Development of Rigorous Adaptive Information Systems

Dr. Nasreddine Aoumeur
FIN, ITI, DB group
aoumeur@iti.cs.uni-magdeburg.de

Course Site:
wwwiti.cs.uni-magdeburg.de/~aoumeur
wwwiti.cs.uni-magdeburg.de/iti_db/lehre/oois/inde
Information Systems: Working definition

- *reactive* systems (i.e. in continuous interaction with their environment), with
- large amount of *immutable and non-immutable data* (i.e. fixed and changing) and, with
- *processes and activities* for exhibiting behaviors on these (state-less and -full) data.
Different generations of CMs: “Entity first”

IS Conceptual Model

Entity-Relationship [1973..]
- Intuitive / Simple
- Revolutionary for IS
- UoD: Entities + Relations
- Mathematically sound
- Rich abstraction mechanisms
- Different variants
  (NIAM / SADT / MERISE / ..)

State-less and -ful DATA

Processes and Rules

Process-centric Formalisms
- CCS, CSP, DFD, Petri Nets, ..
- Synchronous / asynchronous
- Mathematically sound
- Executable / operational
CMs Generation shift: “Entity To Object”

From E/R to “Object-Object”

Processes, Operations and Rules
First generation of CMs: “Entity first”

E/R Conceptual Model

Customer
- Name: String
- Birth-Date: Date
- Address: Address
- Income: Money

(Running) Account
- Number: Nat
- Balance: Money
- Limit: Money
- History: List[Date, Money]

Saving Account
- Number: Nat
- Interest: Percent
- Balance

Processes and Rules
Account USE: First open --- then deposit – then (withdraw-deposit)* - then Close-or-be-closed
CMs generation shift: “From Entity to object”

From E/R to Object Model: Banking Example

**Customer**
- Name: String
- Birth-Date: Date
- Address: Address
- Income: Money
- Own(account): Boolean
- Deposit(amount)
- Withdraw(Amount)

**Bank**
- Open-Date

**Running Account**
- Number: Nat
- Balance: Money
- Limit: Money
- History: List[Date, Money]
- open(date, bank)
- close(date)
- Debit(Amount)
- Credit(Amount)

Processes, operations and rules:
Account USE: First open → then deposit → then (withdraw−deposit)* → then Close or be closed
CM Generation shift: “From Entity to Object”

From E/R Model to Object: ATM example

**Bank-Card**
- Number: Nat
- Account-Nb: Nat
- Code: String
- Create()
- Delete()
- Accepted()
- Rejected()

**Customer**
- Name: String
- Withdraw (ac, amount)
- Deposit (ac, amount)

**AutomaticTellerMachine**
- ATM-Reference: String
- Cash: Hidden
- Bank: String
- Transaction: List[Money]
- History: List[Card-Nb, Acnt-Nb, Money]
- Read-card()
- Enter-Pin(Code)
- Enter-Amount(Money)
- Get-Money(Money)

Processes, operations and Rules:
ATM-use: First enter-card – then enter-code – then enter-transaction—get money
CM Generation shift: “From Entity to Object”

From E/R Model to Object: The Library Example

Student
- Name : String
- Subscription-Nb
- Semester
- Subscribe2Library
- Unsubscribe
- GetCard
- ReceivePenalty

Book
- Reference : String
- Name : String
- Author : String
- Publisher : String
- Add()
- Suppress()
- ToBorrow(Date)
- ToReturn(Date)

Processes, operations and Rules
First subscribe-- Get library-card – (Borrow – Return –or– Penalty)*--(be)Unsubscribe(d)
Object-Oriented Paradigm: General Overview

In real world terms:

- An object represents an individual entity or thing.
- A class represents a group of objects that exhibit some common characteristics or behavior.
- Classes are resulted from classification.

OO philosophy: The real-world consists in a society of interacting objects.

Examples of classes in real world:
- Students
- Graduate students
- Undergraduate students
- MS students
- Ph.D. students
Object-Oriented Paradigm: Main concepts

- **An object has**
  - **state**: defined by the set of fields or attributes.
  - **behavior**: defined by the set of methods or operation that can be applied to the object.
  - **identity**: determined at the creation time **to uniquely referencing the object**.

