IBM Rational University

Rational software

IBM

Pattern Implementation Workshop with IBM Rational Software Architect

RD801/DEV498 April 2007

Student Manual Volume 1

Part No. 800-027312-000

IBM Corporation
Rational University
Pattern Implementation Workshop with IBM Rational Software Architect
Student Manual Volume 1

April 2007

Copyright © International Business Machines Corporation, 2007. All rights reserved.

This document may not be reproduced in whole or in part without the prior written permission of IBM.

The contents of this manual and the associated software are the property of IBM and/or its licensors, and are protected by United States copyright laws, patent laws, and various international treaties. For additional copies of this manual or software, please contact Rational Software.

IBM and the IBM logo are trademarks or registered trademarks of IBM Corporation, in the United States, other countries or both.

Rational, the Rational logo, ClearCase, ClearCase LT, ClearCase MultiSite, Unified Change Management, Rational SoDA, and Rational XDE are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries or both.

WebSphere, the WebSphere logo, and Studio Application Developer, are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries or both.

Microsoft Windows 2000, Microsoft Word, and Internet Explorer, among others, are trademarks or registered trademarks of Microsoft Corporation.

Java and all Java-based marks, among others, are trademarks or registered trademarks of Sun Microsystems in the United States, other countries or both.

UNIX is a registered trademark of The Open Group in the United States, other countries or both.

Other company, product and service names may be trademarks or service marks of others.

Printed in the United States of America.

This manual prepared by:
IBM Rational Software
555 Bailey Ave.
Santa Teresa Lab
San Jose CA 95141-1003
USA

Contents

Module 0: About This Course

Introductions 0-2
Course Outline 0-7

Module 1: Best Practices

Objectives 1-2
Patterns in Software Development 1-11
Pattern Authoring Process 1-23
Review 1-34

Module 2: Reusable Assets and Artifacts

Objectives 2-2
Extending Rational Software Architect 2-8
Plug-ins and Pluglets 2-15
Artifacts and UML 2-19

Module 3: Templating 101

Objectives 3-2
Original JET 3-5
EMFT JET 3-9
Review 3-13
Further Information 3-14

Module 4: JET2 Data Model

Objectives 4-2
JET Data Model 4-4
XPath 4-8
Review 4-28
Further Information 4-29

Module 5: Basic JET Tags

Objectives 5-2
The Basic JET Tags 5-3
Review 5-12

Module 6: More JET Tags

Objectives 6-2
Tags and Tag Libraries 6-4
JET2 Control Tags 6-8
Simple Tag Combinations 6-14
Lab 2: Using XPath 6-18

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Review 6-19

Module 7: JET Examples

Objectives 7-2
Writing an Arbitrary List 7-4
Generating an Arbitrary Number of Files 7-8
Attributes and Derived Attributes 7-13
Lookups and De-Normalizations 7-22
Getter Names 7-27
Comma-Separated Lists 7-30
Lab 3: Authoring Transforms Manually 7-32
Review 7-33

Module 8: Exemplar Analysis

Objectives 8-2
Finding a Pattern to Implement 8-4
Preparing to Author a Model-to-Text Transform 8-14
Authoring the Model and Templates 8-18
Lab 4.1: Exemplar Authoring 8-32
Lab 4.2: Exemplar Authoring 8-33
Lab 5: Console Transform 8-34
Review 8-35
Further Information 8-36

Module 9: Introduction to EMF

Objectives 9-2
What is EMF? 9-3
Labs 9-13
Further Information 9-15

Module 10: Introduction to Transformations

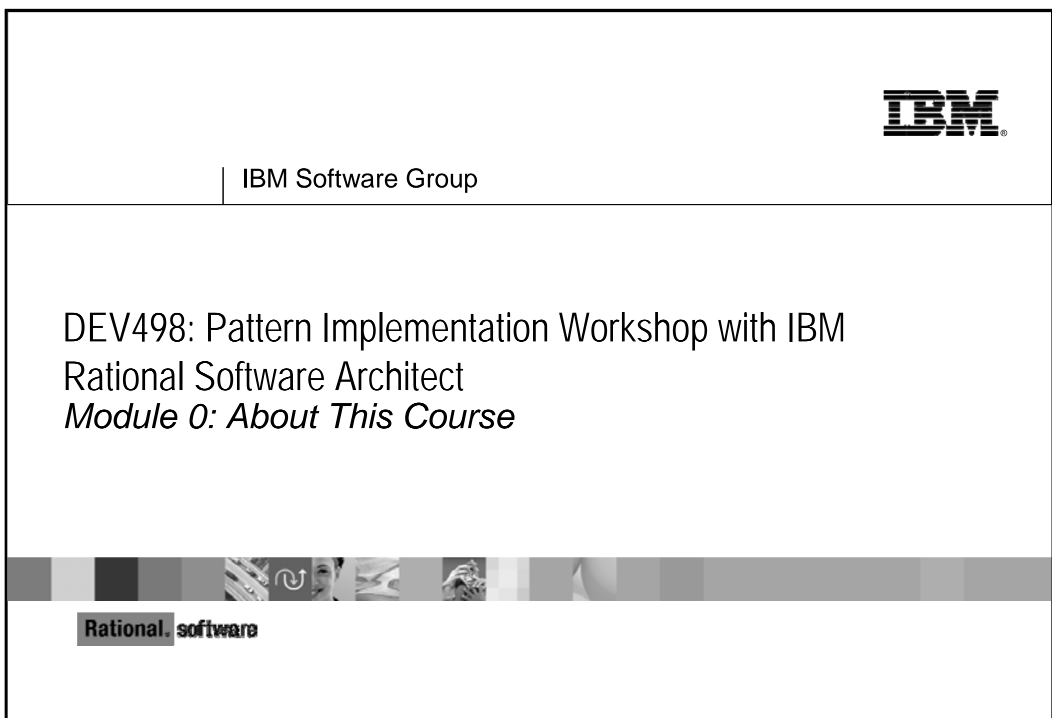
Objectives 10-2
Configuring and Running Transformations 10-7
Lab 7: Customize a Transformation 10-12
Creating a Model-to-Text Transformation 10-14
Lab 8: Create a Model to JET Transformation 10-38
Review 10-39
Further Information 10-40

Module 11: Designing Reusable Assets

Objectives 11-2
Model-Driven Development (MDD) 11-4
Summary 11-25
Review 11-26

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.



Contents

Introductions 0-2
Course Outline 0-7

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Introductions

- Your organization
- Your role
- Your background and experience
  - Software development experience
  - Experience with patterns and reusable assets
- Course expectations

2

IBM

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Intended Audience

- Software developers who use Rational Software Architect and who wish to:
  - Employ Model-Driven Development (MDD) or Model-Driven Architecture (MDA) strategies
  - Design and build pattern implementations and supporting artifacts
    - Patterns
    - Transformations
    - Profiles
    - Model templates

3

IBM

This course is for software architects, designers, and developers who create pattern implementations and related artifacts such as patterns, transformations, profiles, and model templates. The intent is to enable your model-driven development process using automation to design and build a solution according to best practices.

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Prerequisites

This course assumes that students:

- Can read and write Java code
- Model applications with UML
- Have taken the following IBM Rational courses, or have equivalent knowledge or experience:
  - DEV312: Essentials of Eclipse Plug-in Development
  - DEV325: Essentials of Model-Driven Architecture
  - DEV396: Essentials of IBM Rational Software Architect
- (Recommended) Students are familiar with XPath and XML, or have completed the following course:
  - XM301: Introduction to XML and Related Technologies

4

IBM

This course assumes knowledge of and experience with Java™ programming, basics of Eclipse plug-in development and Model-Driven Architecture, as well as familiarity with the basic features of IBM® Rational® Software Architect.

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Course Goals and Objectives

- After completing this course, you will be able to use Rational Software Architect to:
  - Design and create Pattern Implementations and related artifacts, including:
    - Transformations
    - UML Patterns
    - Profiles
    - Model templates
    - Pluglets
  - Package artifacts as a Reusable Asset Specification (RAS) asset

5 IBM

This course shows architects, designers, and lead developers how to develop reusable assets with Rational Software Architect, including artifacts such as plug-ins and pluglets, transformations, and patterns. It also shows you how to package these extensibility artifacts using the Reusable Asset Specification.

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Topics Covered in this Course

	Rational Application Developer	Rational Software Modeler	Rational Software Architect	Rational Systems Developer
Patterns		✓	✓	✓
Transformations	✓*	✓*	✓	✓*
Model Templates		✓	✓	✓
Profiles		✓	✓	✓
Pluglets		✓	✓	✓
Plug-ins	✓	✓	✓	✓

* Rational Systems Developer has a subset of Rational Software Architect transformations.
Rational Software Modeler supports only custom transformations.

6

IBM.

The IBM® Rational® Software Delivery Platform is based on the Eclipse open source platform. This platform enables unprecedented tool integration and artifact traceability throughout the development lifecycle. IBM® was a founding member of the Eclipse Foundation.

That integration extends in two directions:

- It knits together the individual roles on the team, and
- It brings together the shared software development disciplines that you see on this slide: requirements, analysis, design, construction, and so on.

* IBM® Rational® Systems Developer has a subset of Rational Software Architect transformations

* IBM® Rational® Software Modeler has a subset of Rational Software Architect transformations

* IBM® Rational® Application Developer allows you to author and run you own Model to Text Transformations

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Course Outline: Day 1

Morning:

0: About This Course

1: Best Practices for Pattern Implementations

2: Overview of Reusable Assets and Artifacts

Lunch **1 hour**

Afternoon:

3: Templating 101

4: The JET2 Data Model

7

IBM

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Course Outline: Day 2

Morning:

5: Basic JET Tags
6: More JET Tags
7: JET Examples

Lunch **1 hour**

Afternoon:

8: Exemplar Analysis
9: Introduction to EMF

8

IBM

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Course Outline: Day 3

Morning:

10: Introduction to Transformations

11: Designing Reusable Assets

12: Extending Models with Profiles

Lunch **1 hour**

Afternoon:

13: Model to Model Transformations

14: Creating UML Patterns in Rational Software Architect

9

IBM

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Course Outline: Day 4

Morning:

15: Introduction to UML2 API

16: Plug-ins and Pluglets

17: Model Templates

Lunch **1 hour**

Afternoon:

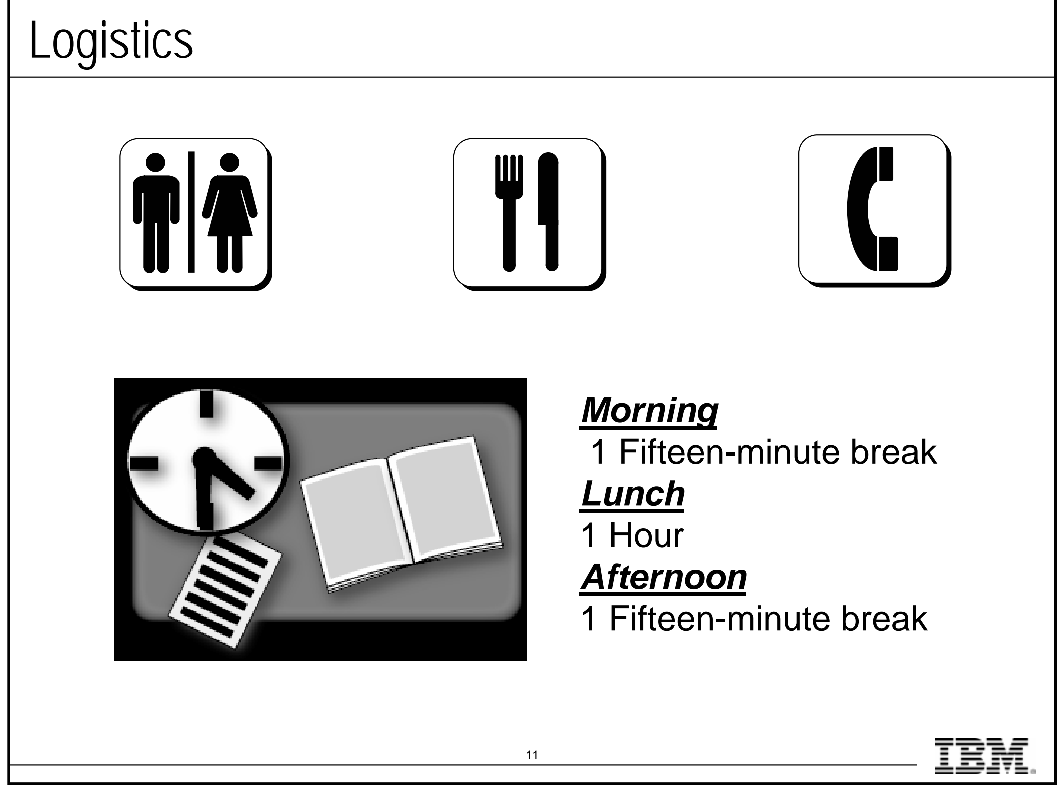
18: Packaging Assets

19: Summary

10

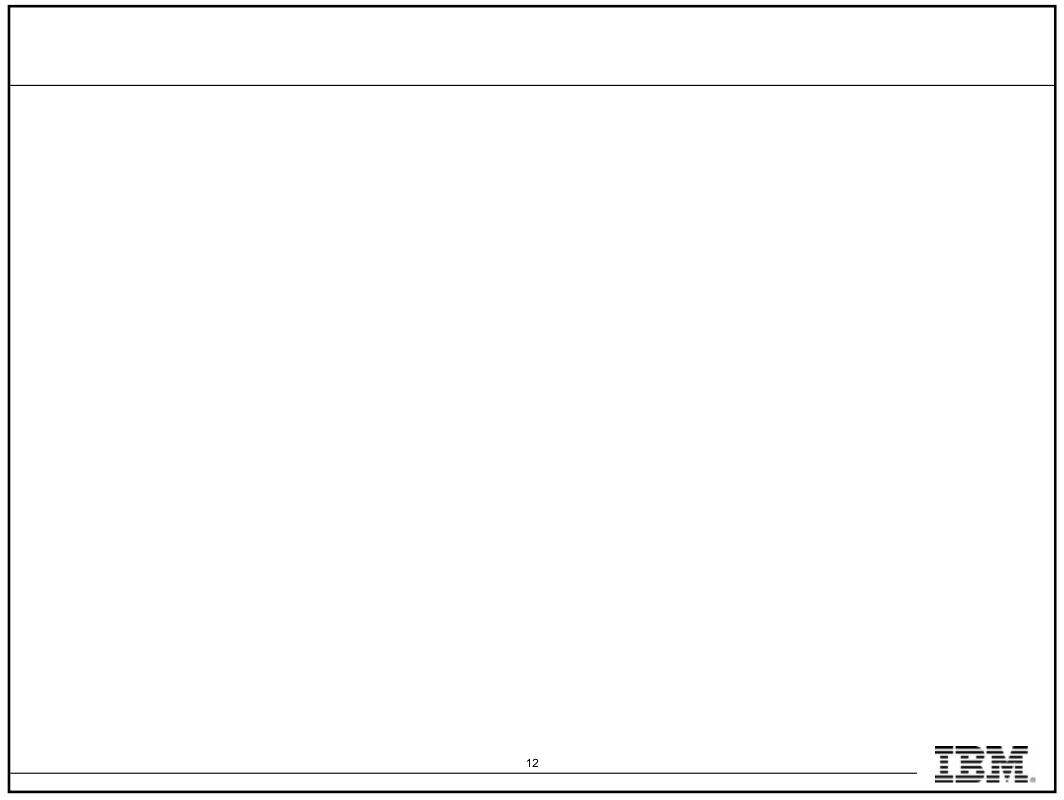
IBM

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.



© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.



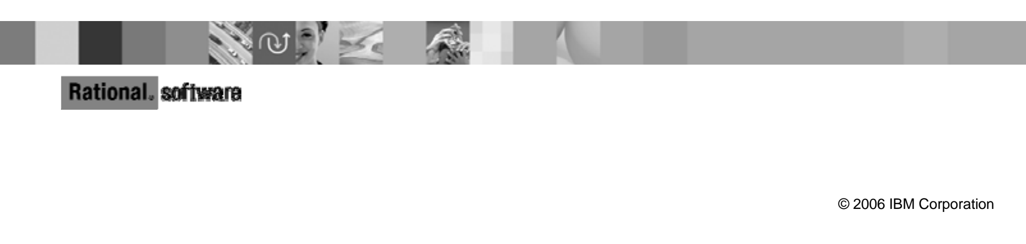
© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

IBM Software Group

DEV498: Pattern Implementation Workshop with IBM Rational Software Architect

Module 1: Best Practices for Pattern Implementations



© 2006 IBM Corporation

Contents

Objectives	1-2
Patterns in Software Development	1-11
Pattern Authoring Process	1-23
Review	1-34

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Best Practices for Pattern Implementations

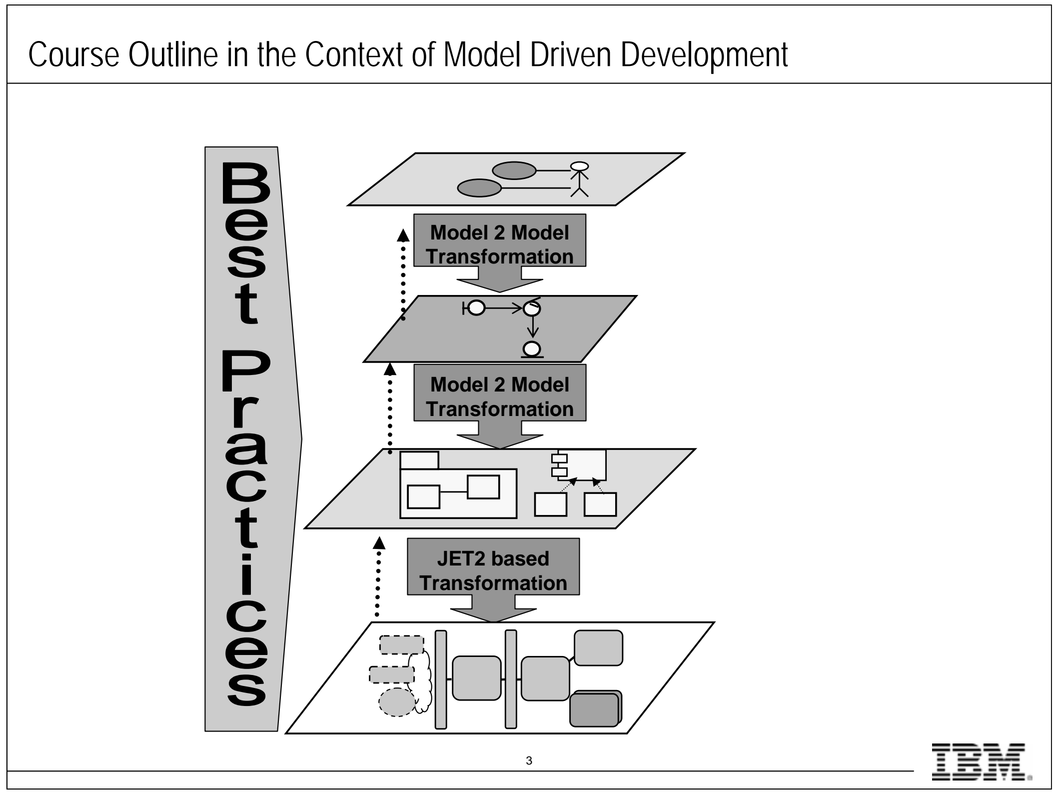
- Objectives:
  - Describe:
    - A tool-based definition of patterns
    - The role of patterns in software development
    - What decisions pattern authors must make
    - How to author a pattern implementation

2

IBM

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.



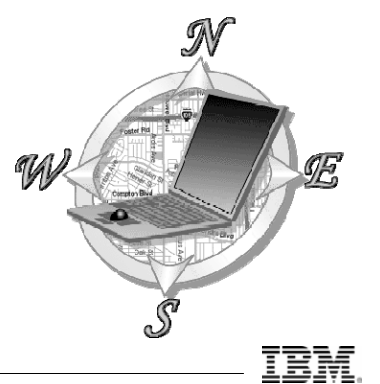
You'll see this slide several times throughout the workshop. It will serve as a visual guide to the skills you are learning and how they fit into Model Driven Development.

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Where Are We?

- **Introduction and Overview**
- Patterns in Software Development
- Pattern Authoring Process



4

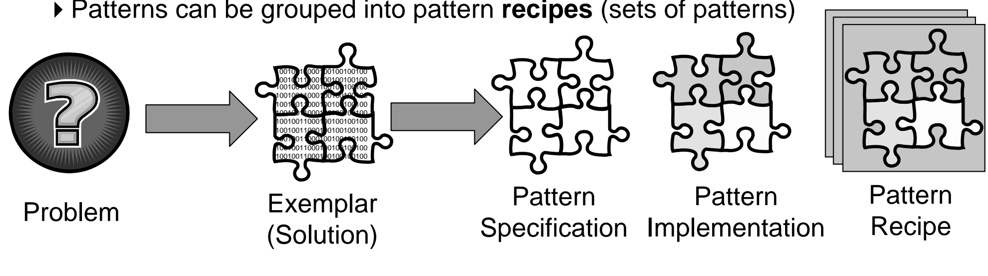
IBM

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Background

- A **pattern** provides a solution to a common problem by:
  - Offering reuse at a higher level than lines of code or individual classes and components, but at a lower level than an entire program
  - Enabling communication, education, and understanding of key development concepts
- A pattern can be implemented using software development tools
  - The pattern is abstracted from a **exemplar** that offers the best available solution to the problem.
  - A **pattern specification** captures a formal pattern description
  - A pattern can be **implemented** using a development tool and easily shared and applied in the development environment
  - Patterns can be grouped into pattern **recipes** (sets of patterns)



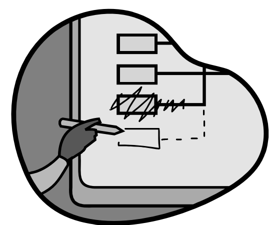
5

IBM

© Copyright IBM Corp. 2007
Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Pattern Specifications

- At first, patterns exist only as an idea in the mind of the developer, as a best practice used in many projects.
- Patterns are often captured as **pattern specifications.**
- A **pattern specification** formally documents:
  - The **problem** the pattern solves
  - The **solution** it provides
  - A **strategy** for applying the pattern in its context
  - **Consequences**, advantages, and disadvantages of applying the pattern



6

IBM

© Copyright IBM Corp. 2007
Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Pattern Specifications

- Pattern specifications are what we traditionally think of as "patterns"
  - Patterns described in books and documentation
    - Capture best practices
    - Are technologically abstract (do not contain technology-specific details)
    - Are often used for educational and communication purposes
  - To use the pattern you must code it yourself, manually

- Pattern specifications are important, but there is much more to patterns than just documentation!

7

IBM

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Pattern Implementations

- A **pattern implementation** automates the application of a pattern in a particular environment
  - Automates the process of applying a pattern in the IDE
  - Provides realized solutions to real problems
  - Makes patterns sharable and reusable
- Patterns become tools, concrete artifacts, in the development environment:
  - Rational Software Architect UML Pattern
  - Rational Software Architect Transformation
  - Plug-in
  - JET2 pattern

8

IBM

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Benefits of Pattern Implementations

- Increased productivity
  - Simplifies and accelerates the building and testing of software
  - Dramatically reduces development cycle times by eliminating repetitive work
  - Offers ease of use for beginners
- Improved software governance
  - Consistently enforce architectural, design, and coding standards
- Increased quality
  - Higher quality end product due to a higher level of consistency
  - Greater leverage of expert skills within the development organization
- Increased openness
  - Less dependency on a specific tool, vendor, or platform

9

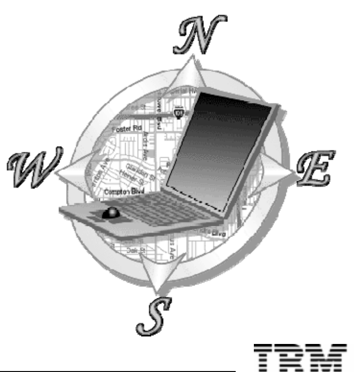
IBM

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Where Are We?

- Introduction and Overview
- **Patterns in Software Development**
- Pattern Authoring Process



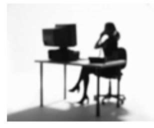
10

IBM

© Copyright IBM Corp. 2007

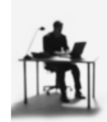
Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Patterns and Development Roles



Pattern User

Anyone who uses the pattern

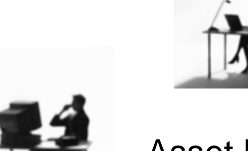


Pattern Author (SME)

Expert in the problem domain explained by the pattern

Pattern Specification Author

Writes the document that describes the pattern in depth



Pattern Implementation Author

Develops the micro tool that implements the pattern

Asset Librarian

Maintains and archives assets for the organization

11

IBM

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Patterns-Enabled Development Process

- Identify the problem
- Select a pattern
- Prepare the pattern
- Implement the pattern
- Execute the pattern
- Analyze the impact

Inputs → Select/Prepare/ Invoke → Execute Pattern → Output

Normally completed by people (Select/Prepare/Invoke)

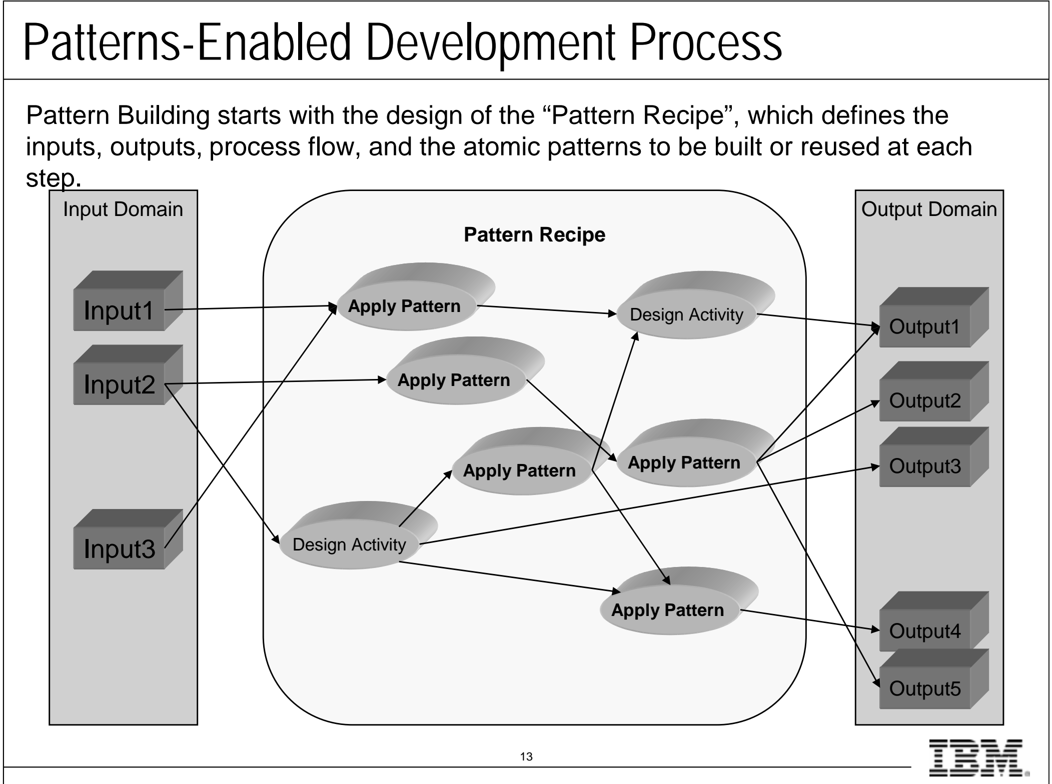
Normally completed by machines (Execute Pattern)

Output can also be used to refine the pattern

12

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.



© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Patterns in Model-Driven Development

- Patterns put the potential of MDD within reach
  - MDD involves using models to drive each stage in the software development process
    - Models abstract out and separate the key information from the details of the target environment or platform
    - The developer specifies the solution using models
    - Patterns and tool automations transform each input model into a target that is closer to the final artifacts

High-level model

Design decisions made using models.

Patterns automate the transitions from models to working solution.

Code artifacts

14

IBM

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Eclipse-Based Pattern Implementation Frameworks

- Eclipse Modeling Framework Technologies (EMFT) Java™ Emitter Template (JET or JET2)
  - Template engine for generating applications based on customizable, model-to-text transformations
  - Features exemplar analysis tools, and a template editor
- IBM® Rational® software design and construction tools support patterns and transformations:
  - **Rational Software Architect**

    Patterns: Applied in a single model and within the same level of abstraction
    - Examples: Business Delegate, Session Facade patterns applied in design model
  - **Transformations:** Applied across meta-models, models, and different levels of abstractions
    - Examples: UML to Java, UML to EJB, Java to UML

Rational Software Architect
(Java, J2EE, C++)

Rational Systems Developer
(C/C++, Java, J2SE, CORBA)

Rational Software Modeler

The Eclipse Platform
EMFT JET

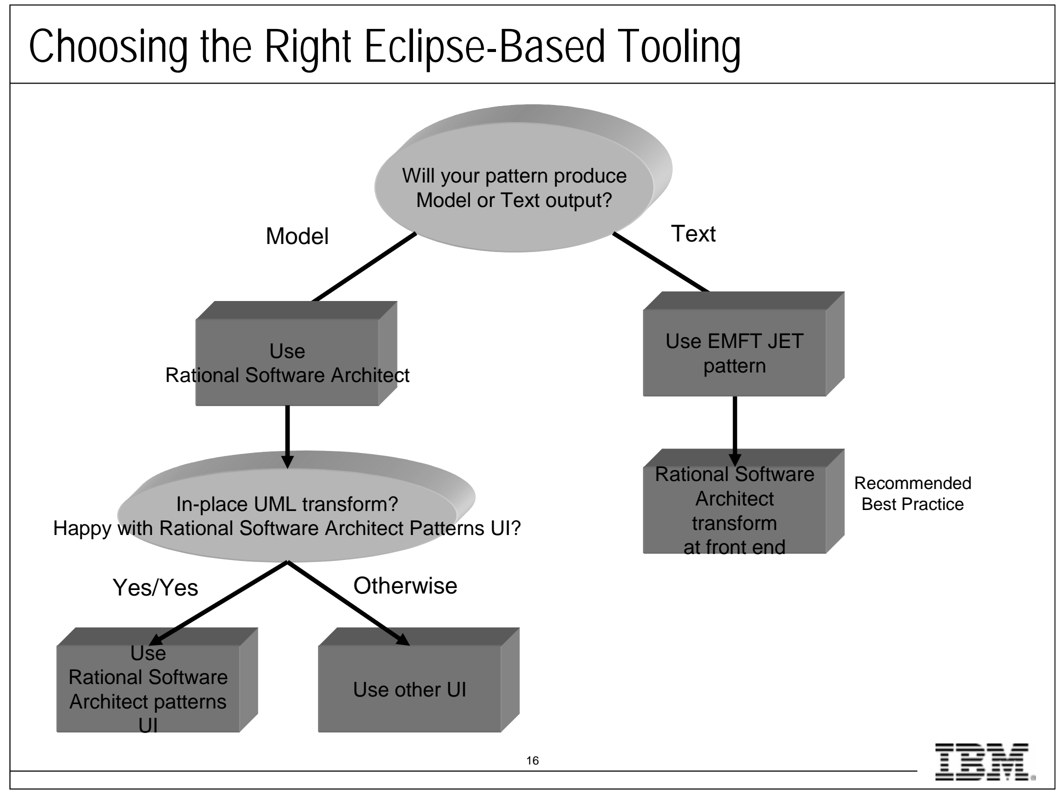
15

Rational Software Architect is for software architects and senior developers developing applications for the Java platform or in C++. Rational Software Architect is a design and construction tool for developing well-architected applications, including applications on a Service Oriented Architecture. Rational Software Architect unifies UML modeling, Java structural analysis, Web Services, Java or Java™ 2 Platform, Enterprise Edition (J2EE) technology, Data, XML, Web development, and process guidance.

Rational Software Modeler is for architects, system analysts, and designers who need to ensure that their specifications, architecture, and designs are clearly defined and communicated with their stakeholders. Rational Software Modeler is a visual modeling and design tool that leverages UML to document and communicate.

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.



Use this Decision tree to make the core technology selection.

The “patterns” referred to here are “atomic patterns,” which address a single use case step. A full use case usually involves you in selecting and applying a series of atomic patterns following a “Recipe”

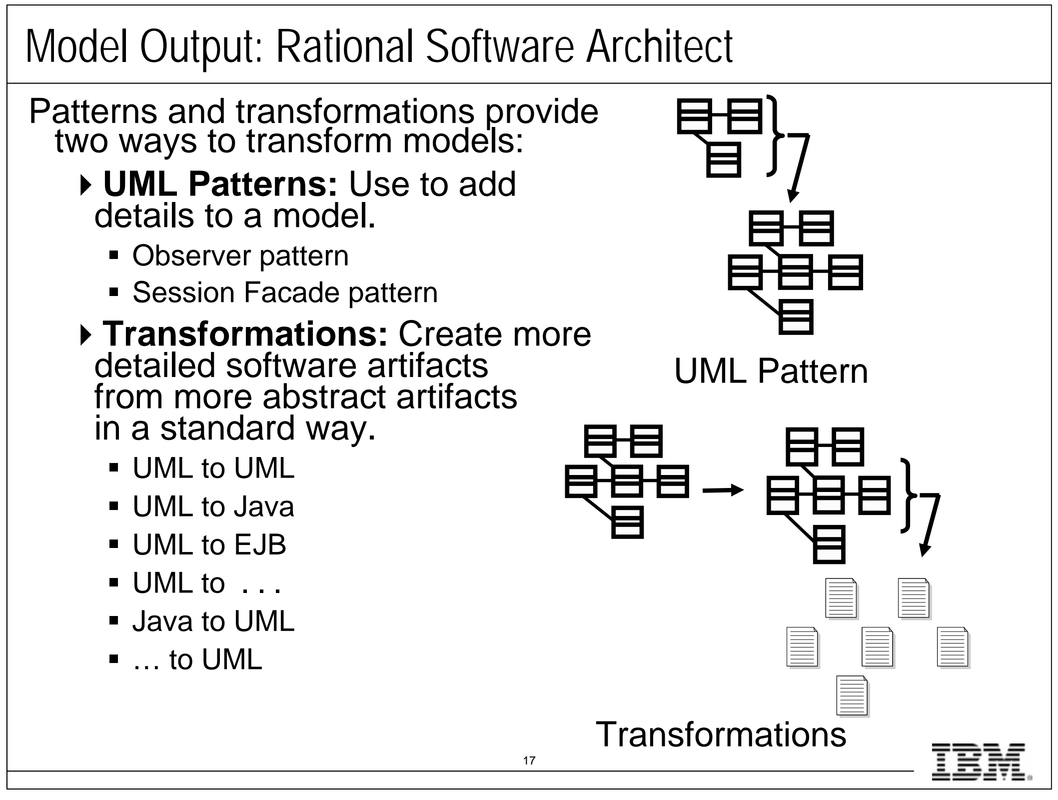
By “Model” (as output), we mean a structure that is intended for further manipulation in memory. The model could be text, EMF, or UML. The output may be a new model, or a modified input model.

EMFT JET (Java™ Emitter Template) requires Eclipse 3.2. If this is not available, use the tech preview subset of EMFT JET available as a design pattern toolkit (DPTK).

If using a tech preview is also unacceptable, use JET.

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

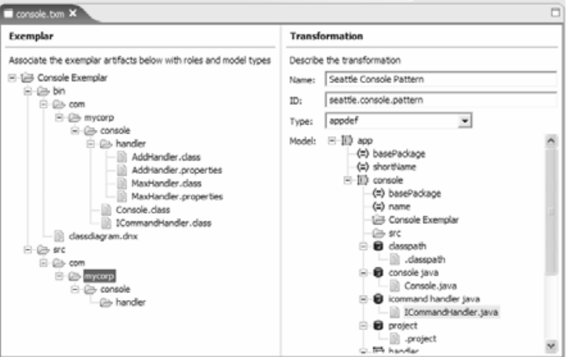


© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Text Output: EMFT JET

- Use EMFT JET to develop patterns with text output
  - Capture best practices for design and implementation as text-to-text transformations
- EMFT JET has the following components:
  - Transformation-based Eclipse resource generator
  - Transformation development and distribution features
- EMFT JET evolved from DPTK (Design Pattern Toolkit)



18

IBM

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Rational Software Design and Construction Products

- Automate design and construction, allowing you to create and customize:
  - UML Profiles
  - Transformations
  - UML Patterns
  - Pluglets
  - Model templates
- Support the Reusable Asset Specification for storing and sharing these resources

19

IBM

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Working with Models without Rational Software Architect

- If Rational Software Architect is not available, you can use the following technologies from the Eclipse Tools Project to work with models in Eclipse:
  - **Eclipse Modeling Framework (EMF):** Specify models using annotated Java, XML, or modeling tools like Rational Rose then import them into EMF for building tools and other applications based on a structured data model.
  - **Graphical Modeling Framework (GMF):** Develop graphical editors based on EMF and GEF in Eclipse
  - **EMFT JET:** Use a generic template engine that can be used to generate SQL, XML, Java source code, and other output from templates.

20

IBM

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Recommendations

- ✓ Use the productivity tools in Rational Software Architect wherever possible.
- ✓ If Rational Software Architect cannot be used, use EMFT JET, and add GMF if graphical modeling capability is essential
- ✓ Model-to-Text transformations should be implemented using EMFT JET or Rational Software Architect with EMFT JET.
  - ✓ Rational Software Architect can be used to build a front-end transformation and GUI
- ✓ By default, artifacts should be treated as text, and generated or manipulated with EMFT JET.

21

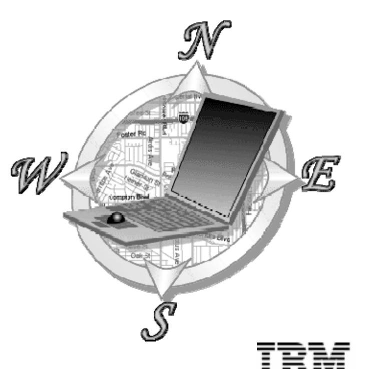
IBM

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Where Are We?

- Introduction and Overview
- Patterns in Software Development
- **Pattern Authoring Process**

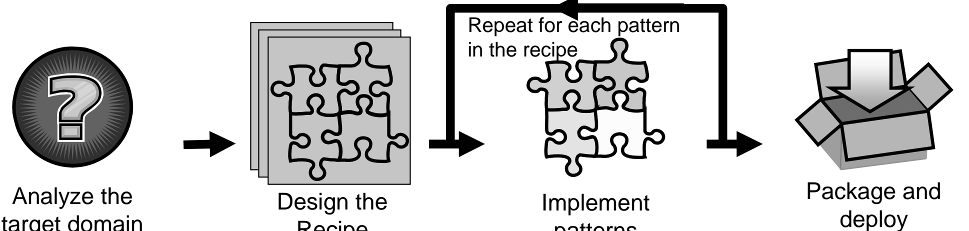


© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Pattern Recipe Authoring

- Pattern implementations are developed in sets (recipes) to address a specific target domain



- Examine the problem
- Find an exemplar solution
- Identify points of variability: what input should be left to the user, and what should the pattern provide?

- Determine a natural and realistic entry point for design
- Figure out what information is available at the start, and what information is needed at each stage
- Create an outline of what will happen in each stage

- For each pattern, a microtool is needed
- **Pattern implementation is our focus here**

Results are packaged and paired with documentation for the intended user community.

23

IBM

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Patterns: What to Implement and Specify

- You need to be able to customize patterns and assemble them using specifications and tools, without having to understand all the details of the implementations

- Pattern Implementation
  - Patterns can be customized with each use
  - Variability is supported by identifying places in exemplars where custom information can be substituted

- Pattern Specification
  - Patterns have to be documented in a standard way
  - All pattern specs provide:
    - Context: When to apply the pattern
    - Problem: What problem the pattern solves
    - Solution: How the pattern solves the problem

24

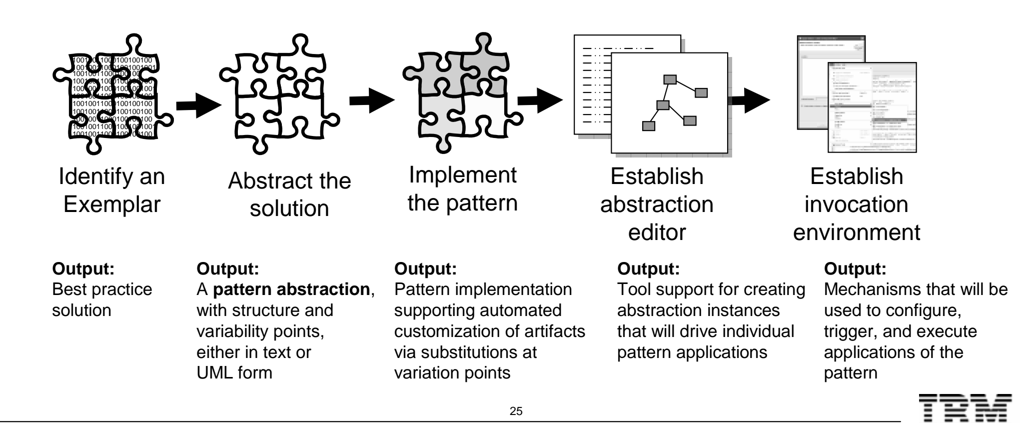
IBM

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Implementing a Pattern

- Pattern implementation consists of two parts, which should be kept independent to maximize reuse potential:
  - The pattern implementation
  - A user-interface for applying the pattern
- This is true regardless of technology choice or pattern type.

