A PROGRAM OR TO PROGRAM?

AARHUS
UNIVERSITET
INSTITUT FOR ELEKTRO OG COMPUTERTEKNOLOGI

JENS BENNEDSEN
INGENIØRDOCENT

Me

- **30+ years of teaching experience**
  - introductory programming for 30+ years
  - introductory object-oriented programming for 25+ years

- **My current (research) focus**
  - Programming education
  - Curriculum development
  - Lifelong learning

Michael E. Caspersen
Managing Director, Honorary Professor, PhD

Three Perspectives on Programming

Instructing the computer (coding)

- the purpose of programming is to instruct the computer
- focus is on aspects of program execution such as storage layout, control flow, parameter passing, etc.

Managing the program description

- the purpose of programming is to create a software architecture that provides overview and understanding of the entire program
- focus is on aspects such as visibility, scope, encapsulation, modularity, software design etc.

Conceptual modeling

- the purpose of programming is to express concepts, structure and relations
- focus is on constructs for describing concepts, phenomena and relations between these

Characterization and conditions

- **Characterize the introductory programming course you know**
  - other views on programming?
  - how are the views balanced in the intro course?
  - what defines the progression in your intro course?

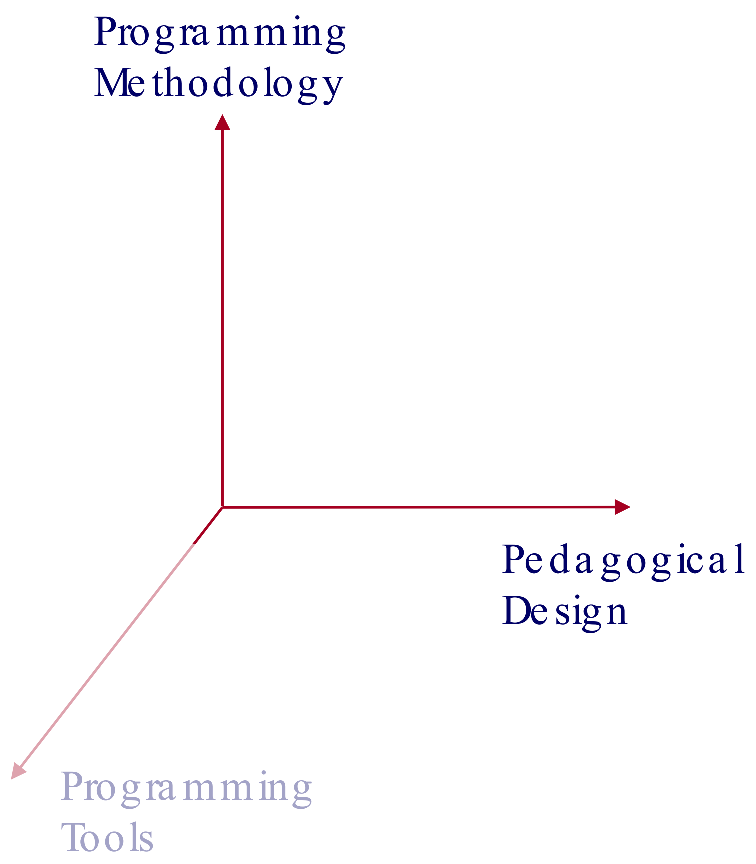
- **Bounding condition!**
  - The programming paradigm is object orientation

Programming Education in Perspective

Programming Education

Programming Education

= Programming Methodology +
Pedagogical Design +
Programming Tools



Programming Methodology

= Theory/ Techniques + Process

Pedagogical Design

= Organization + Dissemination

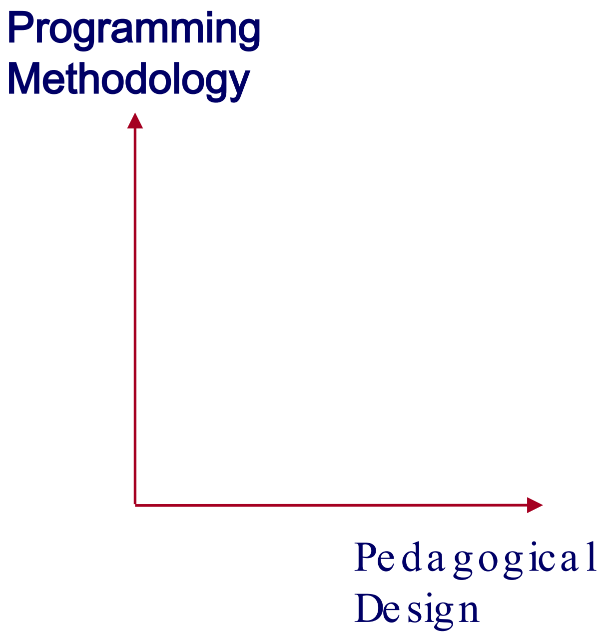
Programming Tools

= Language + Environment

Programming Methodology

= Theory/Techniques + Process

- **Theory/Techniques**
  - model-driven development
  - design by contract (assertions)
  - patterns
  - ...
- **Process**
  - incremental development
  - non-linearity
  - refactoring
  - test
  - ...

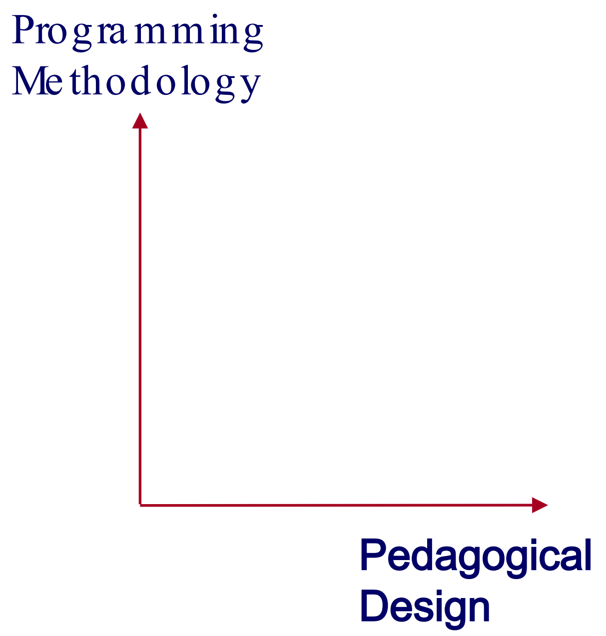


Pedagogical Design

= Organization + Dissemination

- **Organization**
  - graduated exposure to complexity
  - spiral, early bird, fill in the blanks
  - apprenticeship
  - ...

- **Dissemination**
  - text
  - labs
  - videos
  - lectures
  - net-based learning objects



A Conceptual Framework for Object-Oriented Programming

There is more to OO than Java/C++/...

Kristen Nygaard on Object-Orientation

A program execution is regarded as a physical model system simulating the behavior of either a real or imaginary part of the world.

Physical modeling is based upon the conception of reality in terms of phe-nomena and concepts.

A physical model system is construc-ted, modeling phenomena by objects and concepts by categories of objects.

