Software Engineering Fundamentals

Object-Oriented Analysis

Dr. Petru Florin Mihancea

Based on:
R. Pressman - Software Engineering, Ch.20 - Object-Oriented Concepts and Principles Ch. 21 - Object-Oriented Analysis
D. Rubin - Introduction to CRC Cards
K. Beck, W. Cunningham - A Laboratory for Teaching Object-Oriented Thinking
A. Riel - Object-Oriented Design Heuristics, Ch.2 - Classes and Objects, Ch.3
-Topologies of AO vs. OO Applications

V20240423

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

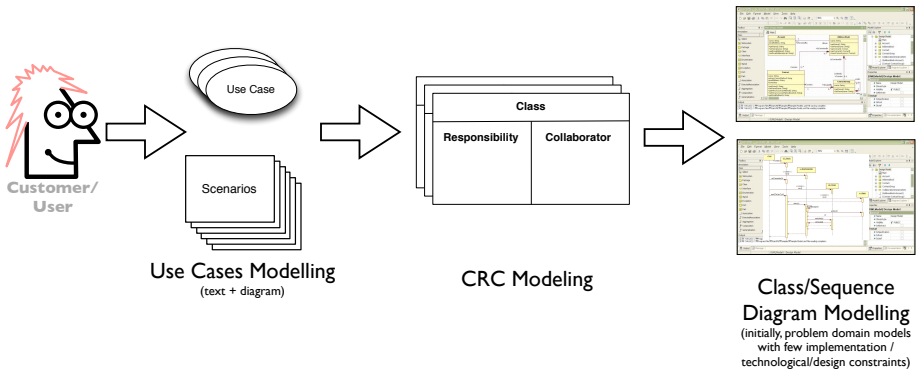
Important questions

What are the relevant objects in the problem to be solved ?

How do they relate and interact with one another ?

How does an object behave ?

Process Overview



Use Cases Modelling
(text + diagram)

CRC Modeling

Class/Sequence Diagram Modelling
(initially, problem domain models with few implementation / technological/design constraints)

1

Class-Responsibility-Collaborator (CRC) Modeling

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 :)

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)

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)**

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

CRC Modelling Session (3)

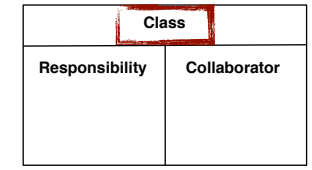
2. "Execute" the scenario and identify classes, responsibilities and collaborators

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



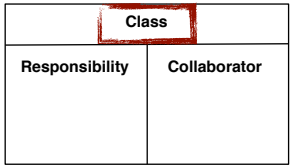
Classes

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



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)

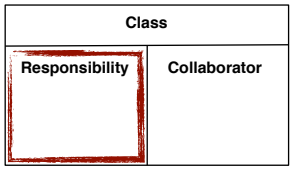
events (e.g., a completion/occurrence of an action)

roles (e.g., manager, engineer)

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

places (e.g., classroom)

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



Responsibility

What are them and how to identify them ?

Anything a class knows or does

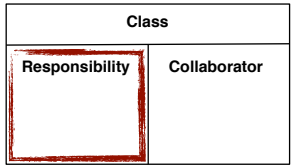
attributes and operations

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



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

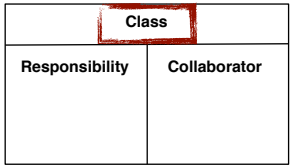
What do they usually do ?

a. computations

b. data manipulation (e.g., add, remove)

c. query (e.g., about the state of the object)

d. monitor an object (e.g., the occurrence of an event)



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

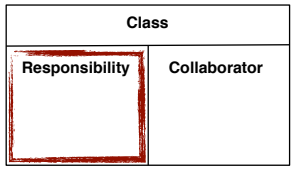
an object with a single attribute should be represented as an attribute of another object during analysis to enable the focus on "major" information

4. Common attributes and operations

attributes and operations are common to all occurrences of an object

5. Essential requirements

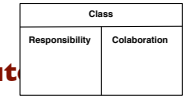
entities that produce or consume information essential to the operation of any solution

