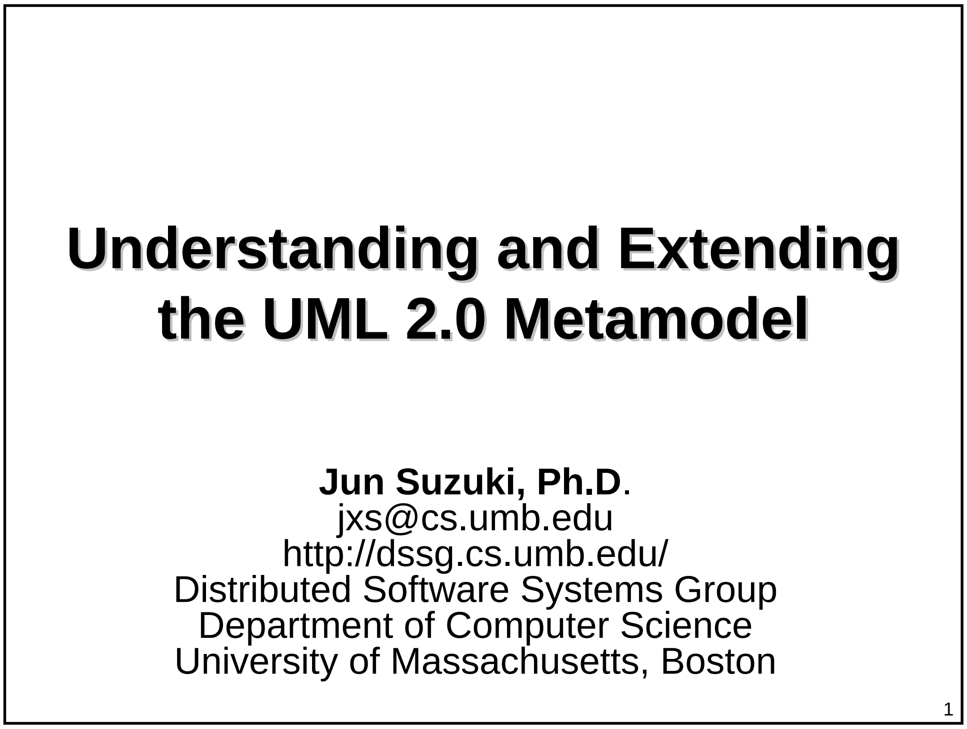
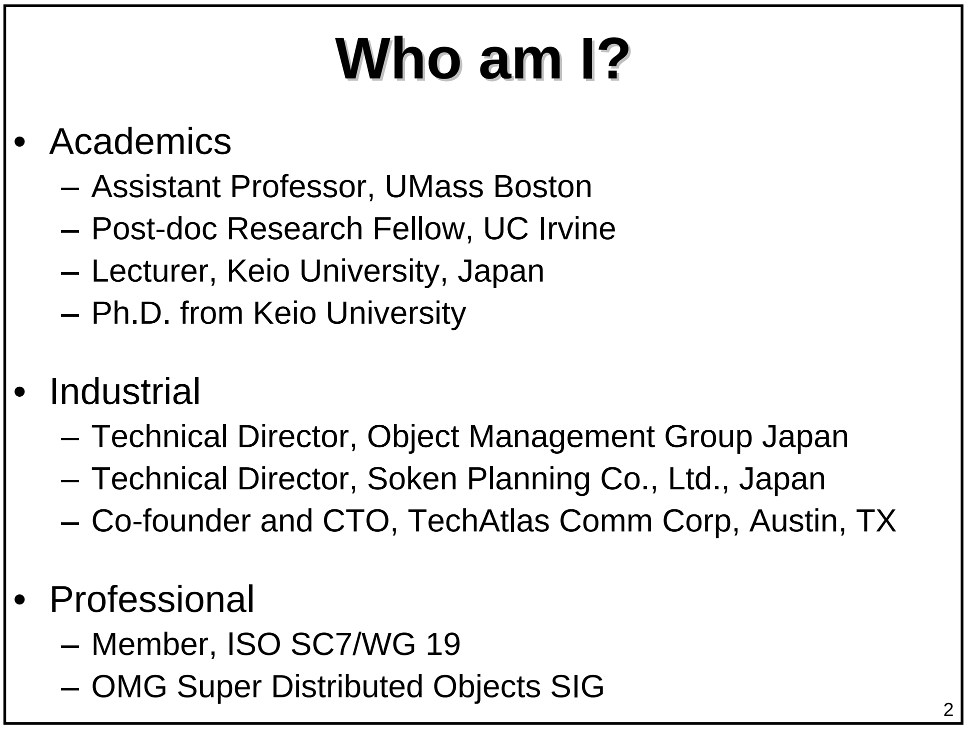


Understanding and Extending the UML 2.0 Metamodel

Jun Suzuki, Ph.D.
jxs@cs.umb.edu
http://dssg.cs.umb.edu/
Distributed Software Systems Group
Department of Computer Science
University of Massachusetts, Boston



Who am I?

- Academics
  - Assistant Professor, UMass Boston
  - Post-doc Research Fellow, UC Irvine
  - Lecturer, Keio University, Japan
  - Ph.D. from Keio University
- Industrial
  - Technical Director, Object Management Group Japan
  - Technical Director, Soken Planning Co., Ltd., Japan
  - Co-founder and CTO, TechAtlas Comm Corp, Austin, TX
- Professional
  - Member, ISO SC7/WG 19
  - OMG Super Distributed Objects SIG

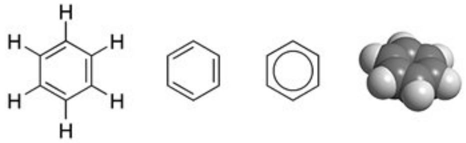
UMass Boston

- One of the 5 UMass system universities

UML Metamodel

UML

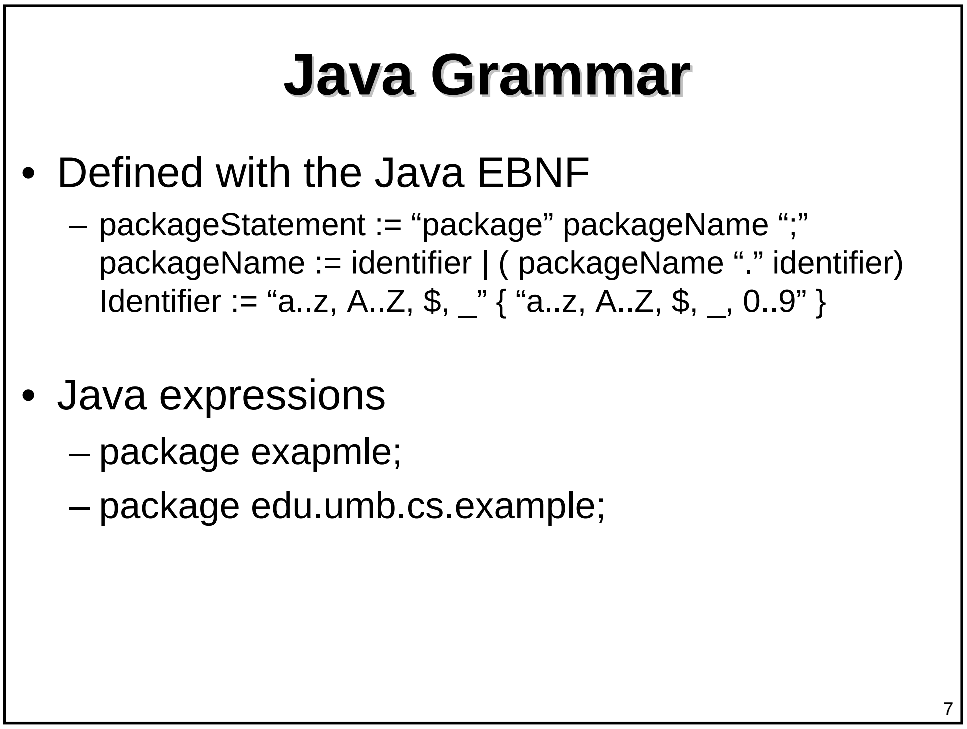
- Visual and intuitive
  - Easy to read and understand
  - Good communication tool among developers
    - c.f. electric diagrams, chemical equations, floor plans, musical scores, baseball scores
      - H_2O



Boston Red Sox	000 000 300	3
New York Yankees	000 000 000	0

UML Diagrams

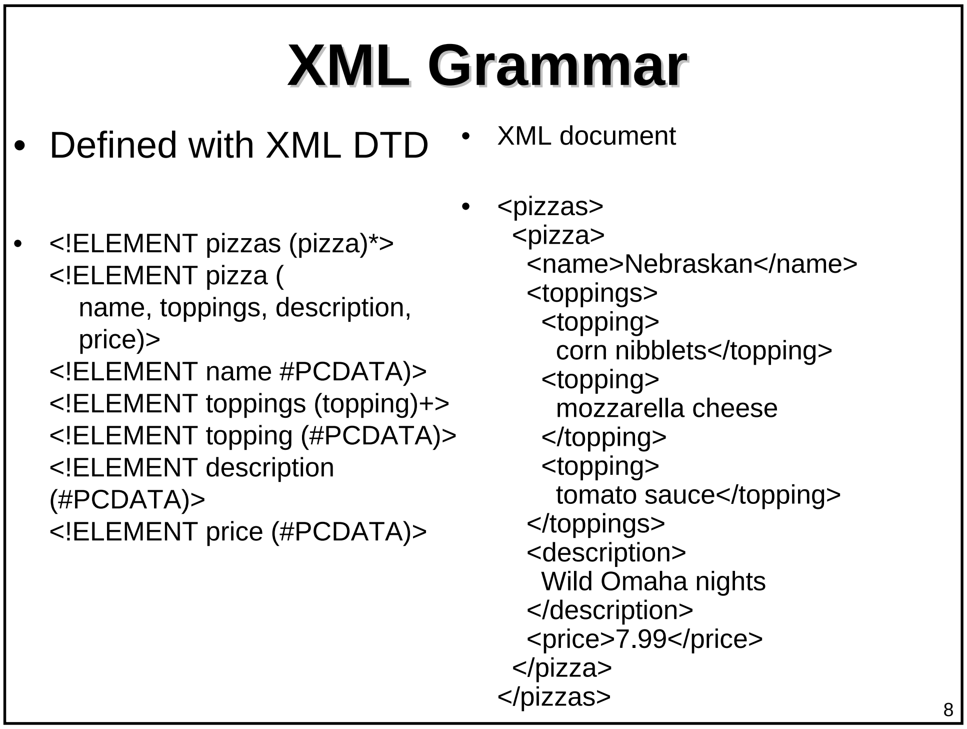
- UML diagrams
  - Class diagram
  - Sequence diagram
  - Interaction diagram
  - Activity diagram
  - …etc.
- UML is a modeling ***language***.
- In general, any language has its own grammar.
  - Syntax
    - The form or structure of expressions
  - Semantics
    - The meaning of expressions



Java Grammar

- Defined with the Java EBNF
  - packageStatement := “package” packageName “;”
    packageName := identifier **|** (packageName “.” identifier)
    Identifier := “a..z, A..Z, $, _” { “a..z, A..Z, $, _, 0..9” }

- Java expressions
  - package exapmle;
  - package edu.umb.cs.example;



XML Grammar

- Defined with XML DTD

-
```
<!ELEMENT pizzas (pizza)*>
<!ELEMENT pizza (
   name, toppings, description,
   price)>
<!ELEMENT name #PCDATA)>
<!ELEMENT toppings (topping)+>
<!ELEMENT topping (#PCDATA)>
<!ELEMENT description
(#PCDATA)>
<!ELEMENT price (#PCDATA)>
```

- XML document

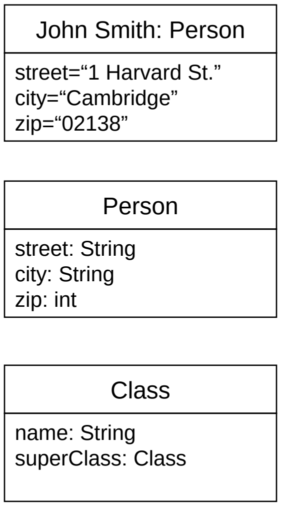
-
```
<pizzas>
  <pizza>
    <name>Nebraskan</name>
    <toppings>
      <topping>
        corn nibblets</topping>
      <topping>
        mozzarella cheese
      </topping>
      <topping>
        tomato sauce</topping>
    </toppings>
    <description>
      Wild Omaha nights
    </description>
    <price>7.99</price>
  </pizza>
</pizzas>
```

UML Grammar

- Defined with the UML metamodel
  - – Syntax
    - Abstract syntax
      - – How each UML model element is structured.
      - – How it is related to other model elements.
      - – using UML class diagrams
    - Well-formedness rules
      - – using OCL
    - Notation
      - – how each UML model element should be drawn.
  - – Semantics
    - using natural language

Metamodel Architecture

- M0 layer: instances
  - – Instances in a running system.
- M1: The model of instances
  - – Defines the concepts used to model M0 instances.
- M2: The model of M1 models
  - – Metamodel (modeling language)
  - – Defines the concepts used to model an M1 models.
- M3: The model of M2 models
  - – Meta-metamodel
  - – Defines the concepts used to model an M2 models.