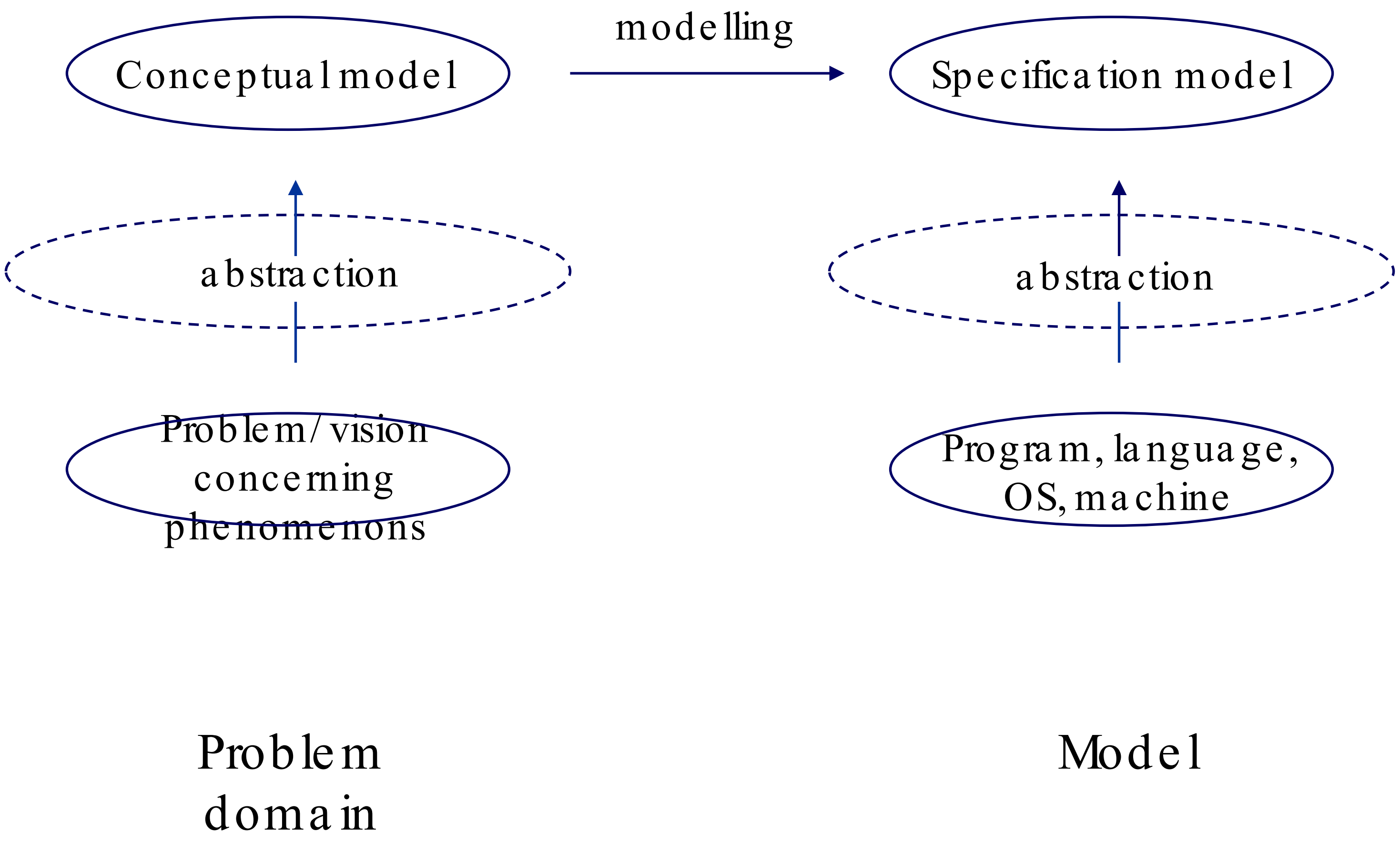
Kristen Nygaard, 1926-2002

Overview

- **Conceptual framework for object-orientation**
  - Concepts and modelling
  - Structure: aggregation, association, specialization
  - is used for organizing **knowledge about a problem domain** and **structure in the solution domain**
  - Is (to some extend) supported by language constructs in OO languages

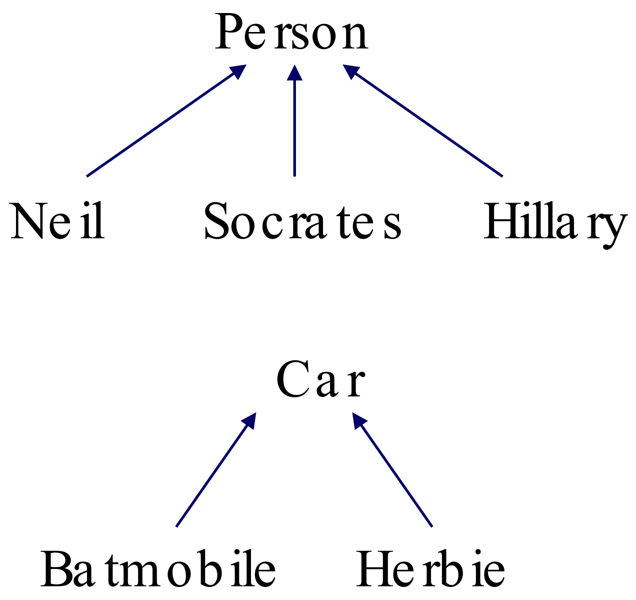
- **Modelling examples**
  - Abstract models in UML
  - Implementation in Java
  - Smaller examples from textbook [Barnes & Kölling] – which does not explicitly present the models...

Conceptual Modelling

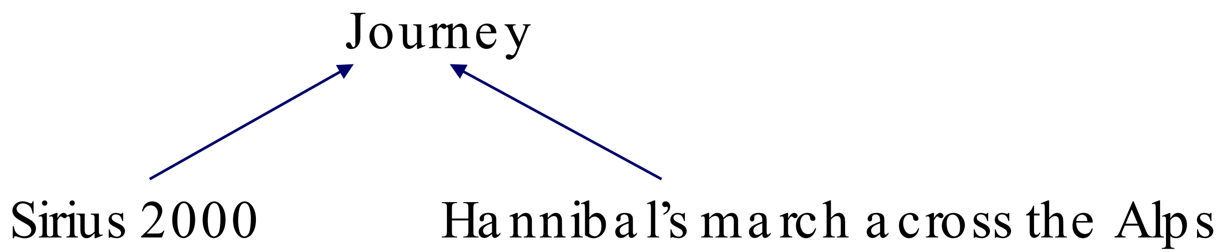


Concept Formation

- **Identification of phenomena**
  - Socrates
  - Batmobile
  - Hannibal’s march across the Alps
  - Neil Young
  - Sirius 2000
  - Herbie
  - Hillary Clinton



- **Classification**

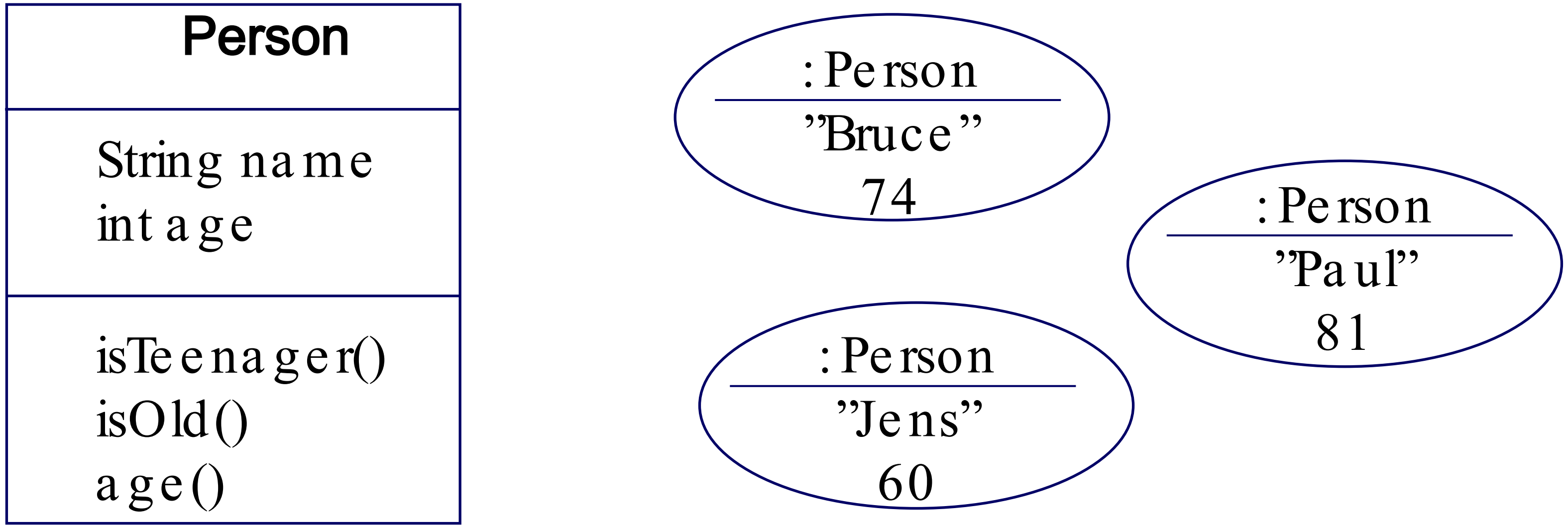


Classification in UML

Classes represent concepts,
objects represent phenomenons.

Example   Concept: Person
            Phenomenons: Bruce, Paul, Jens

Class   Objects



Classification in Java

```
class Person {
  private String name;
  private int age;

  public Person(String name, int age) {
    this.name = name;
    this.age = age;
  }

  public voi
    age++;
  }

  public isTe
    return (a
  }
}
```

```
List l = new ArrayList();

l.add( new Person("Bruce", 55) );
l.add( new Person("Paul", 62) );
l.add( new Person("Michael", 44) );
```

Relations between concepts

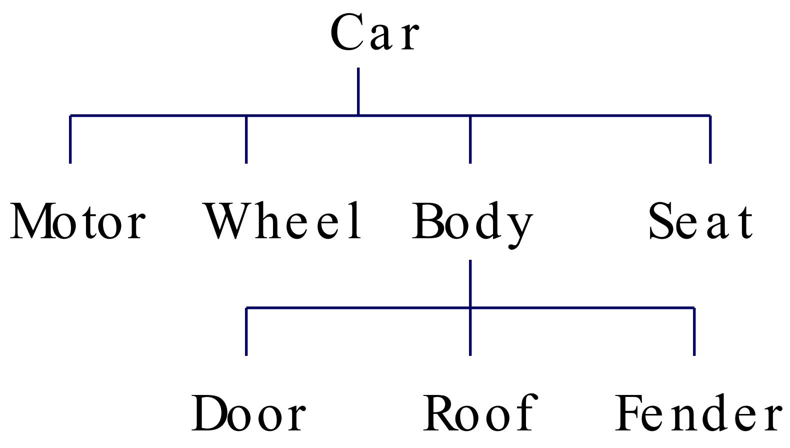
- **Aggregation**
  - has-a
- **Association**
  - X-a
- **Generalization/specialization**
  - is-a

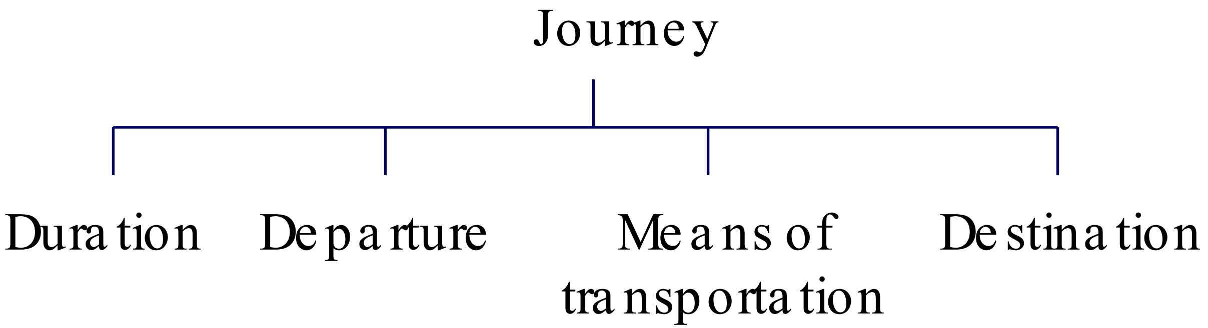
Organization of knowledge...

