

Software Engineering Fundamentals

Software Design



Presentation designed and written by human based on:

I. Sommerville - Software Engineering 8, Ch. 4 Software Processes, Ch. 11 - Architectural Design
R. Pressman - Software Engineering 5th Ed. Ch.14 - Architectural Design, Ch.20 - Object-Oriented Concepts and Principles Ch. 21 - Object-Oriented Analysis
B. Meyer - Object-Oriented Software Construction, Ch. 3 - Modularity
A. Riel - Object-Oriented Design Heuristics, Ch.2 - Classes and Objects, Ch.3 - Topologies of AO vs. OO Applications
R. Martin - The Open/Closed Principle
A. Hunt, D. Thomas - Pragmatic Programmer, Ch5. Decoupling and the Law of Demeter
D. Rubin - Introduction to CRC Cards
K. Beck, W. Cunningham - A Laboratory for Teaching Object-Oriented Thinking

Dr. Petru Florin Mihancea

V20260323

1

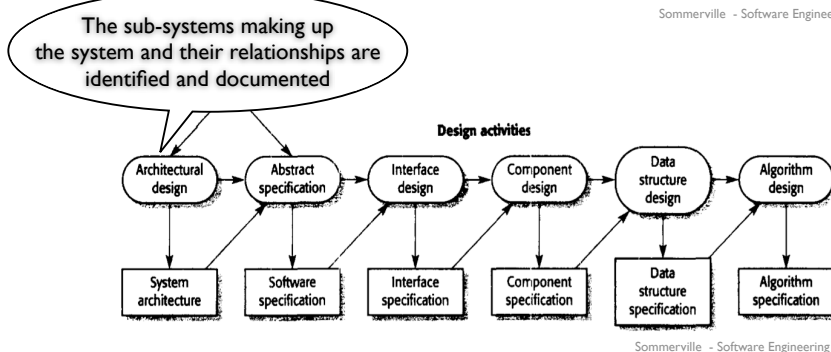
What is Software Design?

Dr. Petru Florin Mihancea

Software Design

A software *design* is a description of the structure of the software to be implemented, the data which is part of the system, the interfaces between system components and, sometimes, the algorithms to be used

Sommerville - Software Engineering



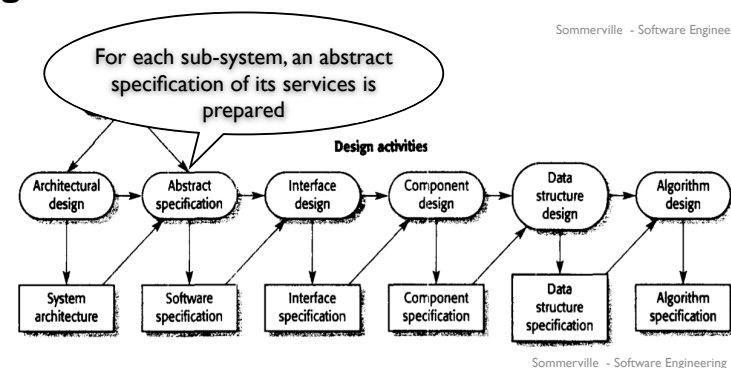
Sommerville - Software Engineering

Dr. Petru Florin Mihancea

Software Design

A software *design* is a description of the structure of the software to be implemented, the data which is part of the system, the interfaces between system components and, sometimes, the algorithms to be used

Sommerville - Software Engineering

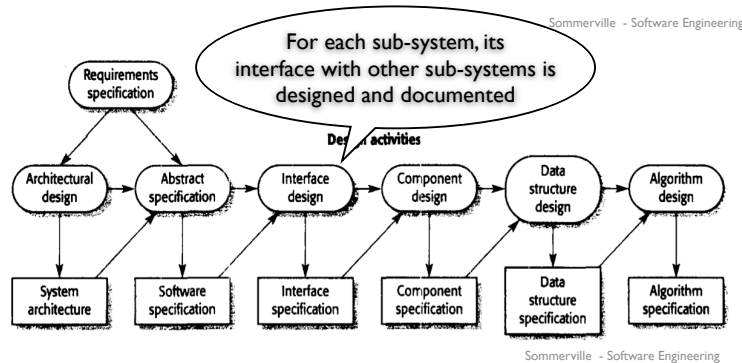


Sommerville - Software Engineering

Dr. Petru Florin Mihancea

Software Design

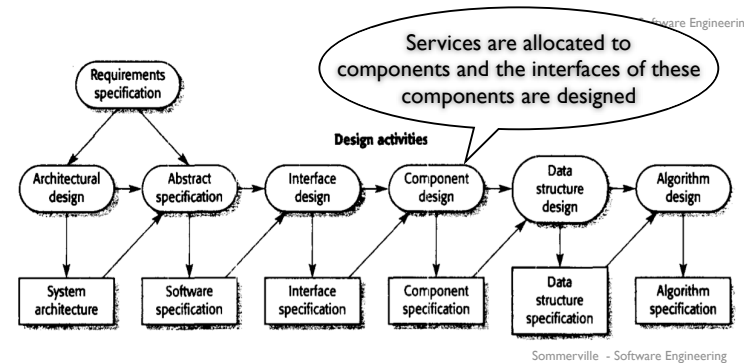
A software **design** is a description of the structure of the software to be implemented, the data which is part of the system, the interfaces between system components and, sometimes, the algorithms to be used



Dr. Petru Florin Mihai

Software Design

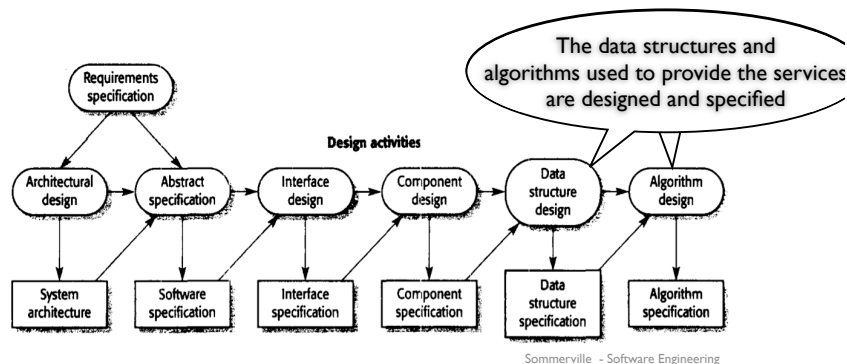
A software **design** is a description of the structure of the software to be implemented, the data which is part of the system, the interfaces between system components and, sometimes, the algorithms to be used



Dr. Petru Florin Mihai

Software Design

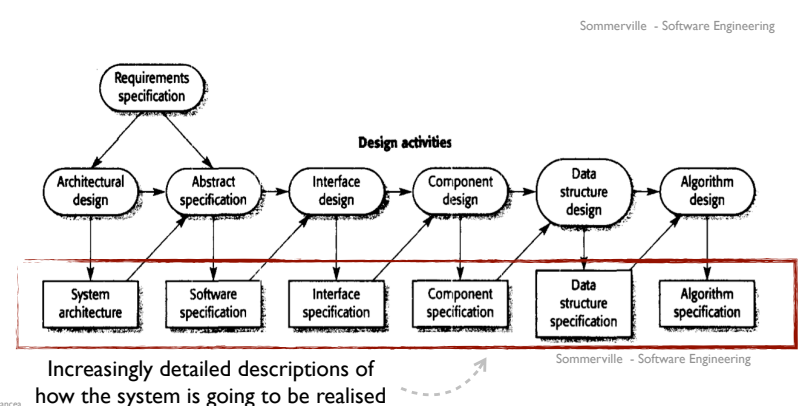
A software **design** is a description of the structure of the software to be implemented, the data which is part of the system, the interfaces between system components and, sometimes, the algorithms to be used



Dr. Petru Florin Mihai

Software Design

A software **design** is a description of the structure of the software to be implemented, the data which is part of the system, the interfaces between system components and, sometimes, the algorithms to be used



Dr. Petru Florin Mihai

Design Levels (1)

1. Architectural Design

Large systems are always decomposed into sub-systems that provide some related set of services; **architectural design** is the process of identifying these sub-systems, and establishing a framework for sub-system control and communication

Sommerville - Software Engineering

2. Detailed Design

After an overall system organisation has been chosen, you need to make a decision on the approach to be used to **decompose sub-systems into modules [components]**

Sommerville - Software Engineering

Dr. Petru Florin Mihailescu

Design Levels (2)

There is **no strict conceptual boundary** between architectural (sub-system) and detailed (module) design ... typically, modules are smaller than sub-systems which allows alternative decomposition styles e.g., **object-oriented decomposition** - you decompose the sub-system into a set of communicating objects

... although in practice we might treat them as separate activities

In the **Waterfall** process model, **both** are performed within a single phase, after the requirements definition phase (previous chapter) and before the implementation (next chapter)

In the **Incremental Delivery** process model, **initial design stages** (i.e., architectural design) are performed **earlier in the process**, while **later design stages** (i.e., detailed design) are carried out **incrementally** (in parallel with activities from the previous and subsequent chapters)

What means good design?

Dr. Petru Florin Mihailescu

Modularity

A software construction method is **modular** if it helps designers produce software systems made of **autonomous elements** connected by a **coherent, simple** structure

Meyer - Object-Oriented Software Construction

Modularity is the property of a system that has been decomposed into a set of **cohesive and **loosely coupled** modules [sub-systems]**

Booch - Object-Oriented Analysis and Design

independent on the design level (thus, modules and sub-systems concepts are interchangeable when discussing modularity) and favours extensibility, reusability and other factors of software maintenance

Dr. Petru Florin Mihailescu

Modularity Criteria (1)

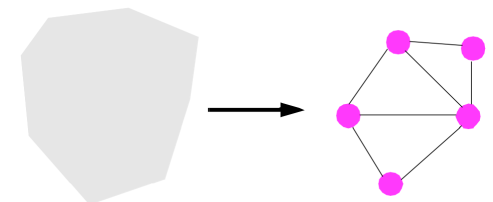
Decomposability

decompose **a problem** into several **less complex sub-problems**, connected by a **simple structure** and **independent enough** to allow work separately on each of them

Goal

labor division: distribute the work on different module to different people

... but dependencies must be kept to a **minimum** and must be **explicit**



Meyer - Object-Oriented Software Construction

Dr. Petru Florin Mihailescu

Modularity Criteria (1)

Decomposability

decompose a problem into several less complex sub-problems, connected by a simple structure and independent enough to allow work separately on each of them

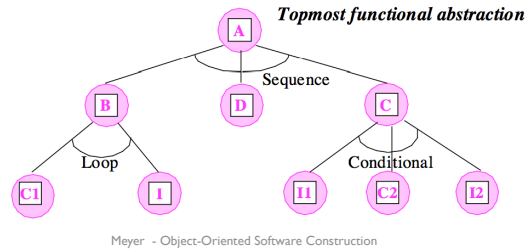
Example

Top-Down Functional Design

Counterexample

Initialisation module endangering the autonomy (e.g. the author of init module will work close to each of the authors of the other modules)

In OO every module is responsible to initialise its data



Jr. Petru Florin Mihances

Modularity Criteria (2)

Composability

the obtained modules should be freely combined to produce new systems (possibly in a different environment)