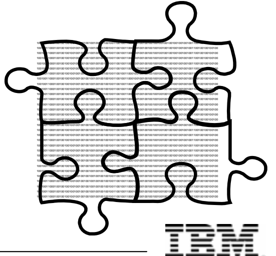


Regardless of the pattern implementation technology you choose (model-to-model or model-to-text), you will follow a similar process when you build the pattern.

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Exemplars

- Solution exemplars are the foundation for patterns and automation
  - They represent the best solution to a given problem, so they must be developed by an expert in the target domain
- Quality exemplars:
  - Follow best practices
    - Best practices in exemplars reflect the pattern and applications of the pattern
  - Include all variations that the pattern automation will support
  - Work



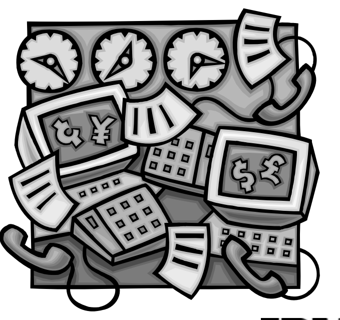
26

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Uses for Exemplars

- Provides the basis for the pattern implementation:
  - **Specification:** Exemplar specifies what the pattern would generate
  - **Test Case:** Exemplar is a test case for the pattern output

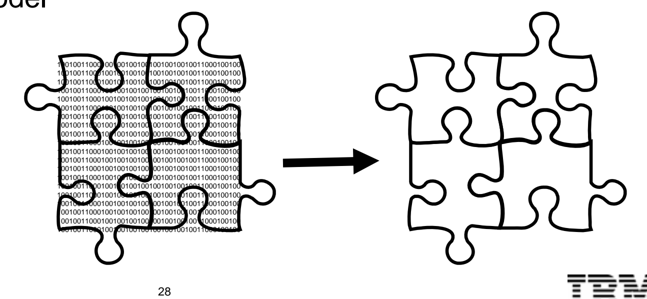


© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Normalization of Reference Solutions (Exemplar)

- Creating a pattern abstraction requires the author to decide:
  - What functional variation points must be exposed
  - What details are invariant and need to be hidden
- Developing pattern abstractions
  - Requires the involvement of a domain expert
  - Develop iteratively
    - The pattern abstraction will have to be revisited many times as you develop and refine the pattern implementation
  - Use a formal metamodel
    - Examples:
      - XML Schema
      - UML Profile



28

IBM

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Implementing the Pattern

Key decision: will the pattern implementation be presented as a model or as text?

- Models can be powerful
  - Ensure that only correct instances are created
  - May offer additional utilities:
    - Default values and implied constructs
    - Might support serialization capabilities for merging previous versions of the generated solution
  - However, APIs may be complicated and very large
    - Pattern authors may have to learn many new APIs
- Patterns based on text substitution (such as JET2) are easier to learn to create

29

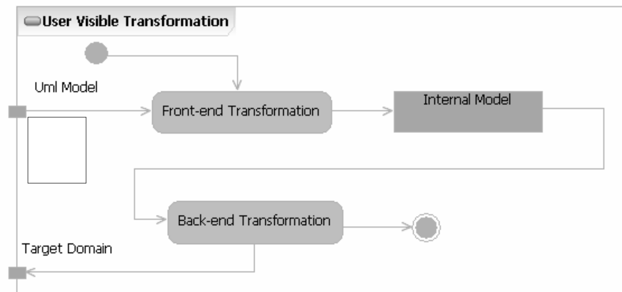
IBM

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Implementing UML Model to Text Pattern Implementations

- Separate the task of creating the UML model and profile (if needed) from the task of abstracting the exemplar
  - Create a pattern abstraction whose output is simple text (XML) documents
  - Implement a "code generator" with EMFT JET that reads the documents and produces the text output
  - To hide this processing from the pattern user, implement a front-end transformation that:
    - Maps from UML to the pattern abstraction
    - Invokes the code generator



© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Implementing UML Model to UML Model Pattern Implementations

There are two types of model-to-model transformations that you can perform with Rational Software Architect:

- In-place pattern expansion, where the problem and solution domains are the same UML model
  - Use Rational Software Architect UML Patterns in most cases
  - Use Rational Software Architect transformations if input parameters can't be represented in the Rational Software Architect UML Patterns framework
- Other model-to-model patterns: across models, across metamodels, and so on
  - Use the Rational Software Architect Transformation framework

31

IBM

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Establish an Editing Environment for Abstraction

- Based upon best practices, you (the pattern developer) can:
  - Provide no additional input representation.
    - Let the user edit XML documents as input to the pattern
  - Create a UML-to-abstraction transform that wraps the back-end (provide a back-end transformation)
  - Create a custom graphic editor using GMF, including component technologies EMF and GEF.
  - Create some other kind of editor, such as a dialog or wizard, using Eclipse extensibility. This will allow user input, and programmatically trigger the pattern call.

32

IBM

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Summary

- Patterns are a re-usable tool that can help simplify development
- Patterns provide an efficient means for ensuring that development is standardized
- By following protocols and best practices, patterns will help your organization produce high-quality products in an efficient manner

33

IBM

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Review

- What are the two types of model to model transformations you can perform with Rational Software Architect?
- Describe the differences between a pattern specification and a pattern implementation.
- What is the function of exemplars?

IBM

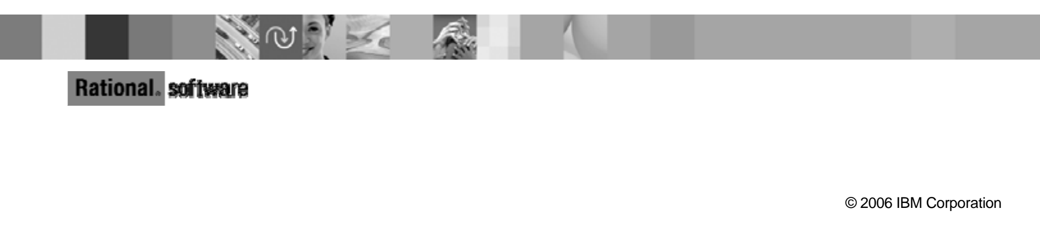
© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

IBM Software Group

DEV498: Pattern Implementation Workshop with IBM Rational Software Architect

Module 2: Overview of Reusable Assets and Artifacts



© 2006 IBM Corporation

Contents

Objectives 2-2
Extending Rational Software Architect 2-8
Plug-ins and Pluglets 2-15
Artifacts and UML 2-19

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Overview of Reusable Assets and Artifacts

- Objectives:
  - Describe what a reusable asset is
  - Describe how reusable assets can be used in software development
  - Describe the extensibility features of Rational Software Architect, and their uses for developing reusable assets.

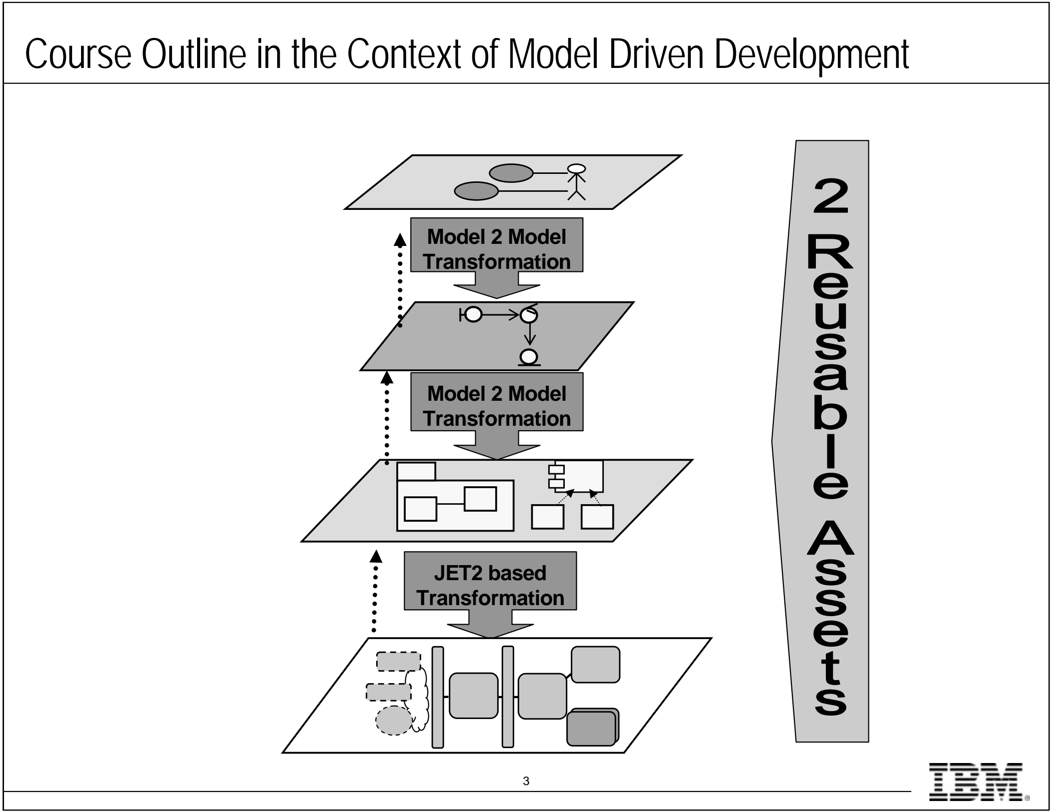
2

IBM

This module discusses reusable assets and the artifacts provided by Rational Software Architect.

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.



You will see this slide several times throughout the workshop. It will serve as a visual guide to the skills that you are learning, and how they fit into model-driven Development.

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

What is a Reusable Asset?

- A reusable asset is an organized collection of artifacts that provides a solution to a problem for a given context
- A reusable asset contains:
  - **Artifacts:** Profiles, patterns, transformations, pluglets, model templates, and so on.
  - **Variability points:** Allow users to customize the asset for a specific project

Problem

Solution

Artifact

Artifact

Artifact

Artifact

Variability Point

4

IBM

A reusable asset is an organized collection of artifacts that provides a solution to a problem for a given context. Assets clearly have much in common with patterns. For example, each:

- Includes instructions or usage rules, to minimize the time needed to discover, analyze, consume, and test the asset
- Includes standard documentation describing the development and business context in which the asset can be used
- Can have variability points, like pattern parameters, that allow users to customize the asset for a specific project

An asset is a more general concept than a pattern, since it is a collection of artifacts.

Asset can contain more than just patterns. An asset for a development project might contain requirements, models, source code, and tests. Assets might also be used to package and share deployable components, Web services, frameworks, and templates.

Reusable Asset Specification (RAS) is the standard structure. The IBM® Rational® brand products use the RAS specification. A RAS asset includes:

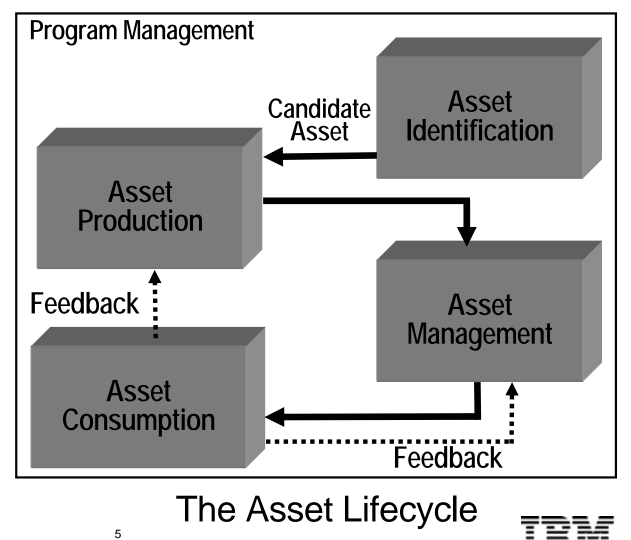
- RAS asset manifest file: The RAS asset is a zipped file that stores the files that make up the asset. At export, a manifest file is created and is included in every RAS asset's file.
- RAS asset profiles: RAS asset profiles allow you to create different assets. A specialized profile extends the contents of the default profile. Every RAS manifest must have a RAS profile.
- Activity task types: Activities should be modified only by users who are familiar with using the RAS to hand code manifest files. Modifications to the activities generated by RAS manifest files can render them incompatible. Activities describe tasks the user should do to reuse the asset. You should not modify generated activities, but you are encouraged to add your own as needed.

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Asset-Driven Development

- Reuse requirements, design, test, and deployment assets to develop new or enhance existing applications
- Supported by IBM Rational tools with:
  - Process guidance
  - Tooling
  - Standards
  - Assets



The Asset Lifecycle

5

Asset-based development (ABD) provides a way to reuse requirements, design, construction, test, and deployment assets to develop new or enhance existing applications.

IBM Rational products support ABD with:

Process Guidance: Provided with the IBM® Rational Unified Process® (RUP®) platform and its Asset-Based Development Plug-in

Tooling: The IBM Rational Software Delivery Platform makes it possible to package and share reusable assets.