Aggregation (has-a)

Relation between concepts describing a whole and (some of) the parts of which constitutes the whole (part-whole structure).

UML: Composition

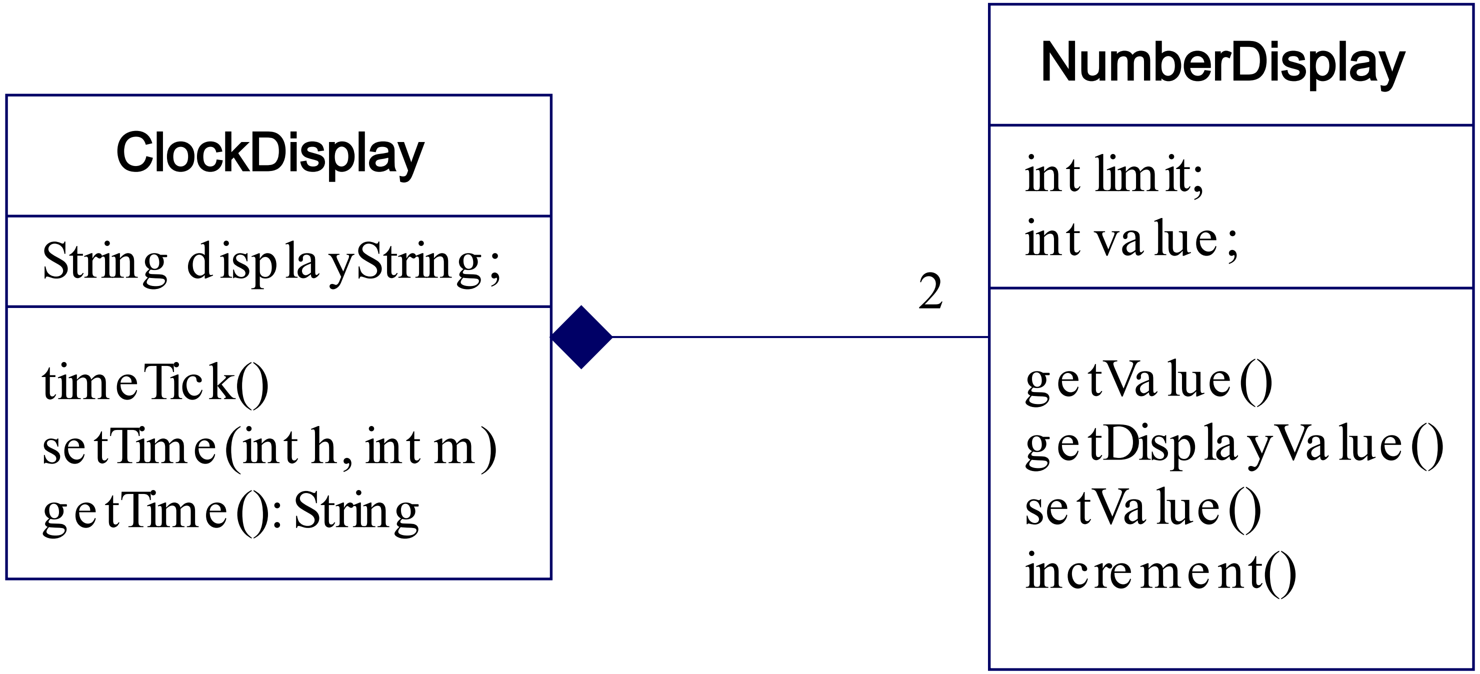




Aggregation in UML (1)

ClockDisplay
NumberDisplay hours; NumberDisplay minutes; String displayString;
timeTick() setTime(int h, int m) getTime(): String

Aggregation in UML (2)



Aggregation in Java

```
class NumberDisplay {
  private int limit, value;

  public NumberDisplay() { ... }
  public int getValue() { ... }
  public String getDisplayValue() { ... }
  pu
  pu
}
```

```
class ClockDisplay {
  private NumberDisplay hours;
  private NumberDisplay minutes;
  private String displayString;

  public ClockDisplay() {
    hours = new NumberDisplay(24);
    minutes = new NumberDisplay(60);
  }
  public void timeTick() { ... }
  public void setTime(int hour, int minutes) { ... }
  public String getTime() { ... }
}
```

Association (X-a)

Relation that describes a dynamic relation between concepts that can exist independently of each other.

MailServer **keeps** MailItem

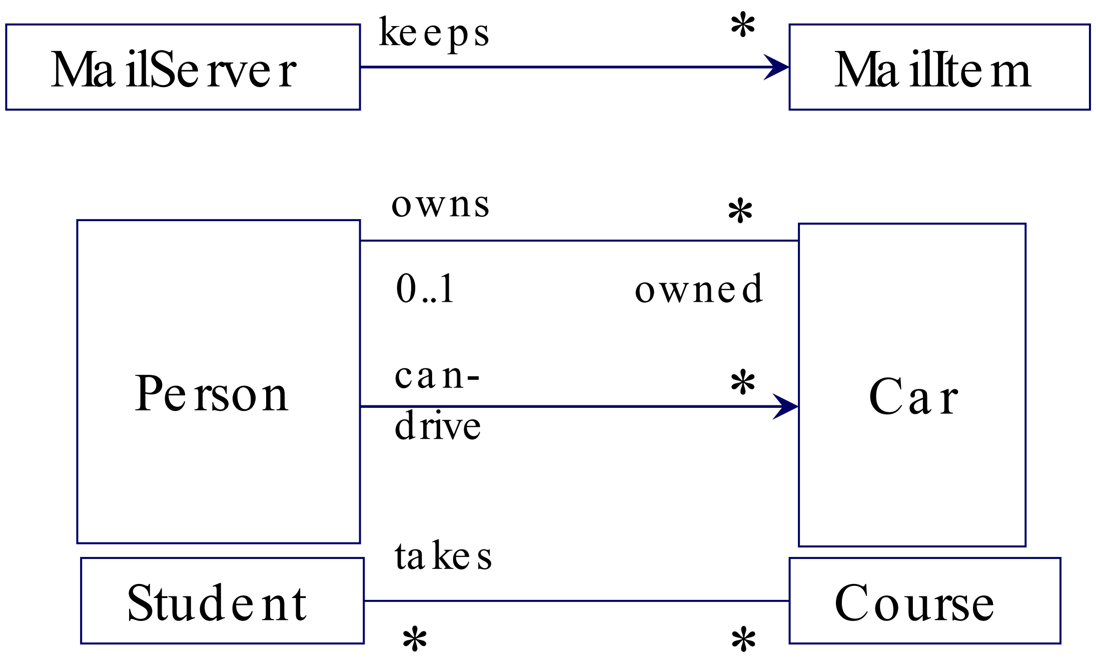
Person **owns** Car
Person **rents** Car

Person **loves** Person
Person **is-friend-with** Person

Student **is-enrolled-at** Course

Patient **have-had** Disease

Association in UML



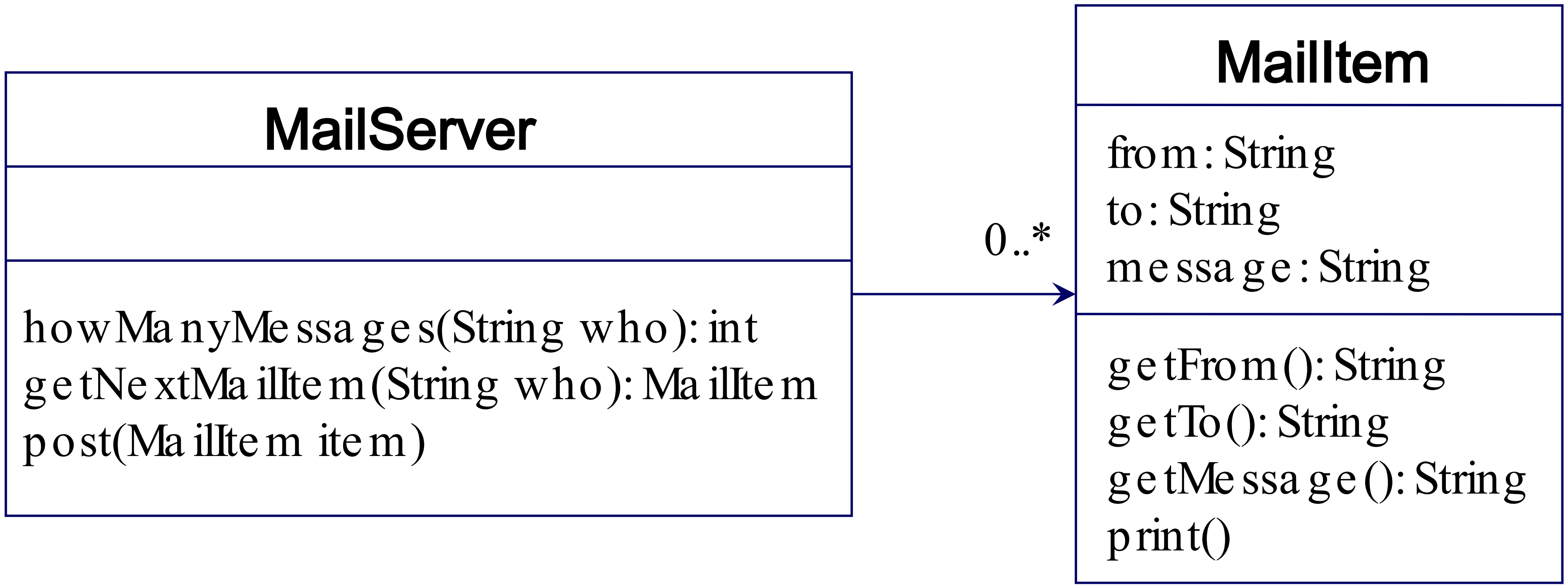
Multiplicity (cardinality): 0..1, 1, n,a..b, 0..* (*)

