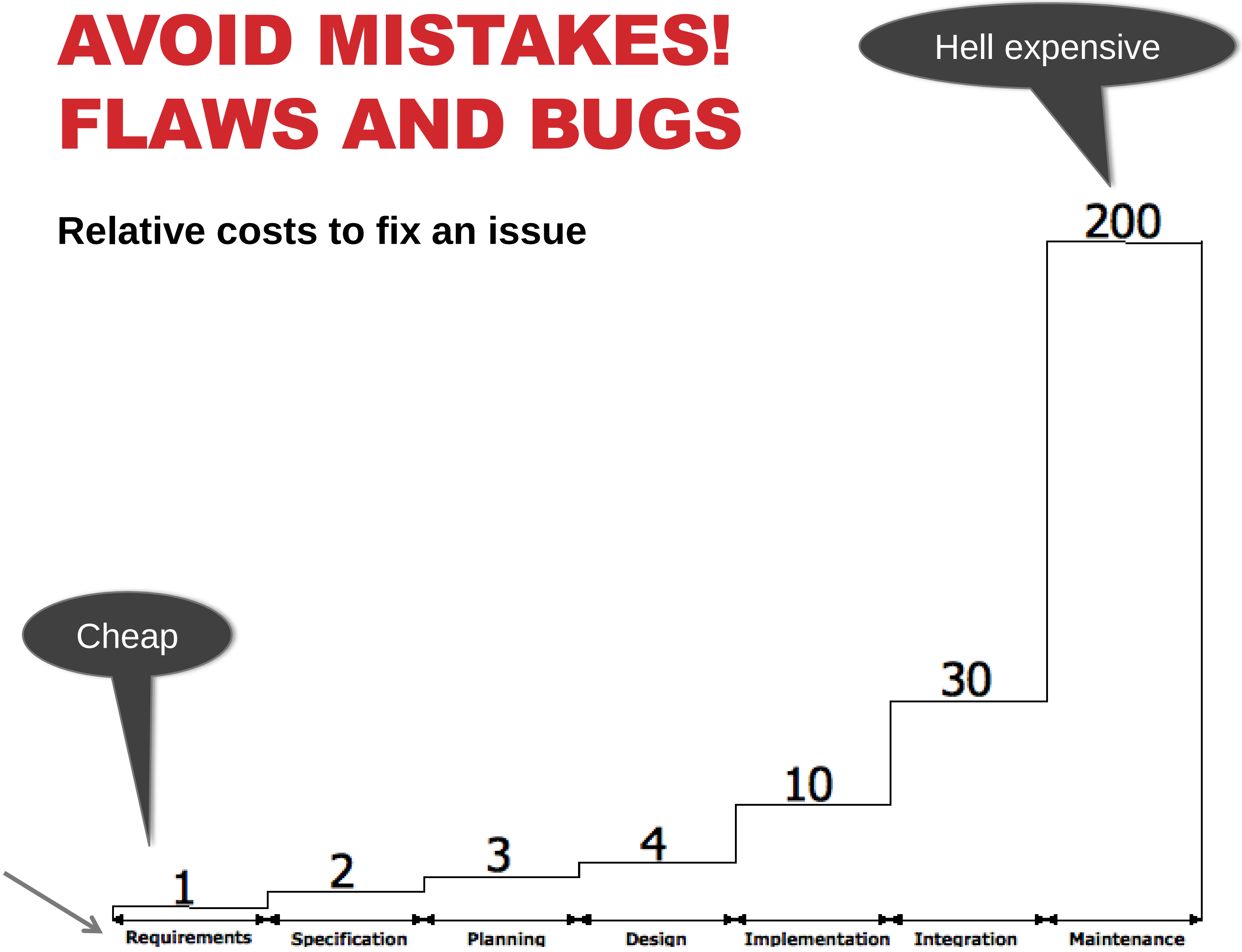
- We know what customer wants
- Vision
- Functional requirements
- Domain model
- Scenarios (use cases)
- Business models

Software design

- **Object oriented, Component based, Industry standards!**

AVOID MISTAKES! FLAWS AND BUGS

Relative costs to fix an issue



THINK TWICE BEFORE YOU START

When designing software

- **Expect long live**
- **Expect you work in team**
  - what you do, should be understood by others
- **When you design something, make it easy for other to understand it**
- **Expect that other spend most of the time <u>reading</u> how it works!**
  - Abstraction!!
  - Interfaces!!
  - Annotations!!

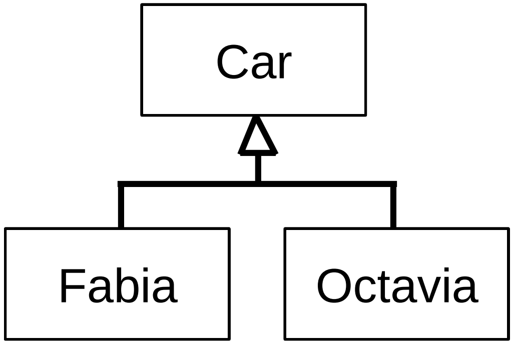
MOTIVATION EXAMPLE

```
Fabia a(ArrayList<Fabia> list) {
    Fabia f = list.get(0);
    f.produce();
    f.test();
    f.ship();
    return f;
}
```

```
Car b(List<Car> list) {
    Car c = list.get(0);
    c.produce();
    c.test();
    c.ship();
    return c;
}
```

Be abstract!

- List vs. ArrayList
- Car vs. Fabia
- Virtual methods!
    - Object-oriented



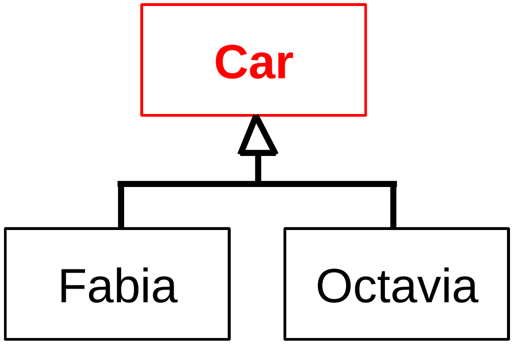
MOTIVATION EXAMPLE

```
Fabia a(ArrayList<Fabia> list) {
    Fabia f = list.get(0);
    f.produce();
    f.test();
    f.ship();
    return f;
}
```

```
Car b(List<Car> list) {
    Car c = list.get(0);
    c.produce();
    c.test();
    c.ship();
    return c;
}
```

Be abstract!

- If you do you get A!



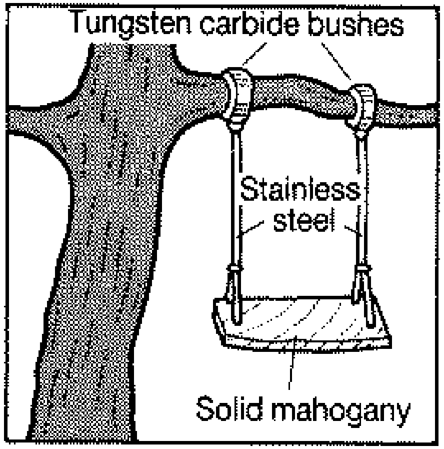
THINK TWICE BEFORE YOU START

Consider these before designing your home nuclear power plant

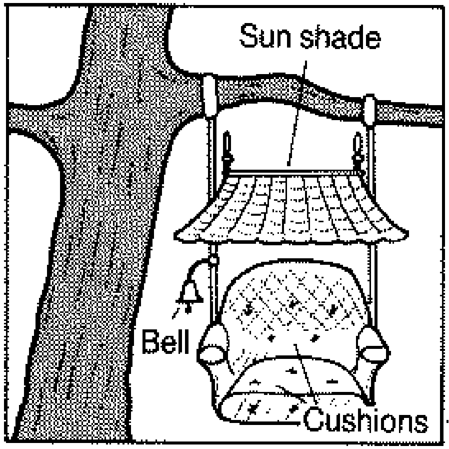
Numbers/Statistics

- **Software maintenance**
  - 65-75% of life-cycle
- **Understanding to the software**
  - 40-60% of the entire maintenance!
- **User Interface**
  - Very expensive!
  - 50% of the entire development of EA

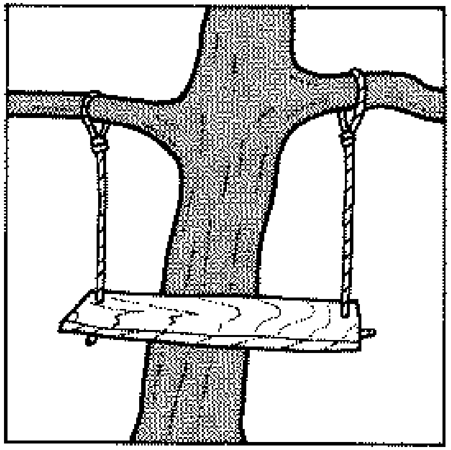
ASSUME NO ONE IS PERFECT, BUT CHARGE THE CUSTOMER



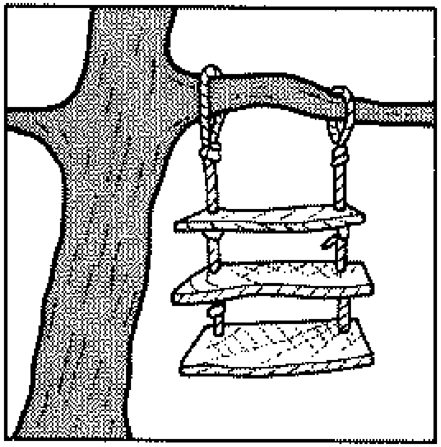
What Product Marketing specified



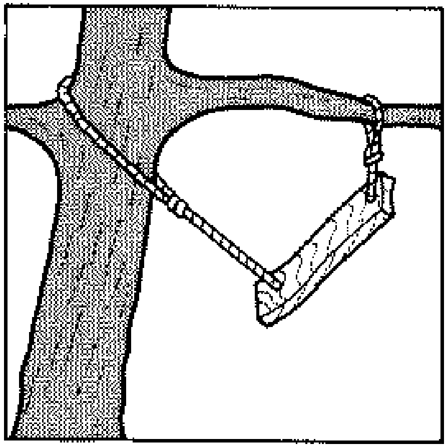
What the salesman promised



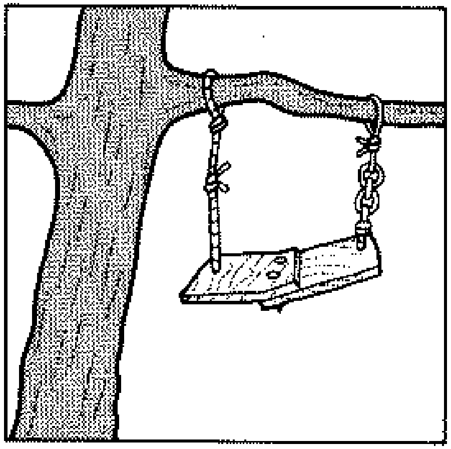
Design group's initial design



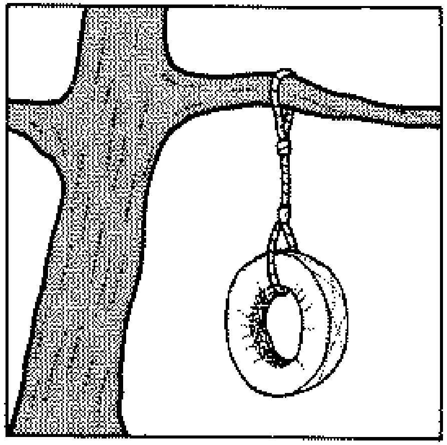
Corp. Product Architecture's modified design



Pre-release version



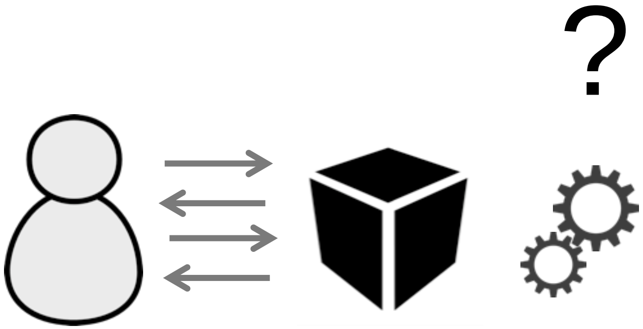
General release version



What the customer actually wanted

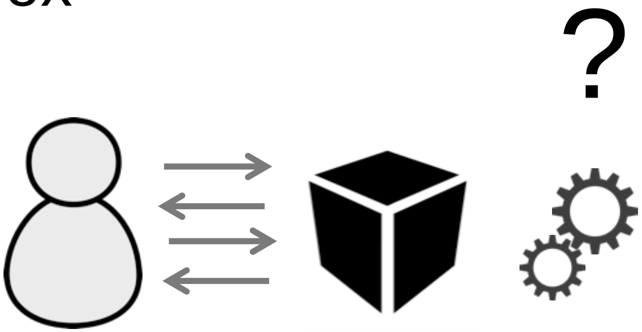
VISUALIZATION OF FIRST STEPS IN DESIGN

- **Keywords**
  - Use Case (Scenario)
  - Domain Model
  - System Sequence Diagram
  - System Operation
  - Operation Contract
  - Design Model



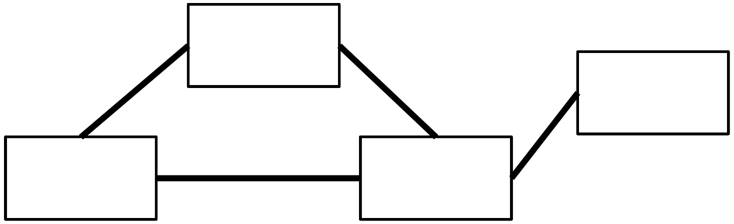
VISUALIZATION
ANALYSIS OUTCOME

- **Domain model**
  - Real-world representation of concepts
  - No software constraints
- **Scenarios**
  - Fully dressed Use Cases
  - Capture requirements
  - Interaction of user and black box