Goal

reuse and thus the modules should be sufficiently autonomous from their immediate goal to be possible to be used in widely different contexts

Example

Libraries

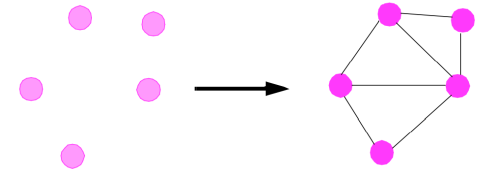
powerful collection, linear algebra, etc. libraries

UNIX Shell Command

Counterexample

Top-Down Functional Design

The modules tend to be closely linked to the immediate context and thus impossible to reuse in different contexts



Jr. Petru Florin Mihances

Modularity Criteria (3)

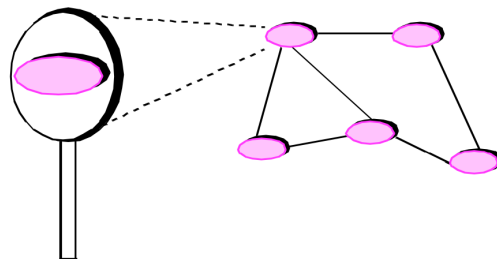
Understandability

a human reader must understand each module without having to know other, or, at worst, by having to examine only a few other modules

Counterexample

sequential dependencies

e.g., A -> B -> C when B works correctly only if it is executed before C and after A



Jr. Petru Florin Mihances

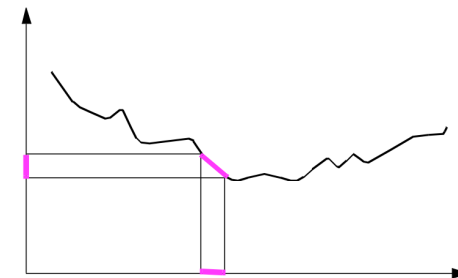
Modularity Criteria (4)

Continuity

a small requirement change triggers a change of just one or of a small number of modules

Example

symbolic constants instead of magic constant (e.g. public static final int MAX = 10;)



Jr. Petru Florin Mihances

Modularity Criteria (5)

Protection

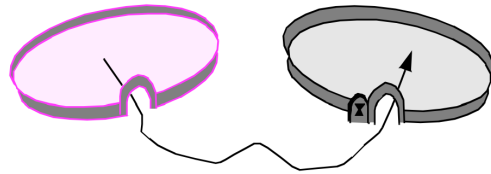
abnormal conditions occurring at run time in a module will remain confined to that module, or at worst will only propagate to a few neighbouring modules

Example

Validate inputs at the source

Counterexample

Undisciplined usage of exception



Meyer - Object-Oriented Software Construction

Jr. Petru Florin Mihances

Rules to Ensure Modularity

Direct Mapping

keep a close mapping between the structure of the solution and the structure of the problem

Favours

Continuity - limits the impact of change

Decomposability - modular decomposition of the problem is a good start for modular decomposition of the software

See later during object-orienter analysis (!)

Jr. Petru Florin Mihances

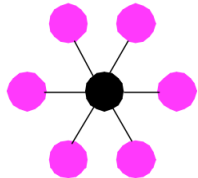
Favours **all** criteria: e.g., composability - how can you reuse a module that depends on many other modules?

Rules (2)

Few Interfaces

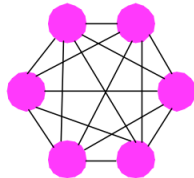
Every module should communicate with as few others as possible

Centralised structure

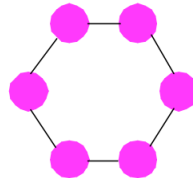


(A)

Decentralised structure found in good object-orientation



(B)



(C)

Meyer - Object-Oriented Software Construction

Jr. Petru Florin Mihances

Particular important for **continuity** and **protection**

Rules (3)

Small Interfaces

if two modules communicate, they should exchange as little information as possible



Meyer - Object-Oriented Software Construction

Jr. Petru Florin Mihances

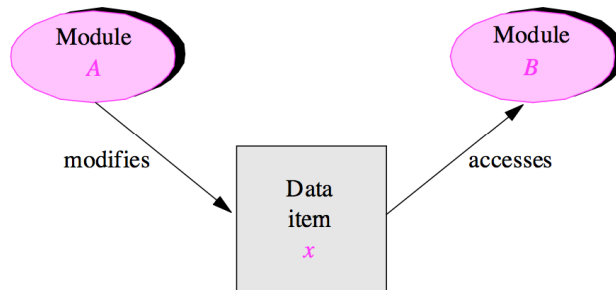
Particular important for
**continuity, protection,
continuity and
understandability**

Rules (4)

Explicit Interfaces

**When two modules communicates this
must be obvious from their text**

Indirect coupled although
there is no apparent
connection (i.e.,
procedure call)



Meyer - Object-Oriented Software Construction

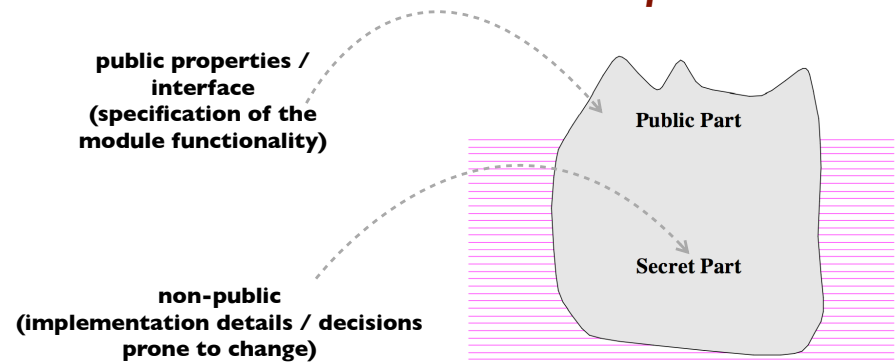
Jr. Petru Florin Mihances

Primary reason: **continuity**
- changing some secret part
will not affect the clients

Rules (5)

Information Hiding

**for a module, we must select a subset of
module's properties as the official information
about the module, to be made available to
authors of client modules**



Meyer - Object-Oriented Software Construction

Jr. Petru Florin Mihances

2

Architectural Design

High-Level Design. Overall System Organisation.

Jr. Petru Florin Mihances

Architectural Design

**Large systems are always decomposed into sub-
systems that provide some related set of services;
architectural design is the process of identifying
these sub-systems, and establishing a framework
for sub-system control and communication**

Sommerville - Software Engineering

Influenced also by non-functional
requirements

Jr. Petru Florin Mihances

How?

Performance

few sub-systems, reduced communication
large-grained sub-systems

Security

layered organisation with critical assets in the innermost layers

Availability

include redundant sub-systems so that to update/replace them without stopping the system

Maintainability

fined-grained, self-contained sub-systems that may be readily changed
avoid shared data structures, separate data producers from consumers

Jr. Petru Florin Mihaices

Architectural Styles

*Architecture styles [...] describe **a named relationship** of sub-systems covering a variety of architecture characteristics. An architecture style name [...] creates a single name that acts as shorthand between experienced architects.*

*An architecture style describes the **topology**, assumed and **default architecture characteristics**, both beneficial and detrimental.*

N. Ford, Mark Richards - Fundamentals of
Software Architecture

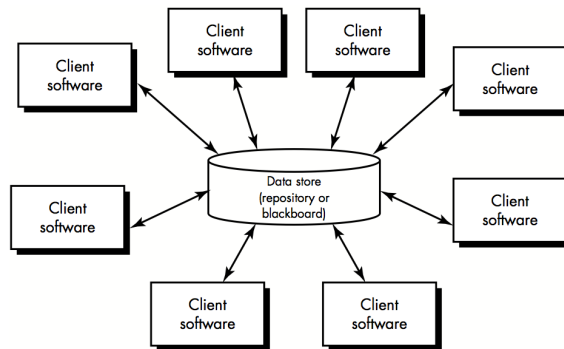
Jr. Petru Florin Mihaices



Repository/Blackboard Style

Repository - the data store is **passive**, the sub-system being responsible to access the repository independently of the activity of other sub-systems / data changes

Blackboard - the data store sends notifications to sub-systems when data of interest changed / becomes available



Pressman - Software Engineering

All data in a system is managed in a central repository that is accessible to all sub-systems. Sub-systems do not interact directly, only through the repository.

Jr. Petru Florin Mihaices

Pros vs. Cons

Pros

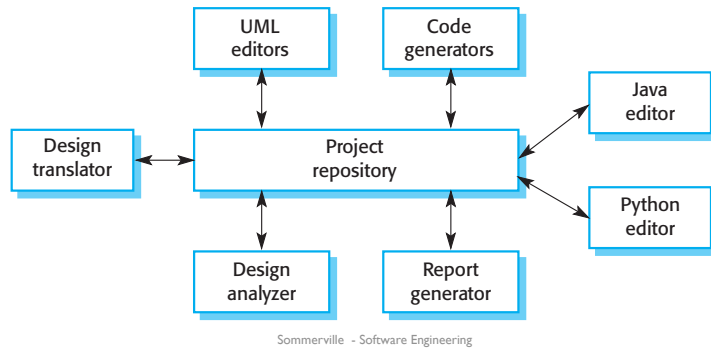
efficient to share large amounts of data
easy to add new sub-systems if they are compatible with the shared data model
data producers are not concerned with how the data is used
centralised activities such as security, backup, recovery; sub-systems can focus on their essential functions

Cons

sub-systems must agree on the repository data model; difficult to integrate other sub-systems if their data models do not fit the agreed schema; migrating to a new shared data model will be expensive;
all sub-systems must have the same backup, security, recovery policy
difficult to distribute the repository on different machines

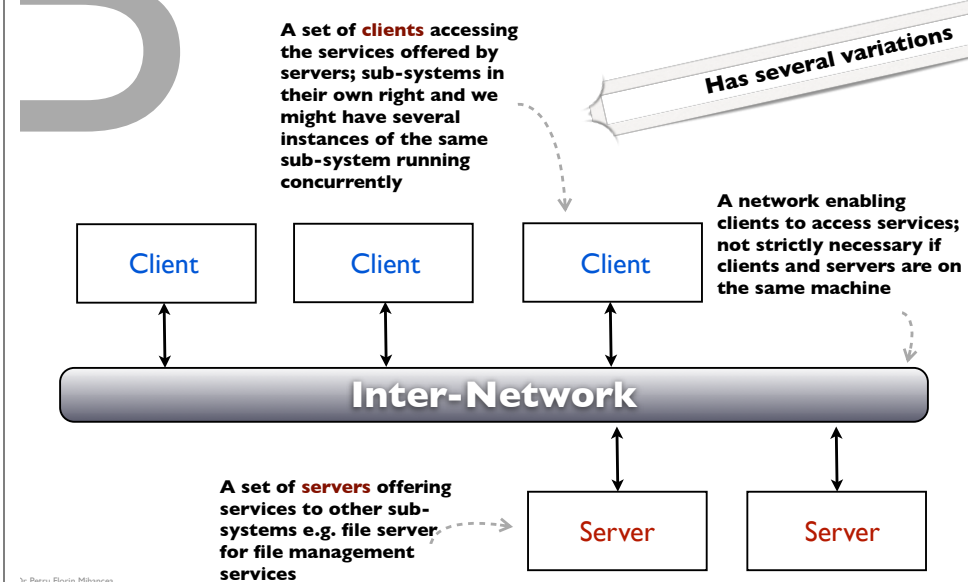
Jr. Petru Florin Mihaices

Example: IDEs



Dr. Petru Florin Mihances

Client-Server Style



Pros vs. Cons

Pros

- distributed architecture
- easy to add new servers and integrate it with the rest of the system
- upgrade servers transparently
- servers are not aware of clients (identity nor number)
- servers have their own data model