John Smith: Person

street=“1 Harvard St.”
city=“Cambridge”
zip=“02138”

Person

street: String
city: String
zip: int

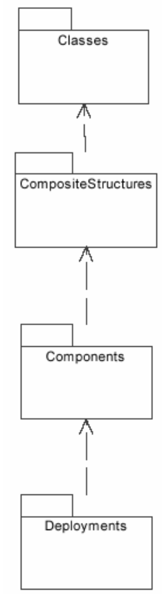
Class

name: String
superClass: Class

UML 2.0 Metamodel

- UML 2.0 superstructure specification
  - Revised final adopted spec
  - http://www.omg.org/cgi-bin/doc?ptc/2004-10-02
- Structural constructs
  - Used in structural diagrams such as class diagrams, object diagrams, composite structure diagrams, component diagrams and deployment diagrams.
- Behavioral constructs
  - Activities, Interactions (Sequence, communication, timing), State Machines, and Use cases
- Supplemental constructs
  - Auxiliary constructs, profile

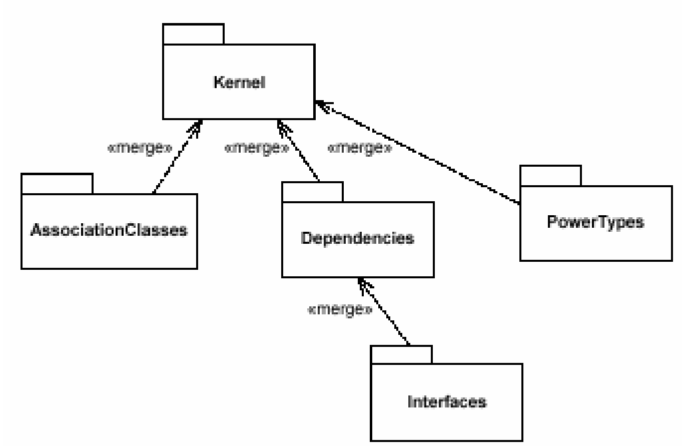
Structural Constructs



- Classes
  - Basic modeling concepts of UML, such as classes and their relationships
- CompositeStructures
  - Composition of interconnected elements representing run-time instances
- Components
  - Component-based system development or system structuring
- Deployments
  - execution architecture of systems that represent the assignment of software artifacts to hardware/software execution environment.

Classes Package

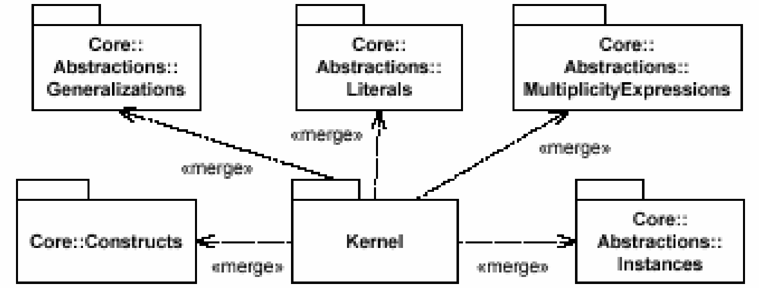
- Used to define basic modeling concepts of UML, such as classes and their relationships

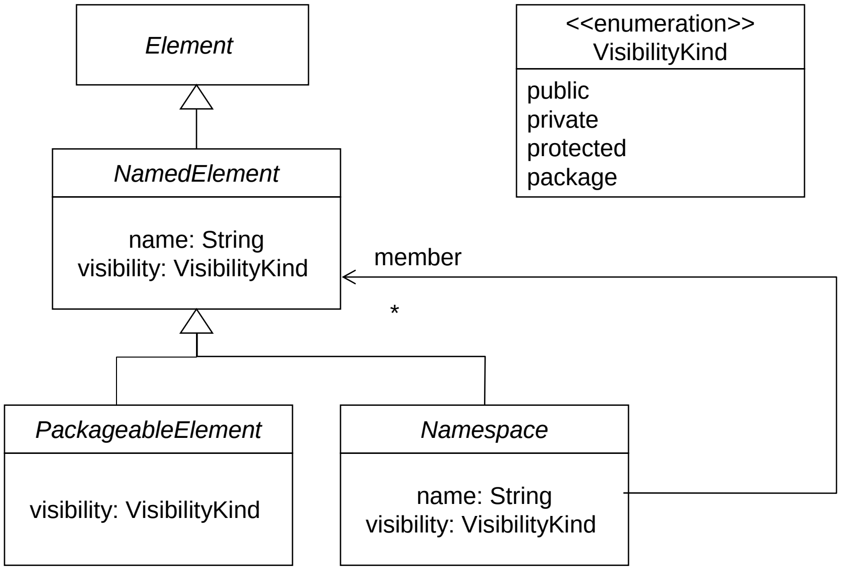


Kernel Package

- Represents core modeling concepts of UML.

- Heavily re-uses (merging) many elements defined in InfrastructureLibrary::Core package.





Element

<<enumeration>>
VisibilityKind

public
private
protected
package

NamedElement

name: String
visibility: VisibilityKind

member

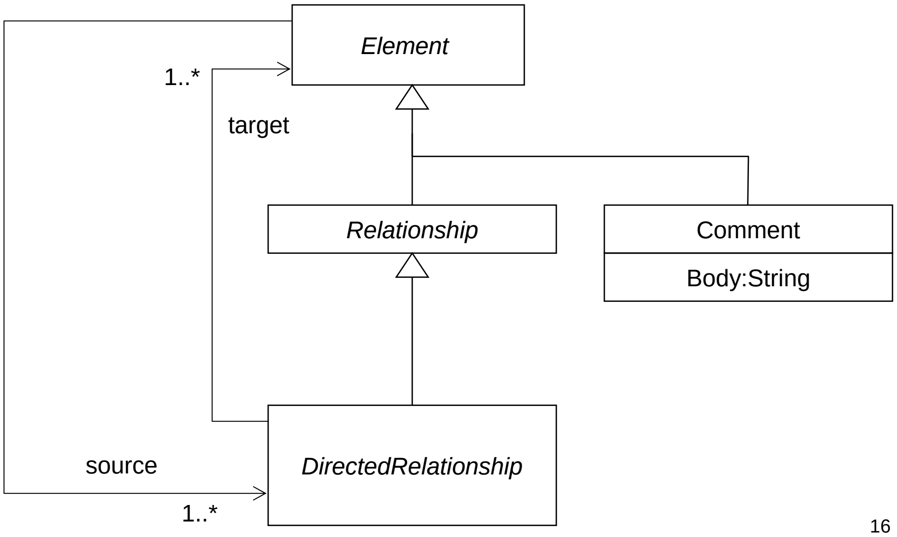
*

PackageableElement

visibility: VisibilityKind

Namespace

name: String
visibility: VisibilityKind



Element

1..*

target

Relationship

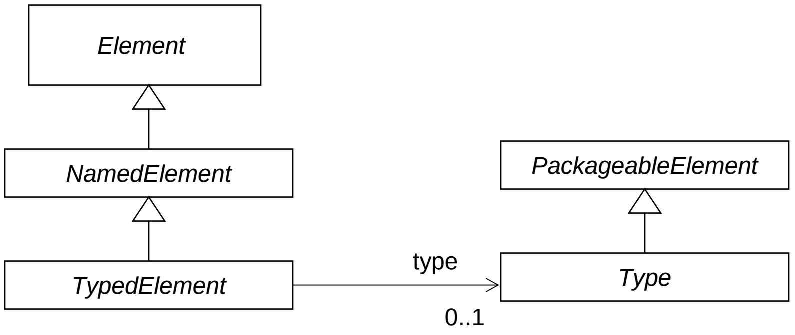
Comment

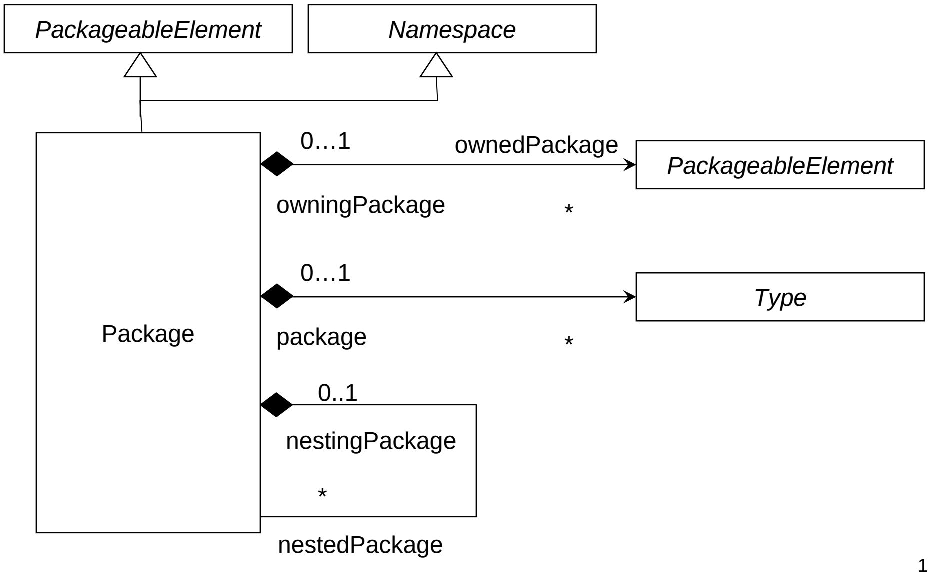
Body:String

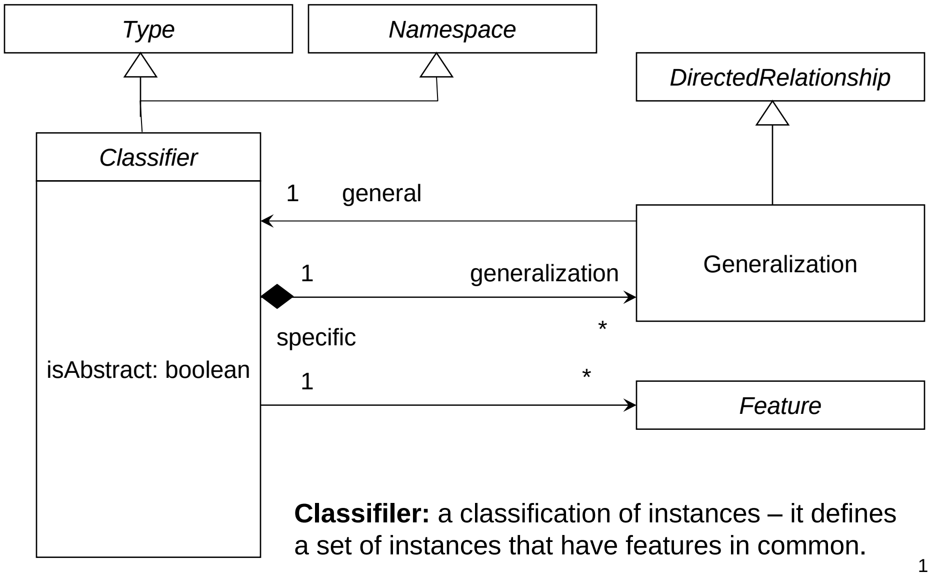
source

DirectedRelationship

1..*


Element
NamedElement
TypedElement
type
0..1
PackageableElement
Type


PackageableElement
Namespace
Package
0…1
ownedPackage
PackageableElement
owningPackage
*
0…1
Type
package
*
0..1
nestingPackage
*
nestedPackage