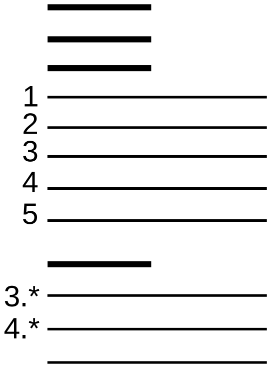


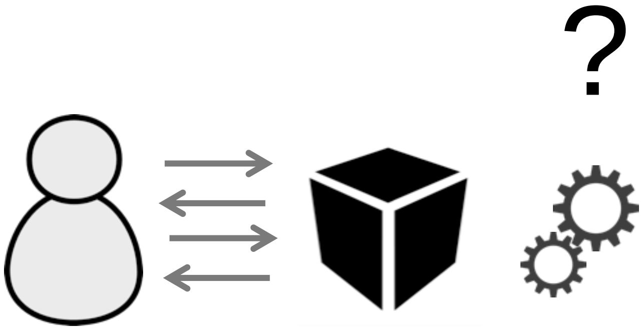
VISUALIZATION
ANALYSIS OUTCOME

- **Domain model**



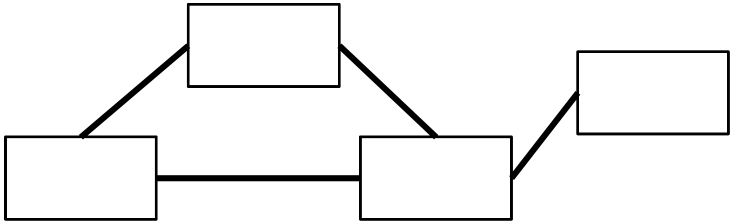
- **Scenarios**





VISUALIZATION
ANALYSIS OUTCOME

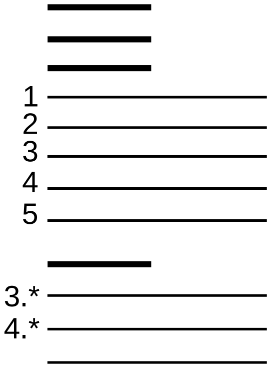
- **Domain model**



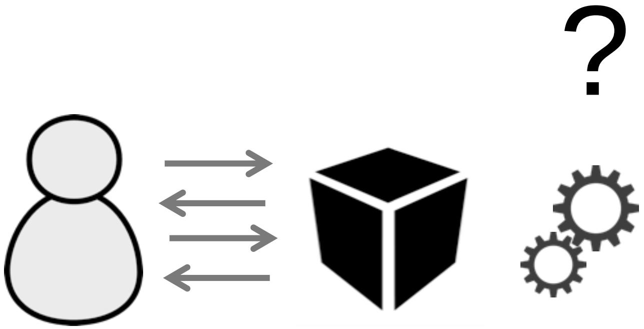
- **Scenarios**

Static structure

Dynamic and Static Information

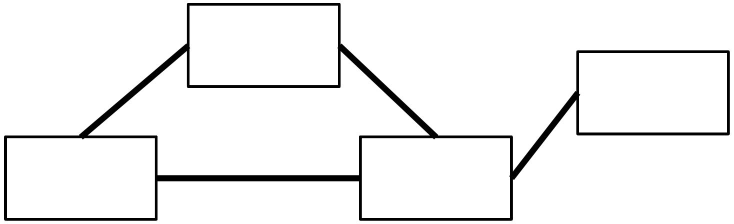


User – system interaction

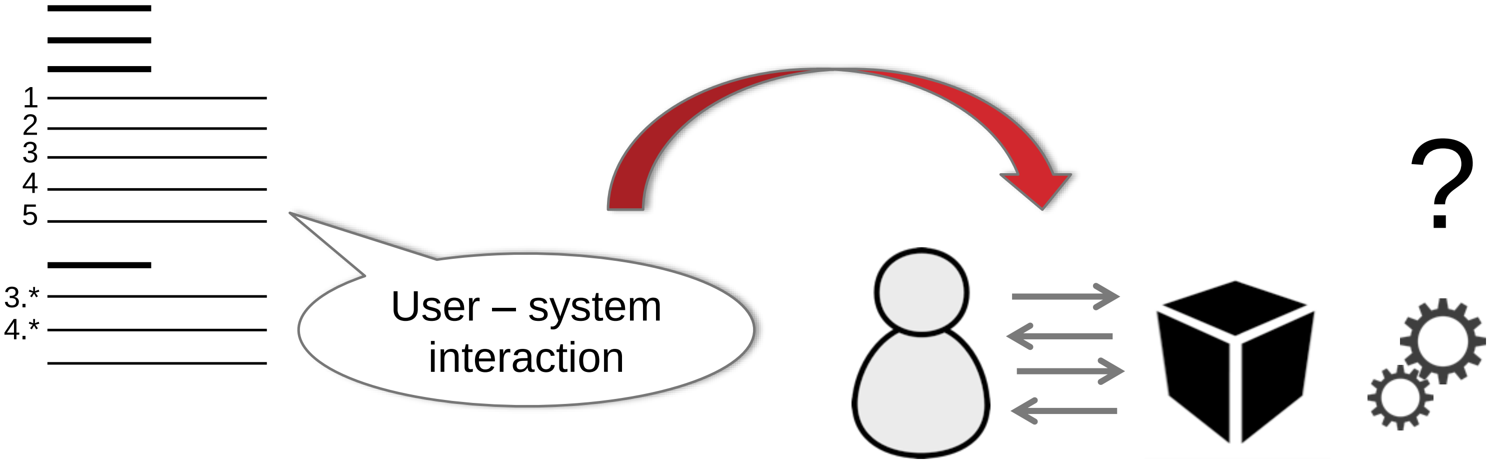


VISUALIZATION
DESIGNING

- Domain model

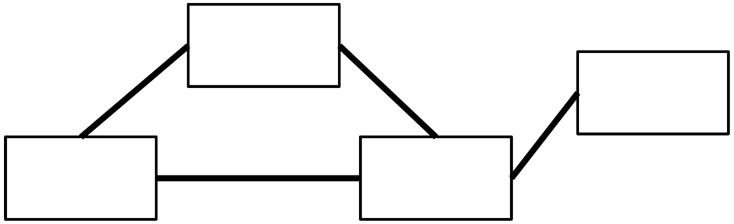


- Scenarios

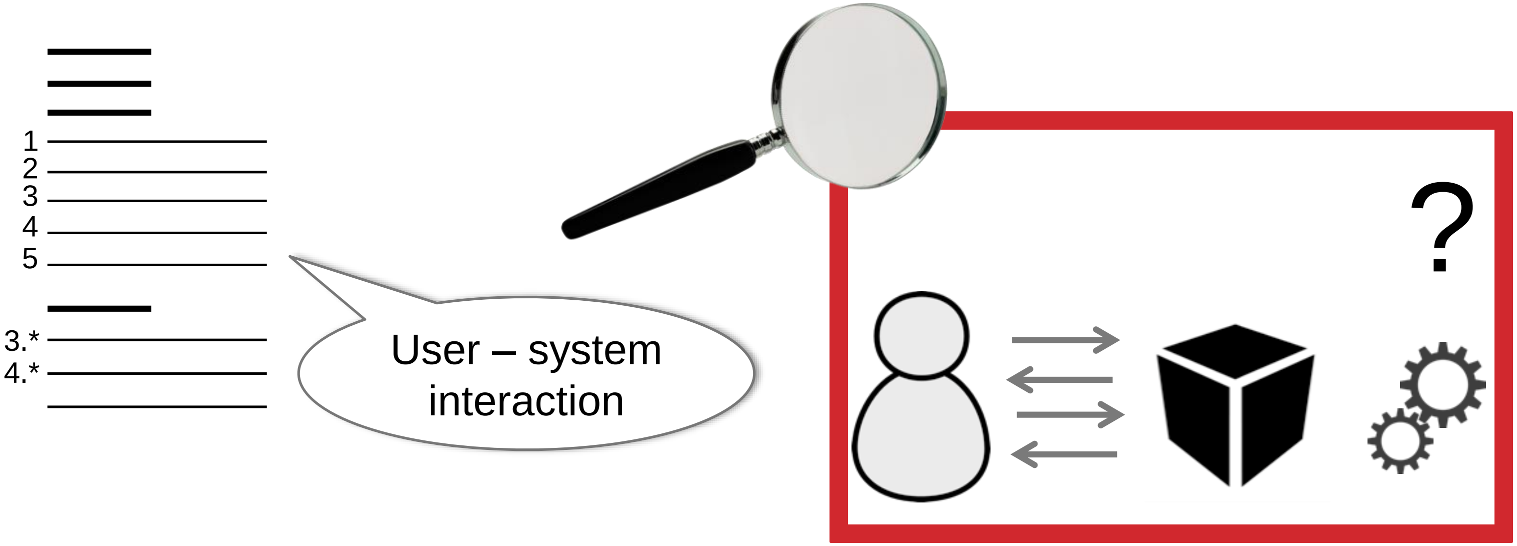


VISUALIZATION
DESIGNING

- **Domain model**



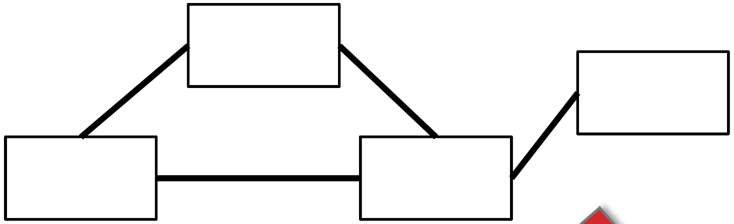
- **Scenarios**



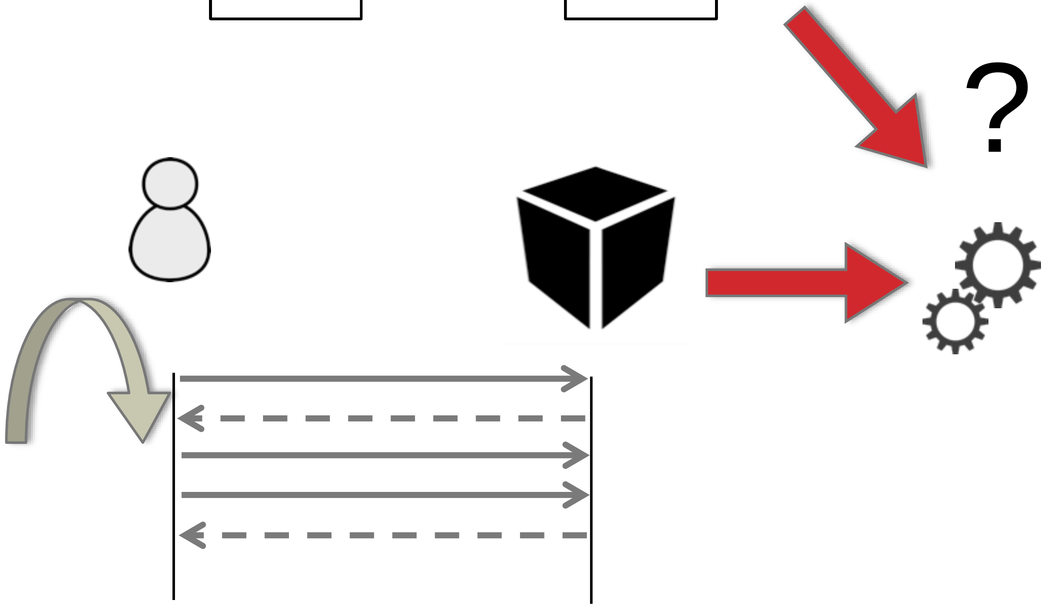
User – system interaction

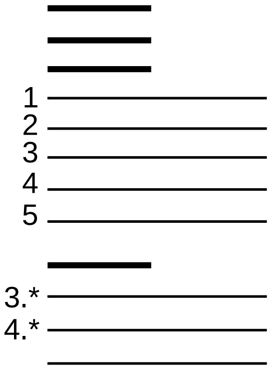
VISUALIZATION
DESIGNING

- **Domain model**



- **Scenarios**

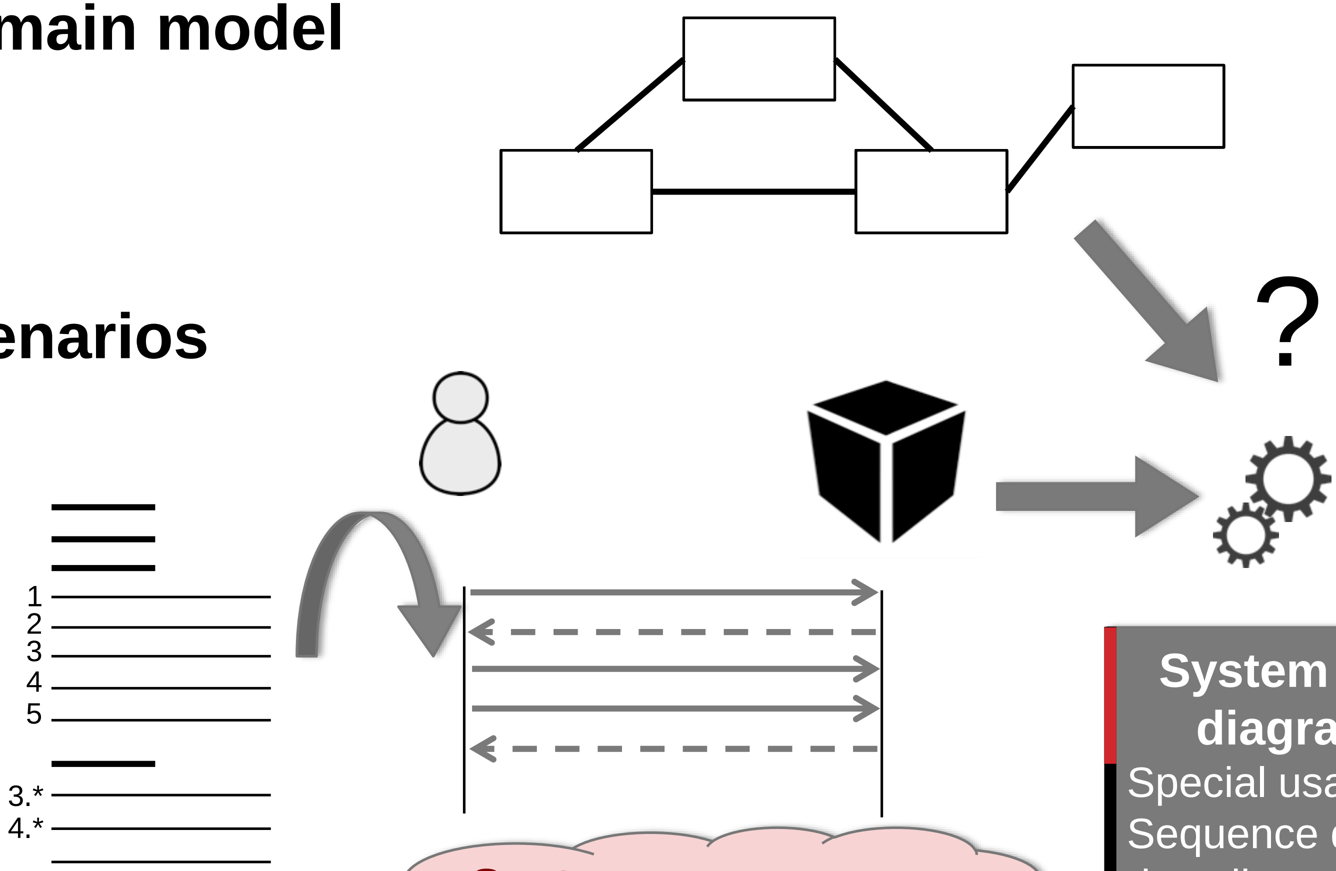




OUTCOME

SYSTEM SEQUENCE DIAGRAM $^{\text{SSD}}$

- **Domain model**
- **Scenarios**



System sequence diagram (SSD)