Role

Orientation (1-way, 2-way)

Association in UML (X-a)

X = keeps



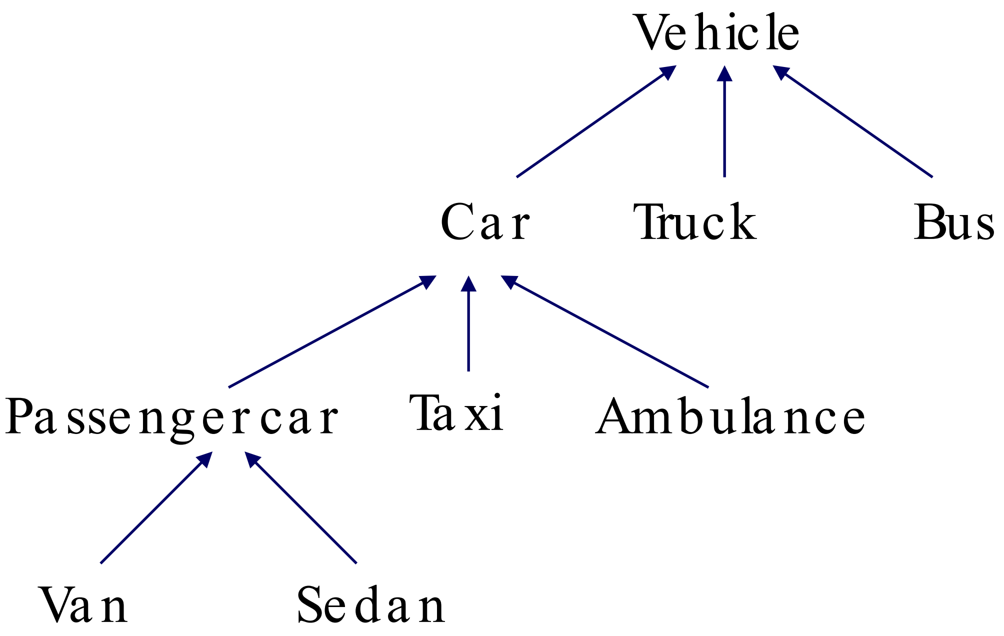
Association in Java

```
class MailServer {

  Set<MailItem> messages;

  public MailServer() {
    messages = new HashSet<MailItem>();
  }

  public int
    howManyMessages (String who)
  { ... }

  public MailItem
    getNextMailItem (String who)
  { ... }

  public void
    post(MailItem item)
  { ... }
}
```

```
class MailItem {

  private String to;
  private String from;
  private String message;

  public MailItem( ... ) {
    ...
  }

  public String getFrom()
  { ... }
  public String getTo ()
  { ... }
  public String getMessage()
  { ... }
  public void print()
  { ... }
}
```

Generalization/specialization (is-a)

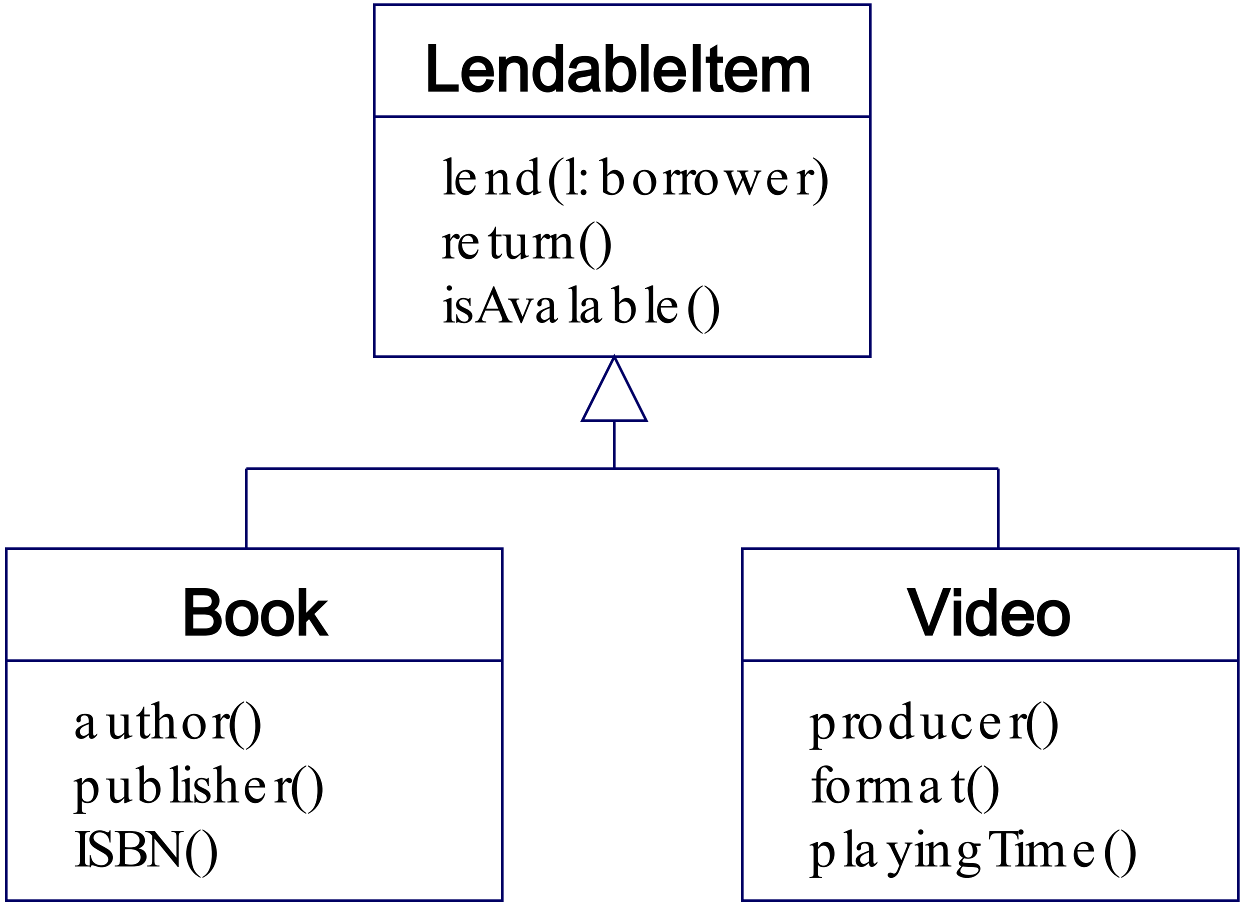
Combine concepts to a more general concept.



Specialization in UML

General concept

Special concepts



Specialization in Java

```java
class LendableItem {
  void lend(Borrower b) {
    // code for lend
  }

  void return() {
    // code for return
  }

  boolean isAvalable() {
    // code for isAvalable
  }

  ...
}
```

```java
class Book extends LendableItem
{
  String author() { ... }
  String puclisher()     { ... }
  String ISBN()         { ... }
  ...
}

class Video extends LendableItem
{
  String producer() { ... }
  String format()    { ... }
  int playingTime()    { ... }
  ...
}
```