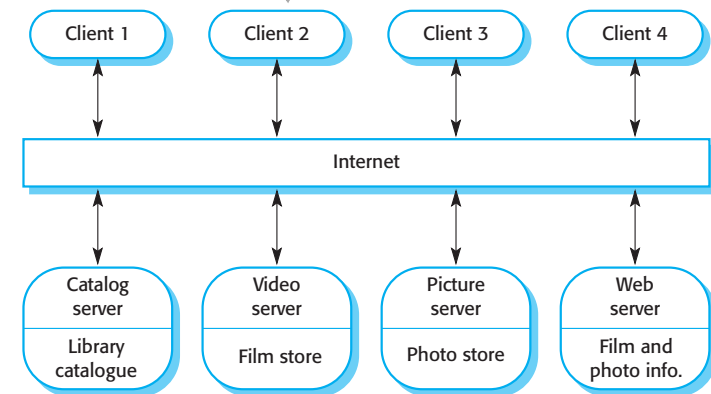
Cons

- changes in existing clients / servers may be required to really benefit of new servers
- performance problems if large amounts of data are exchanged (e.g., data conversion from one representation to another)
- special attention to denial of service attacks or server failures

Dr. Petru Florin Mihances

Example: A Film/Photo Library

In this case, a simple user interface based on HTML (i.e., a web browser)



Links to data about the films, etc.

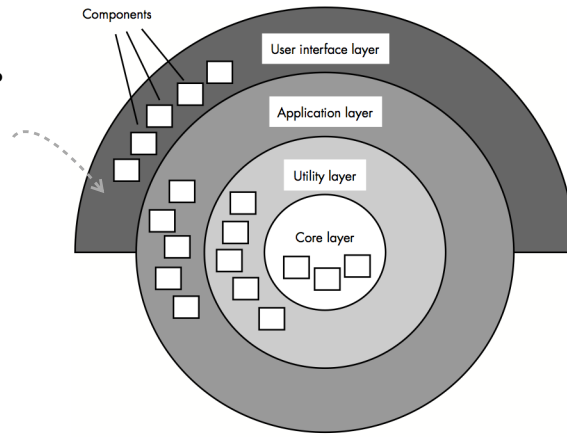
Videos must be sent fast, compressed, decompressed, converted in different formats, etc.

The actual web-server

Layered (aka. Abstract Machine) Style

Each layer is a sub-system similar to a “machine” whose “language” is represented by the services it provides to the upper layer

This “language”/services is implemented based on the “lower-language”/lower-level-services provided by the layer immediately below (i.e., layer N uses only the N-1 layer)



Pressman - Software Engineering

Jr. Petru Florin Mihances

Pros vs. Cons

Pros

supports easily incremental development

changeable and portable

as long as a layer does not modify its interface, a layer can be replaced by another equivalent layer

when a layer interface changes or new facilities are added only the adjacent layer is affected

Cons

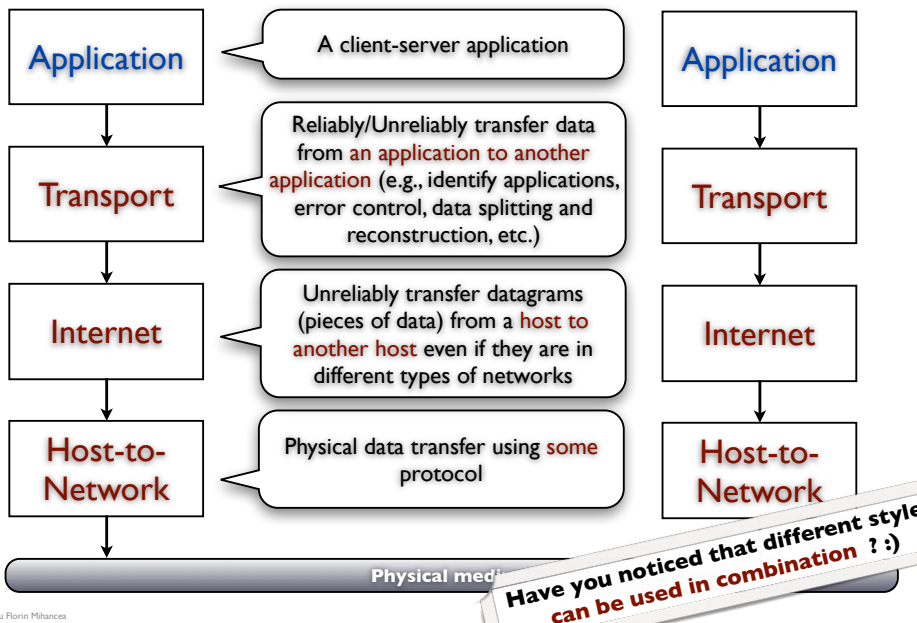
difficult to achieve such a structure

reduces performance because of increased communication through each layer (thus, sometimes, a layer should communicate **directly** with an inner layer and **not via** the intermediate layers between them)

a low-level service might be required at the level of each high-level layer (thus, the same direct communication is considered)

Jr. Petru Florin Mihances

Example: TCP/IP

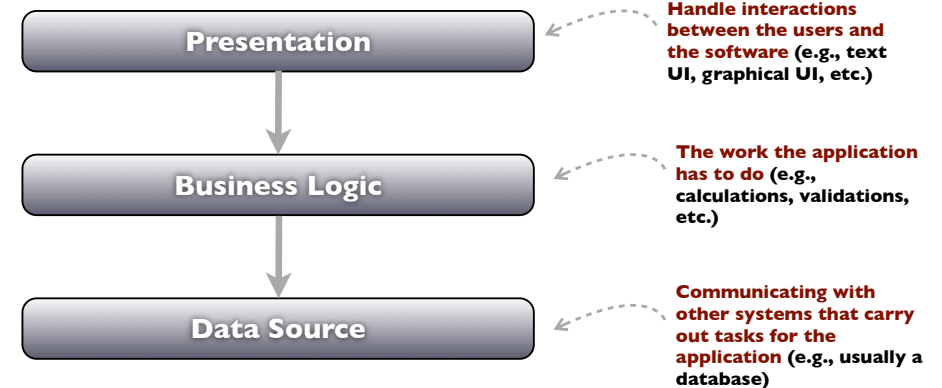


Jr. Petru Florin Mihances

A More Concrete Pattern

Enterprise applications: manipulates & stores much data to support the business of an organization (e.g., payroll, patient records, cost analysis, insurance systems, etc.)

M.Fowler - Patterns of Enterprise Application Architecture

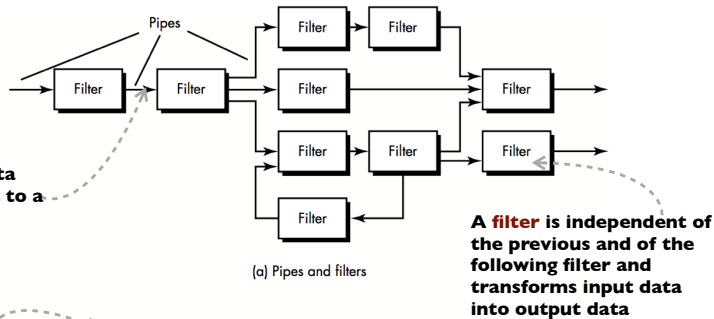


Jr. Petru Florin Mihances

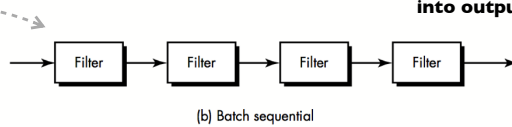
D

Pipes and Filters Style

A **pipe** transfers data from a filter output to a filter input



When transformations are sequential and in batches



Pressman - Software Engineering

This architecture is applied when **input data** are to be transformed through a **series of computational or manipulative components (i.e., filters)** into **output data**

Dr. Petru Florin Mihances

Pros vs. Cons

Pros

a particular filter can be reused

intuitive, since many people think of their work in terms of input output processing

evolving the system by changing/adding new filters should be easy

easy to implement sequentially or concurrently

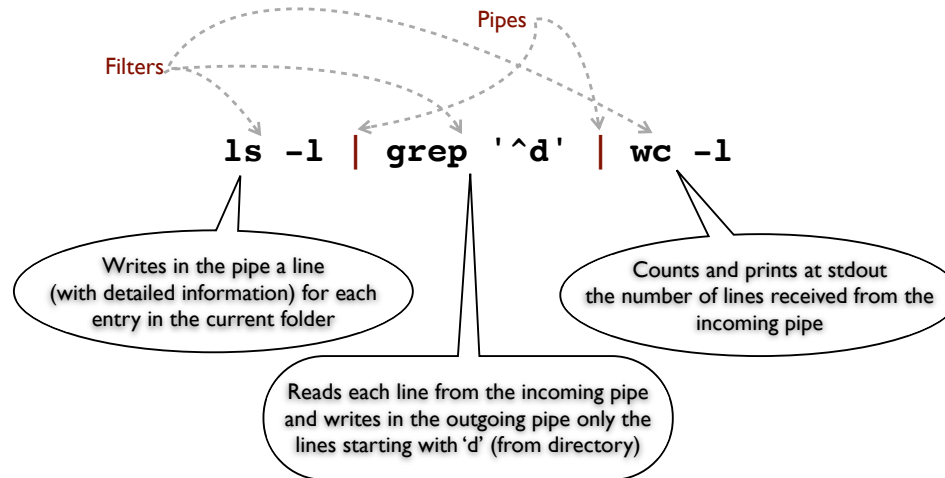
Cons

a filter must either agree with its communicating filters on the schema of the data that is going to be transformed or a **standard data format should be imposed** globally; the later is the only feasible approach in order to have standalone, reusable filters

difficult for interactive systems due to the need of some stream of data to be processed; simple textual in/out can be modelled in this way but for GUI mouse clicks, menu selections, etc. is difficult

Dr. Petru Florin Mihances

Example: UNIX Shell



Dr. Petru Florin Mihances

2

Detailed Design Object-Oriented Analysis and Design

Dr. Petru Florin Mihances

Detailed Design

After an overall system organisation has been chosen, you need to make a decision on the approach to be used to **decompose sub-systems into modules**

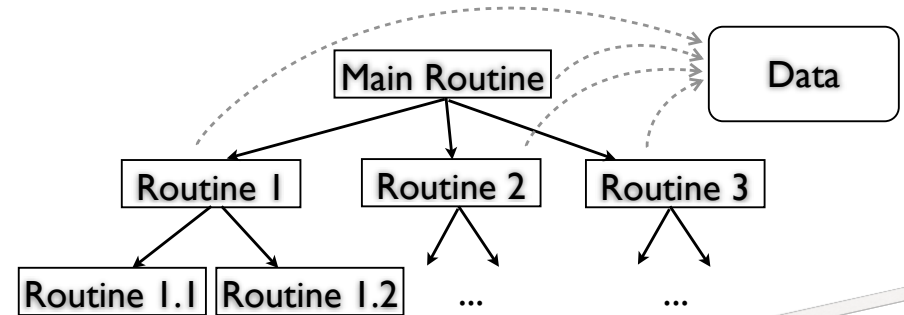
Sommerville - Software Engineering

What are these **modules**?