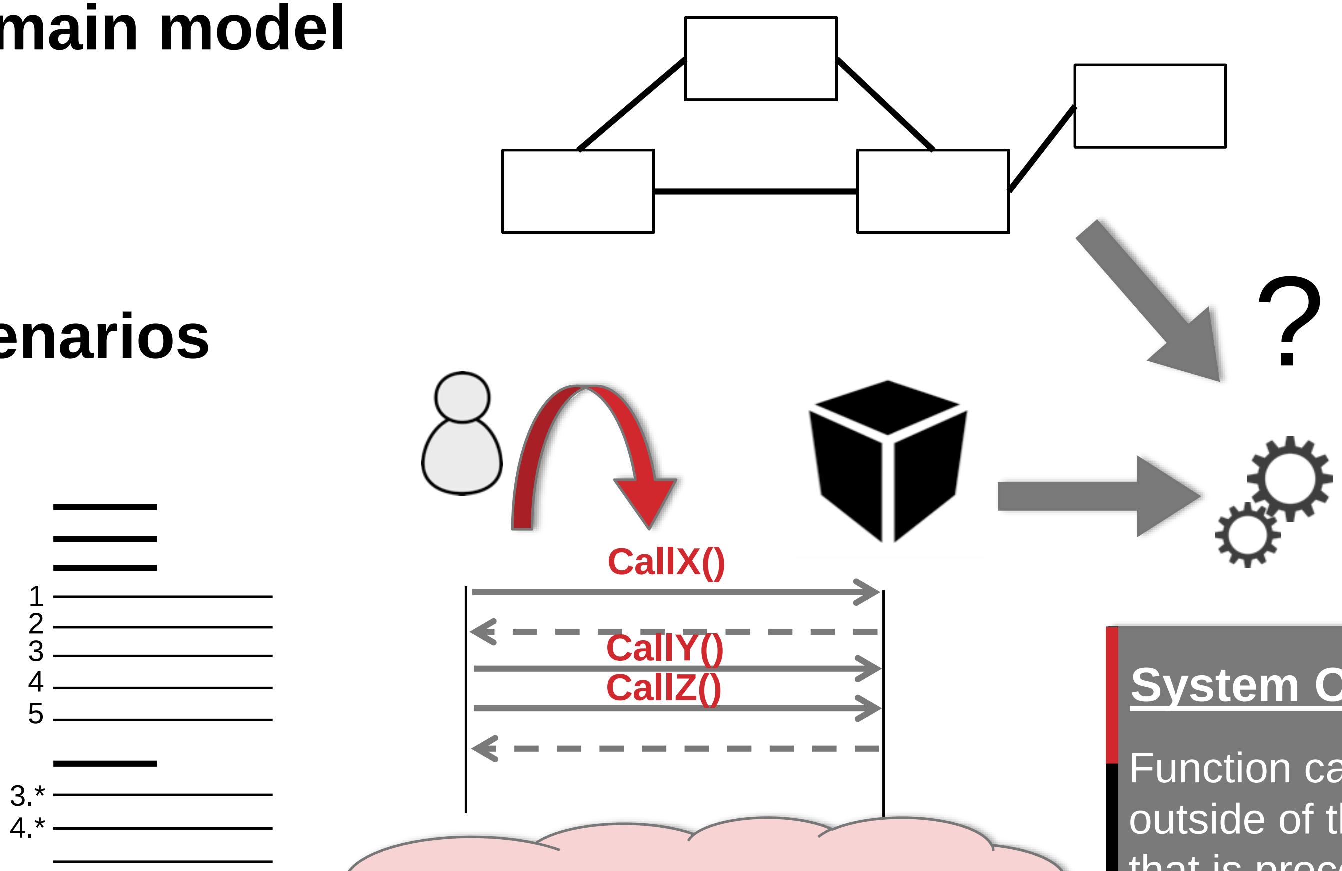
System sequence diagram (SSD)

Special usage of UML Sequence diagram to describe communication with black box and user

OUTCOME

SYSTEM OPERATION

- **Domain model**
- **Scenarios**



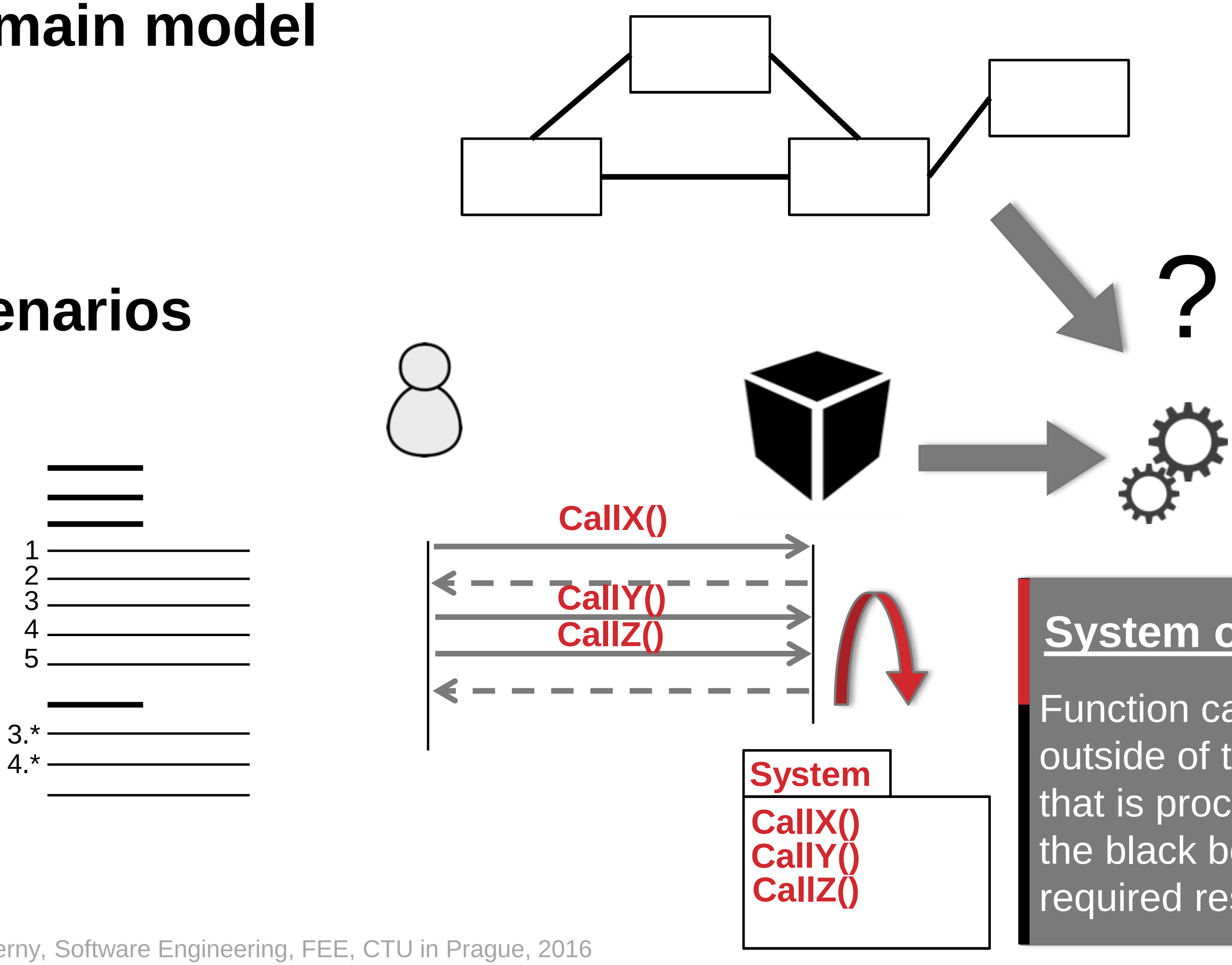
SSD helps to deduce System Operations

System Operation

Function called from outside of the system that is processed by the black box to get required results

OUTCOME
SYSTEM OPERATION

- **Domain model**
- **Scenarios**



System operation

Function called from outside of the system that is processed by the black box to get required results

EXAMPLE:
USE CASE SCENARIO TO SSD

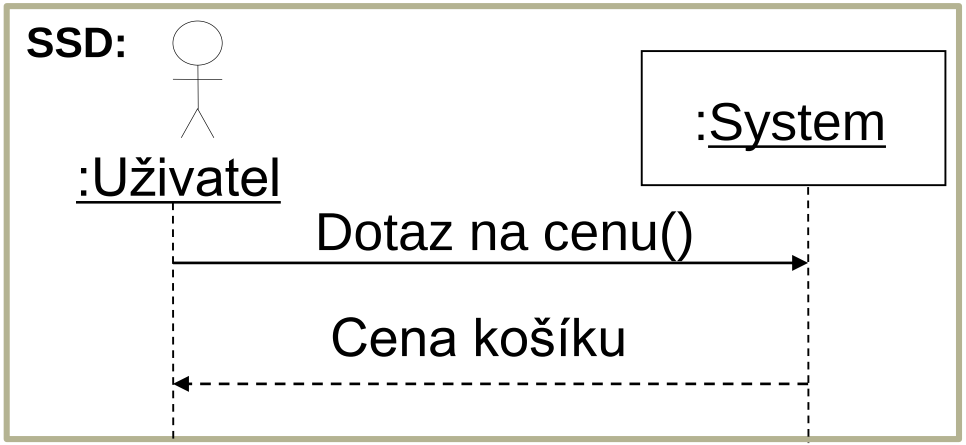
Use case:

1. Uživatel odešle dotaz na cenu košíku
2. Systém zpracuje košík, dohledá ceny a sečte kvantitu vynásobenou cenou a prezentuje uživateli cenu košíku

EXAMPLE:
USE CASE SCENARIO TO SSD

Use case:
1. Uživatel odešle dotaz na cenu košíku
2. Systém zpracuje košík, dohledá ceny a sečte kvantitu vynásobenou cenou a prezentuje uživateli cenu košíku

SSD:

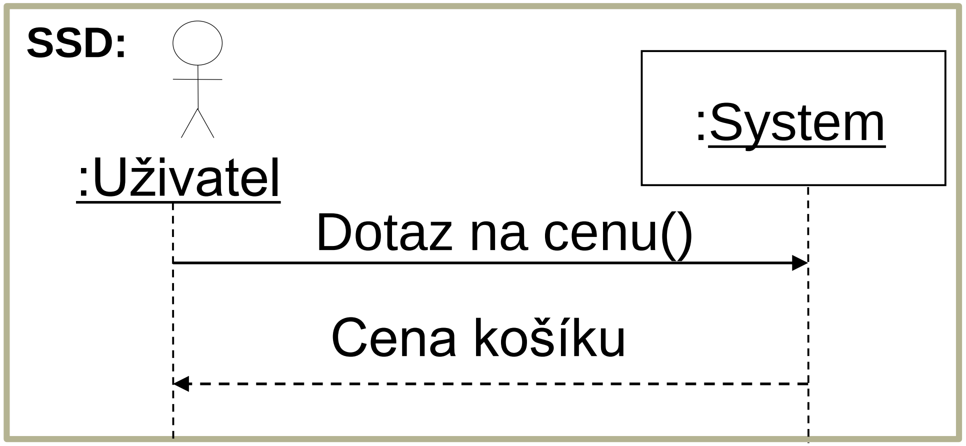


EXAMPLE:
USE CASE SCENARIO TO SSD

Use case:

1. Uživatel odešle dotaz na cenu košíku
2. Systém zpracuje košík, dohledá ceny a sečte kvantitu vynásobenou cenou a prezentuje uživateli cenu košíku

SSD:

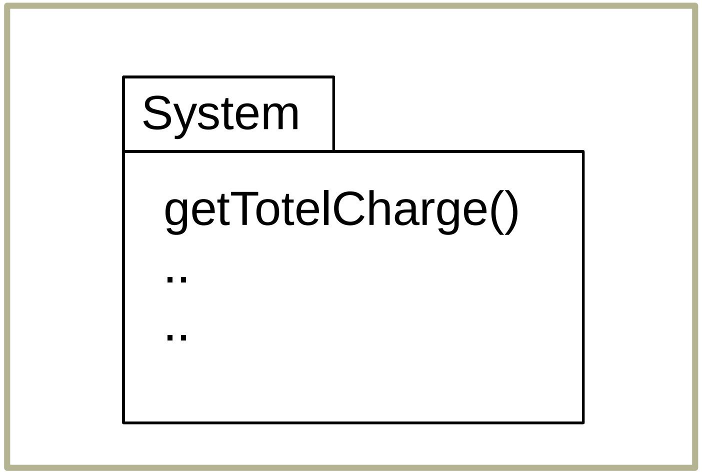


:Uživatel

:System

Dotaz na cenu()

Cena košíku



System

getTotelCharge()

..

..

EXAMPLE II:
USE CASE SCENARIO TO SSD

Simple cash-only Process Sale Scenario

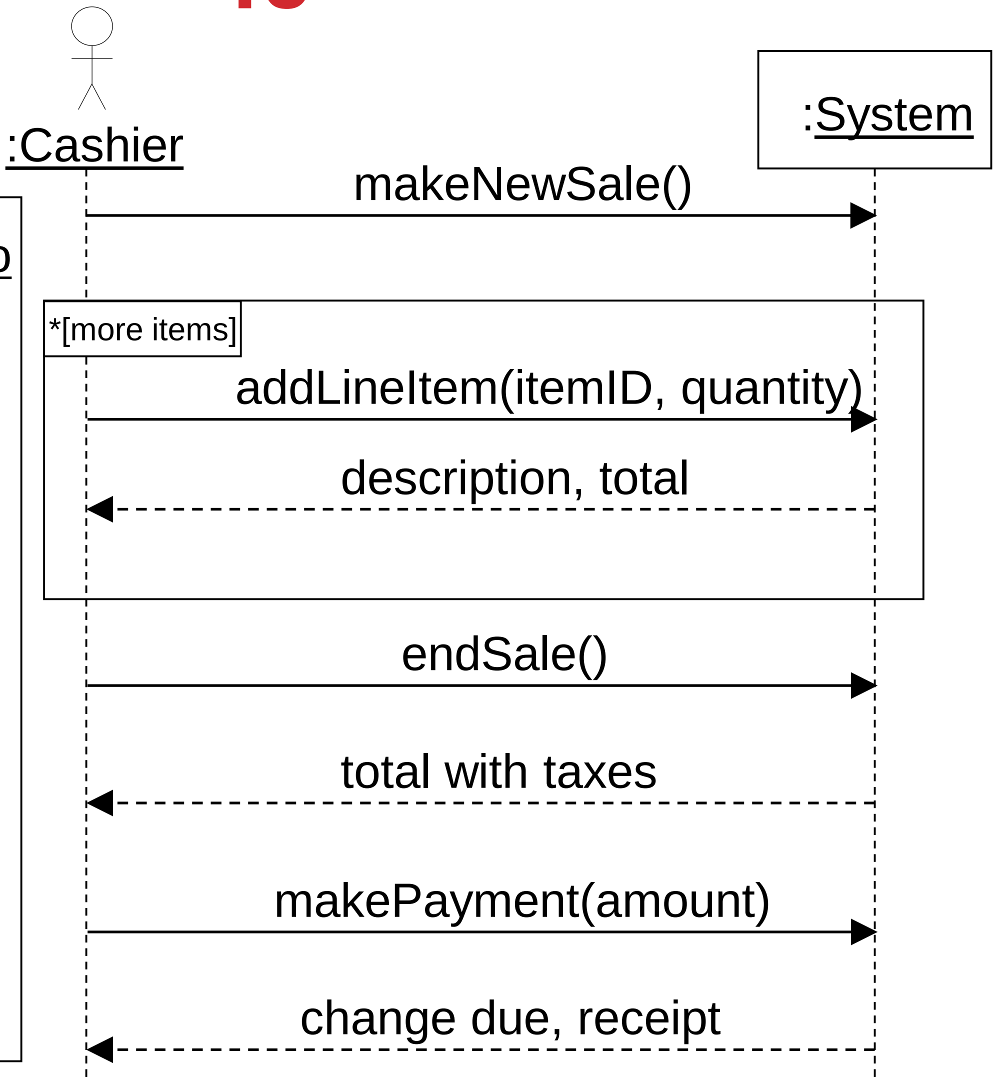
1. Customer arrives at a POS checkout with goods to purchase.
2. Cashier starts a new sale.
3. Cashier enters item identifier.
4. System records sale line item, and presents item description, price and running total.

   cashier repeats steps 3-4 until indicates done.
5. System presents total with taxes calculated.
6. Get payment

EXAMPLE II:
USE CASE SCENARIO TO SSD

Simple cash-only Process Sale Scenario

1. Customer arrives at a POS checkout with goods to purchase.
2. Cashier starts a new sale.
3. Cashier enters item identifier.
4. System records sale line item, and presents item description, price and running total.
   cashier repeats steps 3-4 until indicates done.
5. System presents total with taxes calculated.
6. Get payment

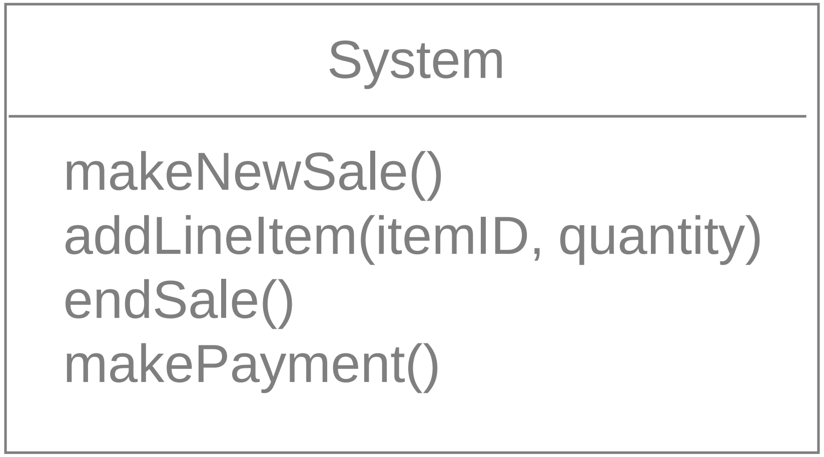


EXAMPLE II:
SSD TO SYSTEM OPERATION

The set of all required system operations is determined by identifying the system events.