- **Class**
  - A template for creating objects.
  - Objects of the same class exhibit the same behavior.
  - But generally, they posses different states (attribute values)
Object-Oriented Paradigm: Main concepts
Object and Class

Like in real world:

- **Id**: IF-43342
- **Title**: „Petri Nets“
- **Author**: „W. Reisig“
- **State**: {available, borrowed, use..}

- has
- one can
- borrowed
- returned
- edited
- 

**Book**

**Object**

**Attributs** (state)

**Methods** (behaviour)
Object-Oriented Paradigm: Main Concepts

Object-Class

The class MyDate

<table>
<thead>
<tr>
<th>MyDate</th>
</tr>
</thead>
<tbody>
<tr>
<td>day</td>
</tr>
<tr>
<td>month</td>
</tr>
<tr>
<td>year</td>
</tr>
</tbody>
</table>

fields (or: variables)

state attributes)

set (d,m,y)
incDays (nDays)
getWeekDay ()

methods

The object d is an instance of class MyDate

<table>
<thead>
<tr>
<th>d:MyDate</th>
</tr>
</thead>
<tbody>
<tr>
<td>day = 31</td>
</tr>
<tr>
<td>month = 1</td>
</tr>
<tr>
<td>year = 2000</td>
</tr>
</tbody>
</table>

set (d,m,y)
incDays (nDays)
getWeekDay ()
### Object-Oriented Paradigm: Main concepts

**Object-Class**

#### The class Accounts

<table>
<thead>
<tr>
<th>Account</th>
<th>Number</th>
<th>balance</th>
<th>limit</th>
<th>history</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ open</td>
<td>- Close</td>
<td>- debit(amount)</td>
<td>- Credit(amount)</td>
<td>- Tranfer(ac1,ac2)</td>
</tr>
</tbody>
</table>

#### The object Ac-Nasr instance

<table>
<thead>
<tr>
<th>Ac-Nasr : Account</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number = 3130888</td>
</tr>
<tr>
<td>Balance = 1200</td>
</tr>
<tr>
<td>Limit = 20</td>
</tr>
<tr>
<td>History = [20.2.2.07,…]</td>
</tr>
</tbody>
</table>

Methods:
- open
- Close
- debit(amount)
- Credit(amount)
- Tranfer(ac1,ac2)
Object-Oriented Paradigm: Main concepts
Encapsulation

The Account Class
- debit(Id, Money)
- Balance(Bal): Money
- Limit(Lm): Money
- History(Hs): List[M.D]
- credit(Id, Money)

The ATM Class
- Transaction(Tr)
- AcntNb(AcN)
- CardNb(AcN)
- Pin(Id)
- Store(Sr): Money
- State(St): State
- Insert-Card(InC)()
- Insert_Code(InsC)(C)
- Choose_Trans(CTr)(M)
### Object-Oriented Paradigm: Main concepts

#### Encapsulation Principles

- In general all attributes should be **private**.
- Use EXCLUSIVELY methods such as **debit** and **credit** to access the internal state.
- Only methods that are part of the "interface" should be public.
- Instance variables and methods are **visible** within that object and other instances of the class.
- With respect to other objects and classes scope can be modified
  - **private**: accessible only within the concerned class
  - **protected**: could be changed at subclasses
  - **public**: could be manipulated everywhere
class MyDate {
    private int day, month, year;
    public void set (int d, int m, int y)
        {day=d, month= m, year= y ... }
    public void incDays (int nDays) {...}
    public int getWeekDay ( ) { ... }
}

class OtherClass {
    void anyMethod {
        MyDate d = new MyDate ( ) ;
        d.set (09, 5, 2007) ;
        d.month = 3; // COMPILATION ERROR !!!
    }
}

Object-Oriented Paradigm: Main concepts
Encapsulation At-work
Object-Oriented Paradigm: Main concepts

Encapsulation At-work

class circle{
    double radius, middlepointX, middlepointY;
    boolean setRadius(double newRadius){
        if(newRadius > 0){
            radius = newRadius;
            return true;
        }
        else return false;
    }
}

//with encapsulation
circle k = new circle();
boolean b = k.setRadius(-88);
//radius never negative!

//without encapsulation:
circle k = new circle();
k.radius = -88;
//radius negative!
Object-Oriented Paradigm: Main concepts

Inheritance Concept

A mechanism to organize classes by commonalities.