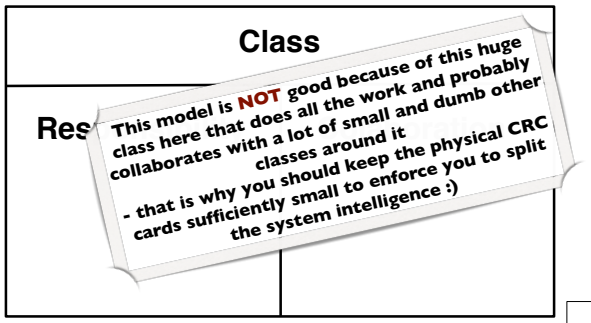


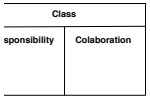
Responsibility

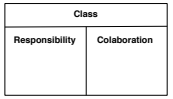
Guidelines

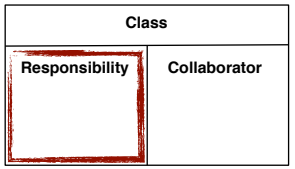
1. System intelligence should be evenly distribut











Responsibility

Guidelines

1. System intelligence should be evenly distributed

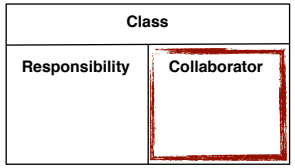
2. Information about one thing should be localized in a single class not distributed across multiple classes

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



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

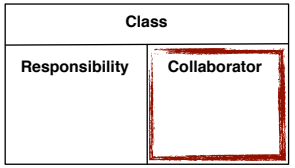
Finding collaborators

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

...



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 :)

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 also the sequence diagrams :)

2

Modelling Heuristics

A

The class proliferation problem

How to (properly) reduce the number of classes ?

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**

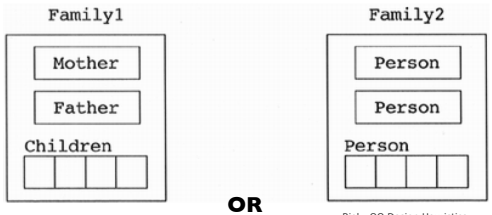
Eliminating such classes will probably imply demoting them to attributes

Special cases for get/set operations

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

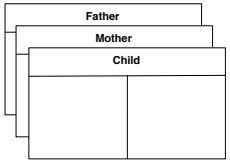
Be sure the abstractions that you model are classes and not simply the roles objects play

Riel's Heuristic 2.11

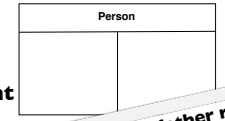


Riel - OO Design Heuristics

OR



If different behaviour then different classes
e.g., go_into_labor()



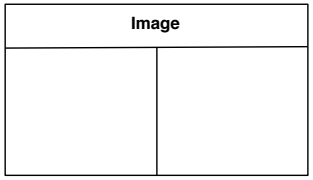
Otherwise, we have different roles of the same class
e.g., change_diapers()

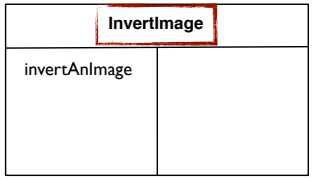
Be careful: a person playing the father role must be able to change_diapers() even if only a mother role does that! Otherwise we are again in the first case

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





Constructive Joke

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

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

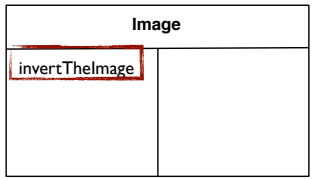
0

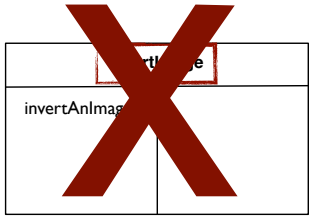
... because an object-oriented developer will teach the light bulb to screw itself !