- makeNewSale()
- addLineItem(itemID, quantity)
- endSale()
- makePayment(amount)

In the UML the system as a whole can be represented as a class.



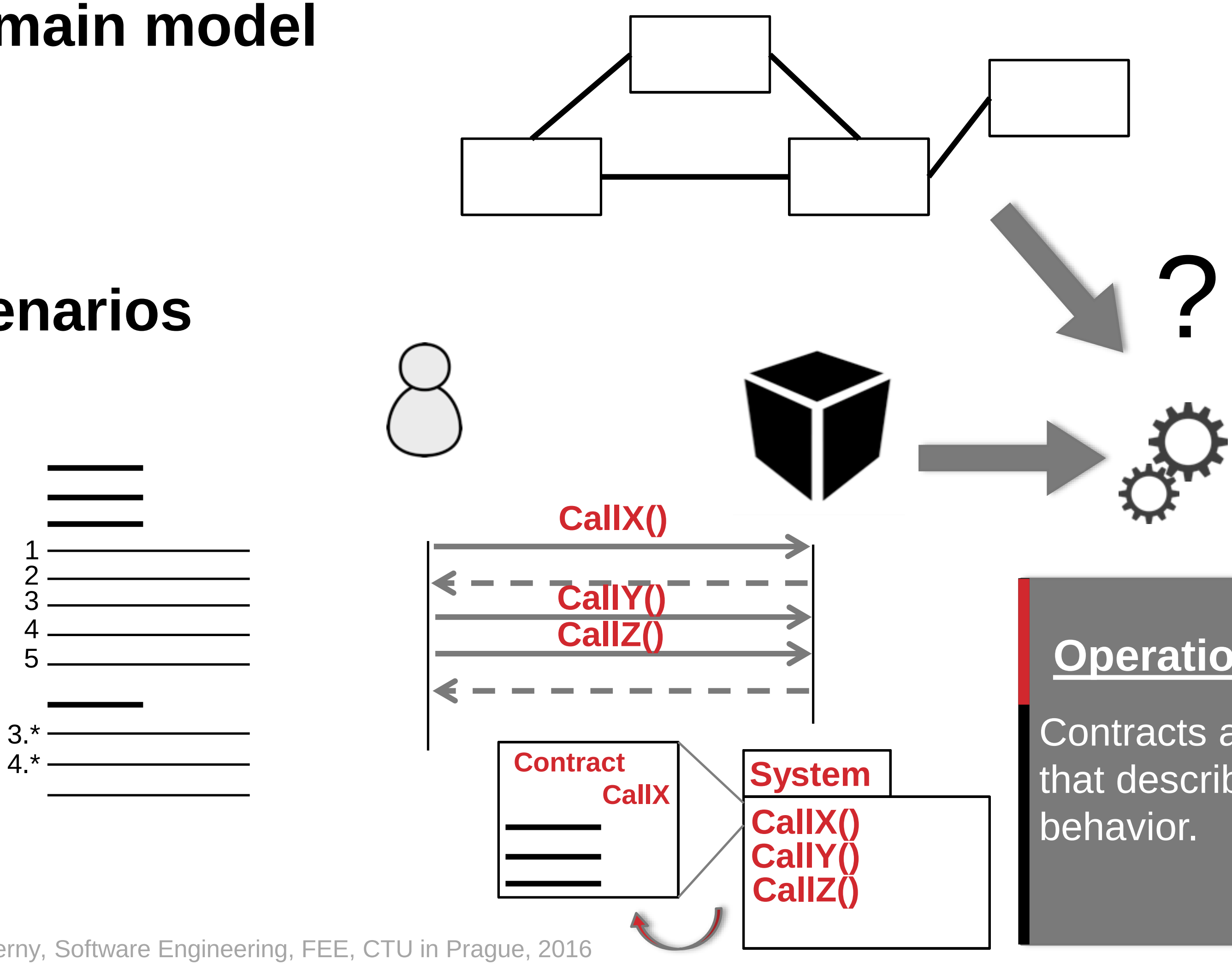
System

makeNewSale()
addLineItem(itemID, quantity)
endSale()
makePayment()

TOWARDS DESIGN

OPERATION CONTRACTS

- **Domain model**
- **Scenarios**



Operation Contracts

Contracts are documents that describe system behavior.

EXAMPLE II:
OPERATION CONRACTS

Contracts <u>may</u> be defined for <u>system operations</u>

- Operations that the system (as a black box) offers in its public interface to handle incoming system events.

The entire set of system operations across all use cases, defines the <u>public system interface.</u>

System

makeNewSale()
addLineItem(itemID, quantity)
endSale()
makePayment()

EXAMPLE II:
OPERATION CONRACT: addLineItem

Contract CO2: addLineItem

Operation: **addLineItem (itemID: ItemID, quantity: integer)**

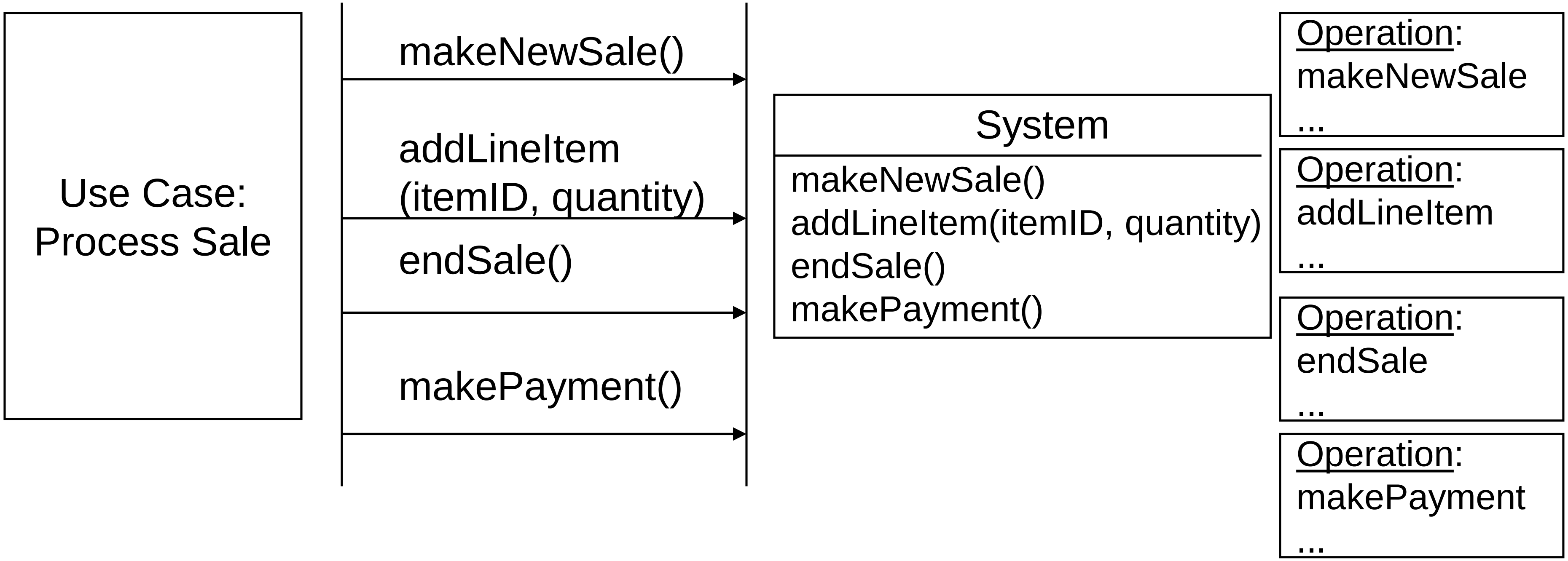
Cross References: **Use Cases: Process Sale.**

Pre-conditions: **There is a sale underway.**

Post-conditions:

- A **SalesLineItem** instance *sli* was created. (instance creation)
- *sli* was associated with the **Sale**. (association formed)
- *sli.quantity* was set to quantity. (attribute modification)
- *sli* was associated with a **ProductSpecification**, based on *itemID* match (association formed)

EXAMPLE II: OVERVIEW



Use Case:
Process Sale

makeNewSale()

addLineItem
(itemID, quantity)

endSale()

makePayment()

System

makeNewSale()
addLineItem(itemID, quantity)
endSale()
makePayment()

Operation: makeNewSale
...

Operation: addLineItem
...

Operation: endSale
...

Operation: makePayment
...

Use Case

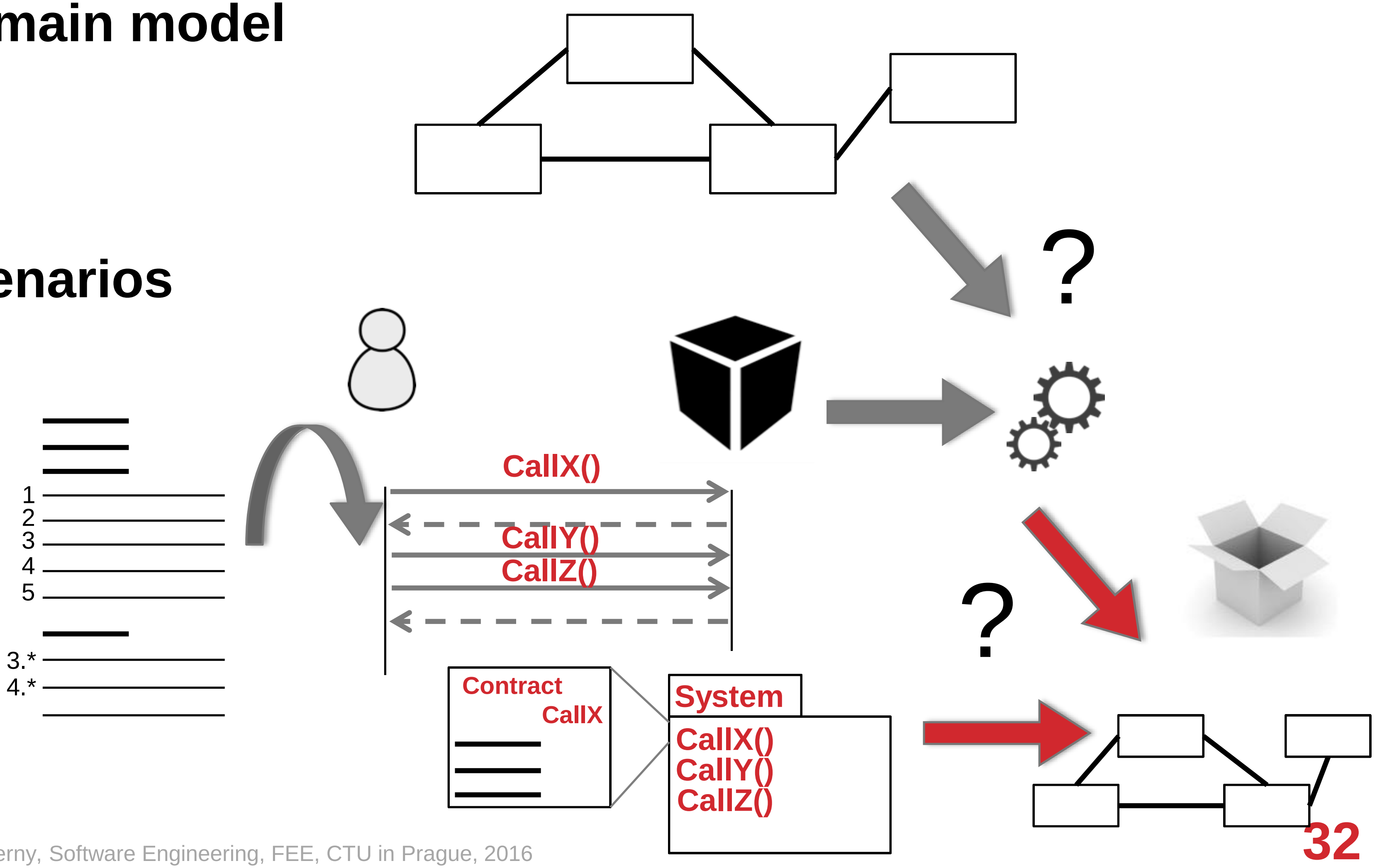
System Sequence Diagram

System Operations

Contracts

TOWARDS DESIGN MODEL
ASSIGN SYSTEM OPERATIONS TO OBJECTS

- **Domain model**
- **Scenarios**



?

1
2
3
4
5
3.*
4.*

CallX()
CallY()
CallZ()

Contract
CallX

System
CallX()
CallY()
CallZ()

?

