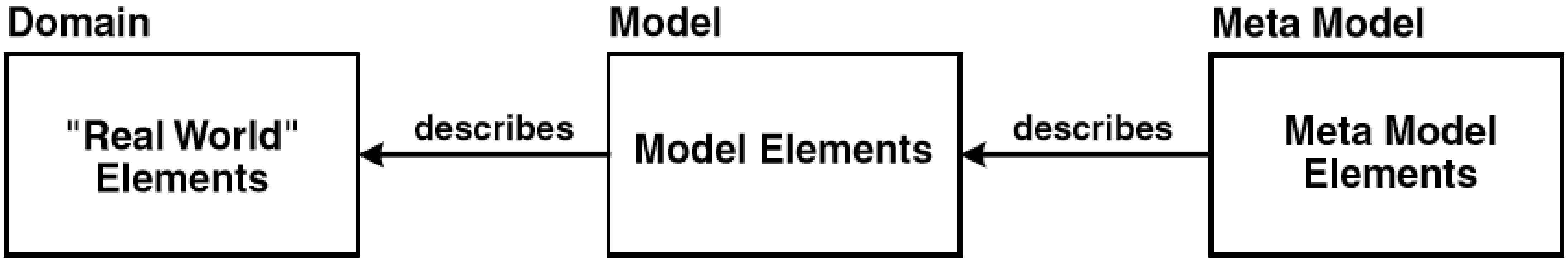
MODEL DRIVEN SOFTWARE DEVELOPMENT

LECTURE : 3

META-MODELLING

- Metamodels are models that make statements about modeling
- Metamodel describes the possible structure of models – in an abstract way
- It defines the constructs of a modeling language and their relationships, as well as constraints and modeling rules – but not the concrete syntax of the language
- Metamodels and models have a class-instance relationship
  - each model is an instance of a metamodel
- To define a metamodel, a metamodeling language is therefore required that in turn is described by a meta meta model

META-MODELLING



Domain

"Real World" Elements

describes

Model

Model Elements

describes

Meta Model

Meta Model Elements

NEED?

Metamodeling knowledge is needed for dealing with the following MDSD challenges:

- **Construction of domain-specific modeling languages (DSLs):**

  The metamodel describes the abstract syntax of such a language

- **Model validation:**

  Models are validated against the constraints defined in the metamodel.

- **Model-to-model transformations:**

  Such transformations are defined as mapping rules between two metamodels.

- **Code generation:**

  The generation templates refer to the metamodel of the DSL.

- **Tool integration:**

  Based on the metamodel, modeling tools can be adapted to the respective domain.

OCL

- Metamodels are models that make statements about modeling
- Metamodel describes the possible structure of models – in an abstract way
- It defines the constructs of a modeling language and their relationships, as well as constraints and modeling rules – but not the concrete syntax of the language
- Metamodels and models have a class-instance relationship
  - each model is an instance of a metamodel
- To define a metamodel, a metamodeling language is therefore required that in turn is described by a meta meta model

OCL

Standard library of

- primitive types and associated operations

Basic types (Boolean, Integer, Real, String)

Collection types:

- Collection
- Set
- Ordered Set (only OCL2)
- Bag
- Sequence

INVARIANT

- **Definition** – An invariant is a constraint that should be true for an object during its complete lifetime
  - Invariants often represent rules that should hold for the real-life objects after which the software objects are modeled
- **Syntax:**
  - context <classifier>
  - inv [<constraint name>]: <Boolean OCL expression>
  - context Meeting inv: self.end > self.start

INVARIANT

- **context Meeting inv: self.participants->size()>=2**

PRE-CONDITION

Definition

- – Constraint that must be true just prior to the execution of an operation

Syntax

- context <classifier>::<operation> (<parameters>)
- pre [<constraint name>]:
- <Boolean OCL expression>

PRE-CONDITION

context Meeting::shift(d:Integer)

pre: self.isConfirmed = false

context Meeting::shift(d:Integer)

pre: d>0 pre: d>0

context Meeting::shift(d:Integer)

pre: self.isConfirmed = false and d>0

POST-CONDITION

Definition

Constraint that must be true just after to the execution of an operation• Postconditions are the way how the actual effect of an operation is described in OCL.

Syntax

context <classifier>::<operation> (<parameters>)

post [<constraint name>]:

<Boolean OCL expression>

POST-CONDITION

context Meeting::duration():Integer

post: result = self.end – self.start

-- keyword result refers to the result of the operation

context Meeting::confirm()

post: self.isConfirmed = true

Let expressions:

Sometimes a sub-expression is used more than once in a constraint.

The let expression allows one to define a variable which can be used in the constraint.

```
context Person inv:
let income : Integer = self.job.salary->sum() in
 if isUnemployed then
 income < 100
 else
 income >= 100
endif
```

The Let expression allows a variable to be used in one Ocl expression. To enable reuse of variables/operations over multiple OCL expressions one can use a Constraint with the stereotype «definition», in which helper variables/operations are defined.