Dr. Petru Florin Mihances

Algorithmic Decomposition

each “module” represent **an execution step** from a task to be performed

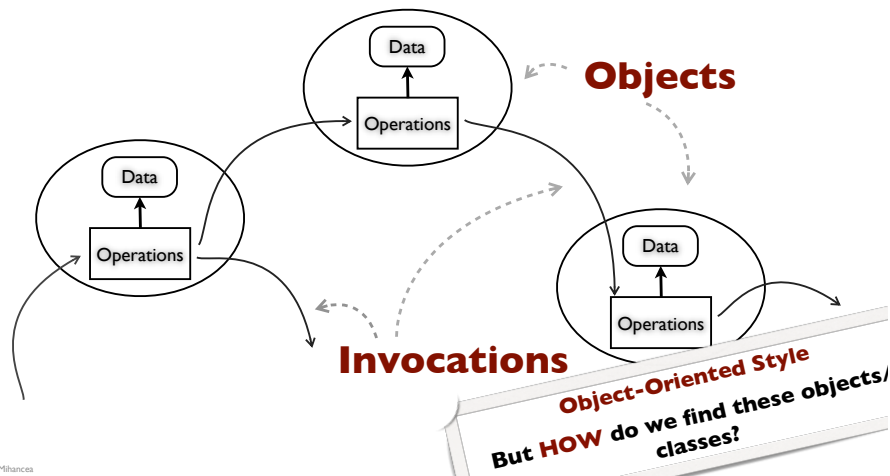


Dr. Petru Florin Mihances

Main Program with Subroutines Style

Object-Oriented Decomposition

“modules” are collaborating objects (classes)



Dr. Petru Florin Mihances

Object-Oriented Analysis

The objective of object-oriented analysis is to develop an [object-centric] model(s) that **describes** computer software as it works to satisfy a set of customer-defined requirements

Pressman - Software Engineering

Important questions

- What are the **relevant objects in the problem** to be solved?
- How do they **relate and interact in the problem** with one another?
- How does an object **behave in the problem**?

Dr. Petru Florin Mihances

Objects in the **problem domain**

Object-Oriented Design

... is concerned with developing an object-oriented model of a software **to implement** the identified requirements

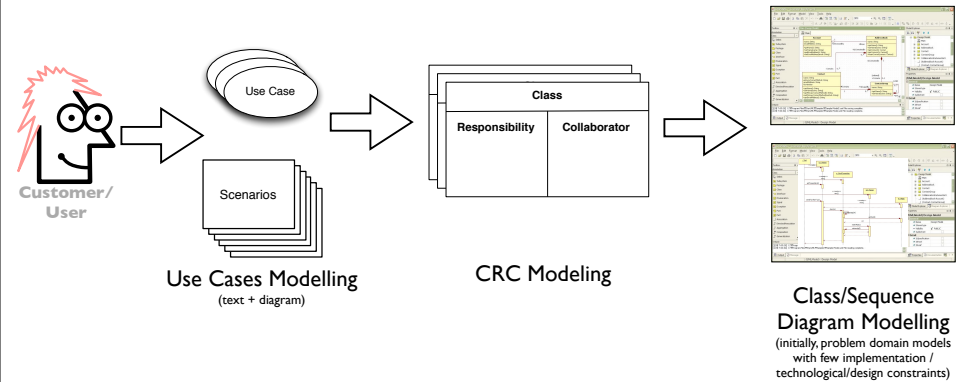
Objects in the **solution** (of the problem) domain
(as they appear in the program)

usually, **close relationship** between **solution objects** and **problem objects** but the **designer** may also **add new objects** and **transform problem objects** to design the product

Do you remember the **Direct Mapping** modularity rule?

Jr. Petru Florin Mihances

Overview



Jr. Petru Florin Mihances

UML Class and Sequence Diagrams / Models

Jr. Petru Florin Mihances

Unified Modeling Language

Family of **graphical** notations

for **modeling** an (OO) system



Jr. Petru Florin Mihances

Types of UML models

Behavioral

e.g. Sequence diagram (SD)

Structural

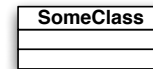
e.g. Class diagram (CD)



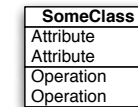
Dr. Petru Florin Mihances

a Class diagram

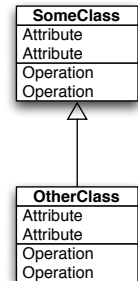
Structural model



Classes



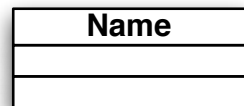
Features



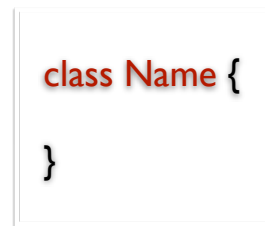
Relations

Dr. Petru Florin Mihances

a Class diagram



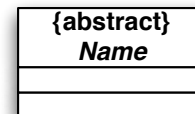
UML sketch



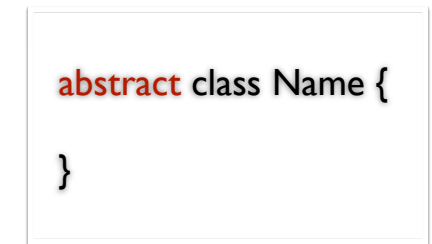
Java sketch

Dr. Petru Florin Mihances

a Class diagram



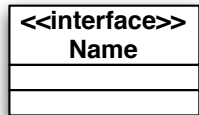
UML sketch



Java sketch

Dr. Petru Florin Mihances

a Class diagram



UML sketch

```

interface Name {

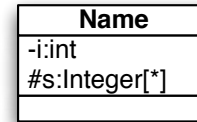
}
  
```

Java sketch

Jr. Petru Florin Mihances

a Class diagram

Attributes



visibility name : type multiplicity
= implicitValue

```

+ public
- private
# protected
~ package
  
```

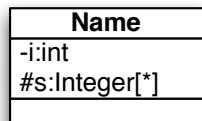
UML sketch

Java sketch

Jr. Petru Florin Mihances

a Class diagram

Attributes



visibility name : type multiplicity
= implicitValue

```

1 - exactly one
0..1 - zero or at most one
0..* or * - zero or more but
NO upper limit
  
```

UML sketch

```

class Name {
    private int i;
    protected List<Integer> s;
    //s must be somehow initialized / created
}
  
```

```

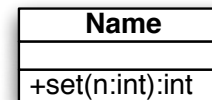
class Name {
    private int i;
    protected Integer[] s;
    //s must be somehow initialized / created
    //an index may be required + you must
    //guarantee NO upper limit if necessary
    //(e.g. re-create & copy the array)
}
  
```

Java sketch

Jr. Petru Florin Mihances

a Class diagram

Operations



visibility name(param_list) : ret_type

direction name : type = default

```

class Name {
    public int set(int n) {
        ...
    }
}
  
```

UML sketch

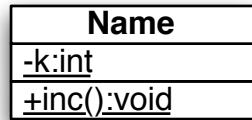
Java sketch

Jr. Petru Florin Mihances



Class diagram

Scope



UML sketch

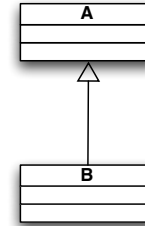
```
class Name {
    private static int k;
    public static void inc() {
        ...
    }
}
```

Java sketch



Class diagram

Generalisation & Realisation



UML sketch

```
class A {
}

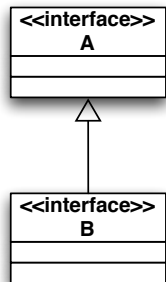
class B extends A {
}
```

Java sketch



Class diagram

Generalisation & Realisation



UML sketch

```
interface A {
}

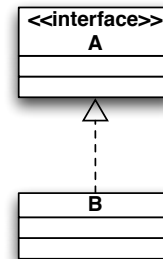
interface B extends A {
}
```

Java sketch



Class diagram

Generalisation & Realisation



UML sketch

```
interface A {
}

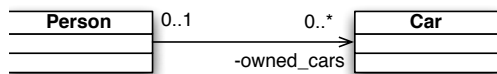
class B implements A {
}
```

Java sketch

a

Class diagram

Association



UML sketch

```

class Person {
    //list must be somehow initialized / created
    private List<Car> owned_car;

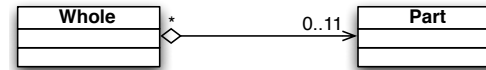
    //add, remove methods usually exist
}
  
```

Java sketch

a

Class diagram

Aggregation



UML sketch

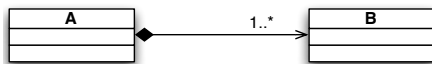
Similar to association

Java sketch

a

Class diagram

Composition



- I. No-sharing
- II. B objects cannot exist without their A object

UML sketch

```

class A {
    private List<B> my_list =
        new ...

    public A(...) {
        my_list.add(new B(...));
    }
    public void add(...) {
        my_list.add(new B(...));
    }
}
  
```

Java sketch

a

Class diagram

Dependency



UML sketch

```

class A {
    public void m(B x) {
        x.doS();
    }
}
  
```

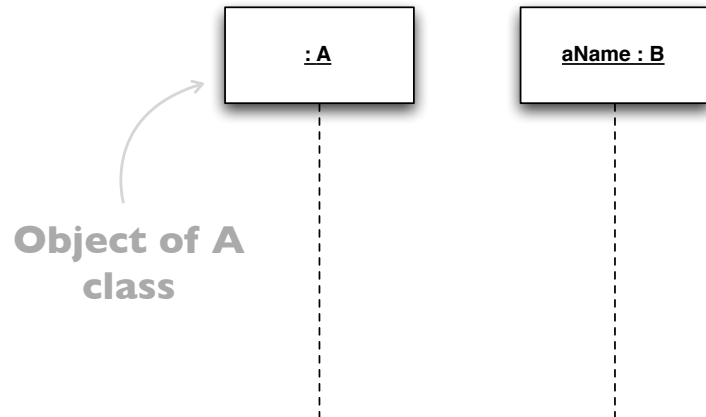
And many other cases ...