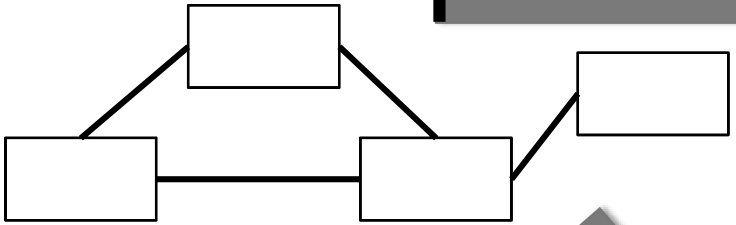
TOWARDS DESIGN
GRASP

GRASP

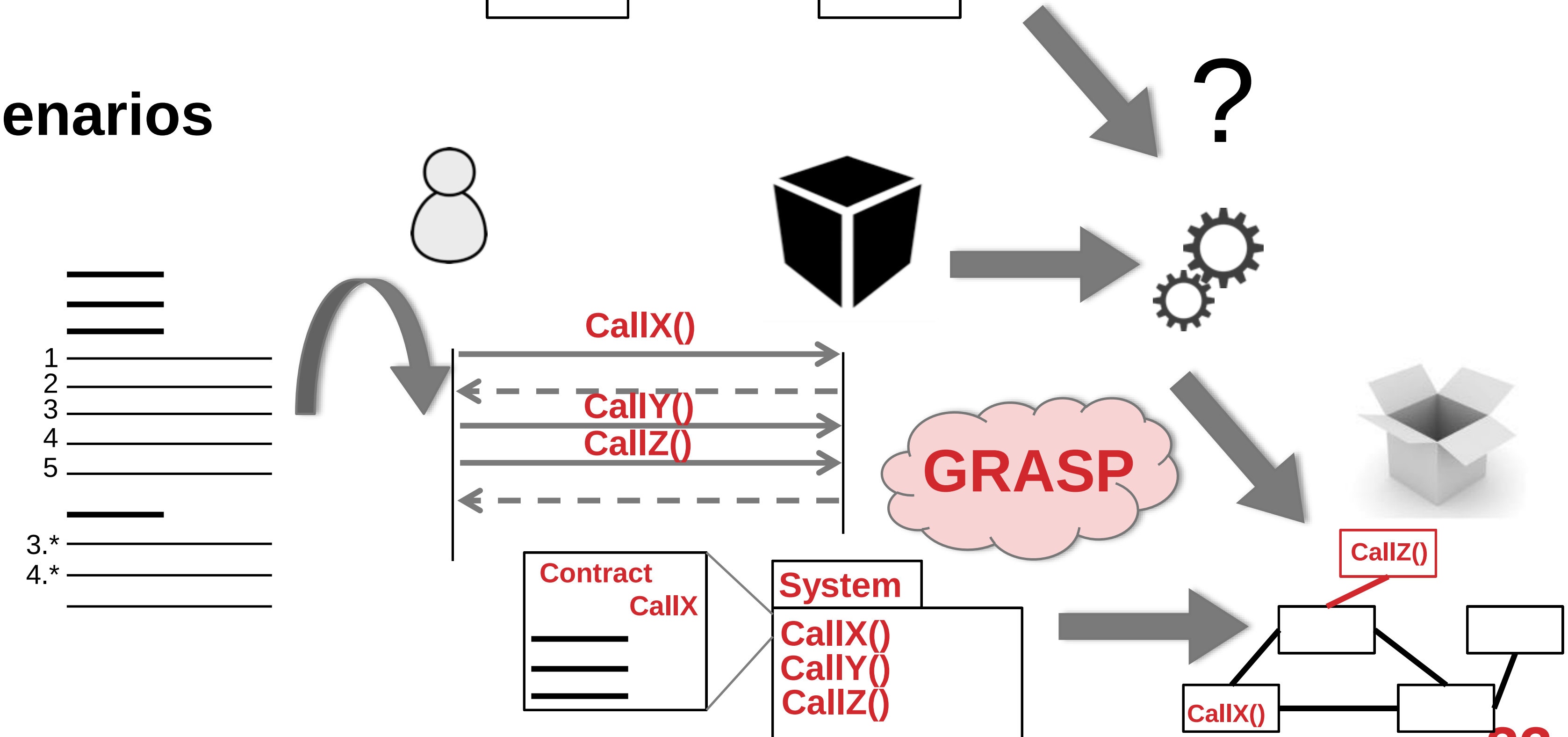
Genenal Responsibility Assignment Software Patterns.

Assign responsibilities to objects (Which operation to which object)

- **Domain model**



- **Scenarios**



GRASP

GRASP

Genenal Responsibility Assignment Software Patterns.
Assign responsibilities to objects
(Which operation to which object)

- Patterns that help us to choose object into which we assign a particular responsibility
- **Name / Problem / Solution**

1. **Information Expert**
2. **Creator**
3. **Low coupling**
4. **High cohesion**
5. **Controller**
6. **Polymorphism**
7. **Pure fabrication**
8. **Indirection**
9. **Do not talk to strangers**

GRASP

1. Information Expert

- Most basic principle to assign responsibility

2. Creator

- Who should create new instances

3. Low coupling

- Support low dependency, low change impact, increase reuse

4. High cohesion

- Keep complexity manageable, keep clarity

5. Controller

- Handle system events

6. Polymorphism

- Variation of behavior through polymorphic methods

7. Pure fabrication

- Make artificial object to help (3) a (4)

8. Indirection

- Get information through a delegation to associated objects

9. Do not talk to strangers

- Talk to associated objects only, avoid indirect objects

GRASP

More about grasps next week

OPENING BLACK BOX

Multiple approaches

Object-oriented design

Framework

Procedural decomposition

Model-driven development

Component-based development

Design pattern

Architectural styles

Objects

Libraries

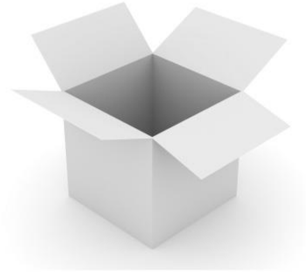
Packages

Layers

Components

Constructs

Artifacts



OBJECT-ORIENTED DESIGN OOD

- **Object-oriented constructs**
  - Generalization, polymorphism, information hiding
  - Delegation, functional decomposition
- **Component design**
- **Layers**
- **Subsystems**
- **Best practice**
  - Design patterns (Solution to a repeatable problems)

QUALITY ATTRIBUTES

- **Runtime**
  - Performance
  - Security
  - Availability
  - Reliability
  - Fault-tolerance
  - **Functionality**
  - Usability
  - Availability
- **Static**
  - Modifiability
  - Readability
  - Integrability
  - Reuse
  - Testability

We cannot have all!!!

*See more at http://en.wikipedia.org/wiki/List_of_system_quality_attributes

ENTERPRISE APPLICATION (EA)

- **Enterprise Application (EA)**
  - An *enterprise application* is the term used to describe applications -- or software -- that a business would use to assist the organization in solving enterprise problems.
  - When the word "enterprise” is combined with "application," it usually refers to a software platform that is too large and too complex for individual or small business use.

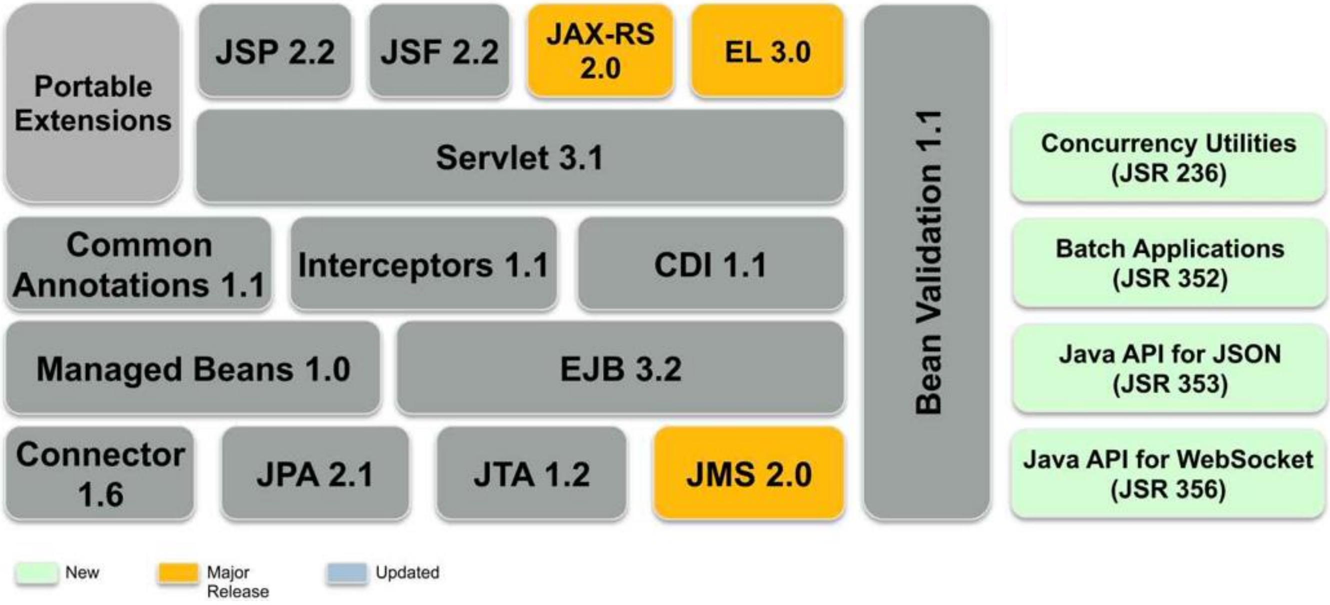
  - Something big that handles a lot of data and business rules
  - Sometimes use term Enterprise System
    - http://en.wikipedia.org/wiki/Enterprise_software

ENTERPRISE APPLICATION (EA)

- **Enterprise Application (EA) Examples**
  - Information system of school, hospital
  - Bank transactions
  - Accounting system
  - Flight booking
  - E-commerce
- **Components**
  - Database
  - Content management
  - Web-services
  - Lot of User Interface to collect and manipulate data
  - Business rules

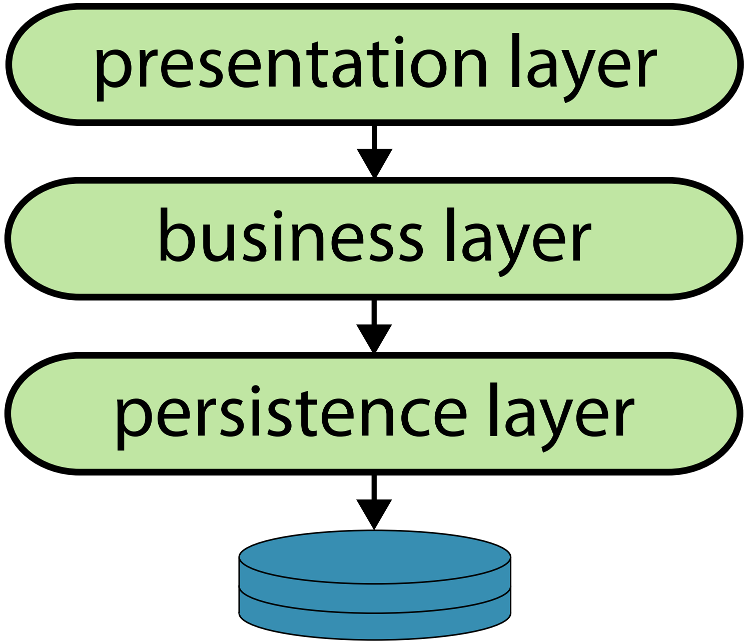
ENTERPRISE APPLICATION ARCHITECTURE (EAA) FRAMEWORK

- **Usually the responsibilities divide to layers**
- **Example: Java EE Framework to build EA with EAA**



ENTERPRISE APPLICATION ARCHITECTURE (EAA)

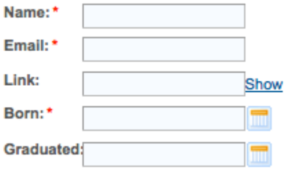
- Usually application responsibilities divide to layers
- EAA with 3 layers



presentation layer

business layer

persistence layer



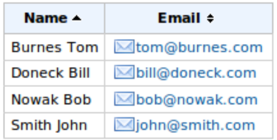
Name:*

Email:*

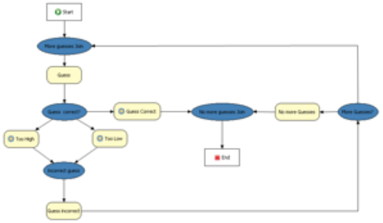
Link: Show

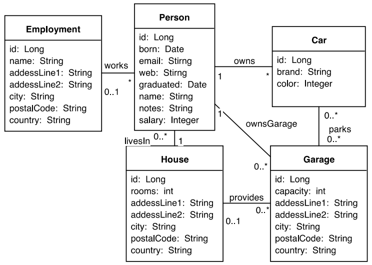
Born:*

Graduated:



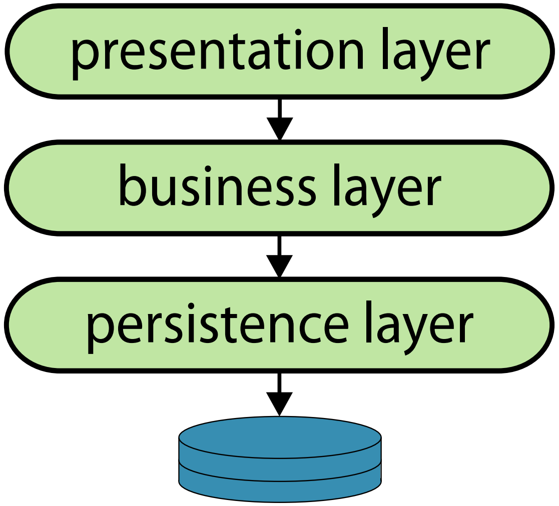
Name ▲	Email ⬍
Burnes Tom	tom@burnes.com
Doneck Bill	bill@doneck.com
Nowak Bob	bob@nowak.com
Smith John	john@smith.com





ENTERPRISE APPLICATION ARCHITECTURE (EAA)

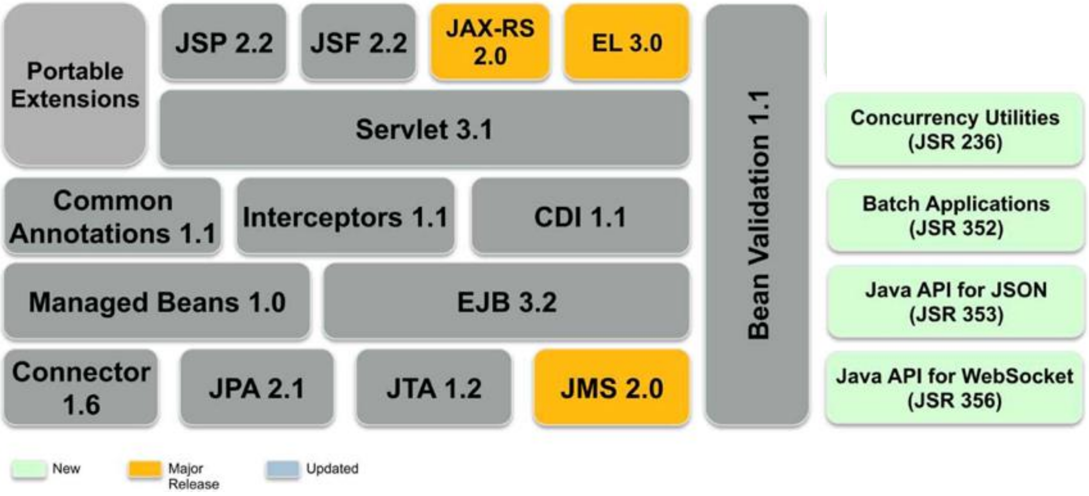
- **EAA in the context of EAA framework**



presentation layer

business layer

persistence layer



Portable Extensions

JSP 2.2

JSF 2.2

JAX-RS 2.0

EL 3.0

Servlet 3.1

Common Annotations 1.1

Interceptors 1.1

CDI 1.1

Managed Beans 1.0

EJB 3.2

Connector 1.6

JPA 2.1

JTA 1.2

JMS 2.0

Bean Validation 1.1

Concurrency Utilities (JSR 236)

Batch Applications (JSR 352)

Java API for JSON (JSR 353)