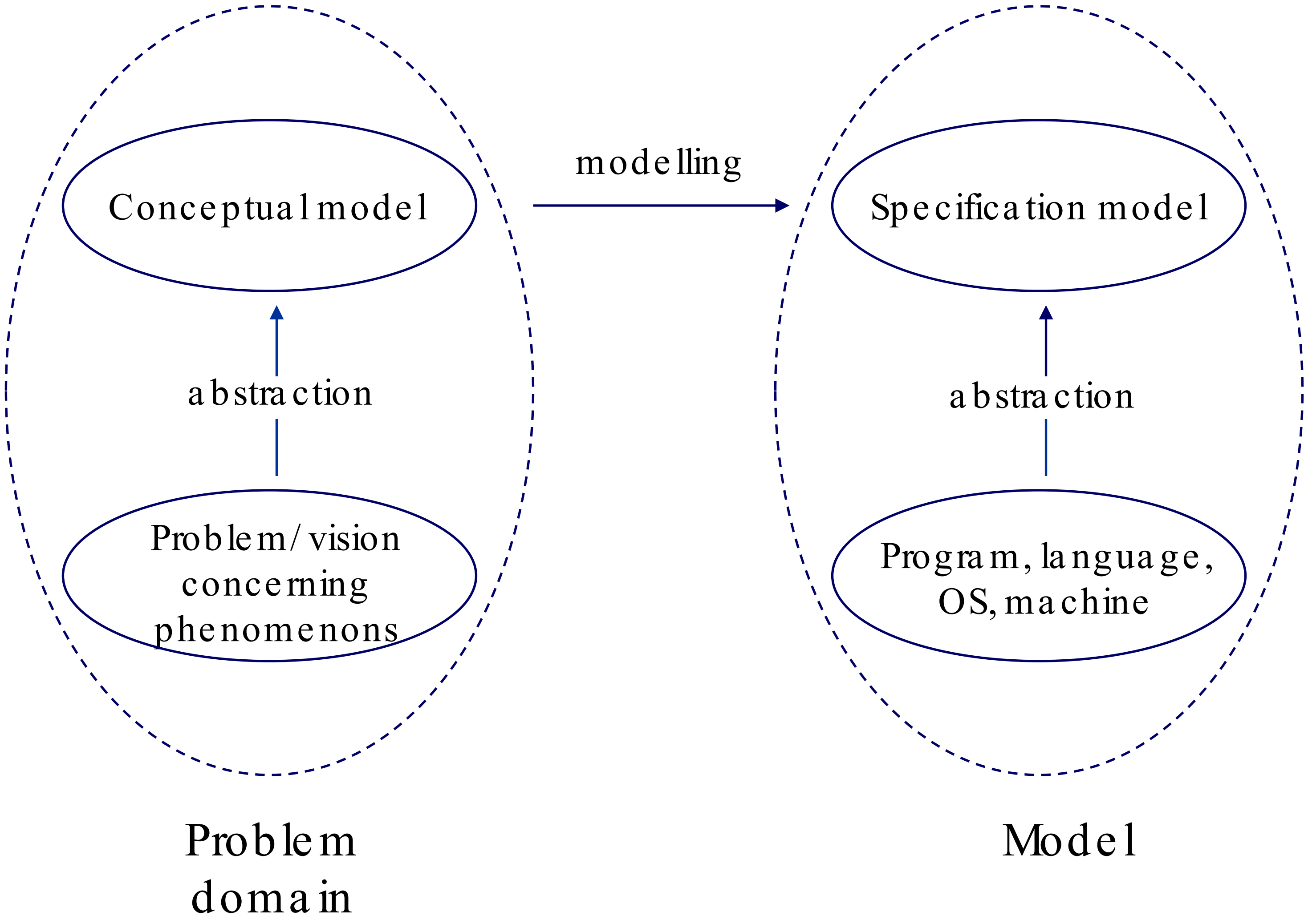
Systematics in OOP

- **Modelling**
  - from problem description to conceptual model
  - refinement of conceptual model to specification model (method signatures and specifications)

- **Implementation**
  - structurally: from specification model to Java code (automatically)
  - body: attributes and methods (creativity and systematics)

Problem domain ⇒ UNIFIED MODELING LANGUAGE UML™ ⇔ JAVA

Conceptual Modelling



Model-Driven Programming

Programming in Context

Hand-in-Hand Modeling and Coding (1)

- **David Gries (Edsger W. Dijkstra)**
  - the loop body and the loop invariant is developed hand-in-hand with the latter leading the way
- **We (Kristen Nygaard)**
  - coding and class modeling is done hand-in-hand with the latter leading the way
- **Design by contract and systematic programming**
  - a class model is a design contract in precisely the same way as a loop invariant is
  - code is introduced on purpose (fulfilling the contract)

Programs as models

K. Nygaard

A program execution is regarded as
a physical model system

E.W. Dijkstra

It's not the purpose of our programs
to instruct the computer;
it's the purpose of the computer
to execute our programs

Contents and Progression

- **Traditional approach**
  - typical textbooks only address the first and to some extend the second perspective
  - topics are organized according to the syntactical structures in the programming language (bottom-up)
  - tendency to completeness in coverage of topics
  - **syntax-driven progression**

- **Model-driven approach**
  - a balanced coverage of all three views
  - conceptual modeling is leading the way
  - systematic programming (killing rabbits)
  - early bird & spiral approach
  - **model-driven progression**

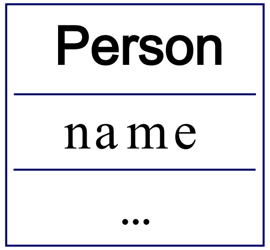
Benefits of MDP

The integration of
conceptual modeling and coding provides
structure, traceability, and a systematic approach to
program development

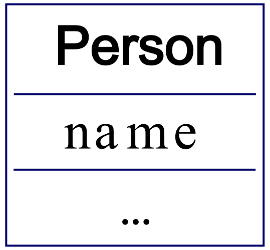
The integrated approach
strongly motivate and support the students
in their understanding and practice of
the programming process

Drop-out rate down from 48% to 13%
(over a five year period)

Hand-in-Hand Modeling and Coding (2)

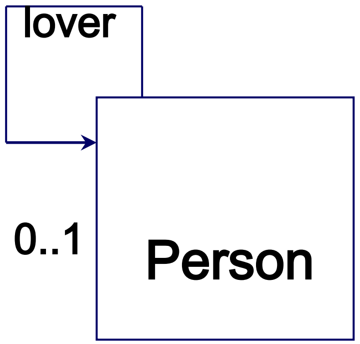


Hand-in-Hand Modeling and Coding (2)



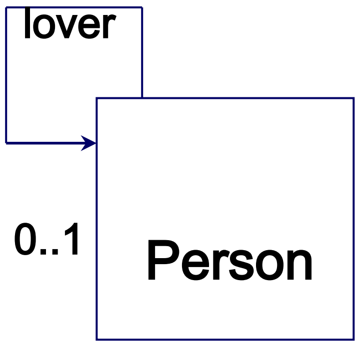

```
class Person {
  private String name;



  public Person(String name) {
    this.name = name;



  }




  ...
}
```

Hand-in-Hand Modeling and Coding (2)



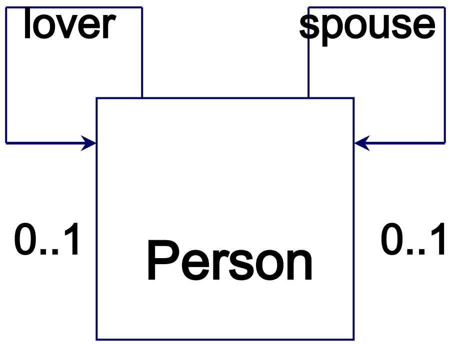

```
class Person {
  private String name;



  public Person(String name) {
    this.name = name;



  }



  ...
}
```

Hand-in-Hand Modeling and Coding (2)



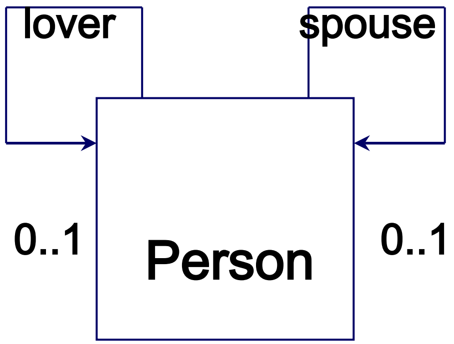

```
class Person {
  private String name;
  private Person lover;



  public Person(String name) {
    this.name = name;
    lover = null;


  }

  public fallsInLoveWith(Person p) ...


   ...
}
```

Hand-in-Hand Modeling and Coding (2)



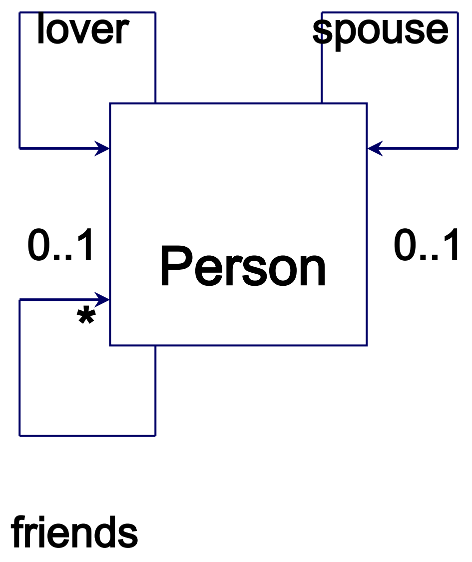

```
class Person {
  private String name;
  private Person lover;



  public Person(String name) {
    this.name = name;
    lover = null;


  }

  public fallsInLoveWith(Person p) ...


   ...
}
```

Hand-in-Hand Modeling and Coding (2)