Java sketch

b

Sequence diagram

Behavioural & Interaction model

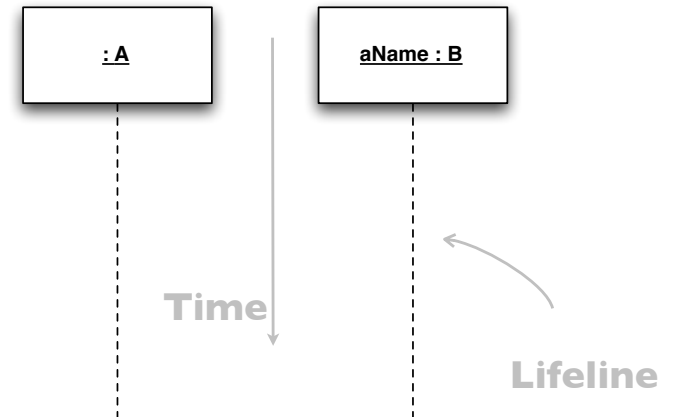


Jr. Petru Florin Mihances

b

Sequence diagram

Behavioural & Interaction model

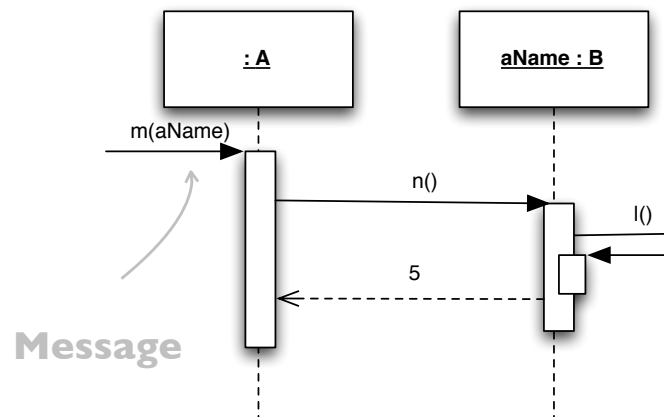


Jr. Petru Florin Mihances

b

Sequence diagram

Behavioural & Interaction model

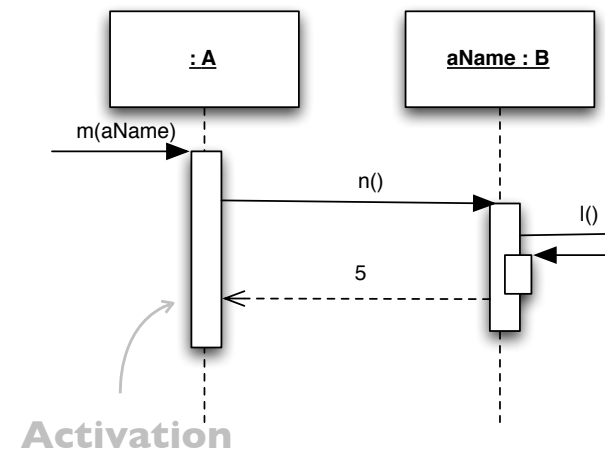


Jr. Petru Florin Mihances

b

Sequence diagram

Behavioural & Interaction model

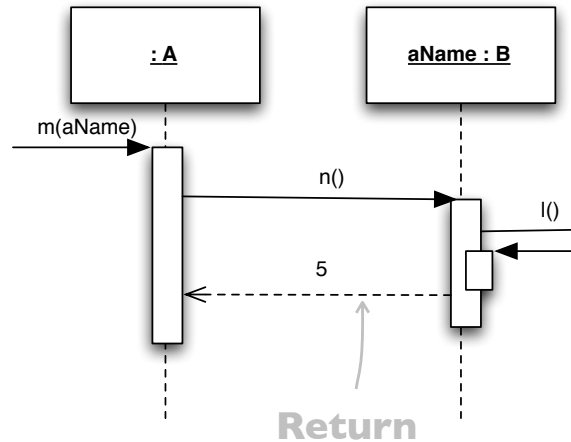


Jr. Petru Florin Mihances

b

Sequence diagram

Behavioural & Interaction model

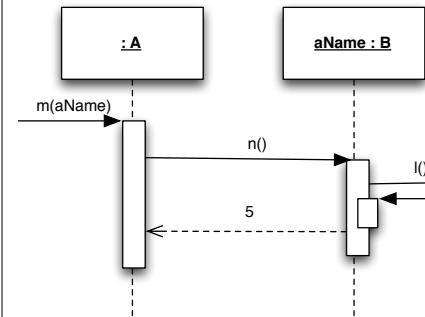


Jr. Petru Florin Mihances

b

Sequence diagram

Behavioural & Interaction model



UML sketch

Jr. Petru Florin Mihances

```

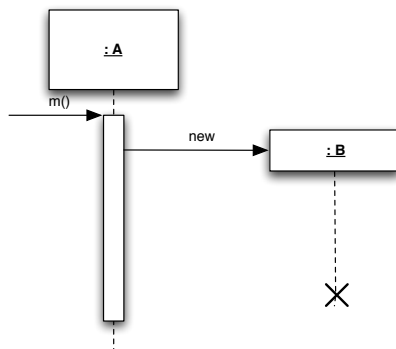
class A {
    public void m(B x) {
        x.n();
    }
}
class B {
    public int n() {
        this.l();
        return 5;
    }
    public void l() {}
}
  
```

Java sketch

b

Sequence diagram

Object creation & deletion



UML sketch

Jr. Petru Florin Mihances

```

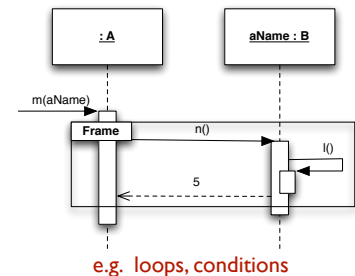
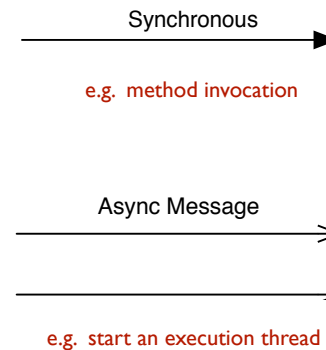
class A {
    public void m() {
        ...
        new B();
        ...
        //the object is
        //no more accessible
    }
}
  
```

Java sketch

b

Sequence diagram

Other notations



Comments

Jr. Petru Florin Mihances



Class-Responsibility-Collaborator (CRC) Modeling

Jr. Petru Florin Mihances

CRC Modeling

*A technique to identify candidate **classes** and indicate their **responsibilities** and **collaborators***

Use simple index cards

Class	
Responsibility	Collaborator

Class name:	
Class type: (e.g., device, property, role, event)	
Class characteristic: (e.g., tangible, atomic, concurrent)	
responsibilities:	collaborations:

Pressman - Software Engineering

Various dimensions are recommended by different authors (e.g., 4"x6", 3"x5"), but a simple approach is to split an A4 paper into 8 cards :)

Jr. Petru Florin Mihances

CRC Modelling Session

The Team (a maximum of 6 people)

Domain users

know the problem domain, good communication skills

Object-Oriented analysts

understand CRC & OO modelling processes, experience in OO development

One facilitator

understand CRC & OO modelling process

chairs the session, **responsible** to keep the session progressing, act as an intermediary when debates occur

May include non-active participants

scribe (capturing information that is not written on the CRC cards),

observers (other users and developers)

Jr. Petru Florin Mihances

CRC Modelling Session (2)

1. Identify one appropriate scenario

should be well documented

select an easy scenario first

start with a **normal execution** scenario

related scenarios should be modelled separately

during this discussion, an **initial set of classes** might be identified; for those classes create a CRC card, and also create cards for classes that already exist (have been previously modelled)

Jr. Petru Florin Mihances

CRC Modelling Session (3)

2. “Execute” the scenario and identify **classes**, **responsibilities** and **collaborators**

In the beginning think that a requester/user asks a **class** to start the scenario execution

That class might ask for some information to start the scenario and the requester could say that the class must know that information; thus we have identified a **responsibility** of that class or another **class** with which our class **collaborates** in order to find that information

When a class, responsibility or collaborator is identified it is written on the corresponding CRC card

Jr. Petru Florin Mihai

CRC Modelling Session (3)

2. “Execute” the scenario and identify **classes**, **responsibilities** and **collaborators**

In the beginning think that a requester/user asks a **class** to start the scenario execution

That class might ask for some information to start the scenario and the requester could say that the class must know that information; thus we have identified a **responsibility** of that class or another **class** with which our class **collaborates** in order to find that information

During the scenario “execution” many other information and actions will be required; the starting class will know that information or how to perform the action (**new responsibility** is found for it) **or** it will ask another **collaborator** class (maybe a new class) for that information or action (and maybe new responsibilities for the collaborator are found)

The team continues until the entire scenario can be executed using the identified classes with their responsibilities and following the links to the collaborators

Jr. Petru Florin Mihai

Class	
Responsibility	Collaborator

Classes

How to **identify** them ?

“Grammatical parse” on the processing narrative

All nouns or noun clauses are **potential** objects/classes

Categorisation

external entities (e.g., devices)

things (e.g., reports, displays)

events (e.g., a completion/occurrence of an action)

roles (e.g., manager, engineer)

organisational units (e.g., division, department)

places (e.g., classroom)

structures (e.g., four-wheel vehicles)

Jr. Petru Florin Mihai

Class	
Responsibility	Collaborator

Classes

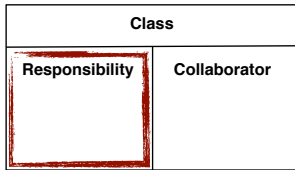
How to **identify** them ?

Identify what the customer interacts with

screens, reports, etc. represent user interface classes, and for the sake of the CRC modelling, use a single class to represent the UI

If a class can't be named with **less than 3 words**, it is **probably not a class** but a **responsibility of another class**

Jr. Petru Florin Mihai



Responsibility

What are them and how to **identify** them ?

Anything a class knows or does

attributes and operations

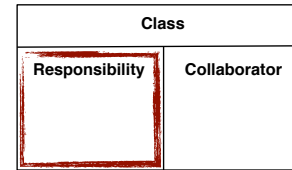
The **attributes** define the object, clarify what is meant by the object in the **context of the problem**

Finding attributes

analyse the processing narrative and select those **things** that reasonably belong to an object

ask what data items fully define the object in the context of the current problem

Jr. Petru Florin Mihances



Responsibility

What are them and how to **identify** them ?

Anything a class knows or does

attributes and operations

The **operations** define the object behaviour

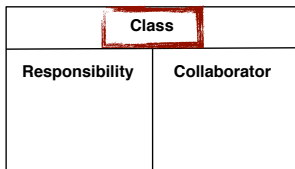
Finding operations

analyse the processing narrative and select those **operations** that reasonably belong to an object; usually they will appear as **verbs**

What do they usually do ?

- computations
- data manipulation (e.g., add, remove)
- query (e.g., about the state of the object)
- monitor an object (e.g., the occurrence of an event)

Jr. Petru Florin Mihances



Classes

How to **identify** them ?