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

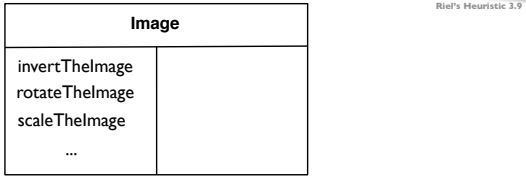


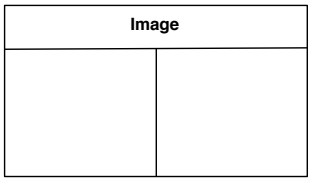


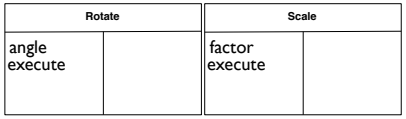
Do not turn an operation into a class.

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







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

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 unmilk yourself message?

Meiler Page-Jones (OOPSLA '87)

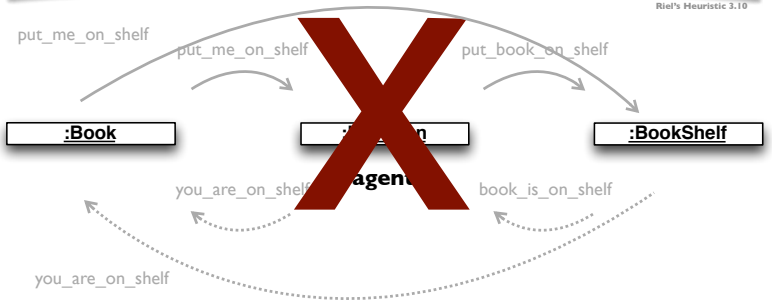
a book should send the bookshelf the book_yourself message or a bookshelf should send the book the shelf_yourself message?

Riel - OO Design Heuristi

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

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



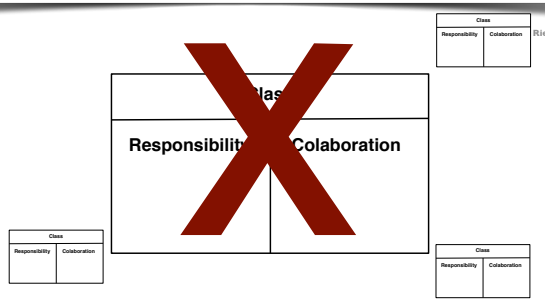
But of what use is the object-oriented librarian ?

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

3 The intelligence uniform distribution problem

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

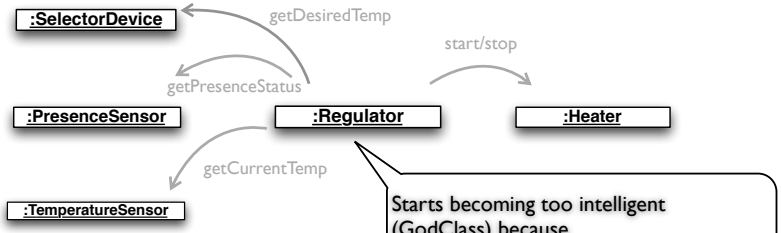


Riel's Heuristic 3.1

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**

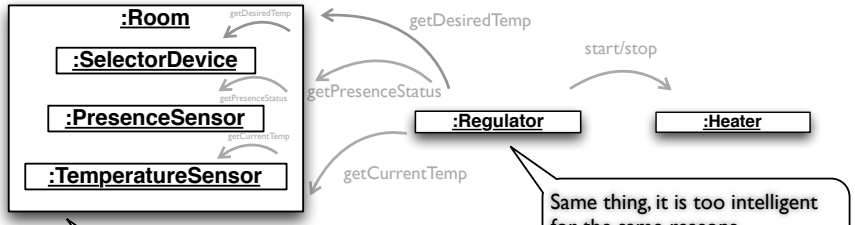


Starts becoming too intelligent (GodClass) because
1. **it controls** the heating process
2. **it detects** the need for heating