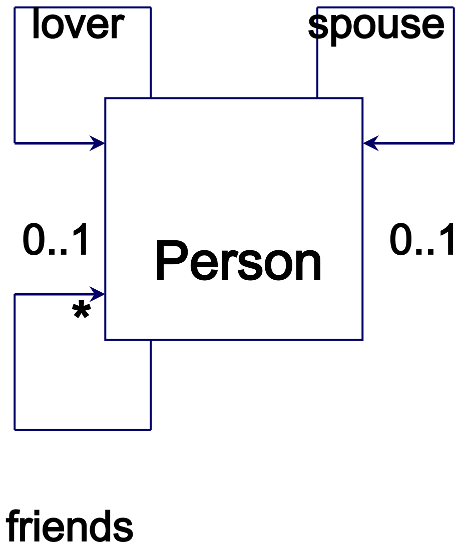

```
class Person {
  private String name;
  private Person lover;
  private Person spouse;


  public Person(String name) {
    this.name = name;
    lover = null;
    spouse = null;

  }

  public fallsInLoveWith(Person p) ...
  public marries(Person p) ...

  ...
}
```

Hand-in-Hand Modeling and Coding (2)



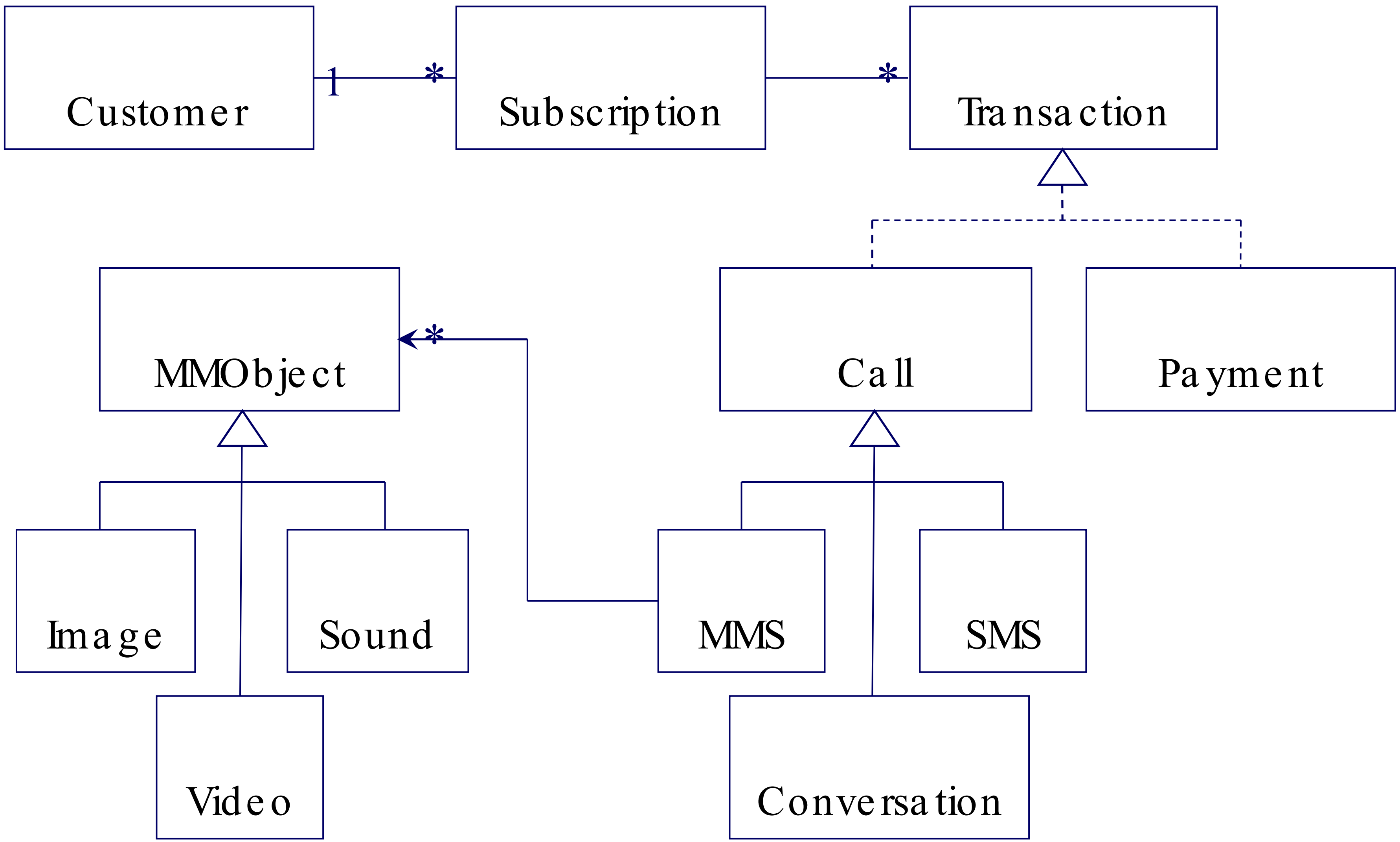

```
class Person {
  private String name;
  private Person lover;
  private Person spouse;


  public Person(String name) {
    this.name = name;
    lover = null;
    spouse = null;

  }

  public fallsInLoveWith(Person p) ...
  public marries(Person p) ...

  ...
}
```

Hand-in-Hand Modeling and Coding (2)




```
class Person {
  private String name;
  private Person lover;
  private Person spouse;
  private Set<Person> friends;

  public Person(String name) {
    this.name = name;
    lover = null;
    spouse = null;
    friends = new HashSet<Person>();
  }

  public fallsInLoveWith(Person p) ...
  public marries(Person p) ...
  public becomesFriendWith(Person p) ...
  ...
}
```

Example: Talkmore Inc.

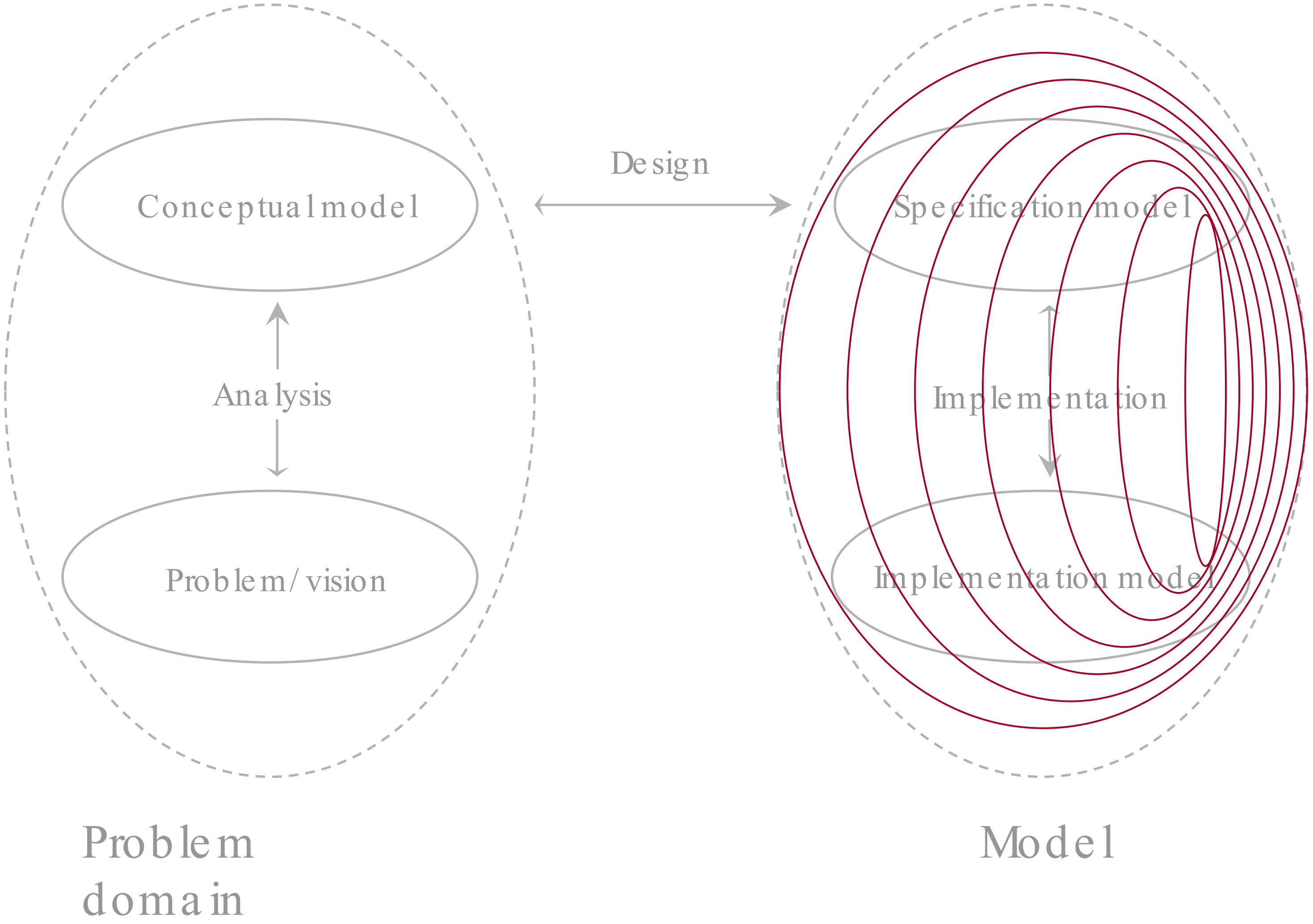


Model-Driven Progression (1)

- **Model-driven**
  - programming tasks starts from a class model
  - mostly, the model is given
  - sometimes, also the model must be developed

- **Progression**
  - models become increasingly complex during the course
  - associated systematic programming techniques
  - language issues covered “by need”

Course Progression



Systematic Implementation Techniques

- **Inter-class structure**
  - implementation **of specification model** using standard patterns for implementing relations between classes

- **Intra-class structure**
  - implementation **of interface or class specifications** using standard techniques for implementing attributes and methods
  - implementation using class invariants
  - explicit process

- **Methods**
  - algorithmic patterns (sweep, search, divide and conquer, ...)
  - loop invariant techniques

Separation of concerns...