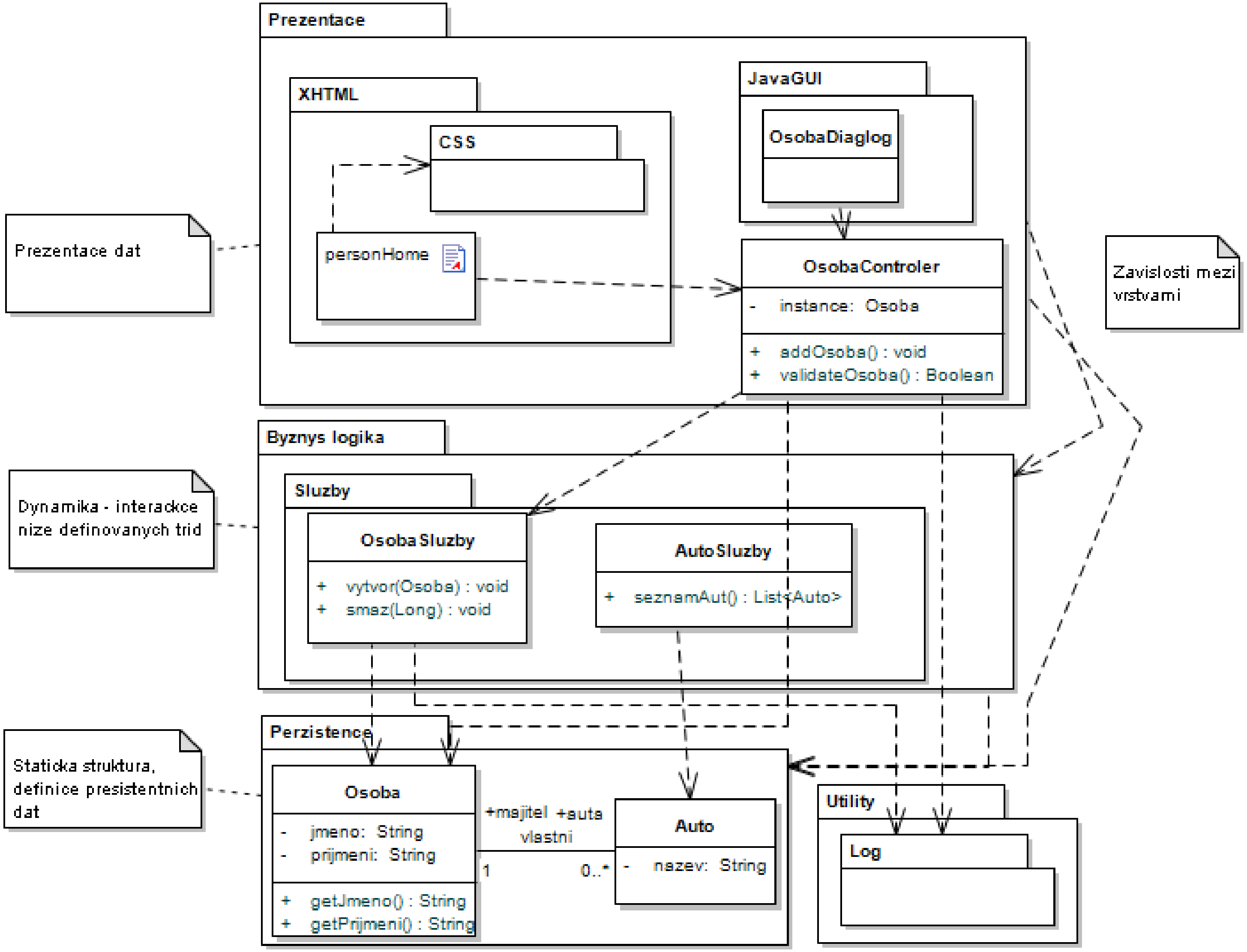
Java API for WebSocket (JSR 356)

🟩 New

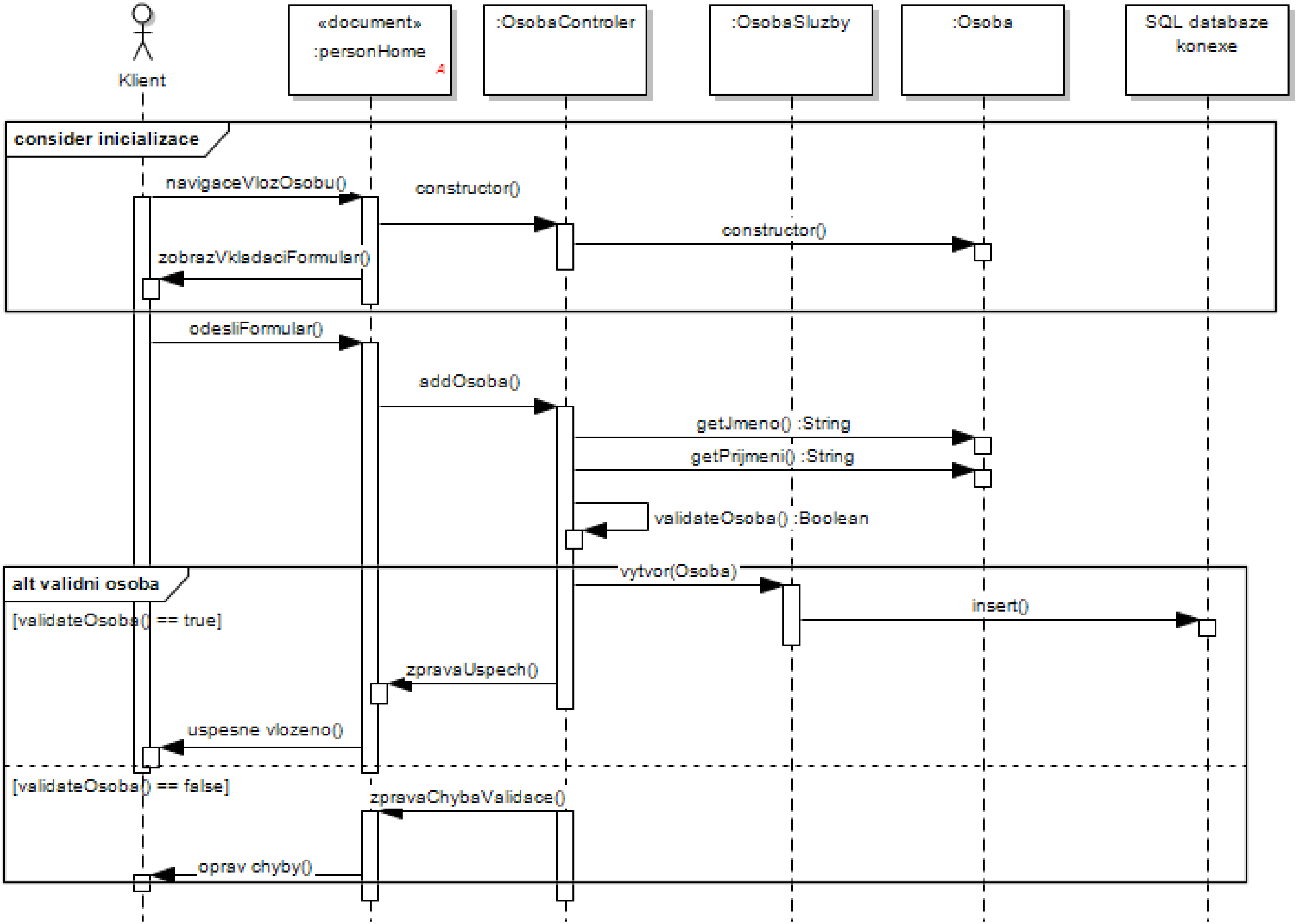
🟧 Major Release

🟦 Updated

ENTERPRISE APPLICATION ARCHITECTURE (EAA)



ENTERPRISE APPLICATION ARCHITECTURE (EAA)



ENTERPRISE APPLICATION ARCHITECTURE (EAA)

- **Layers (Bottom up)**
  - **Persistence**
    - Data model, data persistence, data access
  - **Business logic** **(other names)**
    - Interaction, Business flow, Business constraints/rules
    - Dependency injection, validation, calculation
  - **Presentation**
    - User Interfaces, present the above to user make it easy to process task, automate as much as passible
    - *Do not teach users new things*

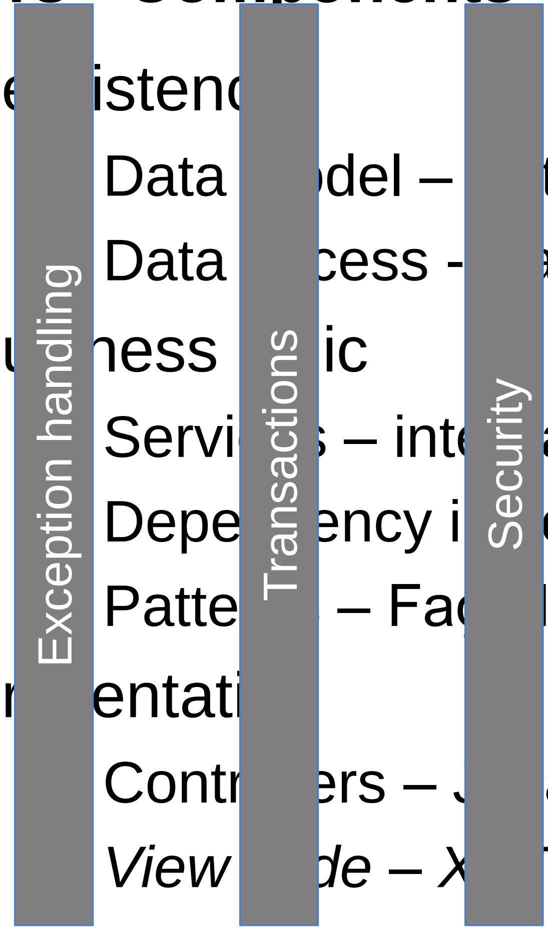
ENTERPRISE APPLICATION ARCHITECTURE (EAA)

- **Layers - Components**
  - **Persistence**
    - Data model – Entity JPA
    - Data Access - Data Access Object (DAO)
  - **Business logic**
    - Services – integrate business flow/constraints/rules - EJB
    - Dependency injection – CDI (which DAO/Service to use)
    - Patterns – Façade, Bridge
    - Transitions
  - **Presentation**
    - Controllers – Java Beans, Session Scopes
    - *View code – XHTML/JSP/JSF*

ENTERPRISE APPLICATION ARCHITECTURE (EAA)

Not everything is that easy!

- **Layers - Components**



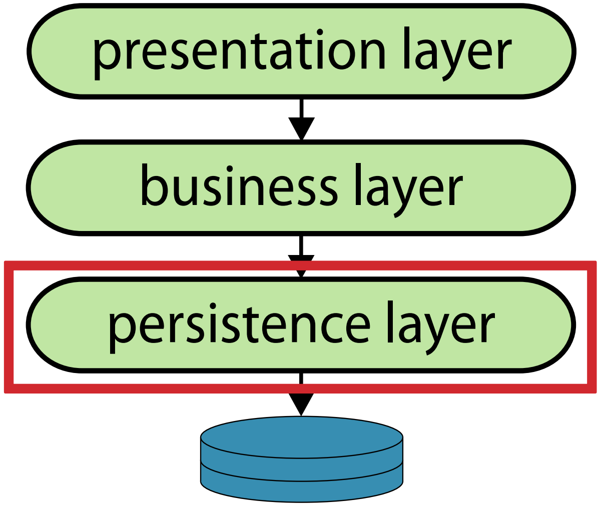
Exception handling

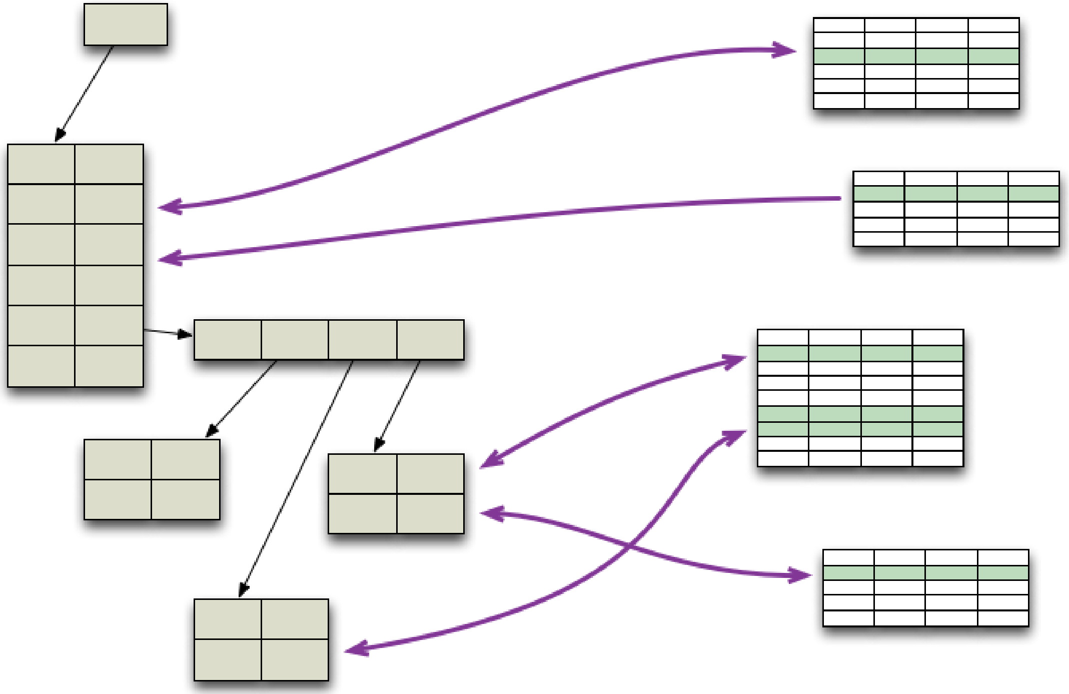
Transactions

Security

  - Pe istenc
    - Data odel – tity JPA
    - Data cess - ata Access Object (DAO)
  - Bu ness ic
    - Servi s – inte ate business flow/constraints/rules - EJB
    - Depe ency i ction – CDI (which DAO/Service to use)
    - Patte s – Fa le, Bridge
  - Pr entati
    - Contr ers – J a Beans, Session Scopes
    - *View de – X TML/JSP/JSF*

PERSISTENCE LAYER





OBJECT-RELATIONAL MAPPING ORM

- **Quotes**
  - Ted Neward, late 2004
    - "Object-Relational mapping is the Vietnam of our industry”
    - I've seen developers struggle for years with the huge mismatch between relational database models and traditional object models. And all the solutions they come up with seem to make the problem worse.

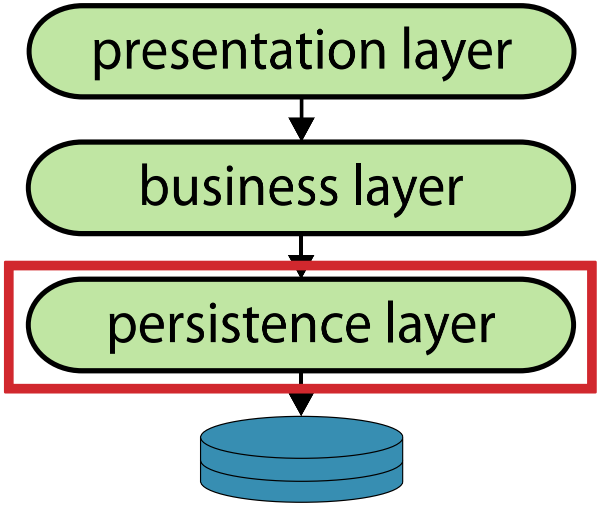
WHAT IS ORM?