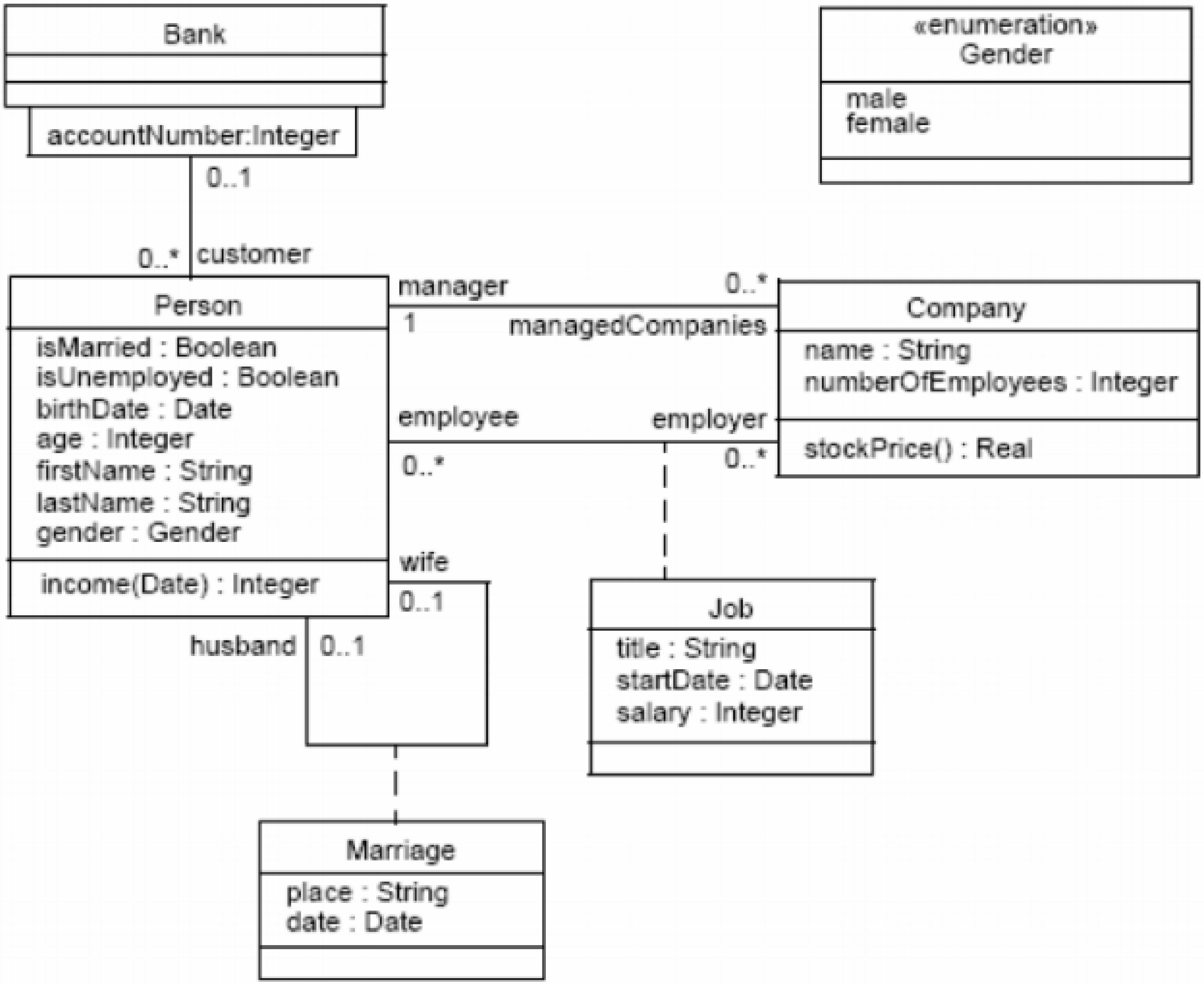
```
context Person
def: income : Integer = self.job.salary->sum()
def: nickname : String = ’Little Red Rooster’
def: hasTitle(t : String) : Boolean = self.job->exists(title = t)
```

PRE, POST AND @PRE

Customer::buy(product)

pre: acctBal-product.price > 0

post: acctBal = acctBal@pre - product.price


Bank
accountNumber:Integer
0..1
0..*
customer
«enumeration»
Gender
male
female
Person
isMarried : Boolean
isUnemployed : Boolean
birthDate : Date
age : Integer
firstName : String
lastName : String
gender : Gender
income(Date) : Integer
manager
1
0..*
managedCompanies
Company
name : String
numberOfEmployees : Integer
stockPrice() : Real
employee
0..*
employer
0..*
wife
0..1
husband
0..1
Job
title : String
startDate : Date
salary : Integer
Marriage
place : String
date : Date

COLLECTION TYPES

- Bag - no order, may contain duplicates
- Set - no order, duplicates removed
- OrderedSet - ordered, duplicates removed
- Sequence - ordered, may contain duplicates