Type

Namespace

DirectedRelationship

Classifier

isAbstract: boolean

1 general

1 generalization

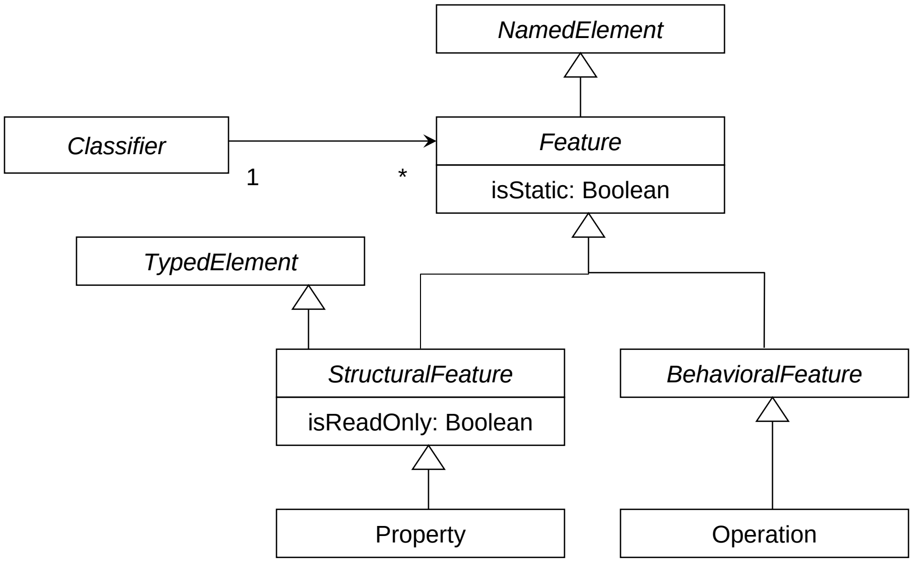
specific *

Generalization

1 *

Feature

Classifiler: a classification of instances – it defines a set of instances that have features in common.