Problem

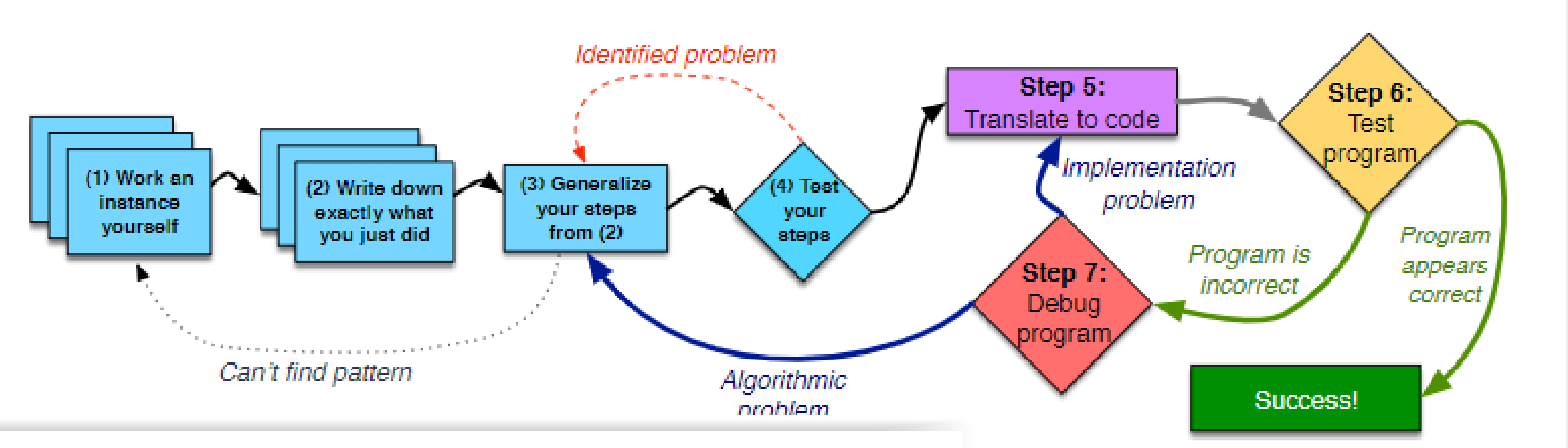
- **Create a class that represents a date (e.g. November 2, 2023). The date should be able to do two things:**
  - nextDate: the date is now representing the next date
  - print: return a string representation of the date (e.g. “02-11-2023”)

- **Discuss with your neighbor: How will you describe your approach to a first year student?**

Photo by RaissaLara Lütolf (-Fasel) on Unsplash

Typical "process"

- **The seven steps**



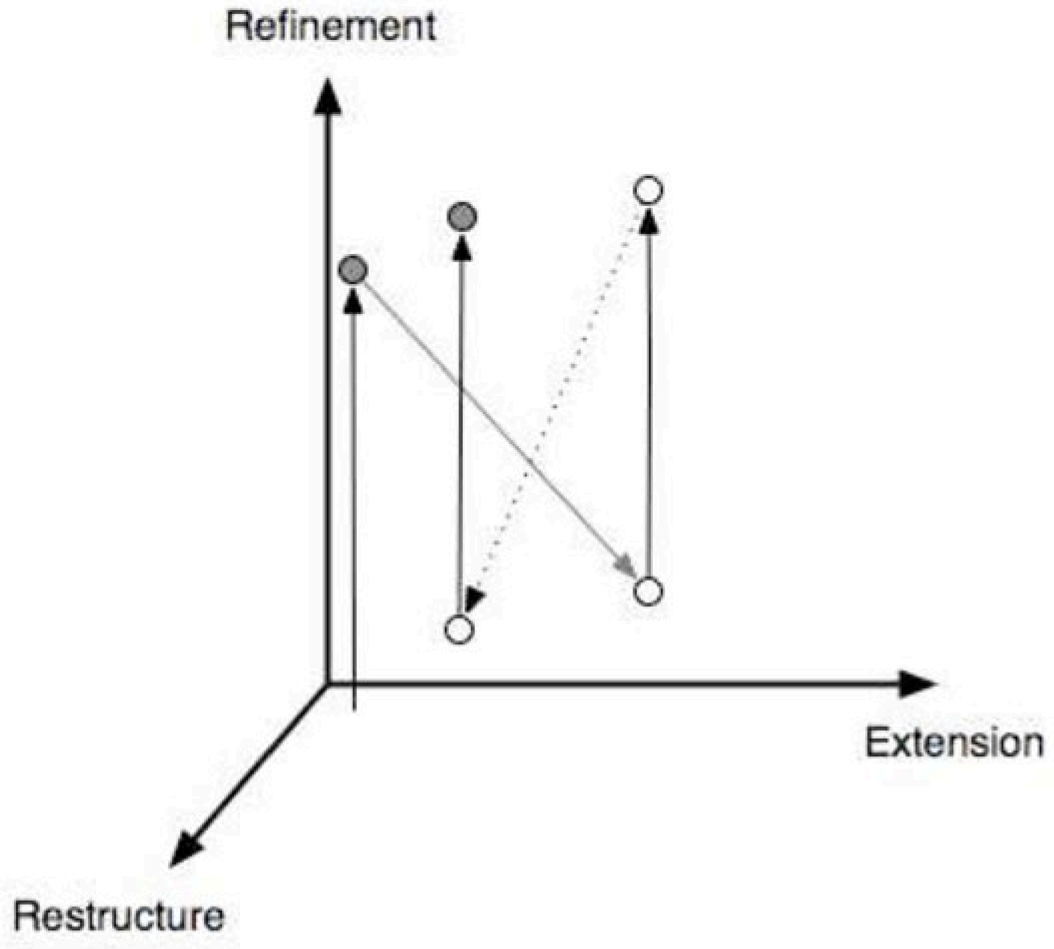

```
make a general pseudo-code that implements the problem
while (pseudo-code is not a program)
    make a more detailed pseudo-code by making (parts of) the pseudo-code more specific
```


Andrew D. Hilton, Genevieve M. Lipp, and Susan H. Rodger. 2019. Translation from Problem to Code in Seven Steps. In Proceedings of the ACM Conference on Global Computing Education (CompEd '19). Association for Computing Machinery, New York, NY, USA, 78–84. https://doi.org/10.1145/3300115.3309508

Alternative

- **Extension:** Extend the specification (abstract description of what the code must do)
- **Refinement:** Make the abstract spec into real code
- **Restructure:** Enhance the quality of the code (without altering the functionality – e.g. private helper-methods)

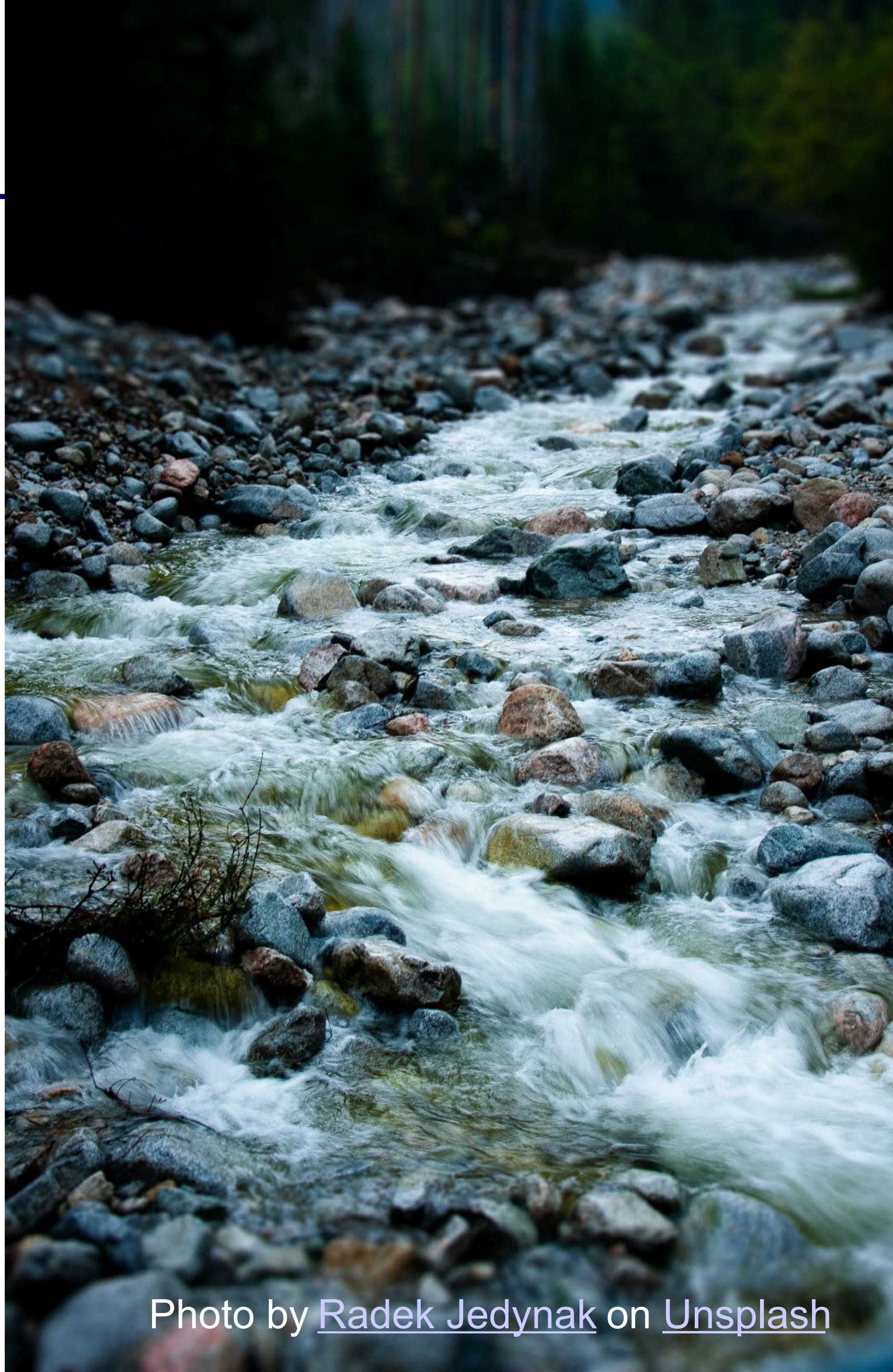
```
spec = the empty specification
impl = the empty program
while (implementation is not runable and does not fulfil the problem)
   if (spec does not fulfil req)
      extend spec;
   if (impl does not meet spec)
      refine impl;
   if (impl need a better quality)
      restructure impl;
```