- **Persistence**
  - Helping to achieve persistence – app data outlive the process
- **Relational database**
  - Persistence to RDBMS in Object-oriented app
- **Object-relational mismatch**
  - Paradigm mismatch – object model and relational model incompatible
- **Granularity**
  - More objects than tables in DB (Person>Address)
- **Subtypes**
  - Inheritance in DB?
- **Identity notion**
  - Primary key vs. a == b and a.equals(b) options
- **Associations**
  - Unidirectional/Bidirectional in OOP vs Foreign Key in RDBMS
- **Data navigation**
  - Fundamental difference in OOP and DRBMS
  - Object network vs. JOIN ? Efficiency number of SQL queries (lazy load)

OBJECT-RELATIONAL MAPPING ORM

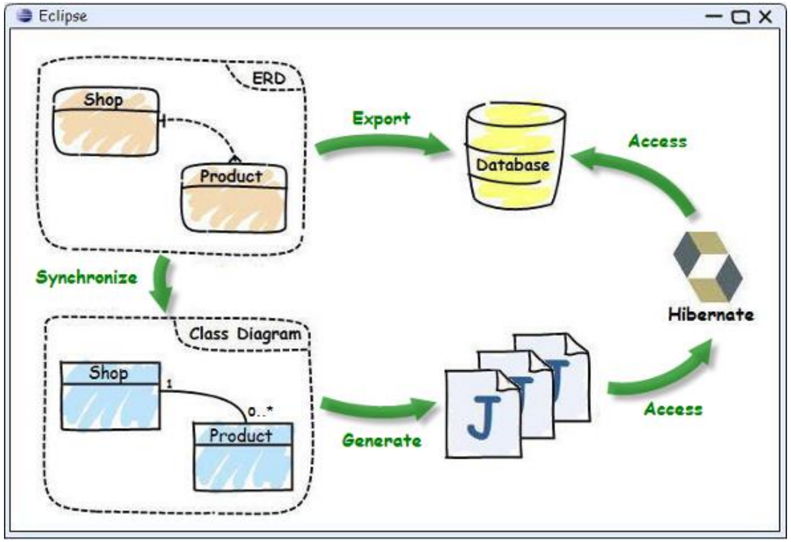
- **Databases are relational**
  - Relations and SQL
  - No inheritance
  - Good performance and centralization
  - ACID, fast to save and retrieve data = <u>GOOD for Persistence</u>
- **Object works**
  - Great for problem decomposition
  - <u>Slow to be used for data persistence</u>
  - Designed to deal with problems not to persist data
  - All nice features and instruments of Object-oriented design
    - Polymorphism, Generics..

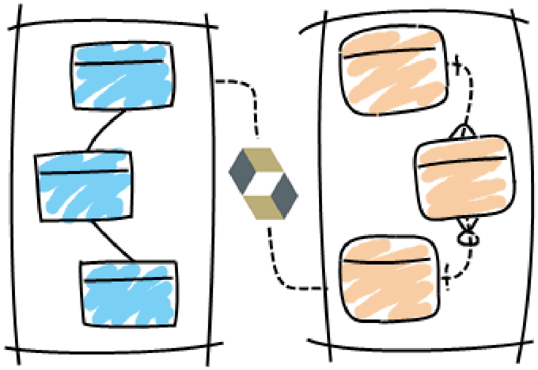
PERSISTENCE LAYER



Object-relational mapping (ORM)

- **Databases are relational RDBMS**
- **Objects are not**
  - Technical incompatibility
- **Solution – mapping to both directions**





OBJECT-RELATIONAL MAPPING (ORM)

- **Create**

```
public void create() {
      Event e = new Event( ″First event!″,new Date())
      Session session = sessionFactory.openSession();
      session.beginTransaction();
      session.save(e);
      session.save(new Event( ″Second event″,new Date()));
      session.getTransaction().commit();
      session.close();
}
```

OBJECT-RELATIONAL MAPPING (ORM)

- **Read**

```
public void read() {
  Session session = sessionFactory.openSession();
  session.beginTransaction();
  List result = session.createQuery("from Event”).list();
  for (Event event : (List<Event>) result) {
   out.print(”Event “+event.getDate()+”):”+event.getTitle());
  }
  session.getTransaction().commit();
  session.close();
 }
```

OBJECT-RELATIONAL MAPPING (ORM)

Note the JavaBeans rules @Component

- **Entity + annotation**

```
@Entity
@Table( name = "EVENTS" )
public class Event {
    ...
 @Id
 @GeneratedValue("increment”)
 @GenericGenerator(
    name="increment",
    strategy = "increment")
 public Long getId() {
    return id;
 }
```

```
public String getTitle() {
    return title;
}

@Temporal(TemporalType.TIMESTAMP)
@Column(name = "EVENT_DATE")
public Date getDate() {
    return date;
}
}
```

OBJECT-RELATIONAL MAPPING (ORM)

- **Read/Write**

```
...
Cat fritz = (Cat) sess.load(Cat.class, generatedId);
..
Cat cat = (Cat) sess.get(Cat.class, id);
if (cat==null) {
    cat = new Cat();
    sess.save(cat, id);
}
..
sess.save(cat);
sess.flush(); //force the SQL INSERT
sess.refresh(cat); //re-read the state (after the trigger executes)

...
```

OBJECT-RELATIONAL MAPPING (ORM)

Note the JavaBeans rules @Component

- **Inheritance**

```
@Entity
@Inheritance(strategy=
    InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(
           name="planetype”,
 discriminatorType=
    DiscriminatorType.STRING
)
@DiscriminatorValue("Plane")
public class Plane { ... }

@Entity
@DiscriminatorValue("A320")
public class A320 extends Plane
{ ... }
```

Associations

```
@Entity
public class FlyingObject

  @ManyToOne
  public PropulsionType
               getPropulsion() {
         return altitude;
  }
  @OneToOne(cascade =
              CascadeType.ALL)
  public Navigation getNavi() {
         return navi;
  }
```

ORM FEATURES

- **Object property mapping**
- **Association mapping (One-one, one-many, many-many)**
  - Lazy loading
- **Inheritance mapping**
  - Single/Table per class/Table per concrete class
- **Generated Keys**
- **Cascades**
- **Locking** – Bob and John modify Person with ID = 1
- **Hibernate Query Language HQL**
- **Mapping to many databases**
- **Cache**
- **Criteria API**

OBJECT-RELATIONAL MAPPING (ORM)

- **Issues**
  - Needs custom equals and hashCode
  - Too much SQL
    - You can make custom/native query

- **More at DOC**
  - http://docs.jboss.org/hibernate/orm/4.2/quickstart/en-US/html/
  - http://docs.jboss.org/hibernate/orm/4.2/devguide/en-US/html/

DESIGN PATTERNS

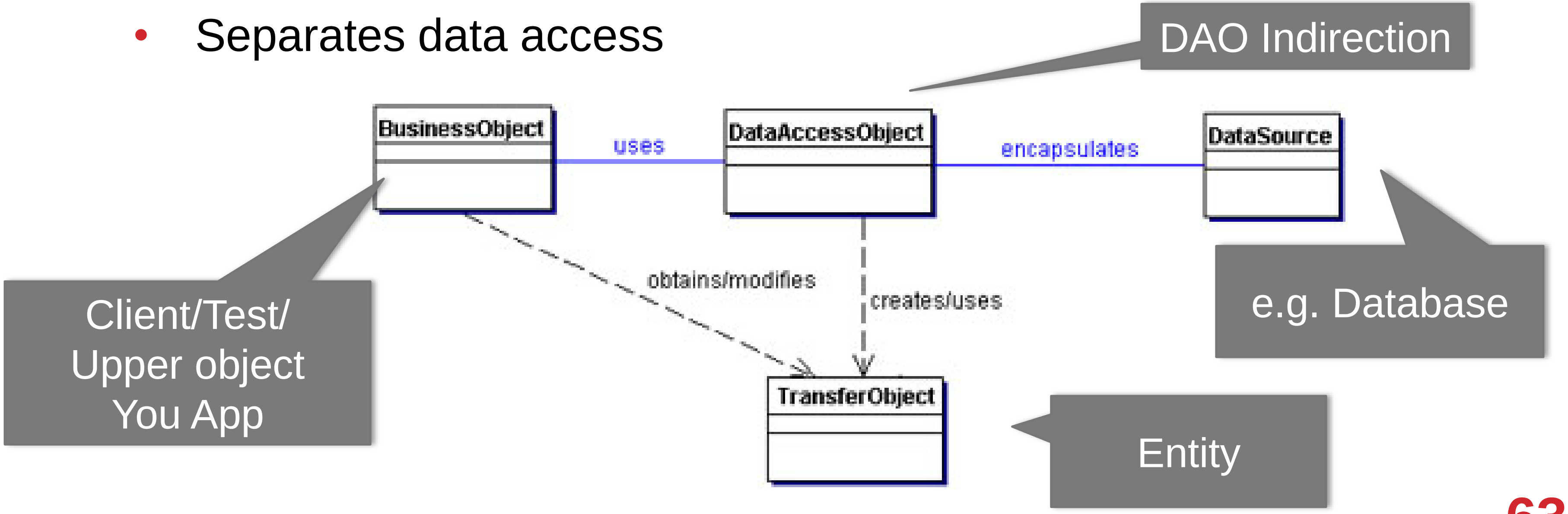
- **Best practice solution to a particular problem**
- **Named problem and solution**
  - Not relevant to a specific contest

- **Good Software Engineer and Developer has good overview of Design Patterns**
  - Programming Bible
    - Erich Gamma
    - Martin Fowler

DATA ACCESS OBJECT (DAO)

DESIGN PATTERN

- Problem:
    - How to access persistent objects
    - Where to persist data
    - Wrap access to Data Source
    - Indirection
- Object that provides interface to persistence mechanism
- Separates data access



DAO Indirection

Client/Test/
Upper object
You App

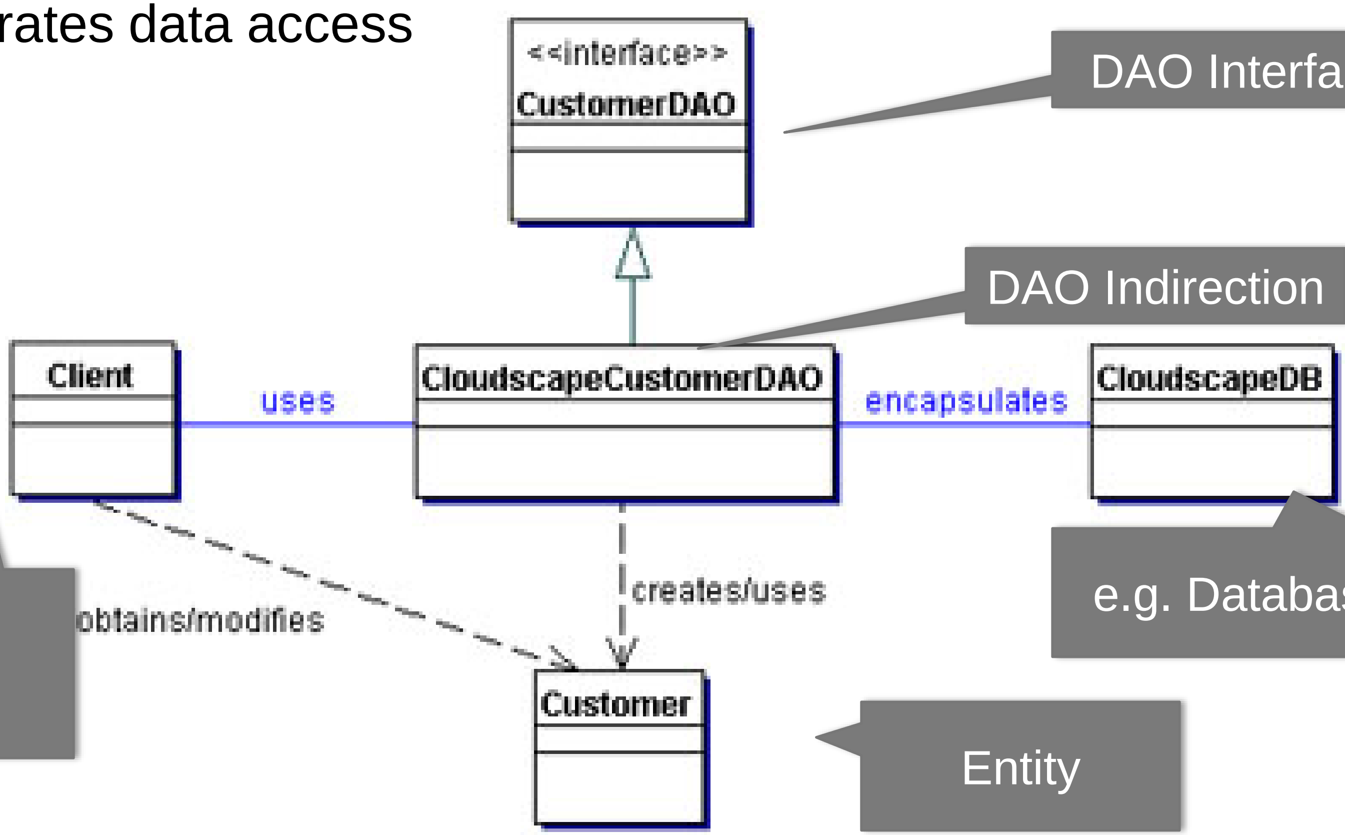
e.g. Database

Entity

DATA ACCESS OBJECT (DAO)

DESIGN PATTERN

- **Design pattern**
  - Object that provides interface to persistence mechanism
  - Separates data access