Important characteristics (to become object/class in the model)

1. Retain information

data about the object must be remembered to enable the system to function

2. Needed services

must have operations that change the value of the object attributes

3. Multiple attributes

an object with a single attribute should be represented as an attribute of another object during analysis to enable the focus on **“major”** information

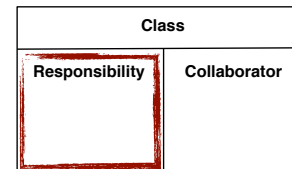
4. Common attributes and operations

attributes and operations are common to all occurrences of an object

5. Essential requirements

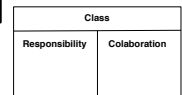
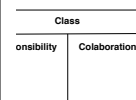
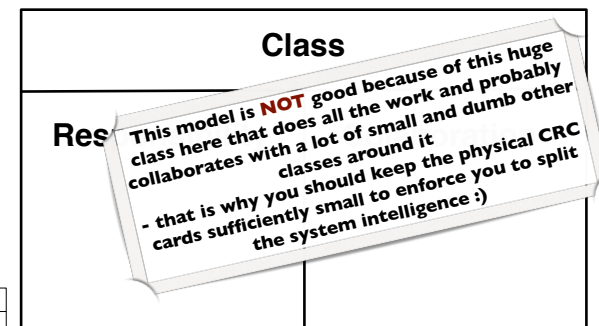
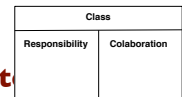
entities that produce or consume information essential to the operation of any solution

Jr. Petru Florin Mihances

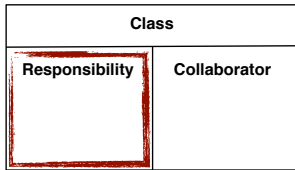


Responsibility Guidelines

I. System intelligence should be **evenly distributed**



Jr. Petru Florin Mihances



Responsibility Guidelines

1. System intelligence should be **evenly distributed**
2. Information about **one thing should be localized in a single class** not distributed across multiple classes

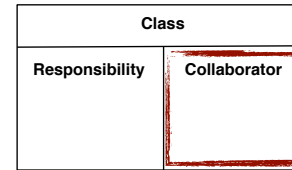
manipulating/storing a specific kind of information should be the responsibility of a single class

3. Information and the behavior related to it **should reside in the same class** (do you remember encapsulation? :))

4. Responsibilities should be **shared among related classes**

the objects *player*, *player-body*, and *player-head* have their own attributes (e.g., position on the screen); a player knows that it must display itself but collaborates with the other objects to actually display the player on the screen

Jr. Petru Florin Mihances



Collaborator

What are them and how to **identify** them ?

A class fulfils its responsibility

1. **exclusively** using its operations and attributes
2. in **collaboration** with another class

Finding collaborators

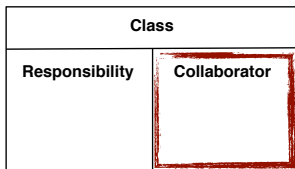
Can this class fulfil each responsibility all by itself? If it cannot, it must interact with another class and thus a collaborator is found

A class may need information that it doesn't have; a collaborator will provide that information

A class may need to modify some information that it doesn't have; a collaborator will probably know to do that

...

Jr. Petru Florin Mihances



Collaborator

What are them and how to **identify** them ?

Usually collaborations help identifying **relations between classes** :)

has-knowledge-of (UML associations)

has-as-part / is-part-of (UML aggregation)

UML composition

UML dependencies

Based also on responsibilities we can start drawing **class diagrams** :)

Jr. Petru Florin Mihances

CRC Modelling Session (4)

3. CRC model **review** (role-playing)

- a. Cards are distributed to participants; collaborating cards must be given to different people

Does this execution sound familiar ?

- b. The facilitator reads the scenario and when a particular object comes to "execution" the person having the corresponding card "becomes" that object (and may hold the card in the air)

- c. Continuing the scenario, it will be found that the current object must perform some responsibility; the person reads it from the card and describes it; during this description the execution could go to a collaborator (and the corresponding person will become the corresponding object and will act in a similar way)

- d. If it is observed that the classes (with their responsibilities and collaborators) can accommodate the current scenario the review is finished; otherwise, the model should be improved (e.g., new responsibilities, collaborations, classes)

Based on collaborations we can start drawing **sequence diagrams** :)

Jr. Petru Florin Mihances



Object-Oriented Analysis and Design **Heuristics**

Jr. Petru Florin Mihances



The class **proliferation** problem

How to (properly) reduce the number of classes?

Jr. Petru Florin Mihances

Eliminate irrelevant classes

Riel's Heuristic 3.7

Irrelevant class

Has **no meaningful** behaviour

i.e., only some get/set/print operations acting on some attributes

e.g., a method returning the colour of a car is not usually an interesting behaviour in a problem domain

Eliminating such classes will probably imply demoting them to attributes

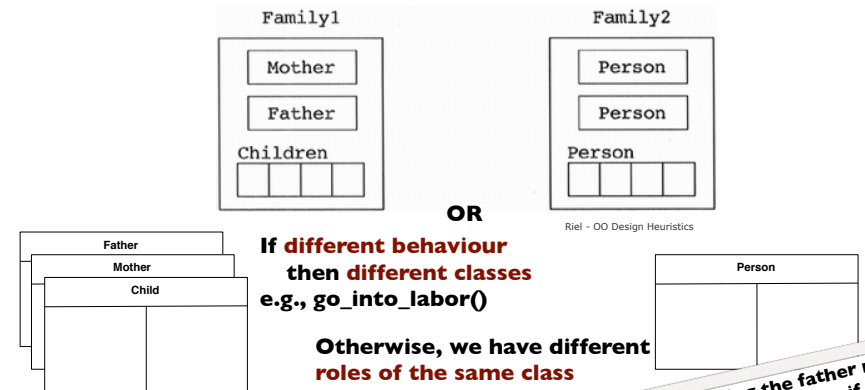
Special cases for get/set operations

e.g., for a sensor getting its status is a relevant behaviour

Jr. Petru Florin Mihances

Be sure the abstractions that you model are **classes** and **not** simply the **roles** objects play

Riel's Heuristic 2.11



Riel - OO Design Heuristics

Otherwise, we have different roles of the same class e.g., change_diapers()

Be careful: a person playing the father role **must be able** to change_diapers() even if only a mother role does that! Otherwise we are again in the first case

Jr. Petru Florin Mihances

Do not turn an operation into a class.

Be suspicious of any class whose name is a verb or is derived from a verb, especially those that have only one piece of meaningful behaviour ...

Riel's Heuristic 3.9

Image	

InvertImage	
invertAnImage	

Jr. Petru Florin Mihances

Constructive Joke

How many object-oriented developers are needed to screw a light bulb ?



http://en.wikipedia.org/wiki/Edison_screw

0

... because an object-oriented developer will teach the light bulb to screw itself !

Jr. Petru Florin Mihances

Do not turn an operation into a class.

Be suspicious of any class whose name is a verb or is derived from a verb, especially those that have only one piece of meaningful behaviour ...

Riel's Heuristic 3.9

Image	
invertTheImage	

InvertTheImage	
invertAnImage	

Jr. Petru Florin Mihances

Do not turn an operation into a class.

Be suspicious of any class whose name is a verb or is derived from a verb, especially those that have only one piece of meaningful behaviour ...

Riel's Heuristic 3.9

Image	
invertTheImage rotateTheImage scaleTheImage ...	

Image	

Rotate		Scale	
angle execute		factor execute	

when requirements treat the actions as atoms/things/objects that can be manipulated individually (e.g., record persistently the history of actions performed on an image, the command design pattern)

Counterexample

Jr. Petru Florin Mihances

Agent classes are often placed in the analysis model of an application. During design time, **many agents are found to be irrelevant and should be removed**

Riel's Heuristic 3.10

On an object-oriented farm there is an object-oriented cow with some object-oriented milk. Should the object-oriented cow send the object-oriented milk the uncow yourself message, or should the object-oriented milk send the object-oriented cow the unmlk yourself message?

Meier Page-Jones (OOPSLA '87)

a book should send the bookshelf the book_yourself message or a bookshelf should send the book the shelf_yourself message?

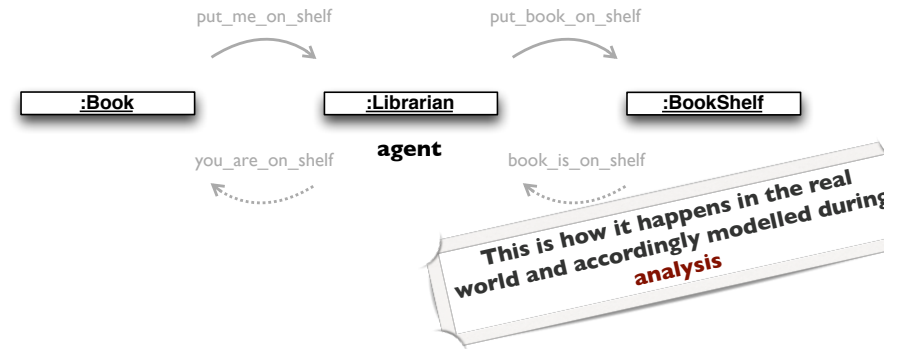
Riel - OO Design Heuristics

There is a key element missing, namely, the object-oriented **farmer**, and the object-oriented **librarian**. And these abstractions classes?

Jr. Petru Florin Mihances

Agent classes are often placed in the analysis model of an application. During design time, **many agents are found to be irrelevant and should be removed**

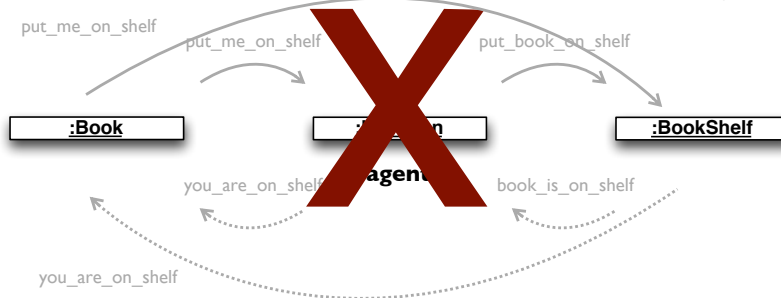
Riel's Heuristic 3.10



Jr. Petru Florin Mihances

Agent classes are often placed in the analysis model of an application. During design time, **many agents are found to be irrelevant and should be removed**

Riel's Heuristic 3.10



But of what use is the object-oriented librarian ?