Michael E. Caspersen and Michael Kolling. 2009. STREAM: A First Programming Process. ACM Trans. Comput. Educ. 9, 1, Article 4 (March 2009), 29 pages. https://doi.org/10.1145/1513593.1513597

STREAM

1. **Stubs**
2. **Tests**
3. **Representations**
    4. Evaluations
    5. Attributes
6. **Methods**



Stub

- **An empty class (h + cpp)**

```
#pragma once
#include <string>

class Date {
public:
    //initialize the date with the date d, month m and year y
    Date(int d, int m, int y);
    //advance the date to the next date
    void toNextDate();
    //return a string representation of the date in the format dd-mm-yyyy
    std::string toString();
};
```

Stub (2)

- **Make a skeleton implementation**

```
#include "Date.h"

//initialize the date with the date d, month m and year y
Date::Date(int d, int m, int y) {
}
//advance the date to the next date
void Date::toNextDate() {
}
//return a string representation of the date in the format dd-mm-yyyy
std::string Date::toString() {
    return "";
}
```

Tests

- **Make tests to ensure that the implementation fulfils the specification**

```
int   main () {
      Date  d1(15, 01, 2007);
      Date  d2(1, 1, 2007);
      Date  d3(31, 05, 2020);
      Date  d4(31, 12, 1999);
      Date  d5(28, 02, 2020);
      Date  d6(28, 02, 2019);
      toStringTest     (d1,   "15 - 01- 2007" );
      toStringTest     (d2,   "01 - 01- 2007" );
      advanceAndCheck (d1,   "16 - 01- 2007" );   //expected 15   - 01- 2007  - > 16 - 01- 2007
      advanceAndCheck (d3,  "01 - 06- 2020" );   //expected 31   - 05- 2020  - > 01 - 06- 2020
      advanceAndCheck (d4,  "01 - 01- 2000" );   //expected 31   - 12- 1999  - > 01 - 01- 2000
      advanceAndCheck (d5,  "29 - 02- 2020" );   //expected 28   - 02- 2020  - > 29 - 02- 2020
      advanceAndCheck (d6,  "01 - 03- 2019" );   //expected 28   - 02- 2019  - > 01 - 03- 2019
      return    0;
}
```

Representations

- **Find (at least two) different ways to represent the knowledge (attributes) the class needs to fulfil its specification**
  - Three integers
  - One integer (days since year X)
  - string

Photo by Shahram Anhari on Unsplash

Evaluation

- **How easy/difficult is it to implement the methods with the given representation**
  - Make a REM: Trivial, Easy, Average, Challenging, Hard

Table 14.3: effort REM for Date

	3int	*daysSince*	*string*
`toNextDate()`	Challenging	Trivial	Hard
`toString()`	Trivial	Hard	Trivial

Photo by Graham Ruttan on Unsplash

Attributes

- **Implement the "easiest" attributes incl a constructor**

```
class Date {
public:
    //initialize the date with the date d, month m and year y
    Date(int d, int m, int y);
    //advance the date to the next date
    void toNextDate();
    //return a string representation of the date in the format dd-mm-yyyy
    std::string toString();
private:
    int day, month, year;
};
```

- **In the cpp file**

```
//initialize the date with the date d, month m and year y
Date::Date(int d, int m, int y): day(d), month(m), year(y) {
}
```

- **Choose the easiest (REM) method and implement (part of) it**

```
//return a string representation of the date in the
format dd - mm-yyyy
std:: string Date ::toString () {
    return to_string ( day ) + "-" + to_string ( month ) + "-" +
    to_string ( year );
}
```

- **Test (one works and one fails)**
- **Day + month as two ciffers – wish-fairy!**

```
std:: string Date ::toString () {
    return formatedString ( day ) + "-" +
    formatedString ( month ) + "-" + to_string ( year );
}
```

Photo by Anthony Tran on Unsplash

formattedString()

```
class Date {
public:
    ...
private:
    //return a stringrepresentation with the with two of the integer dm.
    // precondition: 1 <= dm <=31
    std::string formatedString(int dm);
};
-----
string Date::formatedString(int dm) {
    if (dm <= 9) //only one digit
        return "0" + to_string(dm);
    else // two digits
        return to_string(dm);
}
```

toNextDate()

- **The easiest solution? Add one to the day**

```
//advance the date to the next date
void Date::toNextDate() {
   day += 1;
}
```

- **It actually works in most cases;-)**

Problem

- **What is the problem? Overflow!**
- **Solution: Wish-fairy!**

```
//advance the date to the next date
void Date::toNextDate() {
    day += 1;
    checkDayOverflow(); //fixes the problem if overflow
}
//check if the day and month invariant is violated. If so, fix it
void Date::checkDayOverflow() {
}
```

Overflow?

- **Occurs when the day (day) becomme larger than the number of days in the month**
- **Simple solution: all moths are 30 days**

```
//check if the day and month invariant is violated. If so, fix it
void   Date ::checkDayOverflow    () {
   if   (day > 30)     {  //overflow
        day  = 1 ;
        month  += 1 ;
   }
}
```

New problem – month overflow

- **Same solution – wish-fairy**

```
//check if the day and month invariant is violated. If so,
fix it
void Date::checkDayOverflow() {
    if (day > 30) { //overflow
        day = 1;
        month += 1;
    }
    checkMonthOverflow();
}
```

checkMonthOverflow()

```
//check if the month invariant is violated. If so, set
month = 1 and add one to year
void Date::checkMonthOverflow() {
    if (month > 12) {
        month = 1;
        year += 1;
    }
}
```

A month = 30 days?

- Not all months have 30 days
- Same solution – **wish-fairy**!

```
void Date::checkDayOverflow () {
    if ( day > daysInMonth () ) {
        day = 1;
        month +=1;
    }
    checkMonthOverflow ();
}
//returns the days in the current month
int Date::daysInMonth () {
    return 30;
}
```

Like before: restructure then extend

daysInMonth()

- **Implementing the method – apart from February the length is the same all years, i.e. a table of number of days**

```
//returns the days in the current month
int    Date ::daysInMonth    () {
   //month          1   2   3   4   5   6   7   8   9   10  11  12
   int    days[12] = { 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
   return     days [ month   -   1];
}
```

leapYear()

- **Problem: February**
- **Same solution – wish-fairy!**

```
//returns the days in the current month
int Date::daysInMonth() {
    //month      1   2   3   4   5   6   7   8   9   10  11  12
    int days[12] = { 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
    if (month == 2 && isLeapYear())
        return 29;
    else
        return days[month - 1];
}
//is the current year a leap year?
bool Date::isLeapYear() {
    return false;
}
```

Like before: restructure then extend

isLeapYear()

I den julianske kalender er et år skudår, hvis årstallet er deleligt med 4. Dette er i den gregorianske udvidet således at dette ikke gælder for årstal der er delelige med 100, bortset fra dem, der er delelige med 400 som alligevel er skudår. År 1900 var således ikke et skudår, men år 2000 var. Et år i den gregorianske kalender varer således, i gennemsnit over en periode på 400 år, $365 + \frac{1}{4} - \frac{1}{100} + \frac{1}{400} = 365{,}2425$ dage.

```
//is the current year a leap year?
bool   Date ::isLeapYear    () {
   if    ( month   == 2)
        return     (((year % 4) == 0) && ((year % 100) != 0)) || ((year % 400) == 0);
   else
        return     false   ;
}
```

Wrapping Up: Key Points

- **Conceptual modeling**
  - the defining characteristic of object-orientation
- **Model-driven**
  - programming tasks take-off from a class model
  - coding and conceptual modeling is done hand-in-hand with the latter leading the way
- **Progression**
  - driven by complexity in class models
  - stressing associated techniques of systematic programming
  - language issues covered by need