NamedElement

Classifier

1 *

Feature

isStatic: Boolean

TypedElement

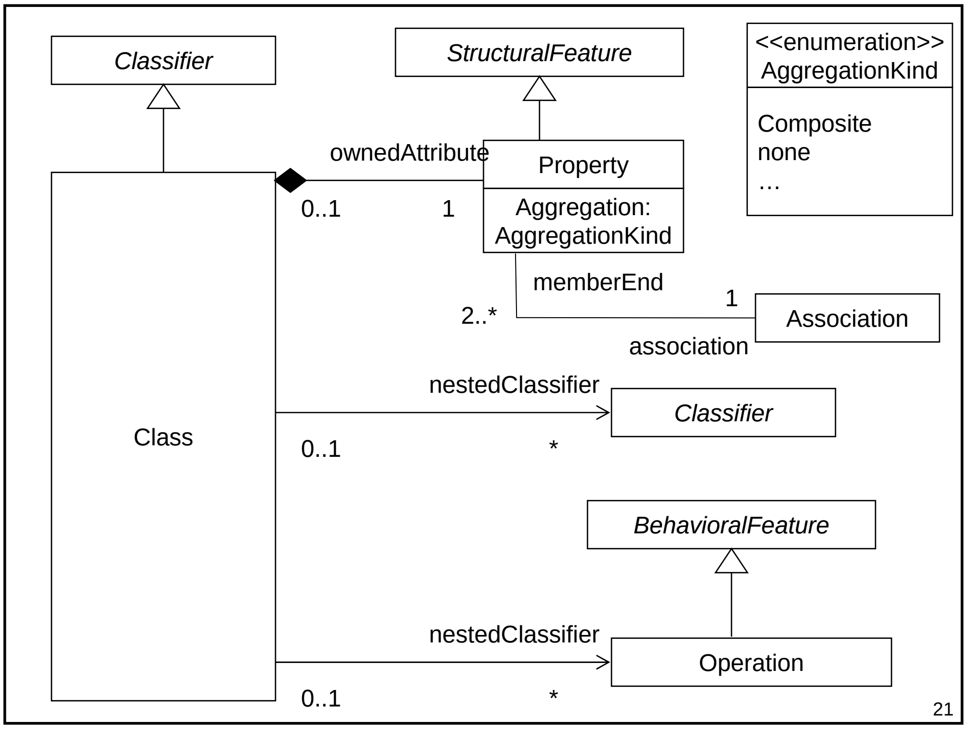
StructuralFeature

isReadOnly: Boolean

BehavioralFeature

Property

Operation



Classifier

StructuralFeature

<<enumeration>>
AggregationKind

Composite
none
…

ownedAttribute

Property

Aggregation:
AggregationKind

0..1

1

memberEnd

1

2..*

Association

association

nestedClassifier

Classifier

Class

0..1

*

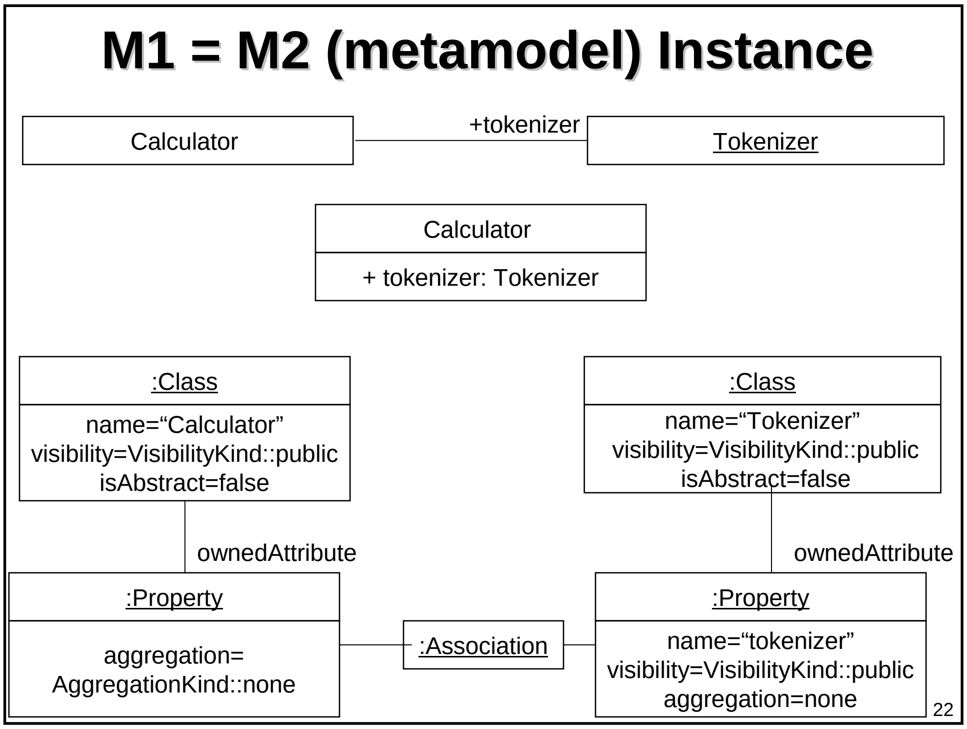
BehavioralFeature

nestedClassifier

Operation

0..1

*



M1 = M2 (metamodel) Instance

Calculator

+tokenizer

Tokenizer

Calculator

+ tokenizer: Tokenizer

:Class

name="Calculator"
visibility=VisibilityKind::public
isAbstract=false

:Class

name="Tokenizer"
visibility=VisibilityKind::public
isAbstract=false

ownedAttribute

ownedAttribute

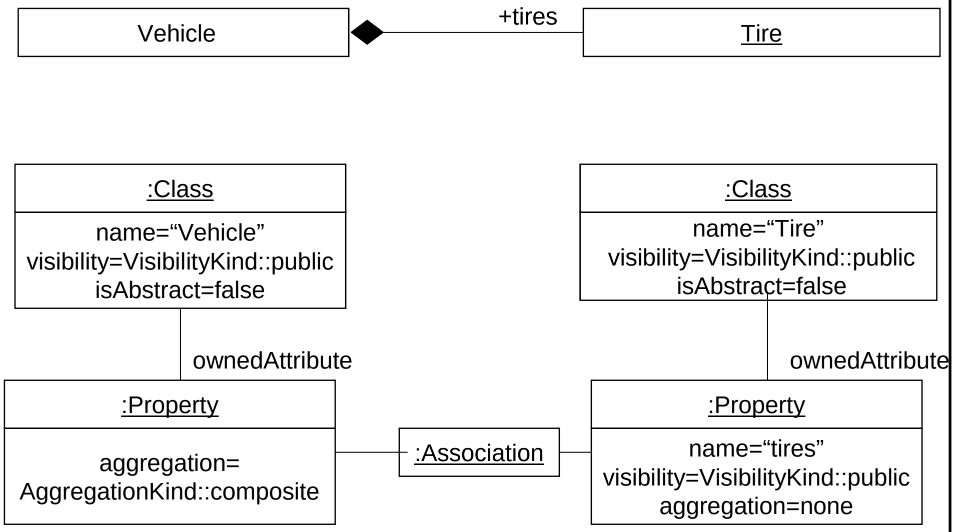
:Property

aggregation=
AggregationKind::none

:Association

:Property

name="tokenizer"
visibility=VisibilityKind::public
aggregation=none



Vehicle

+tires

Tire

:Class

name=“Vehicle”
visibility=VisibilityKind::public
isAbstract=false

:Class

name=“Tire”
visibility=VisibilityKind::public
isAbstract=false

ownedAttribute

ownedAttribute

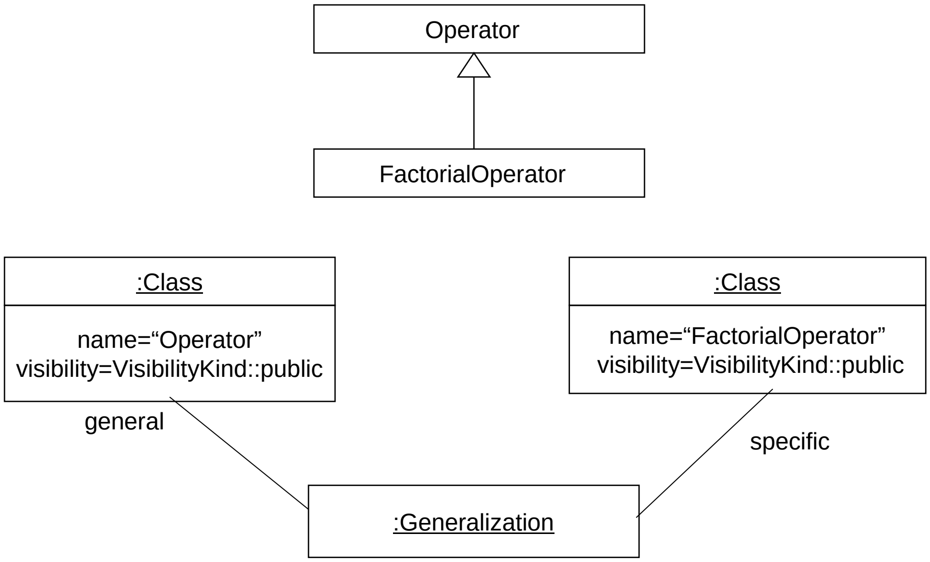
:Property

aggregation=
AggregationKind::composite

:Association

:Property

name=“tires”
visibility=VisibilityKind::public
aggregation=none



Operator

FactorialOperator

:Class

name=“Operator”
visibility=VisibilityKind::public

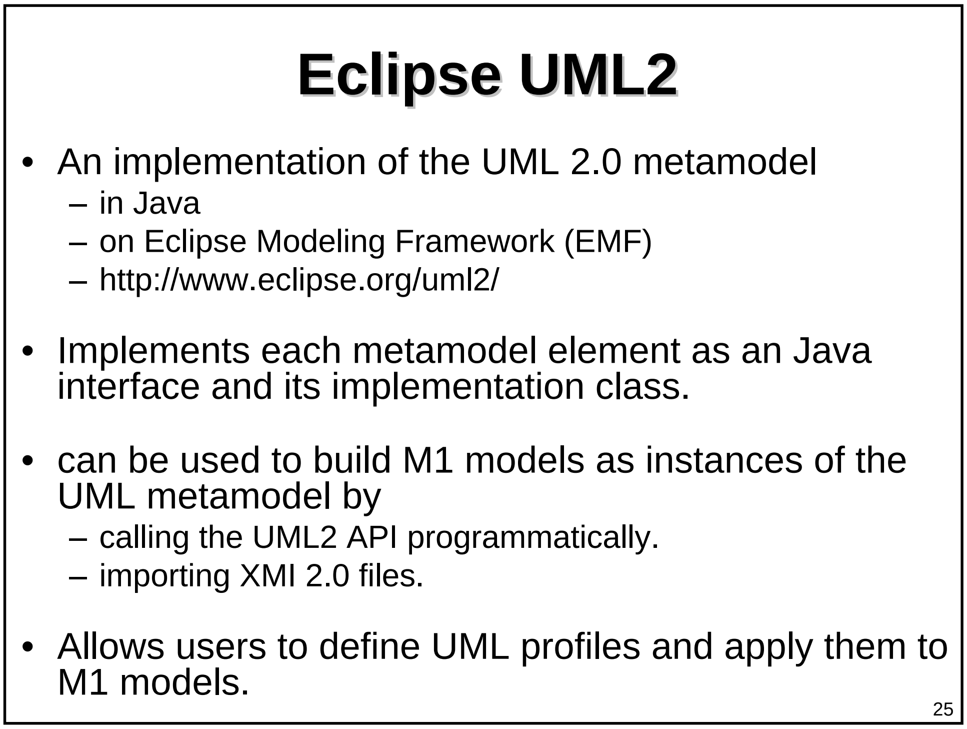
:Class

name=“FactorialOperator”
visibility=VisibilityKind::public

general

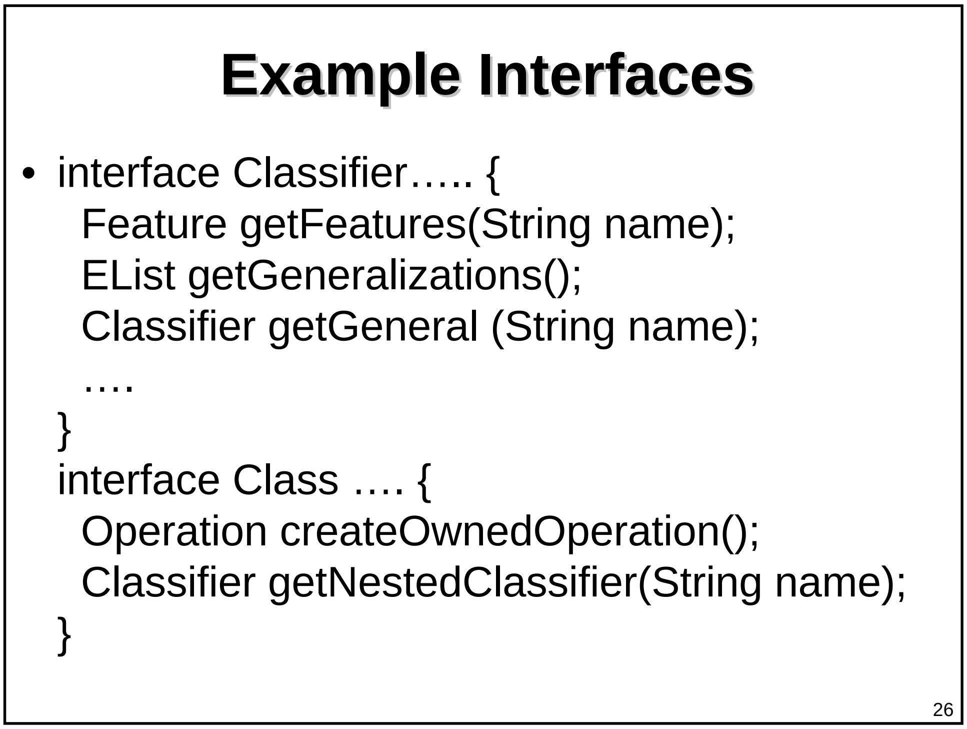
specific

:Generalization



Eclipse UML2

- An implementation of the UML 2.0 metamodel
  - in Java
  - on Eclipse Modeling Framework (EMF)
  - http://www.eclipse.org/uml2/
- Implements each metamodel element as an Java interface and its implementation class.
- can be used to build M1 models as instances of the UML metamodel by
  - calling the UML2 API programmatically.
  - importing XMI 2.0 files.
- Allows users to define UML profiles and apply them to M1 models.

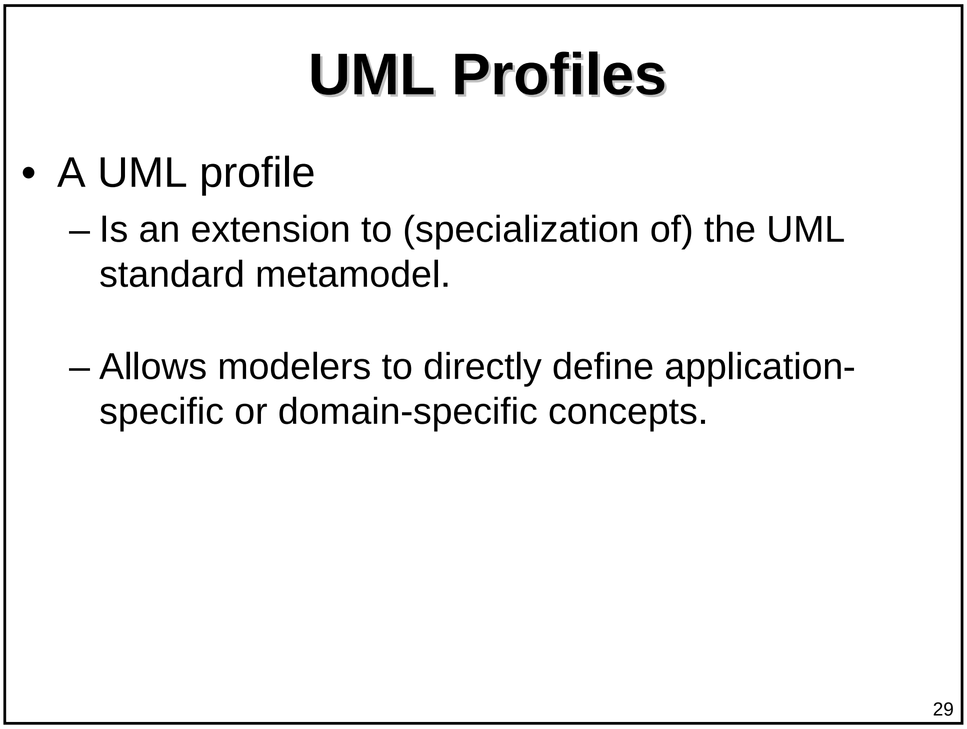


Example Interfaces

-
```
interface Classifier….. {
  Feature getFeatures(String name);
  EList getGeneralizations();
  Classifier getGeneral (String name);
  ….
}
interface Class …. {
  Operation createOwnedOperation();
  Classifier getNestedClassifier(String name);
}
```

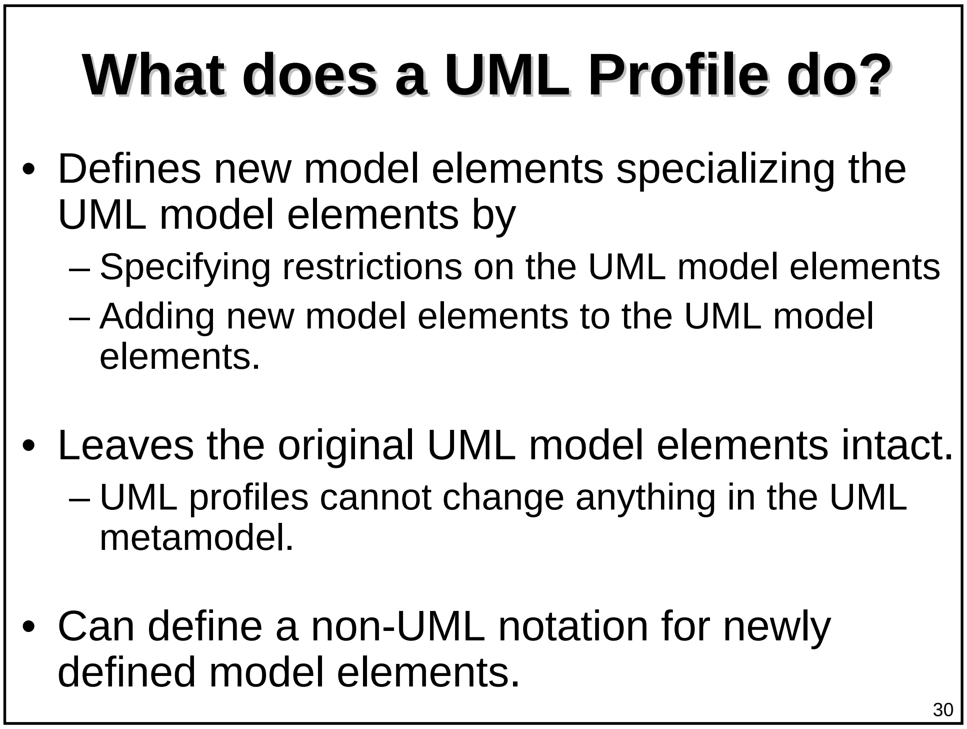
- MTF: Model Tranformation Framework
  - – From IBM
  - – On UML2

UML Profiles



UML Profiles

- A UML profile
  - Is an extension to (specialization of) the UML standard metamodel.
  - Allows modelers to directly define application-specific or domain-specific concepts.



What does a UML Profile do?

- Defines new model elements specializing the UML model elements by
  - Specifying restrictions on the UML model elements
  - Adding new model elements to the UML model elements.
- Leaves the original UML model elements intact.
  - UML profiles cannot change anything in the UML metamodel.
- Can define a non-UML notation for newly defined model elements.

When to Use UML Profiles?

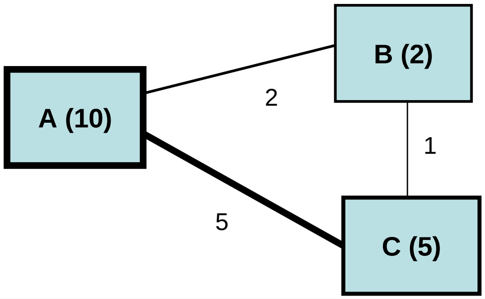
- Have domain/application/platform specific terminology
  - e.g. EJB entity bean
- Add semantics that do not exist in the standard metamodel
- Add constraints that restrict the way to use the standard metamodel.
- Add information used to transform a model to another model or code.
- Have a different notation for standard symbols/icons in UML

How to Define a UML Profile?

- Each UML profile is defined with the UML extension mechanism.
  - Stereotypes
  - Constraints
  - Tagged values
- You will define stereotypes, constraints and tagged values against the UML metamodel.

A Simple Example

- UML profile for weighted, labeled graphs (WL-graphs)
  - – Concepts
    - Node (or vertex)
    - Label associated with a node
    - Link (or edge)
    - Weight associated with a node and link
  - – Rule
    - A weighted link is connected with weighted nodes.



Users may want to use
WL-graphs for network routing
analysis, airline company's
route optimization, etc. etc.

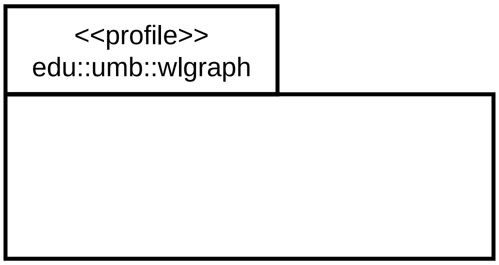
A Typical Process to Define a UML Profile

- Define a UML package that contain profile constructs.
- Define stereotypes.
- Define tagged values, as meta-attributes of stereotypes.
- Define constraints for stereotypes, tagged values and/or UML metamodel elements.