If it just takes messages from one object and passes them to the other then it is irrelevant

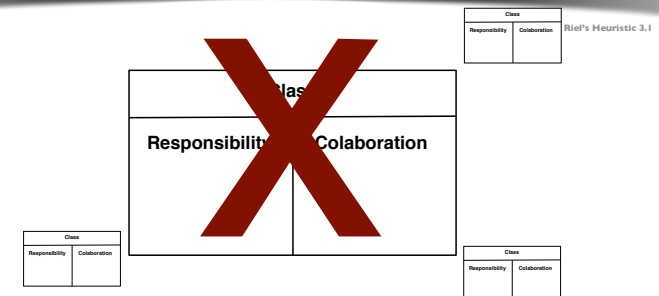
This decision is taken at **design** time. If additional behavior exists in the agent (e.g., check due date, sending fine notices) then it should be kept

Jr. Petru Florin Mihances

b

The intelligence **uniform distribution** problem

Distribute system intelligence horizontally as uniform as possible [...] classes in a design **should share the work uniformly**



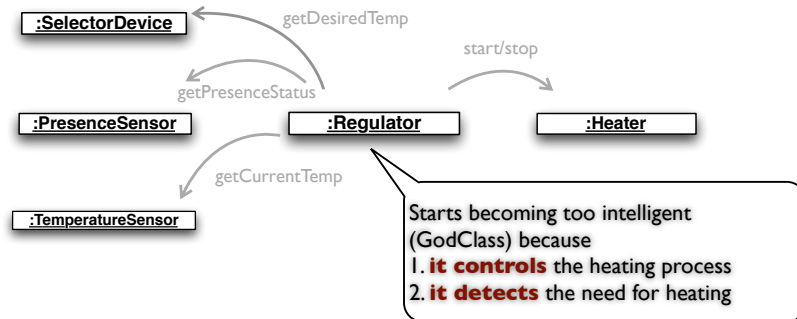
Riel's Heuristic 3.1

Jr. Petru Florin Mihances

Example

Room Temperature Regulator

- Temperature Sensor
- Temperature Selector Device
- Presence Sensor
 - Temperature can decrease up to 5 degree below desired if the room is empty

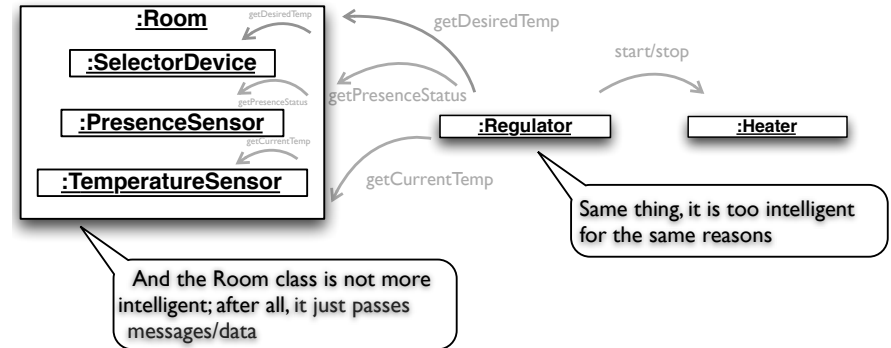


Dr. Petru Florin Mihances

Example

Room Temperature Regulator

- Temperature Sensor
- Temperature Selector Device
- Presence Sensor
 - Temperature can decrease up to 5 degree below desired if the room is empty

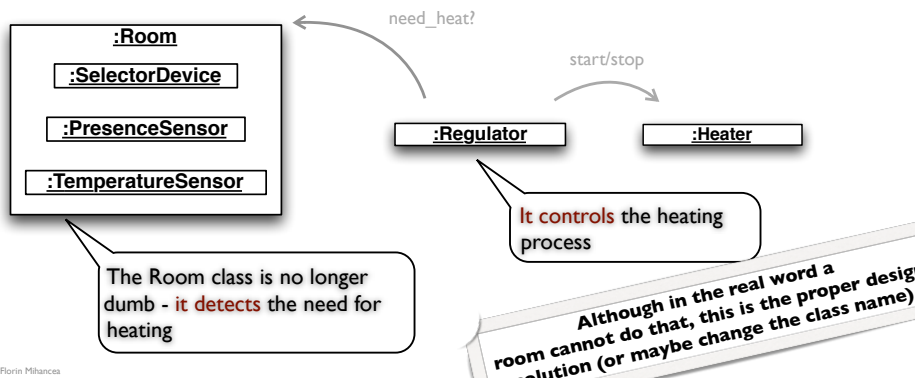


Dr. Petru Florin Mihances

Correct the Example

Let the **Room deciding** the need for heat

- it has all the necessary data



Dr. Petru Florin Mihances



All data **should be hidden** (private) within its class

Why ?

Riel's Heuristic 2.1

```

class A {
    ...
    public void method1(ComplexNumber n) {
        ... // computation based on n.x, n.y
    }
    ...
}

```

```

class D {
    ...
    public void method4(ComplexNumber n) {
        ... // computation based on n.x, n.y
    }
    ...
}

```

```

class ComplexNumber {
    public double x,y;
}

```

```

class B {
    ...
    public void method2(ComplexNumber n) {
        ... // computation based on n.x, n.y
    }
    ...
}

```

```

class C {
    ...
    public void method3(ComplexNumber n) {
        ...
    }
    ...
}

```

For some reason, we decide to change the number representation using its module and angle

Dr. Petru Florin Mihances

C

All data **should be hidden (private)
within its class**

Why ?

Riel's Heuristic 2.1

```
class A {
...
public void method1(ComplexNumber n) {
... // computation based on n.x, n.y
}
...
}
```

```
class D {
...
public void method4(ComplexNumber n) {
... // computation based on n.x, n.y
}
...
}
```

```
class ComplexNumber {
public double module, angle;
}
```

```
class B {
...
public void method2(ComplexNumber n) {
... // computation based on n.x, n.y
}
...
}
```

```
class C {
...
public void method3(ComplexNumber n) {
... // computation based on n.x, n.y
}
...
}
```

**1. Changing the names is
easy (Rename Refactoring)**

© Petru Florin Mihances

C

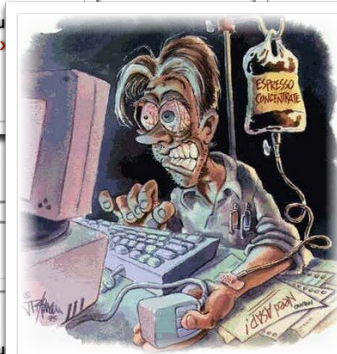
All data **should be hidden (private)
within its class**

Why ?

Riel's Heuristic 2.1

```
class A {
...
public void method1(ComplexNumber n) {
... // computation based on n.x, n.y
}
...
}
```

```
class D {
...
public void method4(ComplexNumber n) {
... // computation based on n.x, n.y
}
...
}
```



```
class B {
...
public void method2(ComplexNumber n) {
... // computation based on n.x, n.y
}
...
}
```

```
class C {
...
public void method3(ComplexNumber n) {
... // computation based on n.x, n.y
}
...
}
```

2. But the data **semantics
changed thus, we must
modify **all the clients** !!!**

© Petru Florin Mihances

C

All data **should be hidden (private)
within its class**

Why ?

Riel's Heuristic 2.1

```
class A {
...
public void method1(ComplexNumber n) {
... // computation based on n.getX/Y()
}
...
}
```

```
class D {
...
public void method4(ComplexNumber n) {
... // computation based on n.getX/Y()
}
...
}
```

```
class ComplexNumber {
private double module, angle; //+ init methods
public double getX() {return module * Math.cos(angle);}
public double getY() {return module * Math.sin(angle);}
}
```

```
class B {
...
public void method2(ComplexNumber n) {
... // computation based on n.getX/Y()
}
...
}
```

**We can limit the modifications
at the class level
(and instantiation code)
Note: We should also bring the
common behaviour
implemented in clients to this
class (bring data and
operations together e.g.,
addition of complex number:**

© Petru Florin Mihances

d

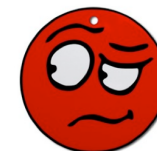
Open-Closed Principle

Software entities (e.g. classes,
methods) should be **open for
extensions** but **closed for modifications**

open for extensions
to be able to **extend their behavior**
closed for modifications
... **but without modifying their code**

Bertrand Meyer
(restated by Robert Martin)

yeah, sure ...



https://www.dailymotion.com/

© Petru Florin Mihances

Example

A program working with geometrical figures

```
class Painter {
    public void drawAll(Object[] figs) {
        for(Object aFig : figs) {
            if(aFig instanceof Circle) {
                ((Circle)aFig).drawCircle();
            } else {
                ((Square)aFig).drawSquare();
            }
        }
    }
}
```

Circle	Square
...	...
+drawCircle() : void	+drawSquare() : void

Jr. Petru Florin Mihances

Example

A program working with geometrical figures

```
class Painter {
    public void drawAll(Object[] figs) {
        for(Object aFig : figs) {
            if(aFig instanceof Circle) {
                ((Circle)aFig).drawCircle();
            } else {
                ((Square)aFig).drawSquare();
            }
        }
    }
}
```

Circle	Square
...	...
+drawCircle() : void	+drawSquare() : void

After a period of time, we must also add triangles

Jr. Petru Florin Mihances

Example

A program working with geometrical figures

```
class Painter {
    public void drawAll(Object[] figs) {
        for(Object aFig : figs) {
            if(aFig instanceof Circle) {
                ((Circle)aFig).drawCircle();
            } else {
                ((Square)aFig).drawSquare();
            }
        }
    }
}
```

**Runtime error -
ClassCastException !!! The
compiler cannot help you**

Circle	Square	Triangle
...
+drawCircle() : void	+drawSquare() : void	+drawTriangle() : void

1. Add the corresponding class

After a period of time, we must also add triangles

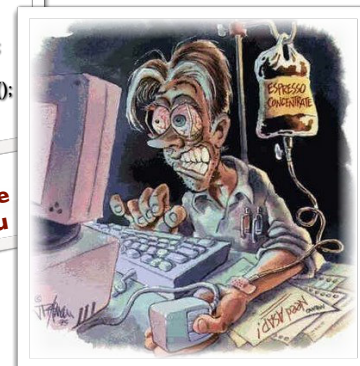
Jr. Petru Florin Mihances

Example

A program working with geometrical figures

```
class Painter {
    public void drawAll(Object[] figs) {
        for(Object aFig : figs) {
            if(aFig instanceof Circle) {
                ((Circle)aFig).drawCircle();
            } else {
                ((Square)aFig).drawSquare();
            }
        }
    }
}
```

**Runtime error -
ClassCastException !!! The
compiler cannot help you**



Triangle
...
+drawTriangle() : void

1. Add the corresponding class

After a period of time, we must also add triangles

2. In all places where we distinguished between several types of figures we should add an additional **if-instanceof-else**

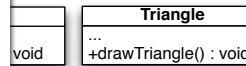
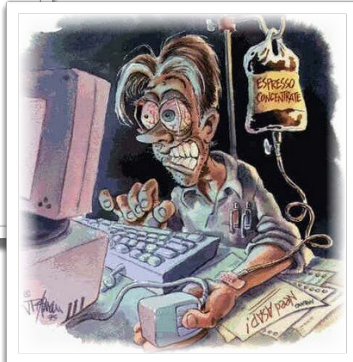
Painter class does not comply to OCP with respect to the addition of new types of figures

Jr. Petru Florin Mihances

Example

A program working with geometrical figures

```
class Painter {
    public void drawAll(Object[] figs) {
        for(Object aFig : figs) {
            if(aFig instanceof Circle) {
                ((Circle)aFig).drawCircle();
            } else if(aFig instanceof Square) {
                ((Square)aFig).drawSquare();
            } else {
                ((Triangle)aFig).drawTriangle();
            }
        }
    }
}
```



After a period of time, we must also add triangles

1. Add the corresponding class

2. In all places where we distinguished between several types of figures we should add an additional **if-instanceof-else**

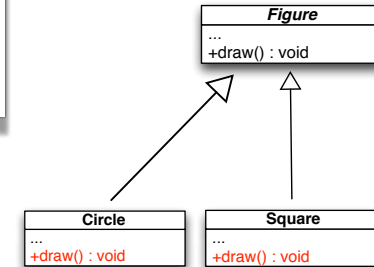
Painter class does not comply to OCP with respect to the addition of new types of figures

Dr. Petru Florin Mihances

Example

A program working with geometrical figures

```
class Painter {
    public void drawAll(Figure[] figs) {
        for(Figure aFig : figs) {
            aFig.draw();
        }
    }
}
```