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**



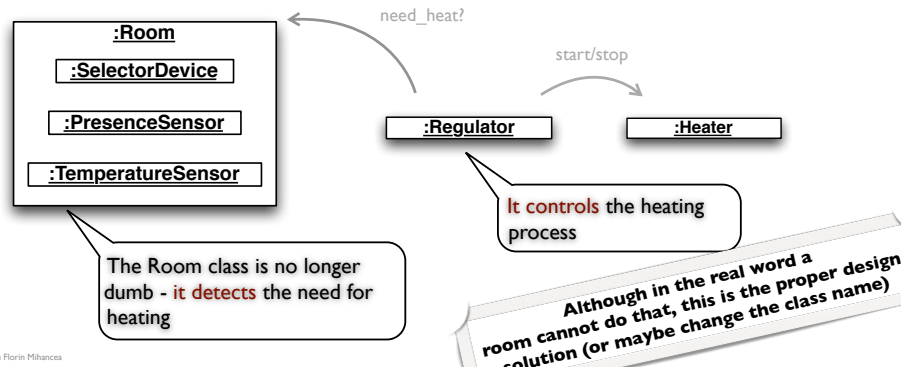
And the Room class is not more intelligent; after all, it just passes messages/data

Same thing, it is too intelligent for the same reasons

Correct the Example

Let the Room deciding the need for heat

- **it has all the necessary data**



3

UML Class and Sequence Diagrams

Unified Modeling Language

Family of graphical notations

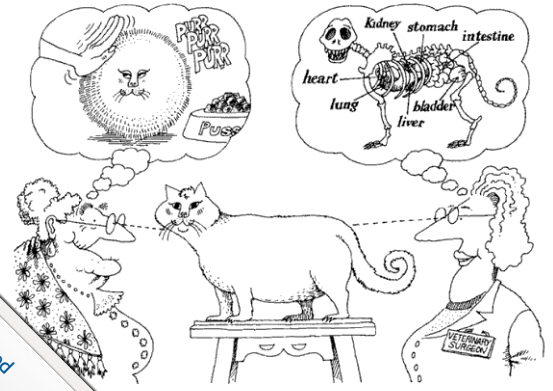
for modeling an (OO) system

Goal
Learn the notations

Types of UML models

Behavioral
e.g. Sequence diagram (SD)

Structural
e.g. Class diagram (CD)

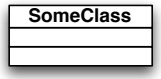


Booch - OO Analysis and Design

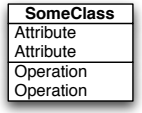
Goal Refined
Learn the notations for
CD and **SD**

1 Class diagram

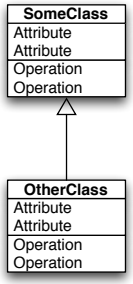
Structural model



Classes

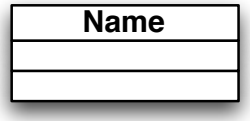


Features



Relations

1 Class diagram

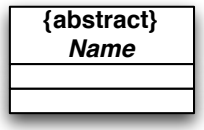



```
class Name {

}
```

UML sketch

Java sketch

1 Class diagram

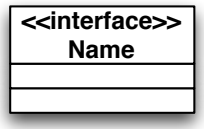



```
abstract class Name {

}
```

UML sketch

Java sketch

1 Class diagram

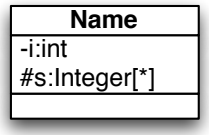



```
interface Name {

}
```

UML sketch

Java sketch

1 Class diagram
Attributes



visibility name : type multiplicity
= implicitValue

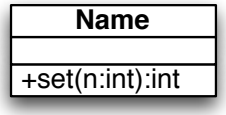
1 - exactly one
0..1 - zero or at most one
0..* or * - zero or more but
NO upper limit

```
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)
}
```

UML sketch Java sketch

1 Class diagram
Operations



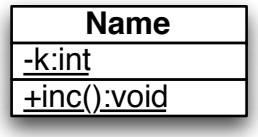
visibility name(param_list) : ret_type

direction name : type = default

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

UML sketch Java sketch

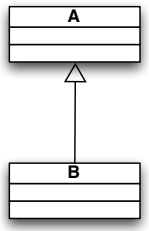
1 Class diagram
Scope




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

UML sketch Java sketch

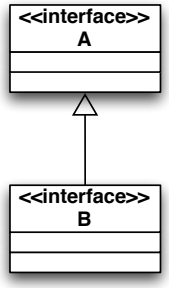
1 Class diagram
Generalisation & Realisation




```
class A {
}

class B extends A {
}
```