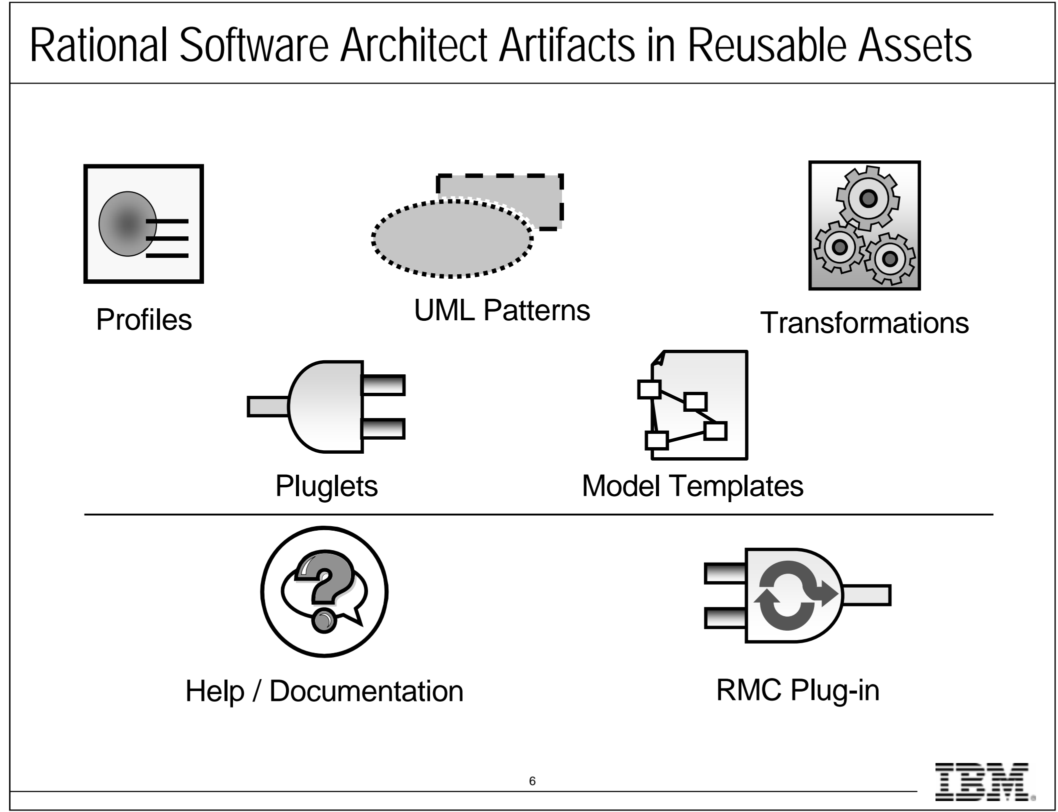
Standards: UML, Model-Driven Architecture, RAS, Middleware

Assets: Patterns, existing components, and new applications

The RUP platform and its ABD plug-in help team members learn who is expected to do what tasks, and when, with Rational Software Architect and other Rational brand tools. Teams develop architected solutions, models, and other artifacts based on a set of well-defined standards, including RAS and UML. Every project can consume assets and produce assets for other projects in an efficient way.

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.



The following artifacts are used to extend Rational Software Architect:

- UML profiles are sets of stereotypes, tag value definitions, and constraints that you can use to create model elements that reflect the semantics of a specific domain or platform. Profiles can tailor the UML for a specific domain or platform. You can use them in patterns to apply stereotypes to pattern participants. They are also used in transformation definitions to specify how model elements should be transformed.
- UML Patterns: You can use the UML Patterns Service and Framework to create implementations that codify specific patterns.
- Transformations: Rational Software Architect provides support using the PDE, Exemplar Authoring, Model Mapping and a Transformations API for you to create custom transformations.
- Pluglets are Java applications that provide an alternative to plug-ins for extending the workbench. Pluglets can be thought of as a lightweight plug-in, usually created to handle routine tasks.
- Model templates: You can export a model as a template so that its structure can be reused as standard model structure, or as a transformation or pattern target. Model templates are similar to patterns in the sense that they can provide whole sets of model elements automatically.

The following artifacts can be bundled with these artifacts:

- Help: You can create custom help documentation to support any artifact you create, and it can be integrated with the standard help documentation for the tool.
- IBM® Rational® Method Composer Plug-in: Provide RUP content along with your artifacts.

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Where Are We?

- **Extending Rational Software Architect**
- Plug-ins and Pluglets
- Artifacts
  - Profiles
  - Model templates
  - Patterns and Transformations

7

This section provides an overview of how you can extend the capabilities of Rational Software Architect.

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Why Use Rational Software Architect to Build Reusable Assets?

- Configure the tool specifically for your environment
- Capture and codify best practices
- Automate tasks related to the specific problem domain and underlying technology
- Reduce manual effort, which leads to:
  - More quickly developed solutions
  - Higher quality solutions
- Leverages Eclipse
  - Extensible
  - Open standards and specifications

8

IBM

Rational Software Architect provides a variety of different customization options, which allow you to tailor tools to respond flexibly to the different needs of different environments and tasks. Architects can deliver tailored tools that directly address the specific needs of developers, allowing them to improve the quality, reusability, and efficiency of the development process.

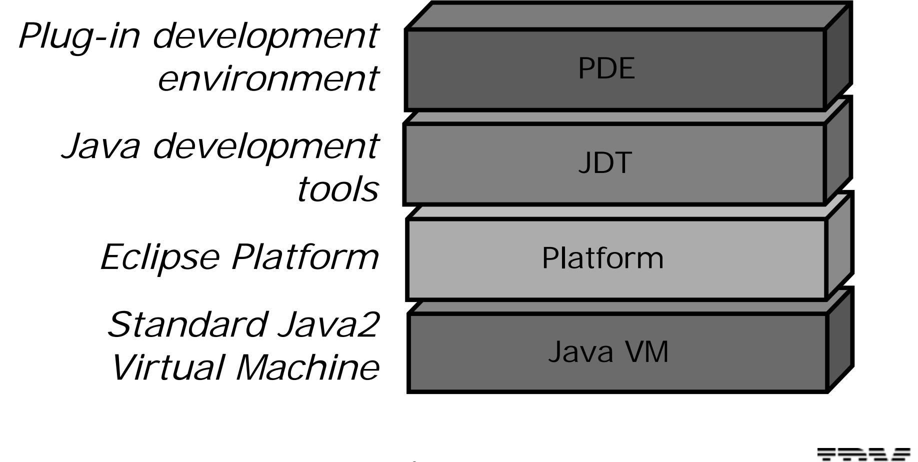
Using this functionality, different organizations that have different needs do not have to compromise on a lowest common denominator tool set to achieve enterprise interoperability and code reuse.

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

What is Eclipse?

- Eclipse is a universal platform for integrating development tools
- Open, extensible architecture based on plug-ins

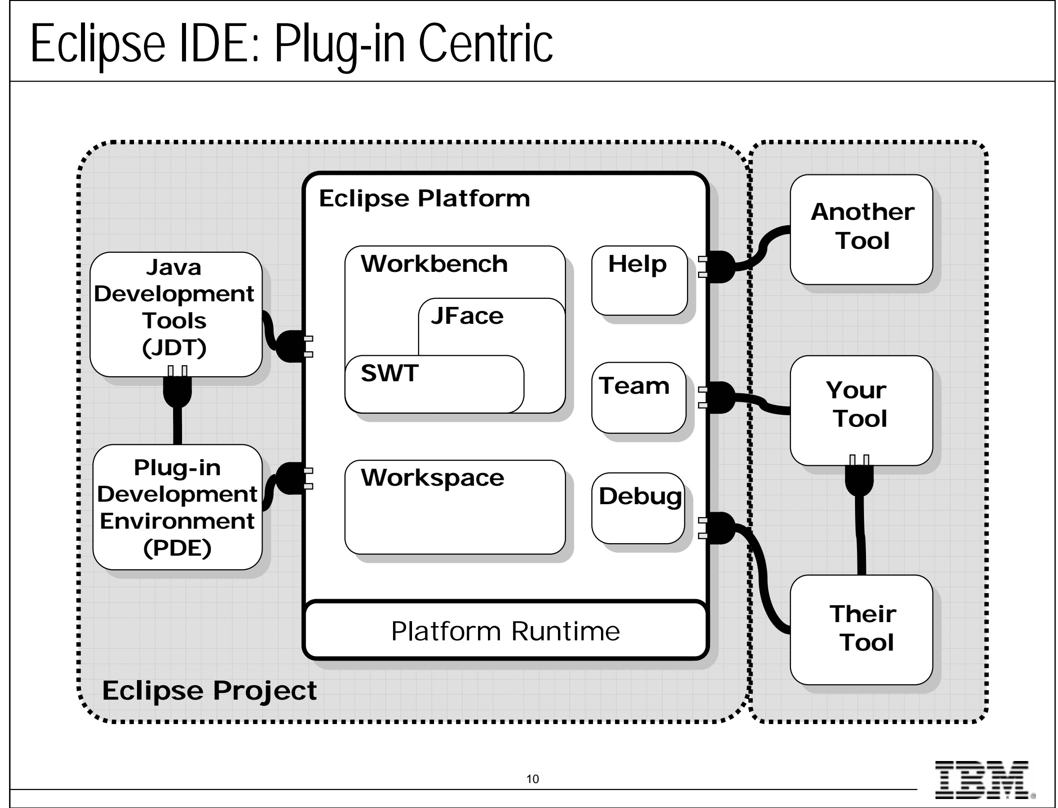


9

IBM

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.



Eclipse is layers:

- **Platform Runtime:** The base engine that makes it all work (plug-ins that provide architecture and functional content)
- **Eclipse Platform:** Built on the Platform Runtime, this is the base for the Workbench. Provides an integration platform for tools and applications.

Platform components, in addition to the Platform Runtime:

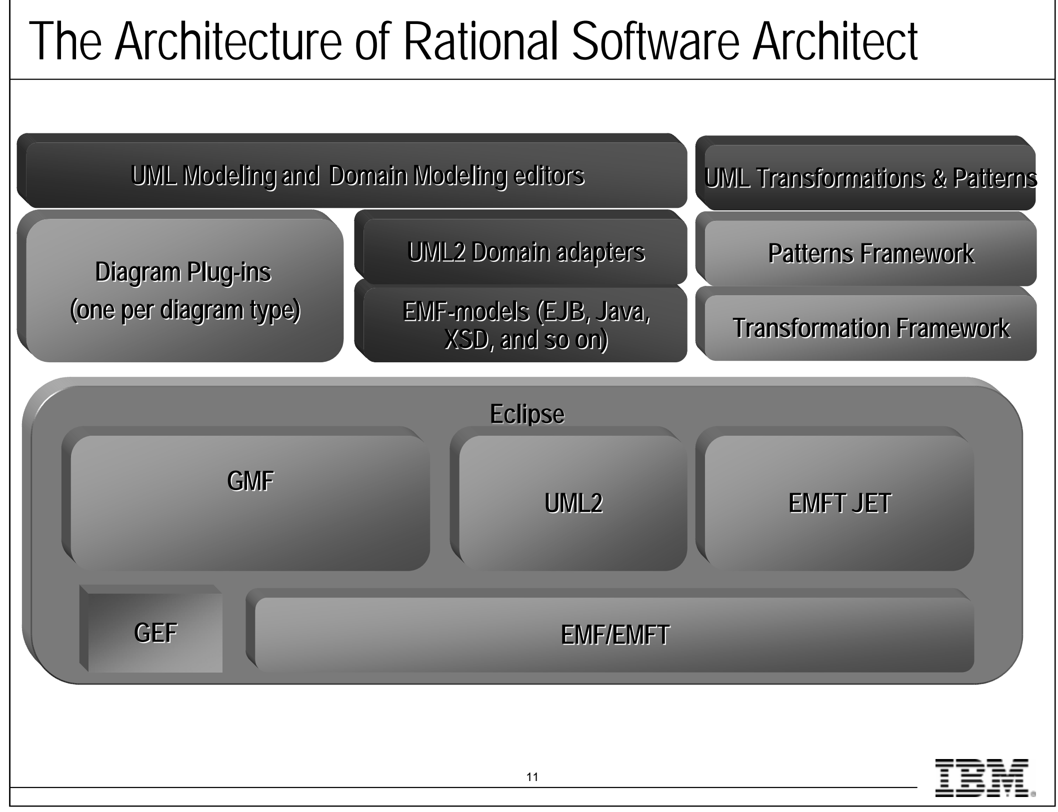
- **Workspace:** Resource model, with support for projects, folders, and files, as well as natures, builders, and markers.
- **SWT, JFace, and Workbench:** Layers in the UI domain that build on each other. SWT is a Java API on Operating System widgets, JFace is an application framework for UI components, and Workbench is the model for an integrated UI with Views and Editors.
- **Help:** The ability to render navigation and content, with APIs for tool-directed navigation (F1) and help invocation of tools. Help can be in a standalone environment.
- **Team:** The framework for team programming and repository access. Eclipse comes with the framework, and a CVS implementation.
- **Debug:** The framework for testing and debugging language-specific programs. It has no functionality as delivered, so it must be taught.
- **Ant** (not shown here): Included and integrated into the Workbench platform. The PDE uses Ant to support feature or plug-in preparation and packaging operations.
- **Update Manager** (not shown): A component and user interface that allows you to manage the active configuration of features known to the workbench.

Java Development Tools: Features and Plug-ins providing a development environment.

Plug-in Development Environment (PDE): Builds on the JDT or Workbench to provide support for developing, testing, building, and deploying feature sets and plug-in sets.