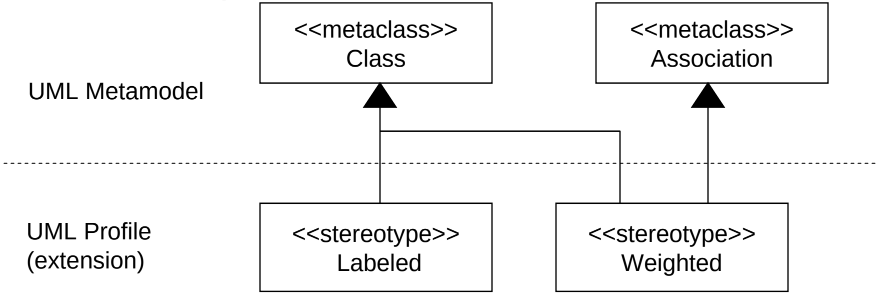
Package of UML Profile for WL-Graphs

- Each UML profile is defined in a UML package stereotyped <<profile>>
  - – Extending the UML metamodel or other profiles.

- A <<profile>> package contains stereotypes and tagged values.

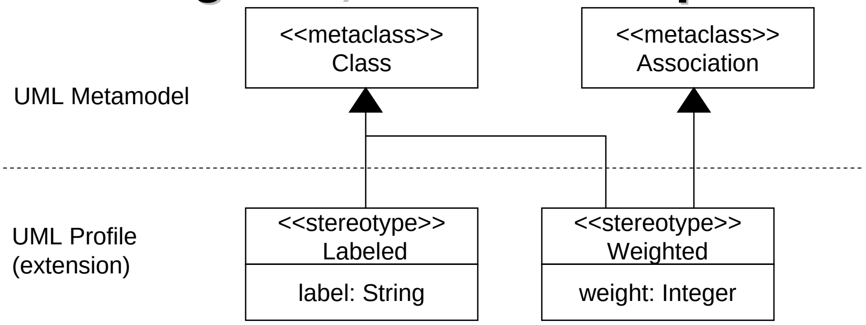


Stereotypes in UML Profile for Weighted, Labeled Graphs



- “Labeled” and “Weighted”
  - – are specializations of Class.
  - – inherits the semantics of Class.
  - – are applied to classes at M1 (user-defined) models.
- “Weighted”
  - – is a specialization of Association.
  - – Inherits the semantics of Association.
  - – Are applied to associations at M1 (user-defined) models.

Tagged-Values in UML Profile for Weighted, Labeled Graphs



- Each tagged value
  - consists of a name and its type.
  - is associated with a specific stereotype.
  - is graphically described as an attribute of a stereotype class.

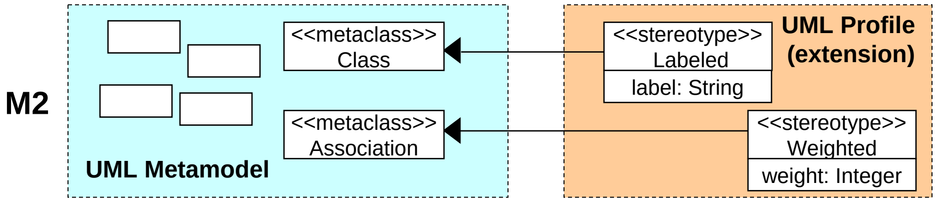
Constraints in UML Profile for Weighted, Labeled Graphs

- Constraints for stereotypes, tagged values and/or UML metamodel elements in
  - Natural language and/or
    - A weighted link is connected with weighted nodes.
    - Weight and label variable shall not be empty.
  - Object Constraint Language (OCL)
    - **context:** UML::infrastructureLibrary::Core::Constructs::Association
      inv: self.isStereotyped(“Weighted”) implies
      self.connection->forAll(isStereotyped(“Weighted”))

      context: edu.umb.wlgraph.Weighted
      inv: self.weight->notEmpty()

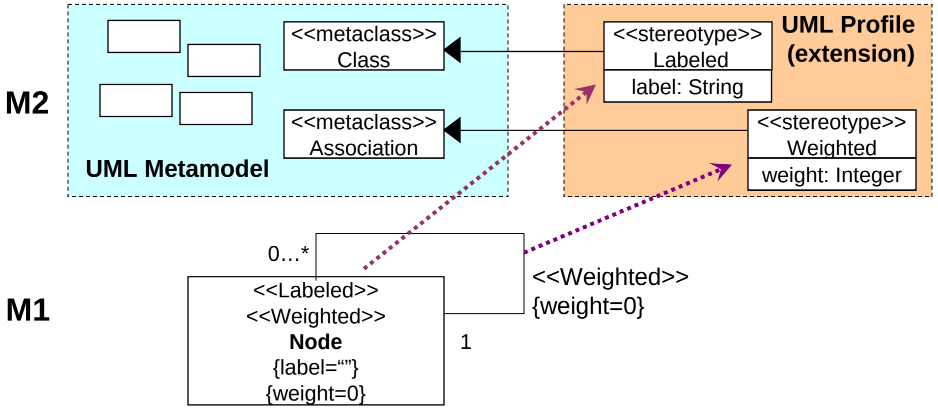
      context: edu.umb.wlgraph.Labeled
      inv: self.label->notEmpty()

How to Use UML Profile for Weighted, Labeled Graphs



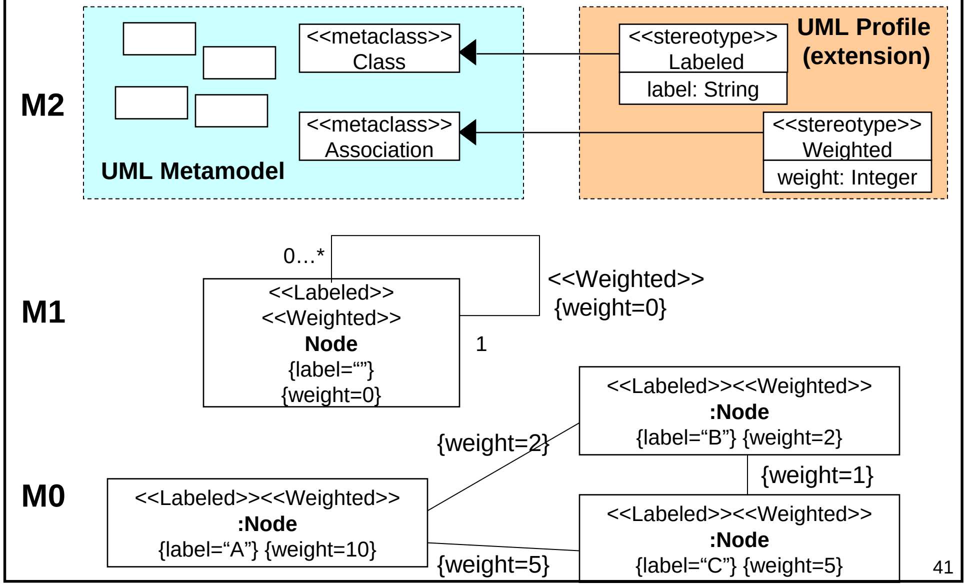
39

How to Use UML Profile for Weighted, Labeled Graphs

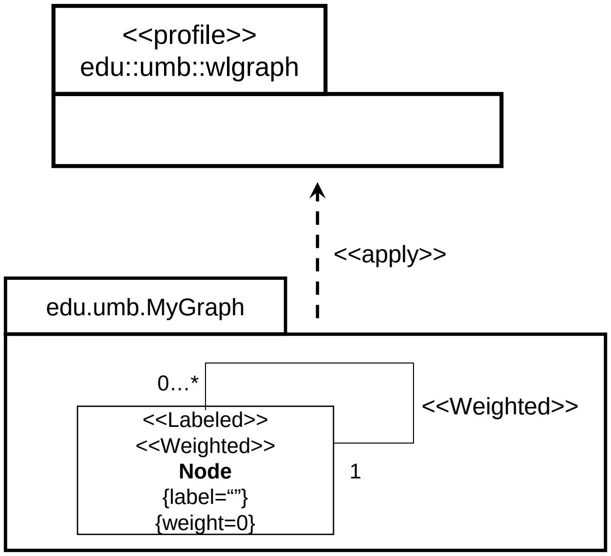


40

How to Use UML Profile for Weighted, Labeled Graphs



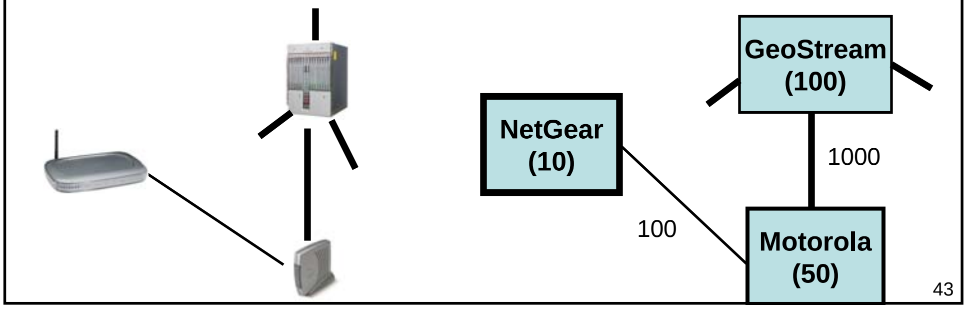
41



42

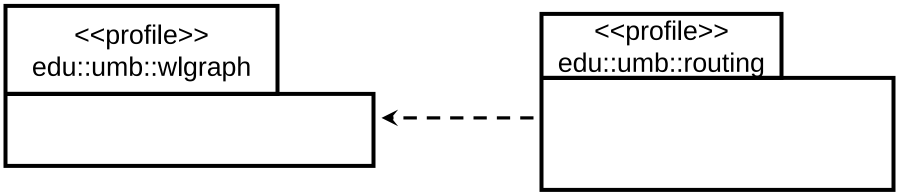
Another Simple Example

- UML profile for network routing analysis
  - Concepts
    - Router
    - Packet queue size in a router
    - Network link
    - Bandwidth available on a network link
  - Rule
    - A weighted link is connected with weighted nodes.

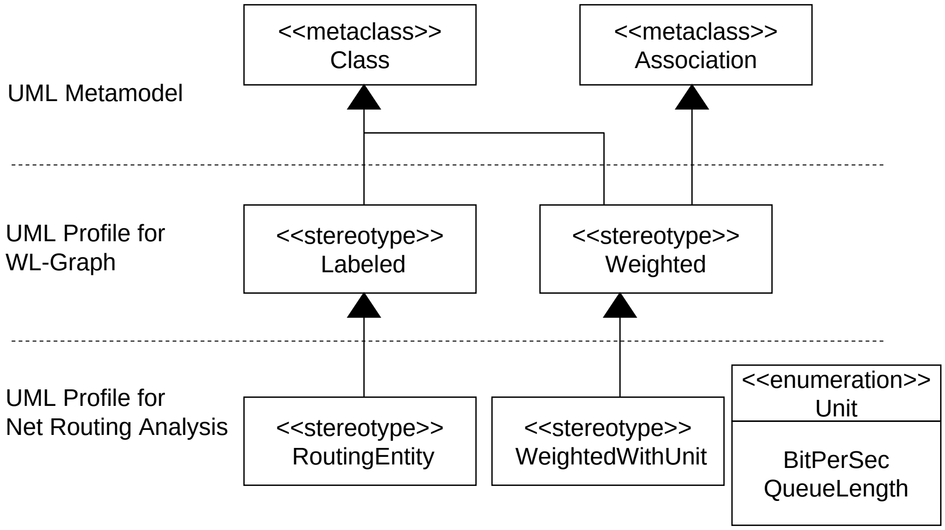


Package of UML Profile for Network Routing Analysis

- Each UML profile is defined in a UML package stereotyped <<profile>>
  - Extending the UML metamodel or other profiles.
- A <<profile>> package contains stereotypes and tagged values.