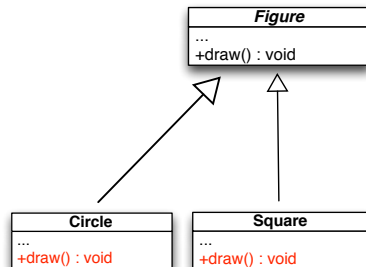


Dr. Petru Florin Mihances

Example

A program working with geometrical figures

```
class Painter {
    public void drawAll(Figure[] figs) {
        for(Figure aFig : figs) {
            aFig.draw();
        }
    }
}
```



After a period of time, we must also add triangles

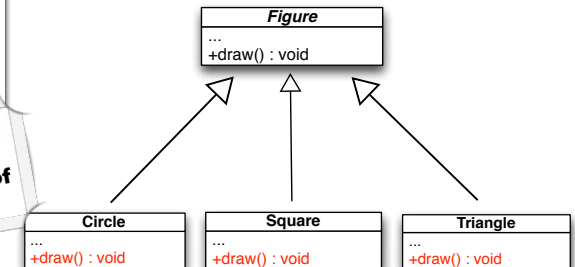
Dr. Petru Florin Mihances

Example

A program working with geometrical figures

```
class Painter {
    public void drawAll(Figure[] figs) {
        for(Figure aFig : figs) {
            aFig.draw();
        }
    }
}
```

Can also draw triangles; due to dynamic binding the corresponding implementation of the draw operation is invoked



After a period of time, we must also add triangles

1. Add the corresponding class and ... done!

Painter class complies to OCP with respect to the addition of new types of figures

Dr. Petru Florin Mihances



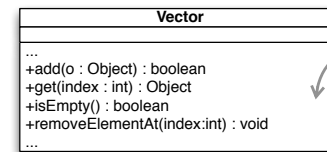
Use Inheritance **Correctly**

Do not use inheritance just to reuse the code of a superclass
[it should also be used for polymorphism]

Favour object composition instead of class inheritance

Jr. Petru Florin Mihances

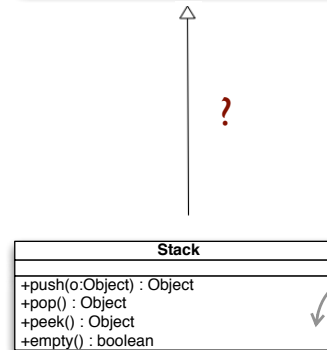
Case Study



“Simulates” an array whose capacity can change.
We assume it is already implemented.

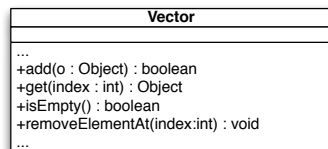
```
class Stack extends Vector {
    public Object push(Object o) {
        this.add(o);
        return o;
    }
    public Object pop() {
        Object r = this.get(this.size() - 1);
        this.removeElementAt(this.size() - 1);
        return r;
    }
    ...
}
```

LIFO (Last-In-First-Out) data structure
with usual operations:
push - adds on top of the stack
pop - extracts the element from the top
We must **implement it** ... what can we do ?

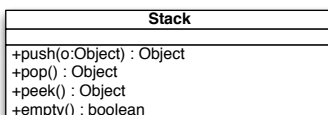


Jr. Petru Florin Mihances

Case Study (2)



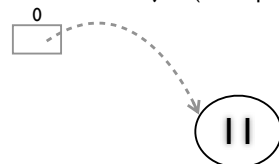
?



```
class Main {
    public static void main(String args[]) {
        Stack stk = new Stack();
        stk.push(new Integer(5));
        stk.push(new Integer(10));
        Object p = stk.pop();
        System.out.println(p);

        stk.add(new Integer(11));
        // with add (?) what does it mean for a stack ?
        stk.removeElementAt(0);
        // what (???) in a stack I should be able to access only
        //the top element
    }
}
```

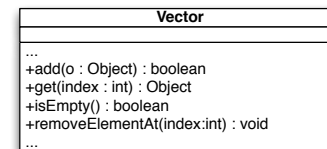
In the stack object (the top is at the right)



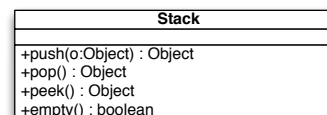
OUTPUT
10

Jr. Petru Florin Mihances

Case Study (2)



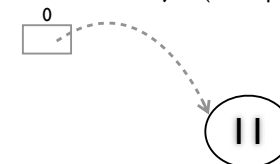
?



```
class Main {
    public static void main(String args[]) {
        Stack stk = new Stack();
        stk.push(new Integer(5));
        stk.push(new Integer(10));
        Object p = stk.pop();
        System.out.println(p);
    }
}
```

Operations that do not characterise a stack can be invoked on a stack object ?????

In the stack object (the top is at the right)



OUTPUT
10

Jr. Petru Florin Mihances

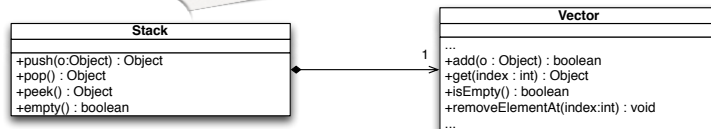
The Right Way

```
class Stack {
    private Vector v = new Vector();
    public Object push(Object o) {
        v.add(o);
        return o;
    }
    public Object pop() {
        Object r = v.get(v.size() - 1);
        v.removeElementAt(v.size() - 1);
        return r;
    }
}
```

```
class Main {
    public static void main(String args[]) {
        Stack stk = new Stack();
        stk.push(new Integer(5));
        stk.push(new Integer(10));
        Object p = stk.pop();
        System.out.println(p);

        stk.add(new Integer(11)); // compile error
        stk.removeElementAt(0); // compile error
    }
}
```

... additionally, it might be possible to change the Vector object with an instance of one of its subclasses even at runtime



Jr. Petru Florin Mihances

NEVER do that ...

```
class Stack extends Vector {
    public Object push(Object o) {
        this.add(o);
        return o;
    }
    public Object pop() {
        Object r = this.get(this.size() - 1);
        this.removeElementAt(this.size() - 1);
        return r;
    }
    public void removeElementAt(int index) {
        //Overriding
        throw new RuntimeException("I don't know how :(");
    }
}
```

```
public class Client {
    public static void doSomething(Vector v) {
        if(v.size() > 0) {
            System.out.println(v.get(0));
            v.removeElementAt(0);
        }
    }
}
```

```
public static void main(String argv[]) {
    Vector v = new Vector();
    v.add(new Integer(5));
    Client.doSomething(v);
}
```

OUTPUT
5

Jr. Petru Florin Mihances

NEVER do that ...

```
public class Client {
    public static void doSomething(Vector v) {
        if(v.size() > 0) {
            System.out.println(v.get(0));
            v.removeElementAt(0);
        }
    }
}
```

```
class Stack extends Vector {
    public Object push(Object o) {
        this.add(o);
        return o;
    }
    public Object pop() {
        Object r = this.get(this.size() - 1);
        this.removeElementAt(this.size() - 1);
        return r;
    }
    public void removeElementAt(int index) {
        //Overriding
        throw new RuntimeException("I don't know how :(");
    }
}
```

```
public static void main(String argv[]) {
    Stack s = new Stack();
    s.push(new Integer(5));
    Client.doSomething(s);
}
```

OUTPUT

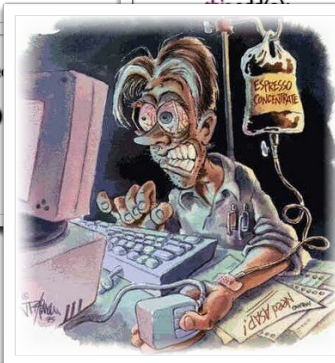
```
Exception in thread "main" java.lang.RuntimeException: I don't know how :(
at Stack.removeElementAt(Stack.java:13)
at Client.doSomething(Client.java:7)
at Client.main(Client.java:18)
```

Jr. Petru Florin Mihances

NEVER do that ...

```
class Stack extends Vector {
    public Object push(Object o) {
        this.add(o);
        return o;
    }
    public Object pop() {
        Object r = this.get(this.size() - 1);
        this.removeElementAt(this.size() - 1);
        return r;
    }
    public void removeElementAt(int index) {
        //Overriding
        throw new RuntimeException("I don't know how :(");
    }
}
```

```
public class Client {
    public static void doSomething(Vector v) {
        if(v.size() > 0) {
            System.out.println(v.get(0));
            v.removeElementAt(0);
        }
    }
}
```



```
public static void main(String argv[]) {
    Stack s = new Stack();
    s.push(new Integer(5));
    Client.doSomething(s);
}
```

OUTPUT

```
Exception in thread "main" java.lang.RuntimeException: I don't know how :(
at Stack.removeElementAt(Stack.java:13)
at Client.doSomething(Client.java:7)
at Client.main(Client.java:18)
```

You will learn about the **Liskov Substitution Principle** next semester

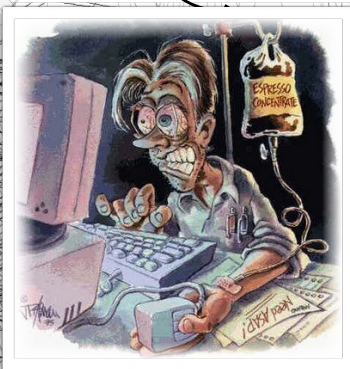
Jr. Petru Florin Mihances



Would you try to **modify** this system?

Dependency

Law of Demeter can help you reducing the number of dependencies



Module

Too many dependencies!

Jr. Petru Florin Mihances

Law of Demeter

The law of Demeter for functions states that any method of an object should call only methods belonging to :

```
class Demeter { //Java
    private A a = new A();
    public void example(B b) {
        //this / super
        this.aMethod();
        //received parameters
        b.executeSomething();
        //objects created in the method or
        //received from a method that creates
        //the objects
        new A().exec();
        //members of the method class
        a.exec();
    }
    ...
}
```

Various forms exist

Jr. Petru Florin Mihances

Example and Correction

```
class Carburetor {
    private boolean openFuelValve = false;
    public void setOpenFuelValve(boolean b) {
        openFuelValve = b;
    }
    ...
}
```

```
class Engine {
    private Carburetor c = new Carburetor();
    public Carburetor getCarburetor() {
        return c;
    }
    ...
}
```

S. Demeyer, S. Ducasse, O. Nierstratz - OO Reengineering Patterns

```
class Car {
    private Engine e = new Engine();
    public void increaseSpeed() {
        e.getCarburetor().setOpenFuelValve(true);
    }
    ...
}
```

Jr. Petru Florin Mihances

Example and Correction

```
class Carburetor {
    private boolean openFuelValve = false;
    public void setOpenFuelValve(boolean b) {
        openFuelValve = b;
    }
    ...
}
```

```
class Engine {
    private Carburetor c = new Carburetor();
    public void speedUp() {
        c.setOpenFuelValve(true);
    }
    ...
}
```

S. Demeyer, S. Ducasse, O. Nierstratz - OO Reengineering Patterns

```
class Car {
    private Engine e = new Engine();
    public void increaseSpeed() {
        e.speedUp();
    }
    ...
}
```

But do not be paranoid (e.g. you can violate the law for optimisation purposes)

Reduces the number of dependencies, reduces the number of get's

Jr. Petru Florin Mihances