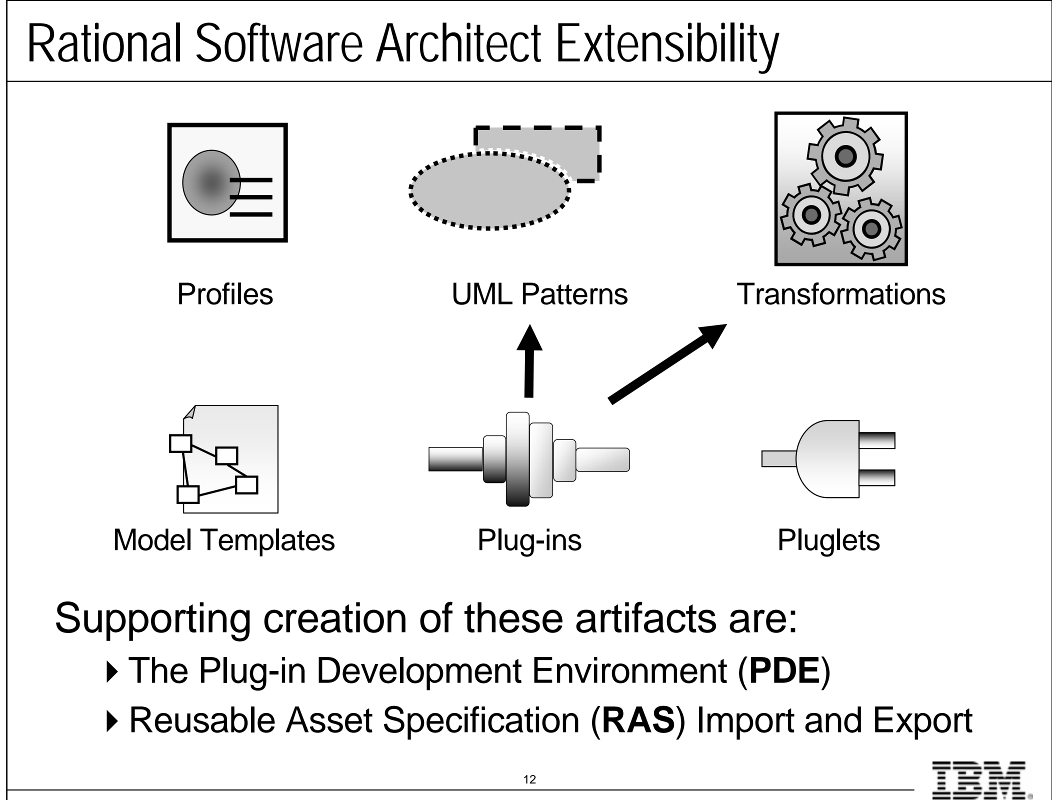
© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.



© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

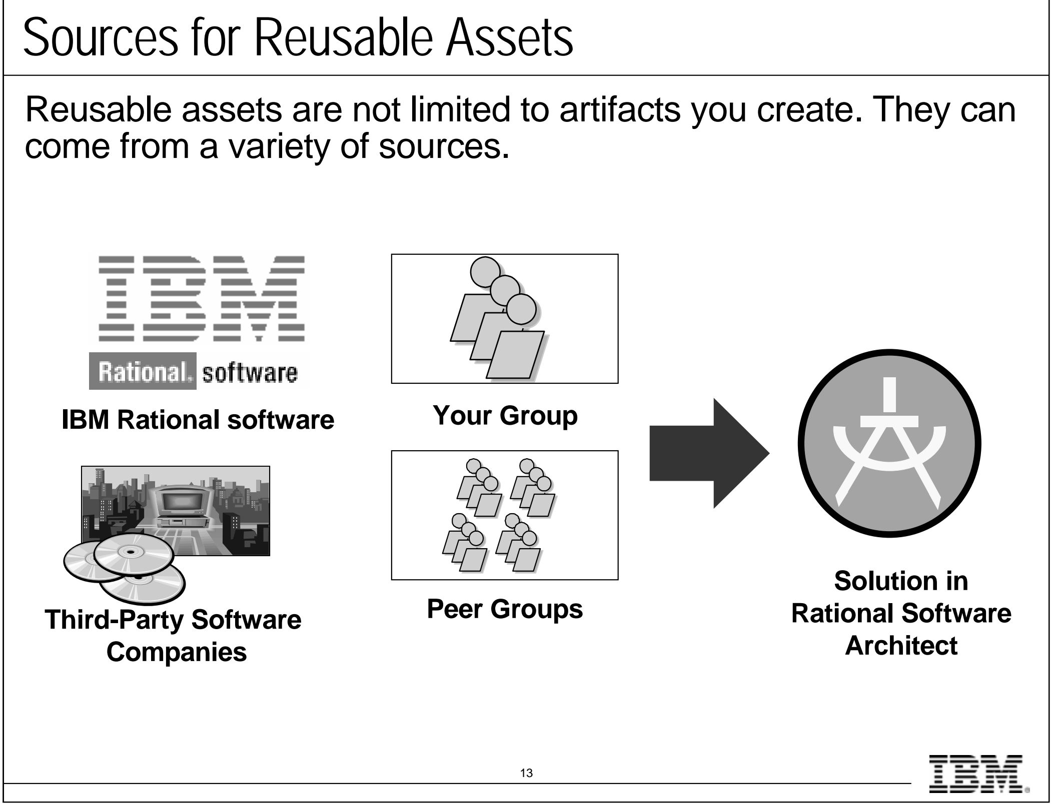


Use the following artifacts to extend Rational Software Architect:

- **Eclipse Plug-ins:** Extension points in Eclipse are available to customize system behavior using plug-ins. The plug-ins that you develop can, in turn, contain their own extensions to existing plug-ins, and make extension points available so that other plug-ins can build on their functionality. Plug-ins are also used to package and exchange many types of resources(such as, in Rational Software Architect, patterns and transformations).
- **UML profiles** are sets of stereotypes, tag value definitions, and constraints that you can use to create model elements that reflect the semantics of a specific domain or platform. Profiles make it possible to tailor the UML for use in a specific domain or platform. You can use them in patterns to apply stereotypes to pattern participants, and in transformation definitions to specify how specific model elements should be transformed.
- **UML Patterns:** You can use the UML Patterns Service and Framework to create implementations that codify patterns that are specific to your organization.
- **Transformations**: Rational Software Architect provides support using the PDE, Exemplar Authoring, Model Mapping and a Transformations API for you to create custom transformations.
- **Pluglets** are Java applications that provide an alternative to plug-ins for extending the workbench. Pluglets can be thought of as a lightweight plug-in, usually created to handle routine tasks.
- The **Plug-in Development Environment (PDE)** is a set of tools in Eclipse for creating, developing, testing, debugging, and deploying Eclipse plug-ins. The PDE includes tools for developing fragments, features, and update sites.

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.



The real power of extensibility resources is that they can be shared between collaborators, projects, groups, or even different organizations.

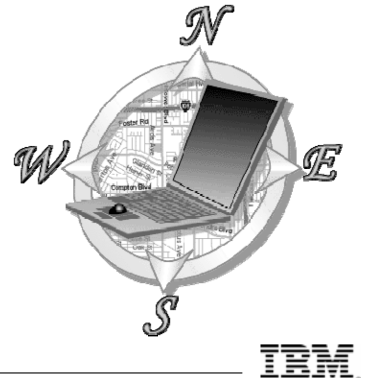
These artifacts can be leveraged into major gains in productivity, not just for single projects but in many different organizations within an enterprise. A good method really deserves more than a single use. By creatively using extensible artifacts, you can share good ideas and use them in new projects simply and effectively.

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Where Are We?

- Extending Rational Software Architect
- **Plug-ins and Pluglets**
- Artifacts
  - Profiles
  - Model templates
  - Patterns and Transformations

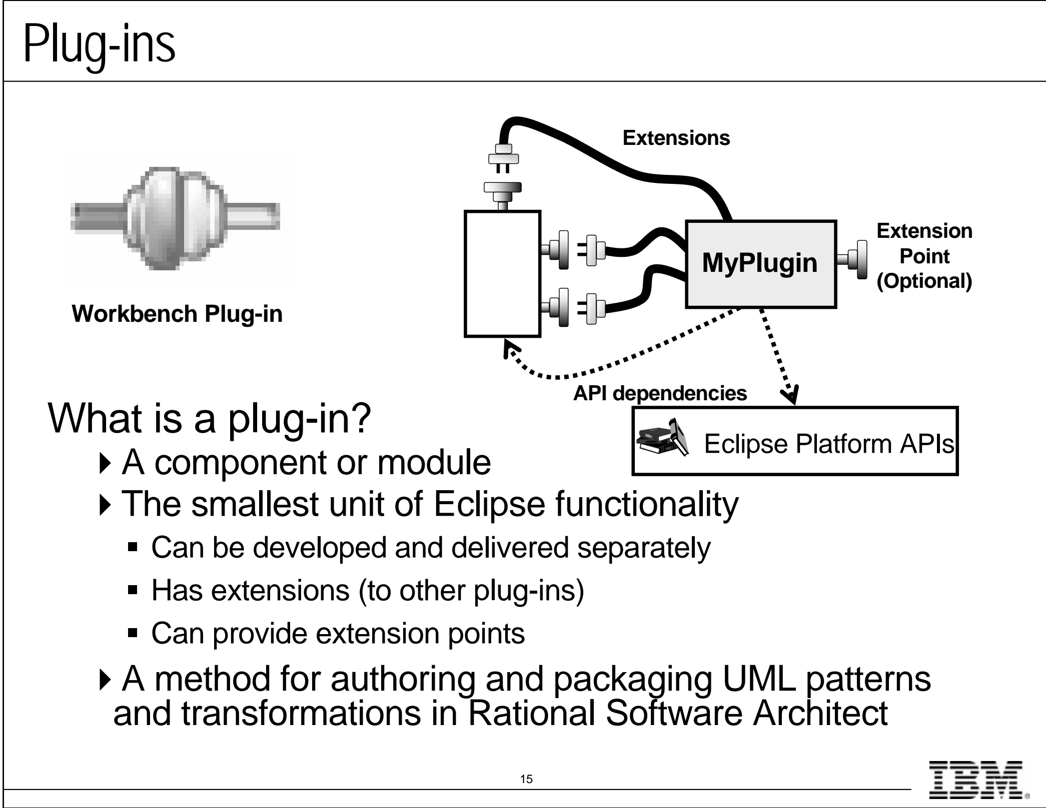


14

IBM

This section provides an overview of plug-ins and pluglets, which are the base technologies for creating assets in Rational Software Architect.

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.



The Eclipse platform is structured as a core runtime engine with a set of additional features installed as platform plug-ins at pre-defined extension points. These extension points are available to developers to contribute to system behavior. The plug-ins you develop can, in turn, contain their own extensions to existing plug-ins, and make extension points available so that other plug-ins can build on your plug-ins' functionality.

As you will see in this module, you can use plug-ins not just to enhance the functionality of Eclipse in the ways that you might expect (like the resource management system, or the workbench plug-ins), but also to package and exchange many resources and assets that you develop in Rational Software Architect.

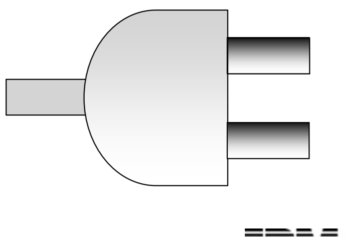
In general, to create a plug-in you would:

- Decide how your plug-in will be integrated with the platform.
- Identify the extension points that you need to contribute to integrate your plug-in.
- Implement these extensions according to the specification for the extension points.
- Provide a manifest file (plugin.xml) that describes the extensions you are providing and the packaging of your code.

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Pluglets

- Pluglets:
  - Are small Java applications used to make minor workbench extensions
  - Are a kind of Java-based scripting mechanism
  - Are executed in the active workbench
    - No target platform needed
  - Provide easy access to workbench plug-in APIs
    - For testing, API exploration, and custom scripts
- They are created in a project
- They use the workbench Java development environment



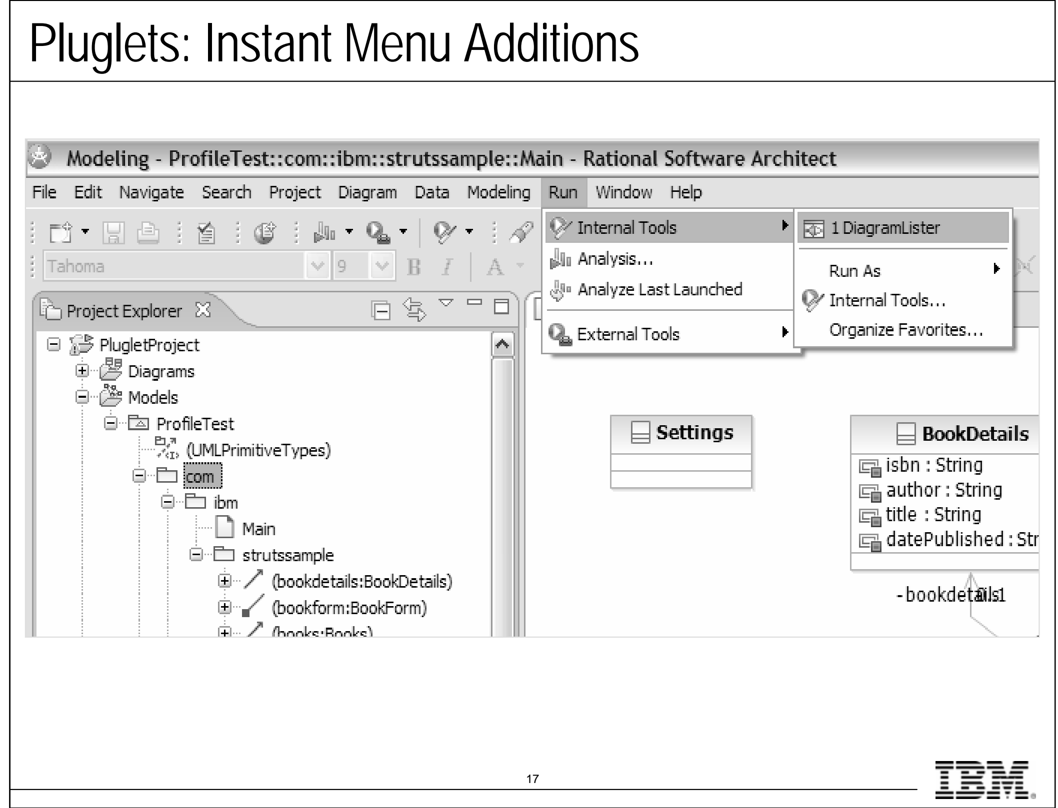
16

IBM

A pluglet is a small Java application that can use any available Eclipse-based API's. It was specifically provided by Rational Software Architect and facilitates code Exploration.

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.