Stereotypes in UML Profile for Network Routing Analysis



UML Metamodel

<<metaclass>>
Class

<<metaclass>>
Association

UML Profile for
WL-Graph

<<stereotype>>
Labeled

<<stereotype>>
Weighted

UML Profile for
Net Routing Analysis

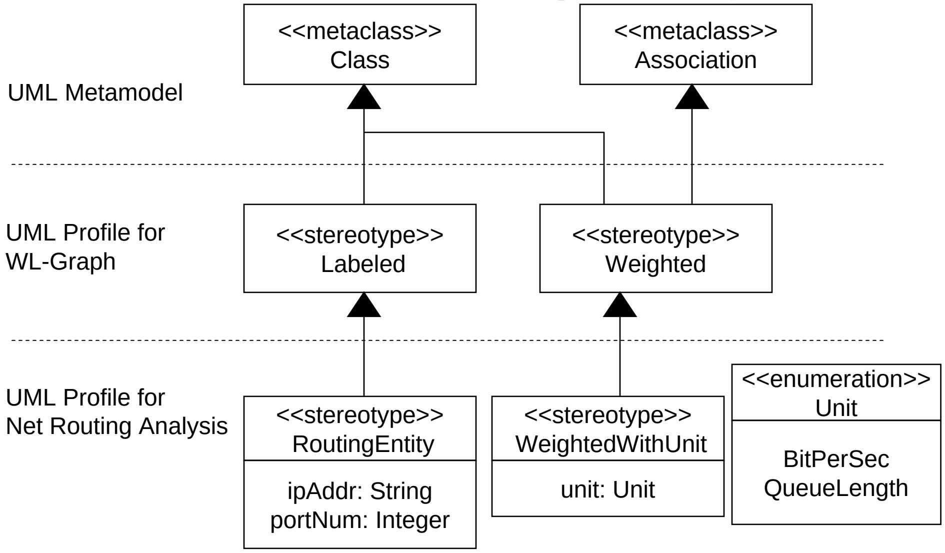
<<stereotype>>
RoutingEntity

<<stereotype>>
WeightedWithUnit

<<enumeration>>
Unit

BitPerSec
QueueLength

Tagged-Values in UML Profile for Network Routing Analysis



UML Metamodel

<<metaclass>>
Class

<<metaclass>>
Association

UML Profile for
WL-Graph

<<stereotype>>
Labeled

<<stereotype>>
Weighted

UML Profile for
Net Routing Analysis

<<stereotype>>
RoutingEntity

ipAddr: String
portNum: Integer

<<stereotype>>
WeightedWithUnit

unit: Unit

<<enumeration>>
Unit

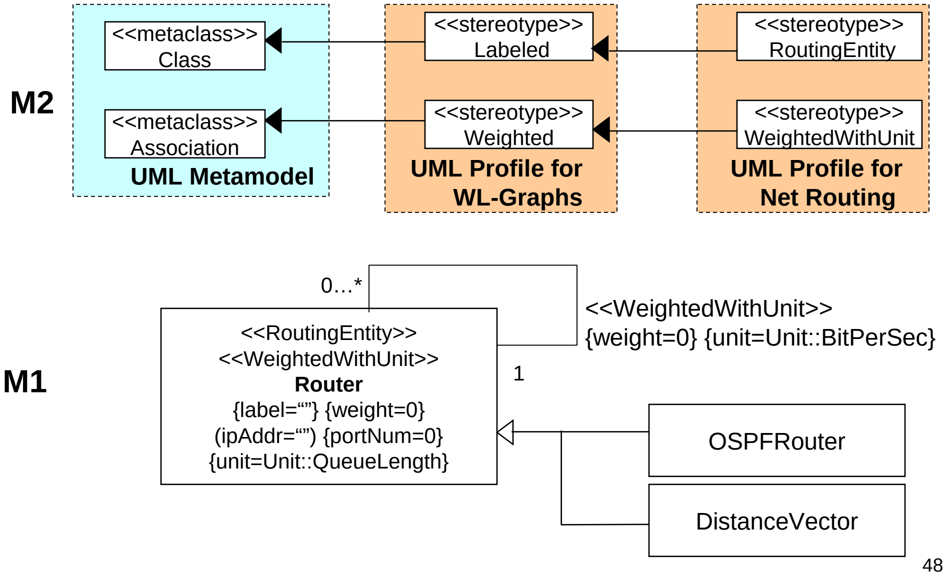
BitPerSec
QueueLength

Constraints in UML Profile for Network Routing Analysis

- Object Constraint Language (OCL)
  - **context:** edu.umb.routing.RoutingEntity
    inv: self.ipAddr->notEmpty()
    inv: self.portNum->notEmpty()

    context: edu.umb.routing.WeightedWithUnit
    inv: self.unit->notEmpty()

How to Use UML Profile for Network Routing Analysis



M2

<<metaclass>>
Class

<<metaclass>>
Association

UML Metamodel

<<stereotype>>
Labeled

<<stereotype>>
Weighted

UML Profile for WL-Graphs

<<stereotype>>
RoutingEntity

<<stereotype>>
WeightedWithUnit

UML Profile for Net Routing

M1

0…*

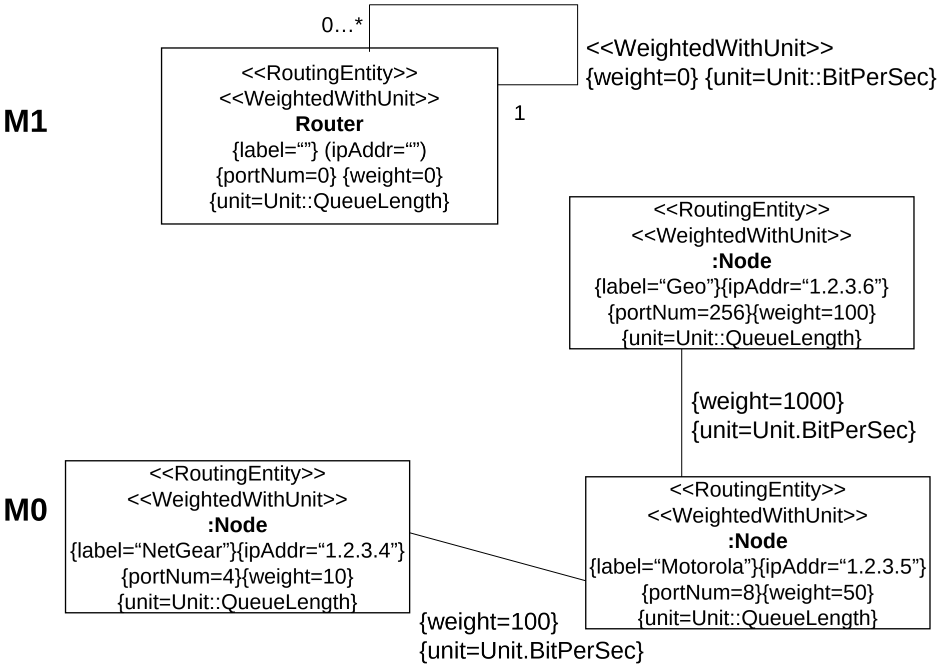
<<WeightedWithUnit>>
{weight=0} {unit=Unit::BitPerSec}

1

<<RoutingEntity>>
<<WeightedWithUnit>>
Router
{label=“”} {weight=0}
(ipAddr=“”) {portNum=0}
{unit=Unit::QueueLength}

OSPFRouter

DistanceVector



M1

<<RoutingEntity>>
<<WeightedWithUnit>>
Router
{label=“”} (ipAddr=“”)
{portNum=0} {weight=0}
{unit=Unit::QueueLength}

0…*

1

<<WeightedWithUnit>>
{weight=0} {unit=Unit::BitPerSec}

<<RoutingEntity>>
<<WeightedWithUnit>>
:Node
{label=“Geo”}{ipAddr=“1.2.3.6”}
{portNum=256}{weight=100}
{unit=Unit::QueueLength}

{weight=1000}
{unit=Unit.BitPerSec}

M0

<<RoutingEntity>>
<<WeightedWithUnit>>
:Node
{label=“NetGear”}{ipAddr=“1.2.3.4”}
{portNum=4}{weight=10}
{unit=Unit::QueueLength}

<<RoutingEntity>>
<<WeightedWithUnit>>
:Node
{label=“Motorola”}{ipAddr=“1.2.3.5”}
{portNum=8}{weight=50}
{unit=Unit::QueueLength}

{weight=100}
{unit=Unit.BitPerSec}

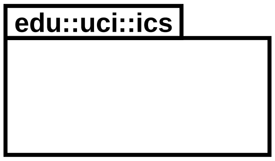
UML Profile for EJB

- JSR26: http://jcp.org/en/jsr/detail?id=026
- Used to…
  - directly model EJB concepts in UML
  - specialize platform independent (UML) models to EJB specific models.

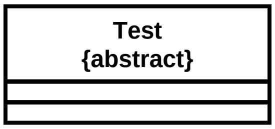
- What *UML Profile for EJB* defines include:
  - – Design model
    - Java design model
    - EJB design model
      - – External model
      - – Internal model
  - – Implementation model
    - Java implementation model
    - EJB implementation model

Java Design Model

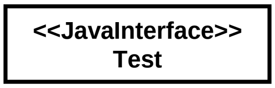
- Defines UML representations of Java language constructs
  - – Java class, interface, etc.
- Java package
  - – mapped to a UML package
  - – e.g. *package edu.uci.ics;*



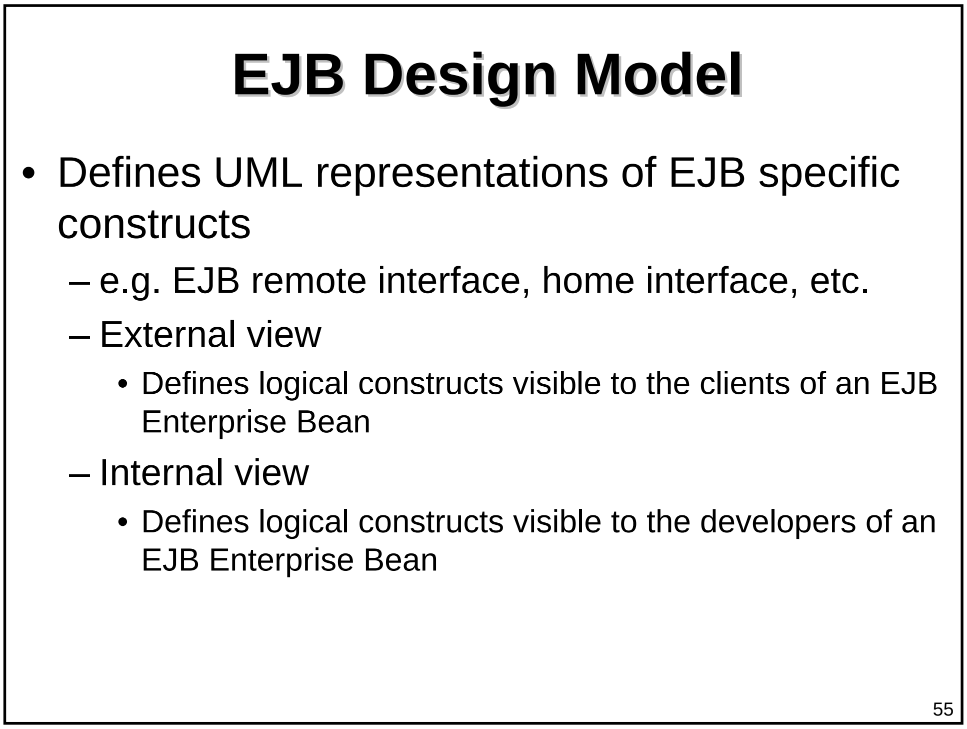
- Java Class
  - – mapped to a UML class
  - – e.g. *public abstract class Test {}*


Test
{abstract}

- Java Interface
  - – mapped to a UML interface or UML class stereotyped as <<JavaInterface>>.
  - – e.g. *public interface Test {}*