UML sketch Java sketch

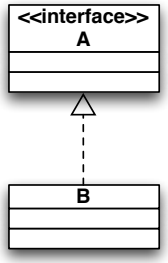
1 Class diagram
Generalisation & Realisation




```
interface A {
}

interface B extends A {
}
```

UML sketch Java sketch

1 Class diagram
Generalisation & Realisation

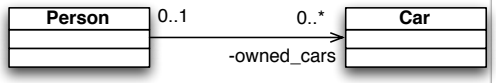



```
interface A {
}

class B implements A {
}
```

UML sketch Java sketch

Dr. Petru Florin Mihancea

1 Class diagram
Association

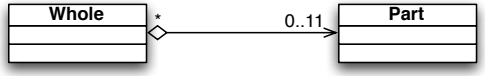



```
class Person {

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

    //add, remove methods usually exist

}
```

UML sketch Java sketch

Dr. Petru Florin Mihancea

1 Class diagram
Aggregation

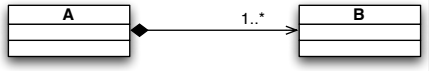


Similar to association

UML sketch Java sketch

Dr. Petru Florin Mihancea

1 Class diagram
Composition

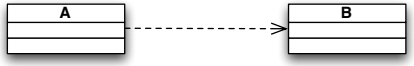


I. No-sharing
II. B objects cannot exists without their A object

```
class  A {

   private List<B> my_list =
      new ...

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

UML sketch Java sketch

1 Class diagram
Dependency

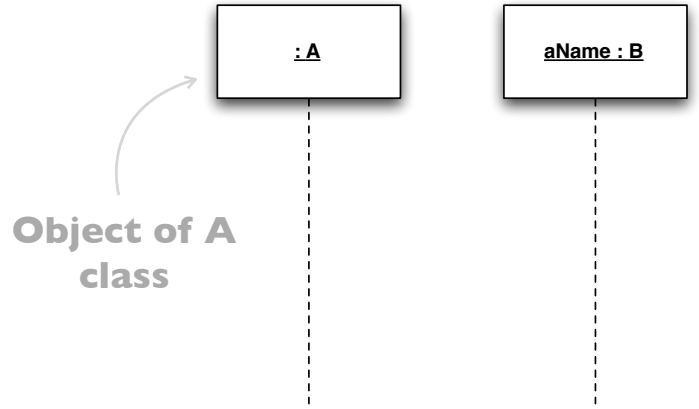