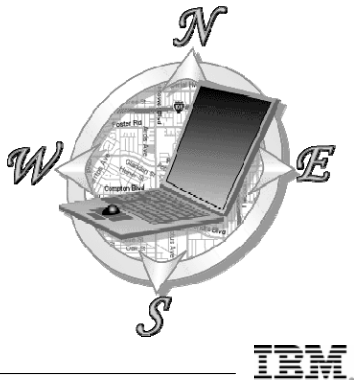
Pluglets can be run instantly, using the **Run Internal Tools** button, in the same session, without having to start a new target instance.

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Where Are We?

- Extending Rational Software Architect
- Plug-ins and Pluglets
- **Artifacts**
  - **Profiles**
  - **Model Templates**
  - **UML Patterns and Transformations**



18

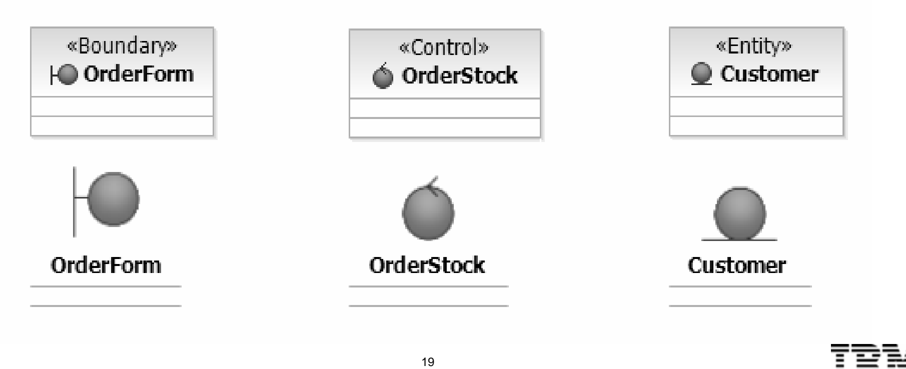
IBM

This section introduces reusable UML artifacts that you can develop in Rational Software Architect, including profiles, model templates, UML patterns, and transformations.

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

The UML Profile: The Language of Reusable Assets

- Profiles enable extending the UML without changing the modeling language.
- Profiles include:
  - **Stereotypes:** A simple textual marker («…») or icon placed on a model element to add semantics to the element
  - **Tagged values:** To add properties that are not supported by the base element
  - **Constraints:** Constraints enforced on the element or its attributes



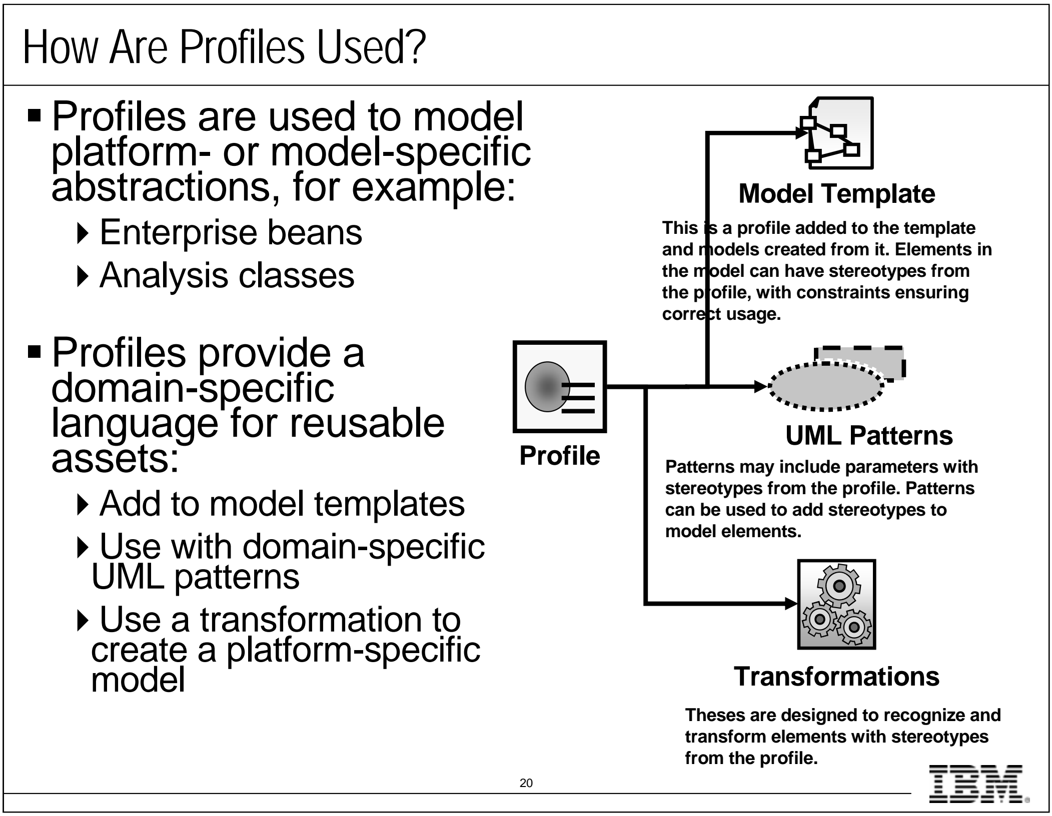
19

Rational Software Architect allows you to develop and apply UML profiles. UML profiles are sets of stereotypes, tag value definitions, and constraints that you can use to create model elements that reflect the semantics of a specific domain or platform.

- **Stereotype:** This is a simple textual marker («…») or icon placed on a model element to add semantics to the element. A stereotype extends UML, but not its structure. You can add stereotypes to model elements to create specialized forms, but you cannot add new elements to UML.
- **Tagged Values:** Typically a string or Boolean value, you can associate tag definitions with specific stereotypes, or with all model elements of a specific type (class, association, operation parameter, and so on). It is common to use tagged values to add values to model elements, and to add information for transformations and code generation.
- **Constraint**: This is a set of rules that you can execute to determine a model or modeling element's correctness. Constraints are usually defined using the Object Constraint Language (OCL), but can also be defined in natural languages and Java.

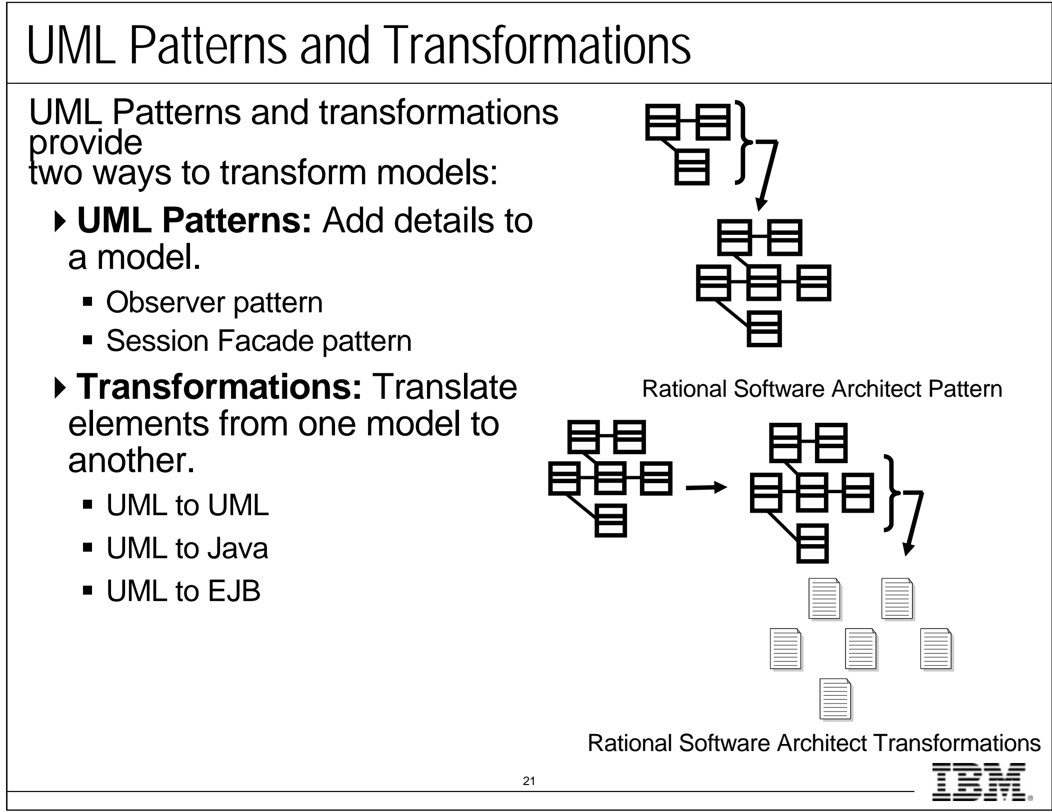
© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.



The profile marks up the template and models based on it, using constraints to enforce correct usage.

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.



In Rational Software Architect and Software Modeler terms, UML patterns and transformations are tool automation features. UML Patterns are sets of model elements that are parameterized to be fitted into any existing model, to speed development and maintain consistency among software solutions (so that, for example, every instance of an Observer is designed the same way). Transformations can be used to translate model elements from one model to another automatically, (in order to speed the transition of elements from, for example, analysis to design or from design to implementation).

UML Patterns

UML Patterns allow you to make use of existing solutions developed in the same type of model. For example, the Observer GoF pattern in Rational Software Architect contains design-level UML model elements that would be applied in a UML design model. You can harvest patterns from an existing model and apply them in multiple models of the same type. You can also harvest a pattern from a model and apply it in a different part of the same model.

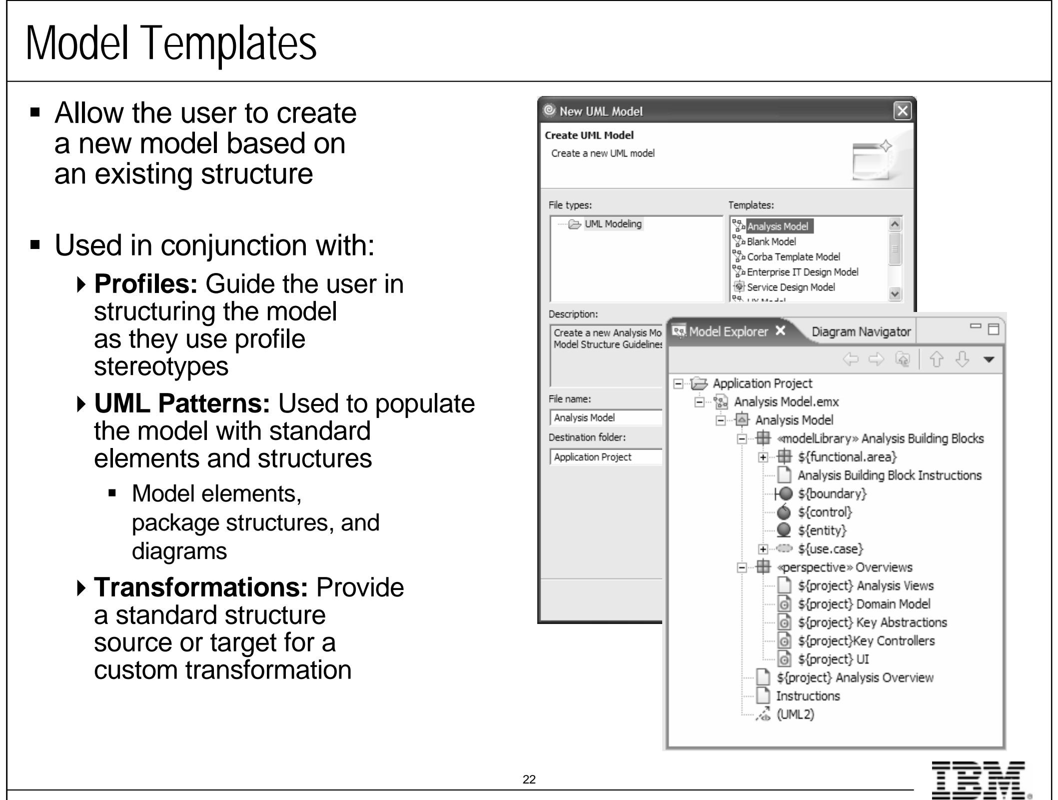
Patterns have parameters so that you can customize them for a specific context, but patterns do not automatically translate themselves to work in different model types. You cannot, for example, apply a design pattern and get a code-level idiom (in Java code) without using transformations.

Transformations

Transformations take elements from one model and translate them into elements of a different model. Transformations are often applied to whole models, but you can apply them to selections from models as well. You could, for example, apply a transformation to move from a platform-independent model to a platform-specific model as you add in details about the platform and get closer to the implementation. When adding levels of refinement, you can transform from a platform-specific model to another, adding more details without changing the model type.

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.



© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Integration of Reusable Artifacts

Profiles, model templates, patterns, and transformations can be used together to increase the ROI of your extensibility investment.

Supply with template

Model Templates

Profiles

UML Patterns

Transformations

Populate the model

Manually mark up the model with profile elements

Automatically mark up the model with profile elements, and/or apply best practices

Transform the model based on rules that recognize model markup

Source Model

Source Model with Markup

Source Model with Markup, UML Patterns Applied

Target Model

23

IBM

As you work with Rational Software Architect in your environment, you will come across situations where plain UML is not able to model the elements of your domain sufficiently. In addition, there will be patterns of usage that will accompany these domain-specific elements. You can develop UML profiles in Rational Software Modeler or Rational Software Architect for these situations.