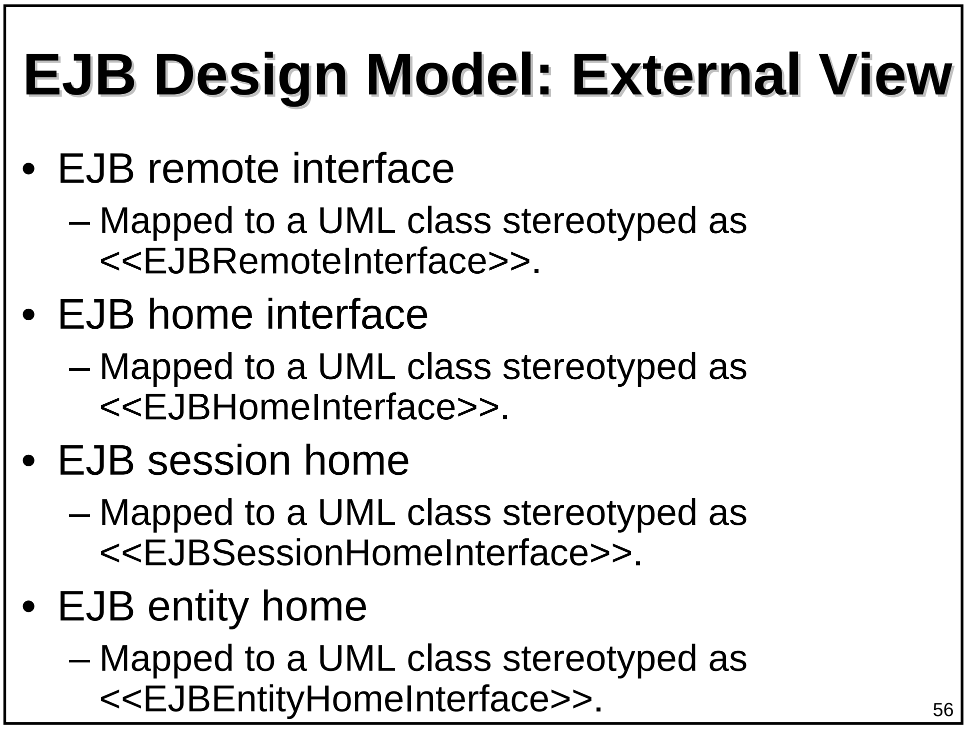

<<JavaInterface>>
Test

- Java method
  - – mapped to a UML operation
  - – e.g. *public void test() throws Foo{}*
    - + test(): void {JavaThrows=Foo}
- Others
  - – Single type import
  - – On demand type import



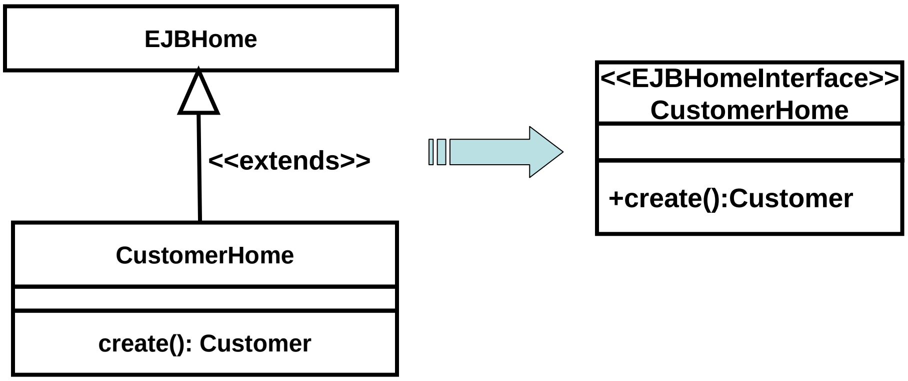
EJB Design Model

- Defines UML representations of EJB specific constructs
  - e.g. EJB remote interface, home interface, etc.
  - External view
    - Defines logical constructs visible to the clients of an EJB Enterprise Bean
  - Internal view
    - Defines logical constructs visible to the developers of an EJB Enterprise Bean

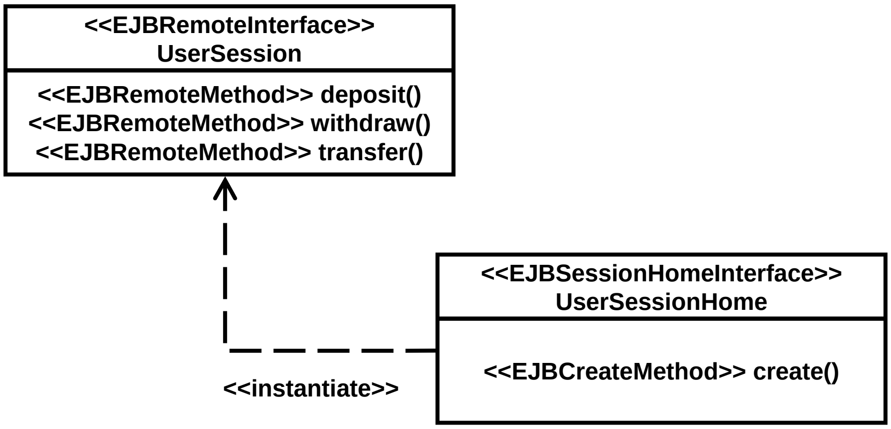


EJB Design Model: External View

- EJB remote interface
  - Mapped to a UML class stereotyped as <<EJBRemoteInterface>>.
- EJB home interface
  - Mapped to a UML class stereotyped as <<EJBHomeInterface>>.
- EJB session home
  - Mapped to a UML class stereotyped as <<EJBSessionHomeInterface>>.
- EJB entity home
  - Mapped to a UML class stereotyped as <<EJBEntityHomeInterface>>.



- EJB Method
  - Means methods declared in EJB Remote and Home interfaces
  - Mapped to a UML operation
  - <<EJBCreateMethod>>
    - Represents a create method in a home interface
  - <<EJBFinderMethod>>
    - Represents a finder interface in a home interface
  - <<EJBRemoteMethod>>
    - Represents a method in a remote interface.



<<EJBRemoteInterface>>
UserSession

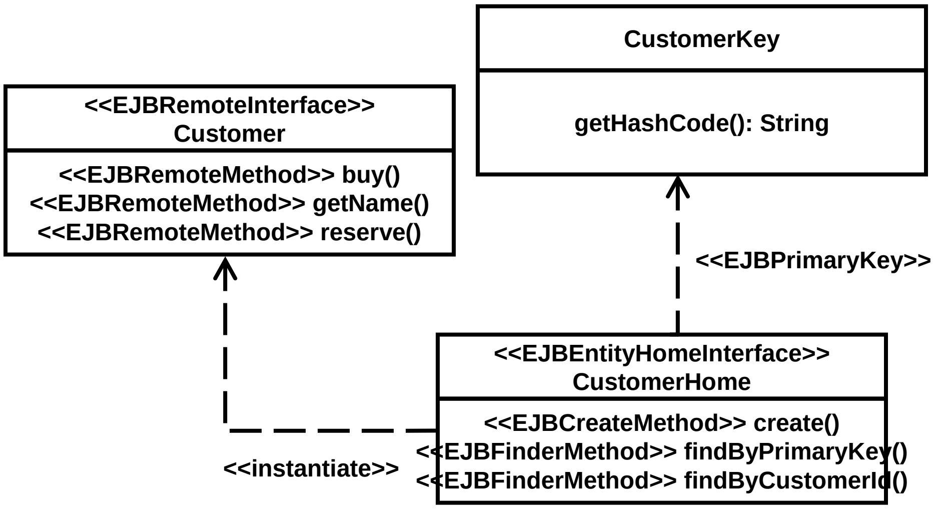
<<EJBRemoteMethod>> deposit()
<<EJBRemoteMethod>> withdraw()
<<EJBRemoteMethod>> transfer()

<<EJBSessionHomeInterface>>
UserSessionHome

<<EJBCreateMethod>> create()

<<instantiate>>

- EJB primary key
  - Mapped to a UML usage association stereotyped as <<EJBPrimaryKey>>.
    - between EJB primary key class and EJB entity home



CustomerKey

getHashCode(): String

<<EJBRemoteInterface>>
Customer

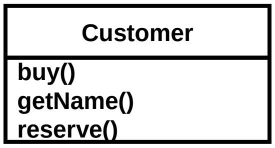
<<EJBRemoteMethod>> buy()
<<EJBRemoteMethod>> getName()
<<EJBRemoteMethod>> reserve()

<<EJBPrimaryKey>>

<<EJBEntityHomeInterface>>
CustomerHome

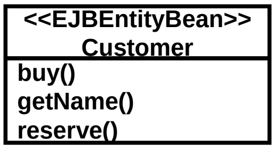
<<EJBCreateMethod>> create()
<<EJBFinderMethod>> findByPrimaryKey()
<<EJBFinderMethod>> findByCustomerId()

<<instantiate>>



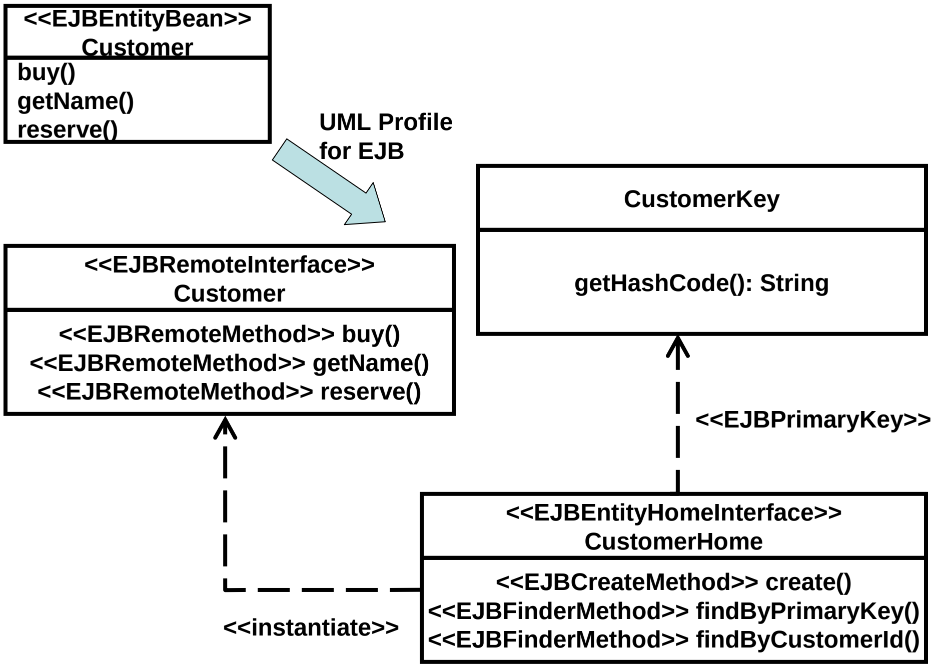
Customer

buy()
getName()
reserve()



<<EJBEntityBean>>
Customer

buy()
getName()
reserve()



<<EJBEntityBean>>
Customer

buy()
getName()
reserve()

UML Profile
for EJB

CustomerKey

getHashCode(): String

<<EJBRemoteInterface>>
Customer

<<EJBRemoteMethod>> buy()
<<EJBRemoteMethod>> getName()
<<EJBRemoteMethod>> reserve()

<<EJBPrimaryKey>>

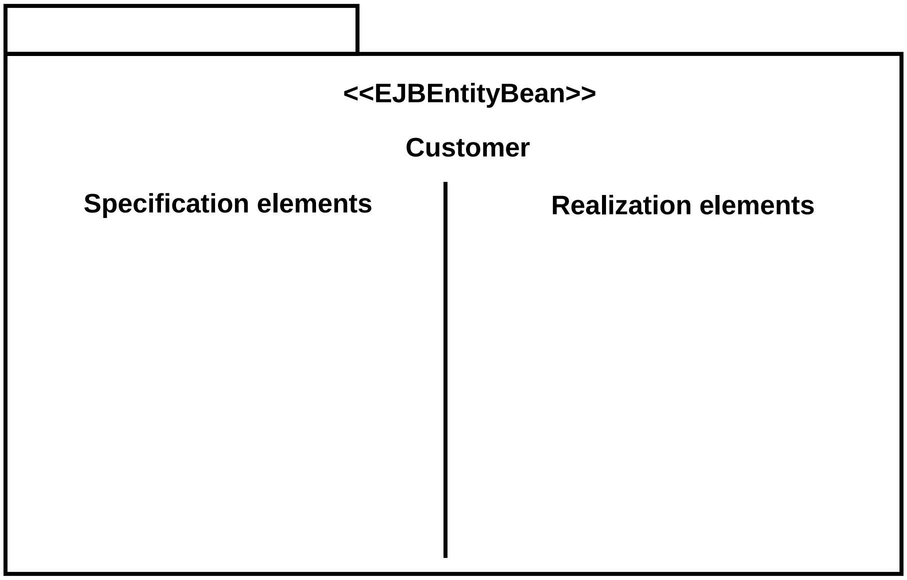
<<EJBEntityHomeInterface>>
CustomerHome

<<EJBCreateMethod>> create()
<<EJBFinderMethod>> findByPrimaryKey()
<<EJBFinderMethod>> findByCustomerId()

<<instantiate>>

EJB Design Model: Internal View

- EJB enterprise bean
  - Mapped to a UML subsystem stereotyped as <<EJBEnterpriseBean>>.
- EJB session bean
  - Mapped to a UML subsystem stereotyped as <<EJBSessionBean>>.
- EJB entity bean
  - Mapped to a UML subsystem stereotyped as <<EJBEntityBean>>
  - <<EJBCmpField>> represents a container-managed field (attribute).