```
class A {

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

}
```

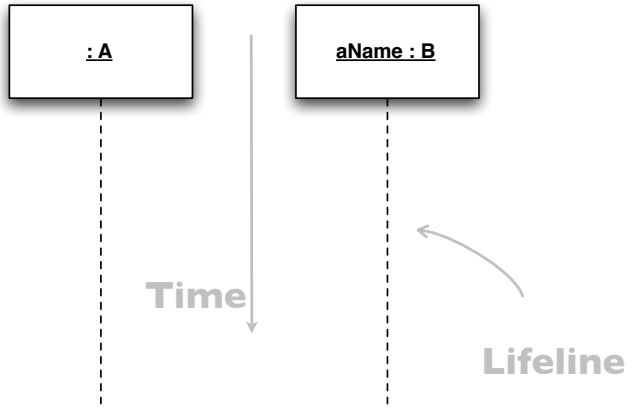
And many other cases ...

UML sketch Java sketch

2 Sequence diagram
Behavioural & Interaction model

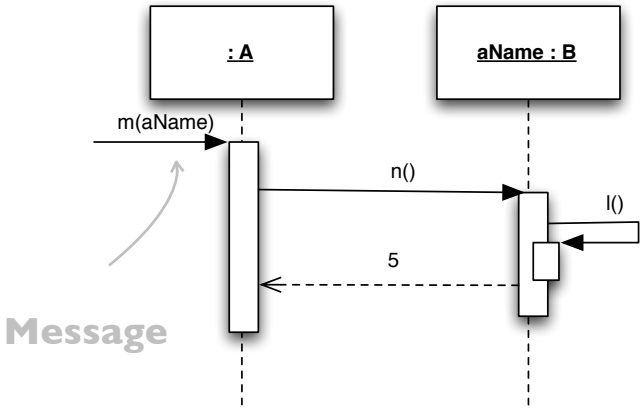


2 Sequence diagram
Behavioural & Interaction model



2 Sequence diagram

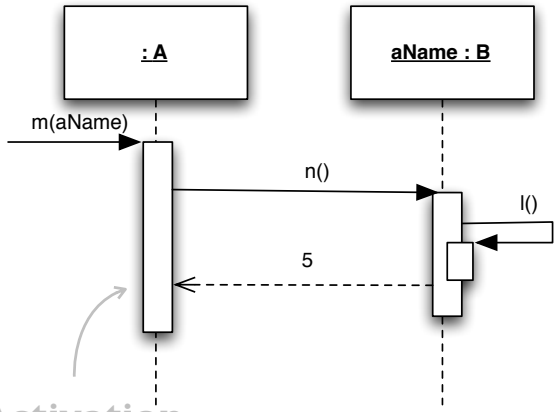
Behavioural & Interaction model



Message

2 Sequence diagram

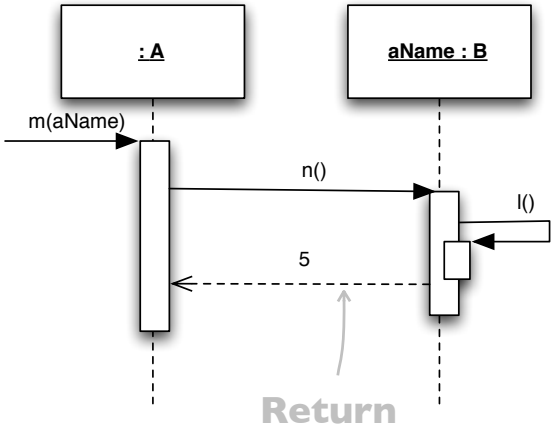
Behavioural & Interaction model



Activation

2 Sequence diagram

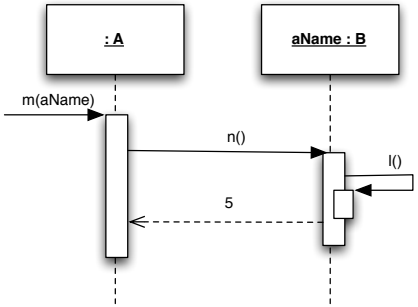
Behavioural & Interaction model



Return

2 Sequence diagram

Behavioural & Interaction model



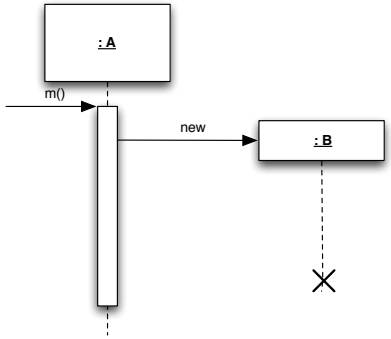
UML sketch

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

Java sketch

2 Sequence diagram

Object creation & deletion



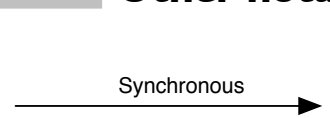
UML sketch

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

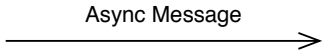
Java sketch

2 Sequence diagram

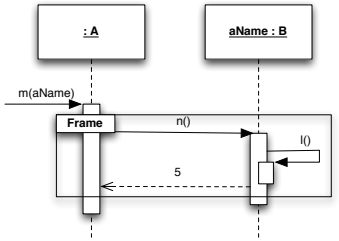
Other notations



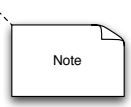
e.g. method invocation



e.g. start an execution thread



e.g. loops, conditions



Comments