- subclasses, specialization
- superclass, generalization

- Is-a relation

Example:

- A graduate student is a student.
- A Master student is a graduate student.
- A Ph.D. student is a graduate student.
- An undergraduate student is a student.
Object-Oriented Paradigm: Main concepts
Inheritance concept and illustration

```
Student

GraduateStudent          UndergraduateStudent

MasterStudent            PhDStudent
```
Object-Oriented Paradigm: Main concepts
Inheritance concept and illustration

Running Account

- Number : Nat
- Balance : Money
- Limit : Money
- History : List[Date,Money]

- open (date, bank)
- close(date)
- Debit(Amount)
- Credit(Amount)

Saving Account

- Interest : Percent
- BalanceSav : Money

- open (date, bank)
- close(date)
- IncreaseInterest(Percent)
- Money(money,account)
Object-Oriented Paradigm: Main concepts
Inheritance concept and illustration

(ordinary) book
- NumberCode : Nat
- Title : String
- Status : {available, borrowed}
- NumberCopies : Natural
- ToBorrow (date)
- ToReturn (Date)

Thesis
- Field
- Supervisor
- Mark
- Date-defense
- GetCopy

Periodics
- Editors
- Publisher

Online-Publications
- URL
- Duration-subscription
- +subscribe
- - unsubscribe
As opposed to C++, it is possible to inherit only from ONE class.

**Pros**  
avoids many potential problems and bugs.

**Cons**  
might cause code replication
Overloading:

• Two or more methods/constructors with the same name but different numbers or different types of parameters:
  
  ```
  void methodB(int i)
  void methodB(float f)
  ```

Overriding

- Replacing the implementation of a method in the superclass with one of your own.
- You can only override a method with the same signature.

Please Avoid overloading !!!!
Polymorphism:
- Inheritance creates an “is a” relation:
- For example, if B inherits from A, then we say that “B is kind of an A”.
- A same method same defined in the class hierarchy
- How to dynamically choose the right methods?
Object-Oriented Paradigm: Main constructions

Polymorphism concept

GuiComponent
- width: int
- height: int
- center: point
- setOptimalSize()
- moveTo(newX, newY)

Editbox
- minValue
- maxValue
- append(String)
- setOptimalSize()

Scrollbar
- text: String
- getValue()
- getText()
Object-Oriented Paradigm: Main constructions
Polymorphism at-work

- A subclass inherits all members of its superclass:
  - Variables
  - Methods

- A subclass can:
  - Add more variables
  - Add more methods
  - Override methods of its superclass

```java
Scrollbar sb = new Scrollbar();
```

**Q**: What are the variables of `sb`?

**What are the methods that `sb` can execute?**
GuiComponent g;
if (scrollbarWasSelected)
    g = new Scrollbar();
else
    g = new EditBox();

g.setOptimalSize(); // dynamic binding

- A reference to a superclass can point to objects of its subclasses.
- The pointer g is a polymorphic pointer.
Object-Oriented Paradigm: Main constructions
Complex Polymorphic Dynamic binding

GuiComponent[] ga = new GuiComponent[3];
ga[0] = new Scrollbar();
ga[1] = new Scrollbar();
ga[2] = new EditBox();
for (int i = 0; i < ga.length; i++) {
    ga[i].setOptimalSize(); //line 6
}

Q: Which version of setOptimalSize is called in line 6?

Polymorphism allows us to work with an object without knowing its exact type.
Suppose we know that $g$ currently points to a Scrollbar, and we want to call: $n = g\.getMinValue()$; This will cause a compilation error (why?) .... Casting problem
Object-Oriented Paradigm: Main constructions

Polymorphism at-work (casting problem)

- The solution is casting (actually down casting).
  - **Casting** = convert a variable from one type to another.
  - **Down Casting** = convert from a superclass to one of its subclasses.

```java
sb = (Scrollbar) g.getMinValue();
```

Or:

```java
Scrollbar sb = (Scrollbar) g;
sb.getMinValue();
```

- We “tell” the compiler that `g` currently points to a `Scrollbar`.
- If `g` does not currently point to a `Scrollbar`, a `ClassCastException` is thrown.
Object-Oriented Paradigm: Main constructions

Association Types

- **Association**: C → D
- **Directed Association**: C → D
- **Aggregation**: C ⊇ D
  - Whole part
  - A stronger form of association
- **Composition**: C ⊂ D
  - If an object d of class D is related to an object c of class C, then d depends existentially on c.
  - A stronger form of aggregation
Each object of class C is related to 1 to 5 objects of class D.
Each object of class D is related to exactly one object of class C.

Each object of class C is related to at least one object of class D.
Each object of class D is related to arbitrary many objects of class C.

Associated objects of class D are ordered.
Object-Oriented Paradigm: Main constructions

Association Types

Line

- lRole: 0..1
- determine
- pRole: 2

Point

- role used to navigate
- association name
- direction: 2 points determine a line
- directed aggregation
Object-Oriented Paradigm: Main constructions

Association Types

Inheritance
Class C inherits from class D

dependency relationship
Class C depends on class D
Object-Oriented Paradigm: Main constructions
Association Types: Aggregation

```
Car

Engine 1..1

Brakes 1..4

Wheel 4

Searing Wheel 1
```