<<EJBEntityBean>>

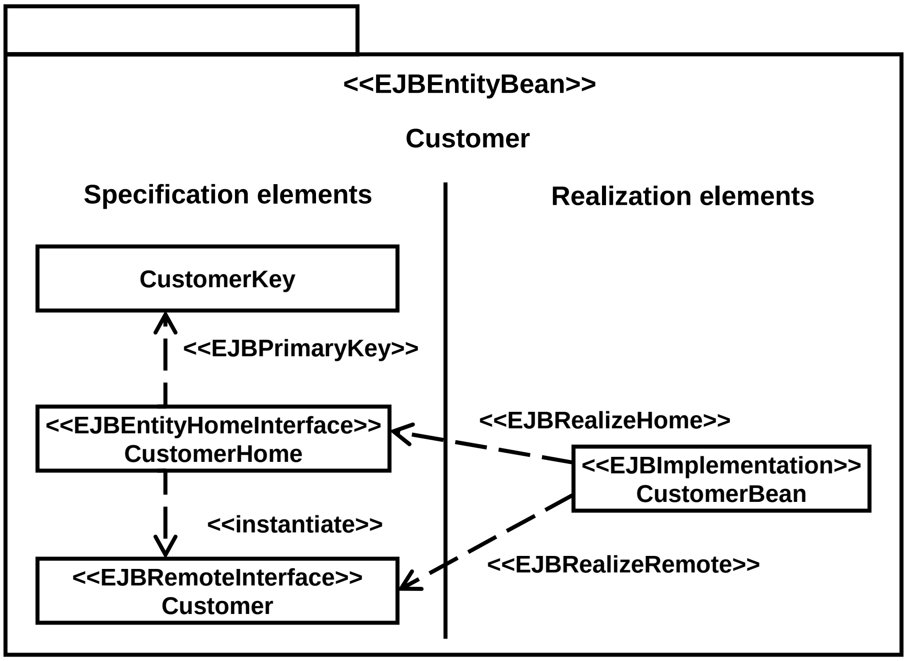
Customer

Specification elements

Realization elements

- EJB enterprise bean is declared by
  - an EJB home interface,
  - an EJB remote interface,
  - an EJB implementation class
  - Supplemental Java classes and interfaces, and
  - EJB deployment descriptor.

- EJB implementation class
  - Mapped to a UML class stereotyped as <<EJBImplementation>>.
- EJB remote interface
  - Mapped to a UML abstraction association stereotyped as <<EJBRealizeRemote>>.
    - between EJB remote interface and EJB implementation class.
- EJB home interface
  - Mapped to a UML abstraction association stereotyped as <<EJBRealizeHome>>.
    - between EJB home interface and EJB implementation class.



<<EJBEntityBean>>

Customer

Specification elements

Realization elements

CustomerKey

<<EJBPrimaryKey>>

<<EJBEntityHomeInterface>>
CustomerHome

<<EJBRealizeHome>>

<<EJBImplementation>>
CustomerBean

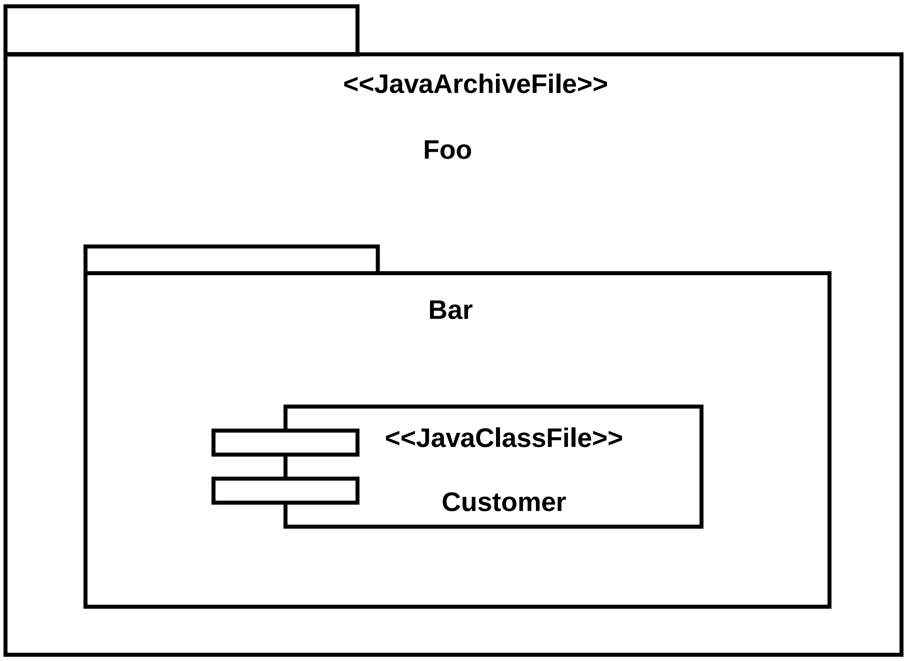
<<instantiate>>

<<EJBRemoteInterface>>
Customer

<<EJBRealizeRemote>>

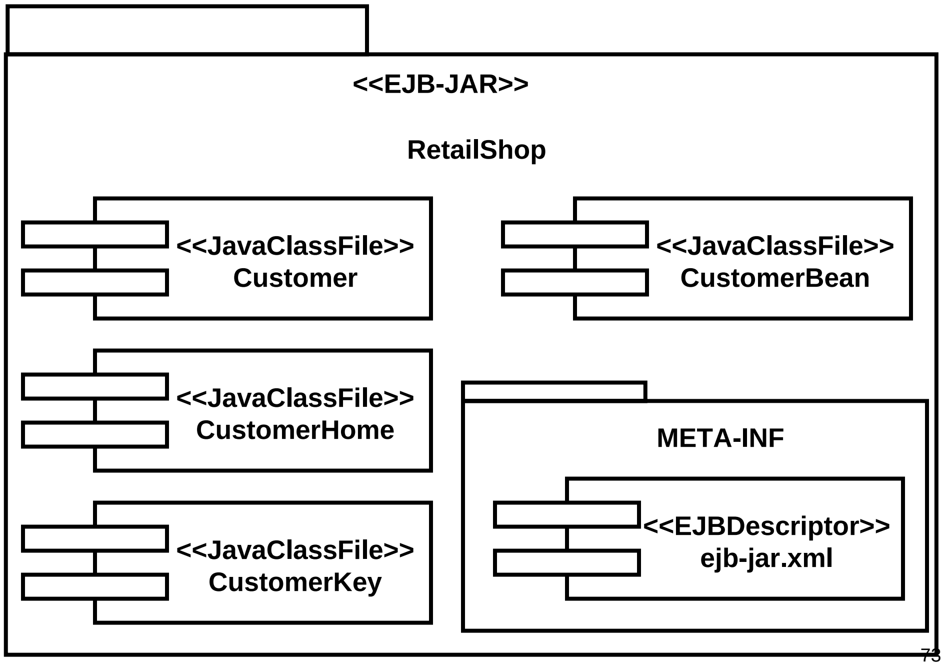
Java Implementation Model

- Java class file
  - Mapped to a UML component stereotyped as <<JavaClassFile>>.
- Java archive (JAR) file
  - Mapped to a UML package stereotyped as <<JavaArchiveFile>>.



EJB Implementation Model

- EJB-JAR
  - Mapped to a UML package stereotyped as <<EJB-JAR>>
- EJB deployment descriptor
  - Mapped to a UML component stereotyped as <<EJBDescriptor>>



<<EJB-JAR>>

RetailShop

<<JavaClassFile>>
Customer

<<JavaClassFile>>
CustomerBean

<<JavaClassFile>>
CustomerHome

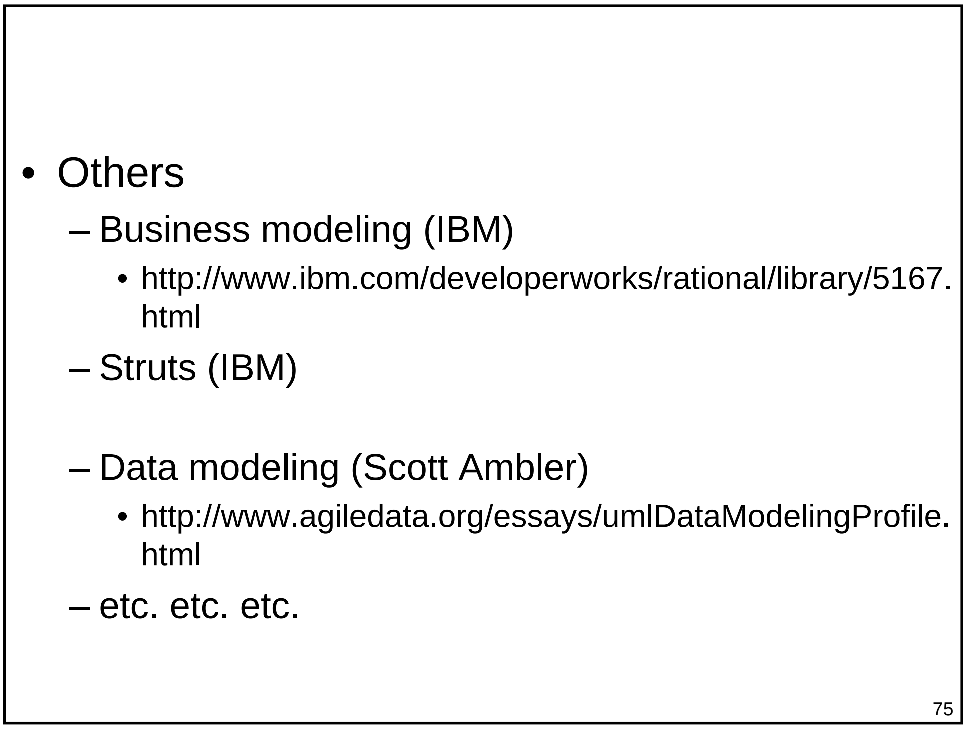
META-INF

<<EJBDescriptor>>
ejb-jar.xml

<<JavaClassFile>>
CustomerKey

More Profiles

- Completed OMG profiles
  - http://www.omg.org/technology/documents/modeling_spec_catalog.htm
    - CORBA
    - CORBA Component Model (CCM)
    - Enterprise Distributed Object Computing (EDOC)
    - Enterprise Application Integration (EAI)
    - QoS and Fault Tolerance
    - Schedulability, Performance and Time (SPT)
- On-going OMG profiles
  - http://www.omg.org/schedule/
  - System on a Chip (SoC)
  - Testing
  - Software radio



- Others
  - Business modeling (IBM)
    - http://www.ibm.com/developerworks/rational/library/5167.html
  - Struts (IBM)
  - Data modeling (Scott Ambler)
    - http://www.agiledata.org/essays/umlDataModelingProfile.html
  - etc. etc. etc.