DAO Interface

DAO Indirection

Client/Test/
Upper object
You App

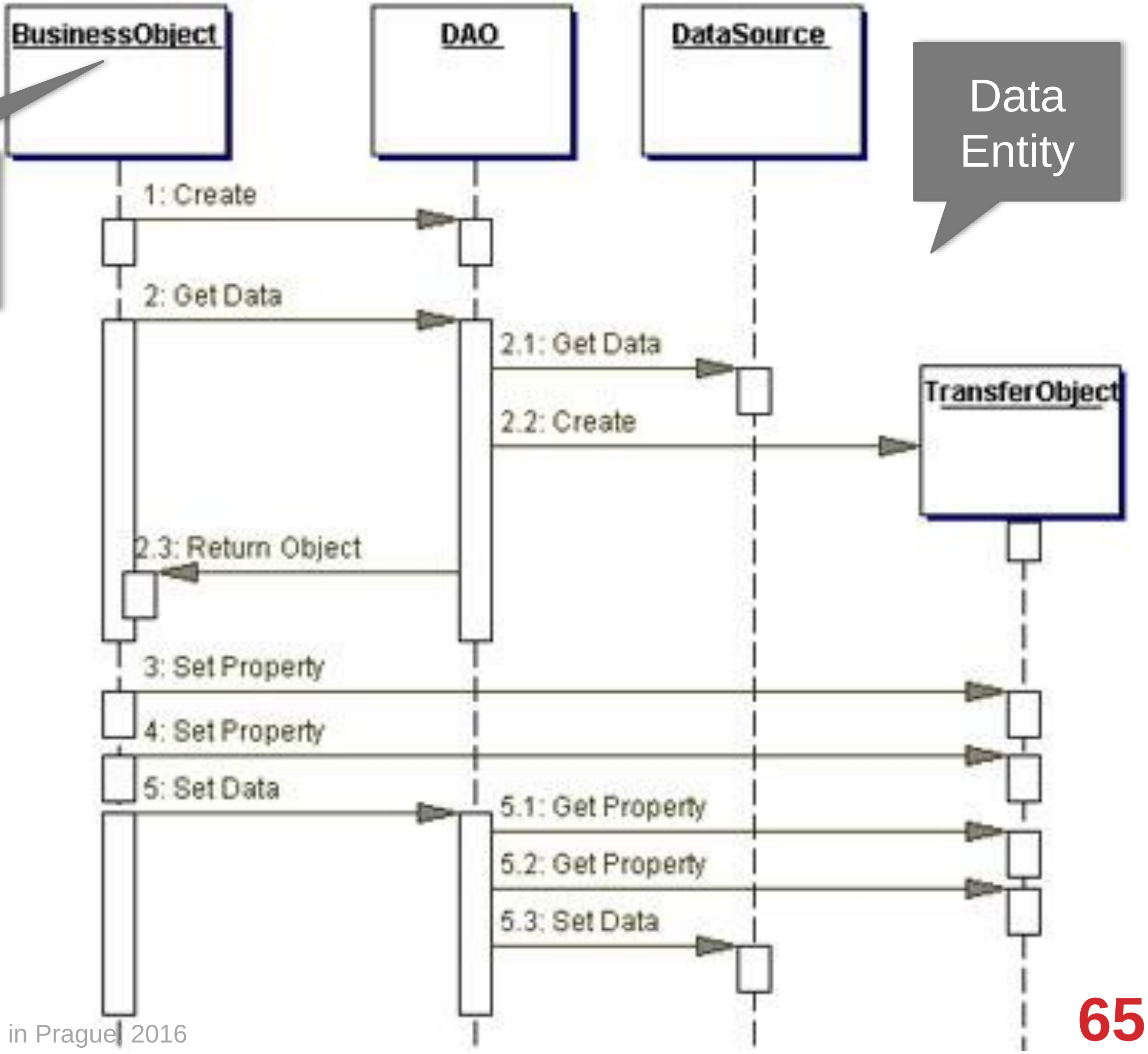
e.g. Database

Entity

DATA ACCESS OBJECT (DAO)

DESIGN PATTERN

- **Interaction**



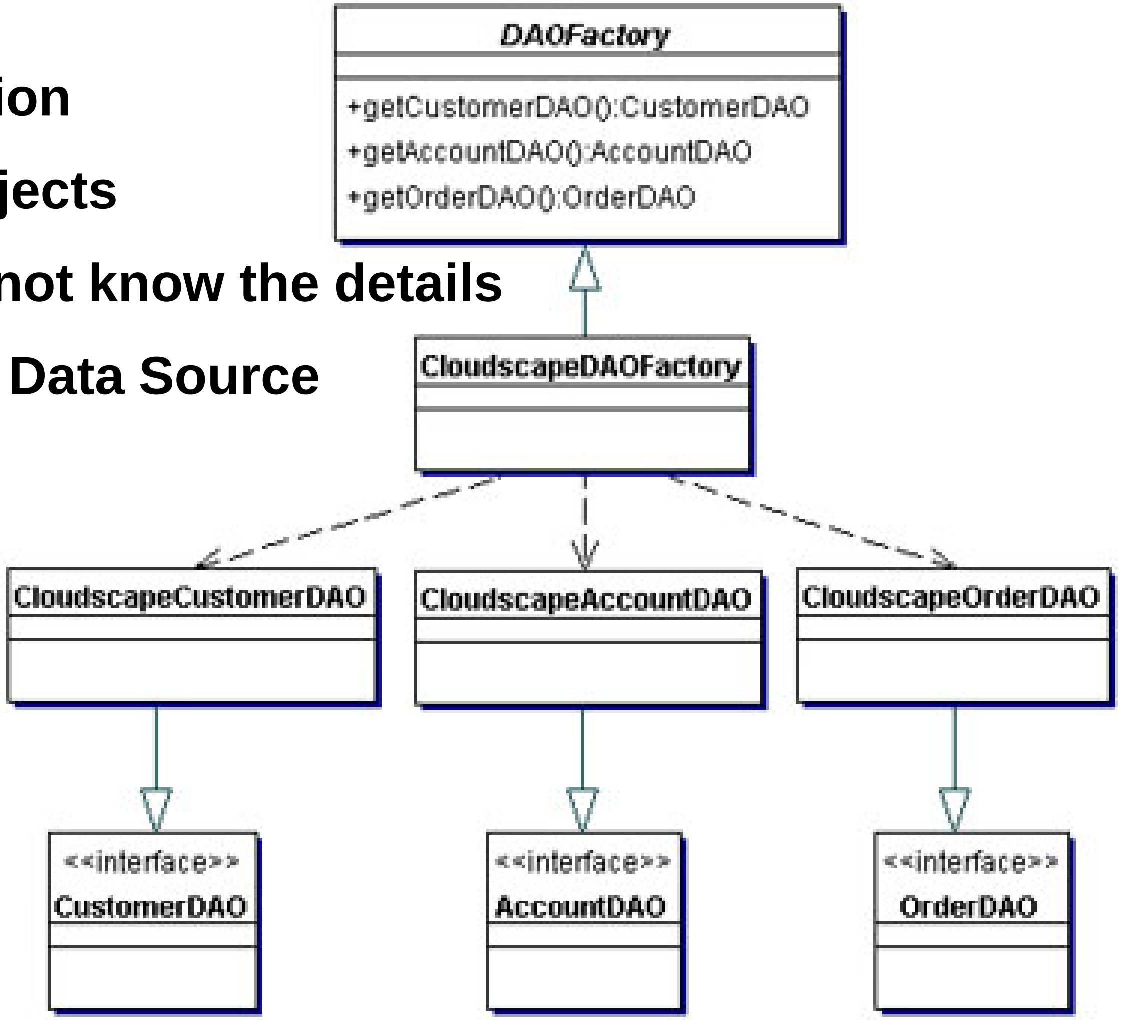
Client/Test/
Upper object

Data
Entity

DATA ACCESS OBJECT (DAO)

DESIGN PATTERN

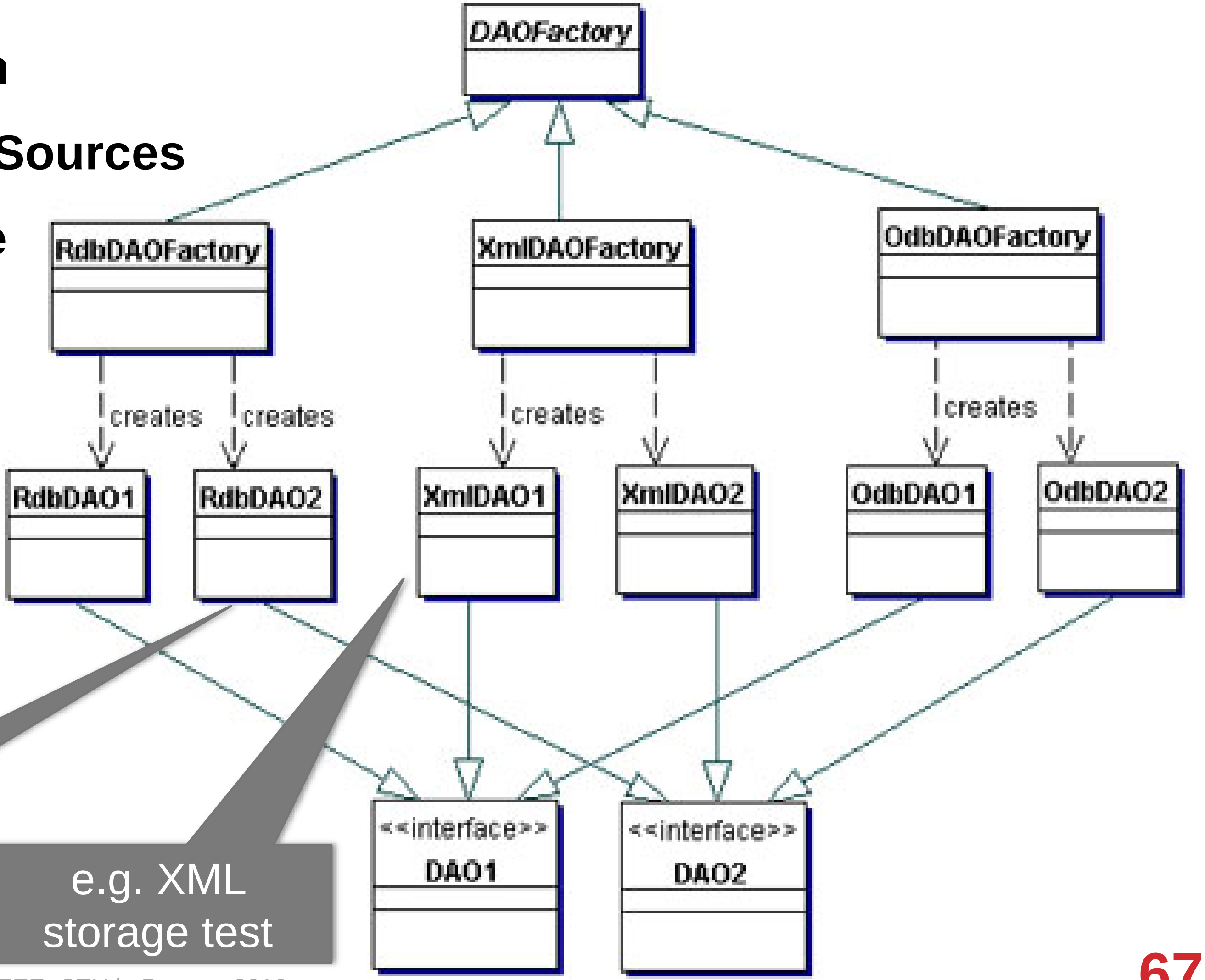
- Usual situation
- Different Objects
- Client does not know the details
- Transparent Data Source



DATA ACCESS OBJECT (DAO)

DESIGN PATTERN

- **Usual situation**
- **Different Data Sources**
- **Client unaware**



e.g. Database test

e.g. Database production

e.g. XML storage test

DATA ACCESS OBJECT (DAO)

DESIGN PATTERN

INTERFACE

```
// Interface that all CustomerDAOs must support
public interface CustomerDAO {
  public int insertCustomer(...);
  public boolean deleteCustomer(...);
  public Customer findCustomer(...);
  public boolean updateCustomer(...);
  public RowSet selectCustomersRS(...);
  public Collection selectCustomersTO(...);
  ...
}
```

DATA ACCESS OBJECT (DAO)

DESIGN PATTERN IMPLEMENTATION

```
public class CloudscapeCustomerDAO
         implements CustomerDAO {

  public int insertCustomer(...) {
    // Implement insert customer here.
    // Return created customer number
    // or a -1 on error
  }
  public boolean deleteCustomer(...) {
    // Implement delete customer here
    // True -success, false -failure
  }

   public Customer findCustomer(...) {
     // Find a customer using supplied
     // argument values -search criteria
     // Return Transfer Object if found,
     // return null on error / not found
   }

 public boolean updateCustomer(...) {
     // update record here using data
     // from the customerData Object
     // True - success, false – failure
 }
 …
```

DATA ACCESS OBJECT (DAO)

DESIGN PATTERN

ENTITY

```
@Entity
public class Customer implements java.io.Serializable {
   // member variables
   int customerNumber;
   String name;
   String streetAddress;
   String city;
   ...

   // getter and setter methods...
   ...
}
```

DATA ACCESS OBJECT (DAO)

DESIGN PATTERN

```
// create the required DAO Factory
DAOFactory cloudscapeFactory = ..
// Create a DAO
CustomerDAO custDAO =
     daoFactory.getCustomerDAO();


// create a new customer
int custNo = custDAO.insertCustomer(...);


// Find a customer object.
Customer cust = custDAO.findCustomer(...);


// modify the values in the Transfer Object. ….
cust.setAddress(...);
cust.setEmail(...);
```

```
// update the customer object using the DAO
custDAO.updateCustomer(cust);


// delete a customer object
custDAO.deleteCustomer(cust);


// select all customers in the same city
Customer criteria=new Customer();
criteria.setCity("New York");
Collection customersList =
    custDAO.selectCustomersTO(criteria);
// returns customersList
```

DATA ACCESS OBJECT (DAO)

DESIGN PATTERN

- **Defines CRUD**
    - Create
    - Read
    - Update
    - Delete
- **See more at**
    - http://www.oracle.com/technetwork/java/dataaccessobject-138824.html

DATA REPOSITORY

DESIGN PATTERN

- **Problem**
  - CRUD
  - Look up a particular object.
  - We know the ID, name and need the object.
  - Find all orders from a customer.

  - Concentration of query construction code.
  - Large number of domain classes or heavy querying.

  - ***The goal of Data Repository is to significantly reduce the amount of boilerplate code required to implement data access layers for various persistence stores.***

DATA REPOSITORY

DESIGN PATTERN BY DELTASPIKE

Sample usage POM.XML

```
<dependency>
    <groupId>org.apache.deltaspike.modules</groupId>
    <artifactId>deltaspike-data-module-api</artifactId>
    <version>${deltaspike.version}</version>
    <scope>compile</scope>
</dependency>

<dependency>
     <groupId>org.apache.deltaspike.modules</groupId>
     <artifactId>deltaspike-data-module-impl</artifactId>
     <version>${deltaspike.version}</version>
     <scope>runtime</scope>
</dependency>
```

DATA REPOSITORY

DESIGN PATTERN BY DELTASPIKE

Sample usage

```
public interface EntityRepository<E, PK extends Serializable> {
  E save(E entity);
  void remove(E entity);
  void refresh(E entity);
  void flush();
  E findBy(PK primaryKey);
  List<E> findAll();
  List<E> findBy(E example, SingularAttribute<E, ?>... attributes);
  List<E> findByLike(E example, SingularAttribute<E, ?>... attributes);
  Long count();
  Long count(E example, SingularAttribute<E, ?>... attributes);
  Long countLike(E example, SingularAttribute<E, ?>... attributes); }
```

DATA REPOSITORY

DESIGN PATTERN BY DELTASPIKE

Sample usage

```
@Repository
 public interface PersonRepository extends EntityRepository<Person, Long> {


    List<Person> findByAgeBetweenAndGender(int minAge, int maxAge, Gender g);


    @Query("select p from Person p where p.ssn = ?1")
    Person findBySSN(String ssn);


    @Query(named=Person.BY_FULL_NAME)
    Person findByFullName(String firstName, String lastName);


    Person findBySsn(String ssn);
}
```

DATA REPOSITORY

DESIGN PATTERN BY DELTASPIKE

Sample usage : Its and injectable component

```
@Stateless
public class MyService {

    @Inject
    private PersonRepositorypersonRepository;

    List<Person> result = personRepository.findAllByAge(18, 65)
        .orderAsc("p.lastName", false).orderDesc("p.age”,false)
        .lockMode(LockModeType.WRITE)
        .hint("org.hibernate.timeout", Integer.valueOf(10))
        .getResultList();

   // Query API style
   QueryResult<Person> paged = personRepository.findByAge(age)
           .maxResults(10) .firstResult(50);
```

DATA REPOSITORY

DESIGN PATTERN

Details:

- **Spring**
- **http://docs.spring.io/spring-data/data-commons/docs/1.6.1.RELEASE/reference/html/repositories.html**

- **DeltaSpike**
- **https://deltaspike.apache.org/documentation/data.html**