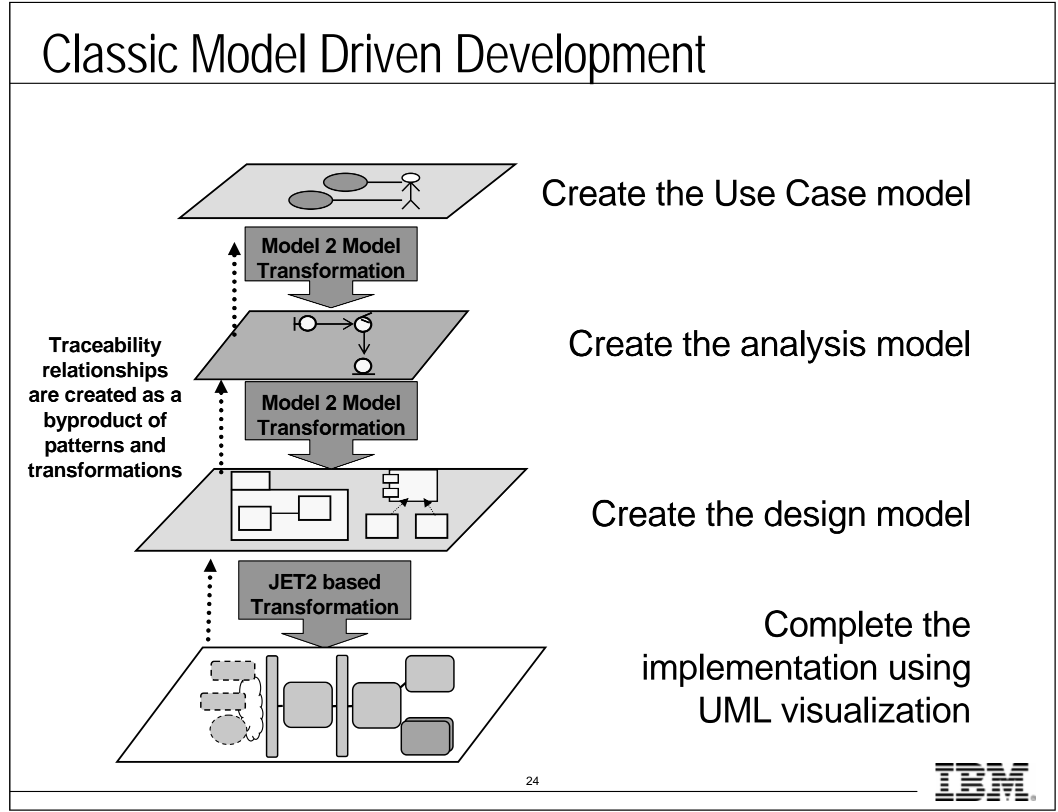
Creating a UML pattern that can understand and use the domain-specific elements of your profile will help ensure that users are following best practices for your organization. As a final step in this workflow, the user would send the model through a transformation. Ideally, the model elements would then be updated according to the profile, with elements structured in a way that makes the best use of those model elements. The transformation will understand the domain-specific elements, and will produce an output model that reflects this understanding.

UML Patterns and transformations can work together and extend each other:

- Transformations can apply patterns
- UML Patterns can execute transformations
- A UML pattern can mark up model elements with the appropriate stereotypes to prepare for a transformation
- Transformations and UML Patterns can be contained in the same plug-in project

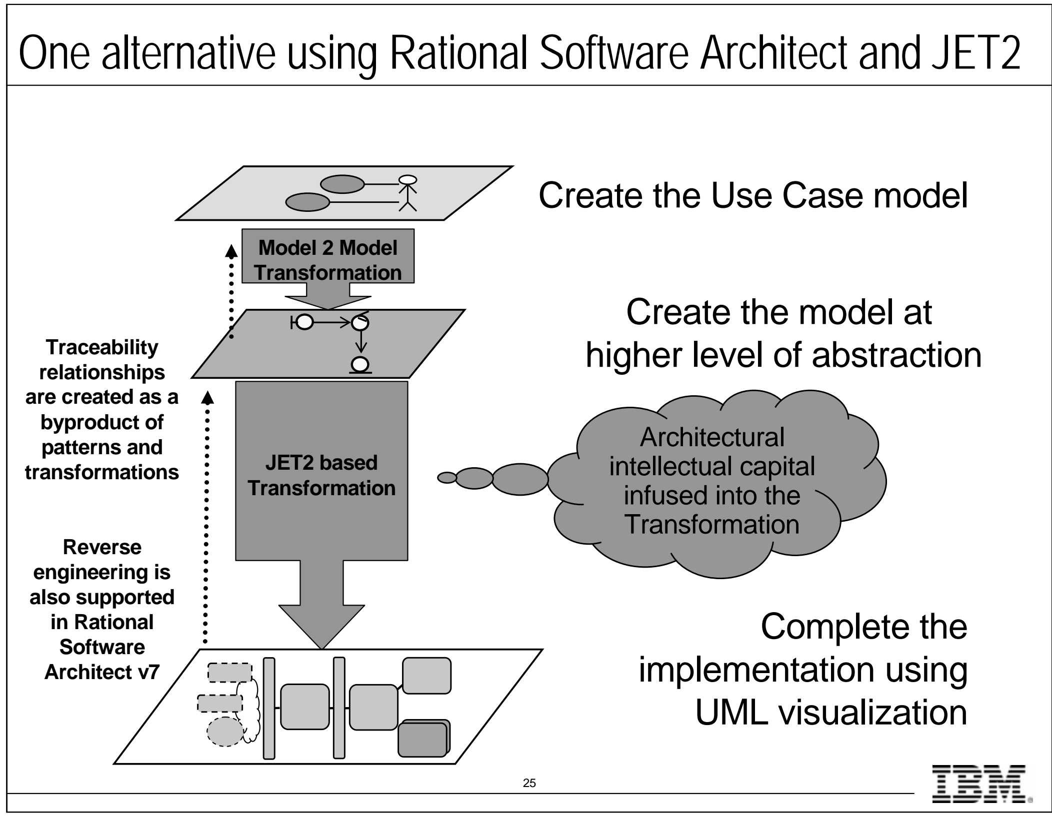
© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.



© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

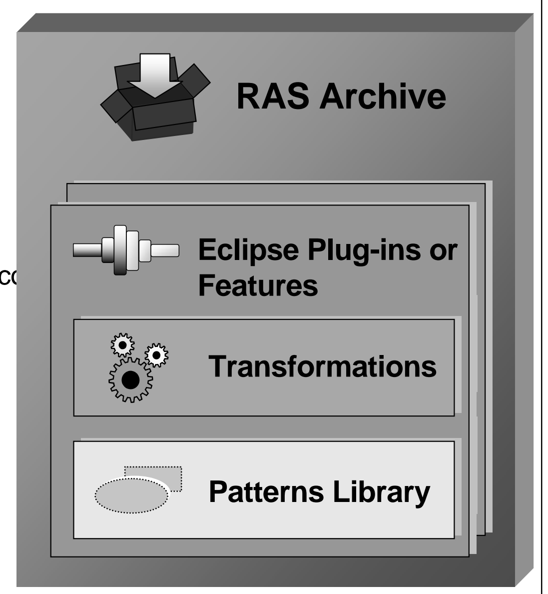


This represents one of various approaches customers have taken to reduce the amount of modeling (thus limiting variability) while infusing consistent architecture in the form of a pattern-based transformation.

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Packaging Artifacts

- Artifacts can be packaged and shared using the Reusable Asset Specification
- Why package artifacts?
  - Group related and dependent artifacts into whole solutions
    - Recipes
    - Pattern Solutions
    - Eclipse Features
  - Install artifacts easily
  - Distribute assets without giving out source files, with no user c
  - Add branding information
- Package artifacts
  - The asset
  - Example models
  - Help and other documentation
  - Workbench menu items, actions and so on



26

Often they are used together as recipes or solutions. You will see during a running example how these assets can come together. Note that this is just the starting point – additional patterns and transformations may be added to this recipe or solution.

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Demonstration: Reusable Artifacts

- The instructor will now show you how to:
  - Use a reusable asset

27

IBM

The instructor will show you how to use a reusable asset.

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Review

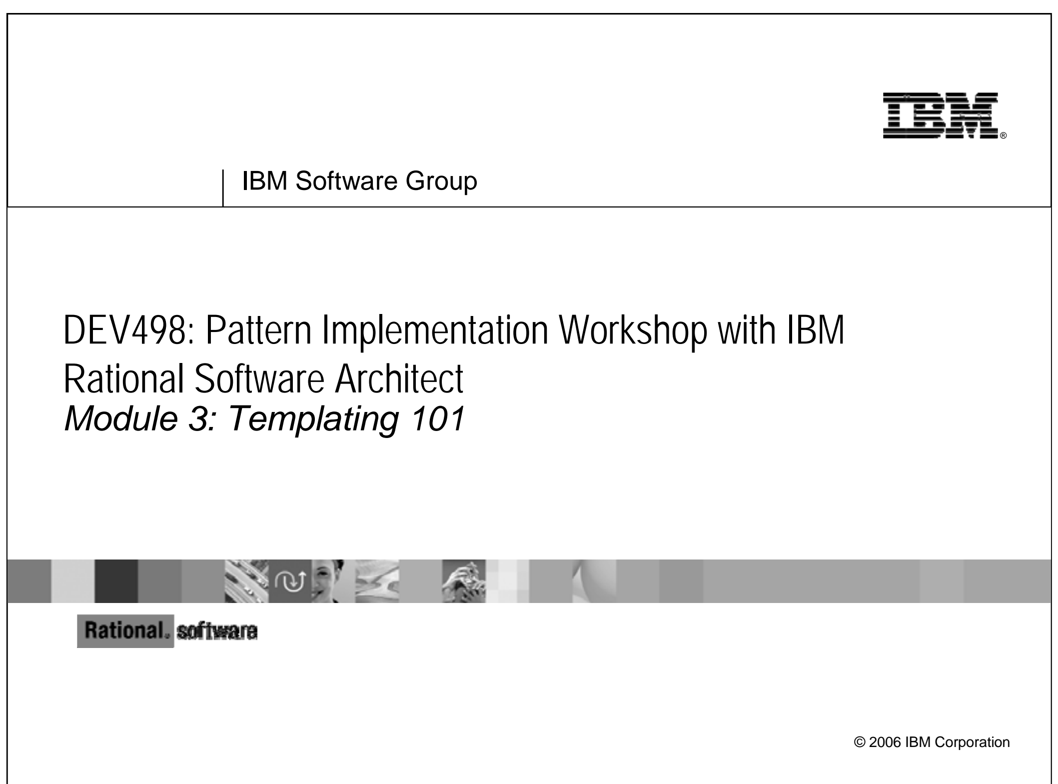
- What are cases where you would combine artifacts into a reusable asset?
- Why extend the functionality of Rational Software Architect?
- What artifacts can you build with Rational Software Architect that promote reusability?
- Why develop pluglets?

28

IBM

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.



Contents

Objectives 3-2
Original JET 3-5
EMFT JET 3-9
Review 3-13
Further Information 3-14

© Copyright IBM Corp. 2007
Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Templating 101

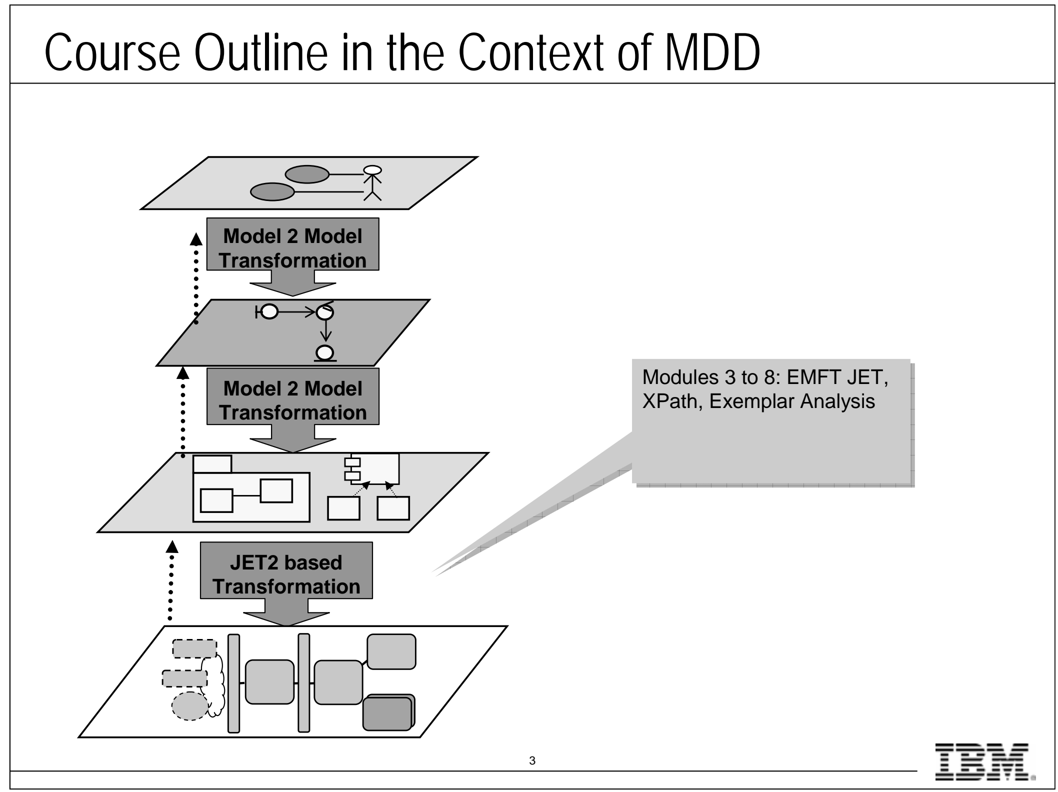
- Objectives:
  - Describe the structure and syntax of a JET template
  - Discuss the differences between JET (Original Templates) and EMFT JET

2

IBM

There are two variants of JET (Java Emitter Templates). There's the original JET function that came out sometime during Eclipse V1 and there's the new set of enhancements that came out recently and upon which this workshop is based.

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.



We will see this slide several times throughout the workshop. It will serve as a visual guide to the skills we are learning and how they fit into model Driven Development.

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Where Are We?

- **Original JET (JET1)**
- EMFT JET (JET2)

IBM
© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Two JETs

- JET1 (original JET)
  - Templates as helper classes
  - Control flow via embedded Java
- JET2 (EMFT JET with Eclipse 3.2 and Rational Software Architect 7.0 and later)
  - JET1 support
  - Stand-alone transformations
  - Control flow, function via tags

5

IBM

Templating is the best way to produce large amounts of text programmatically.

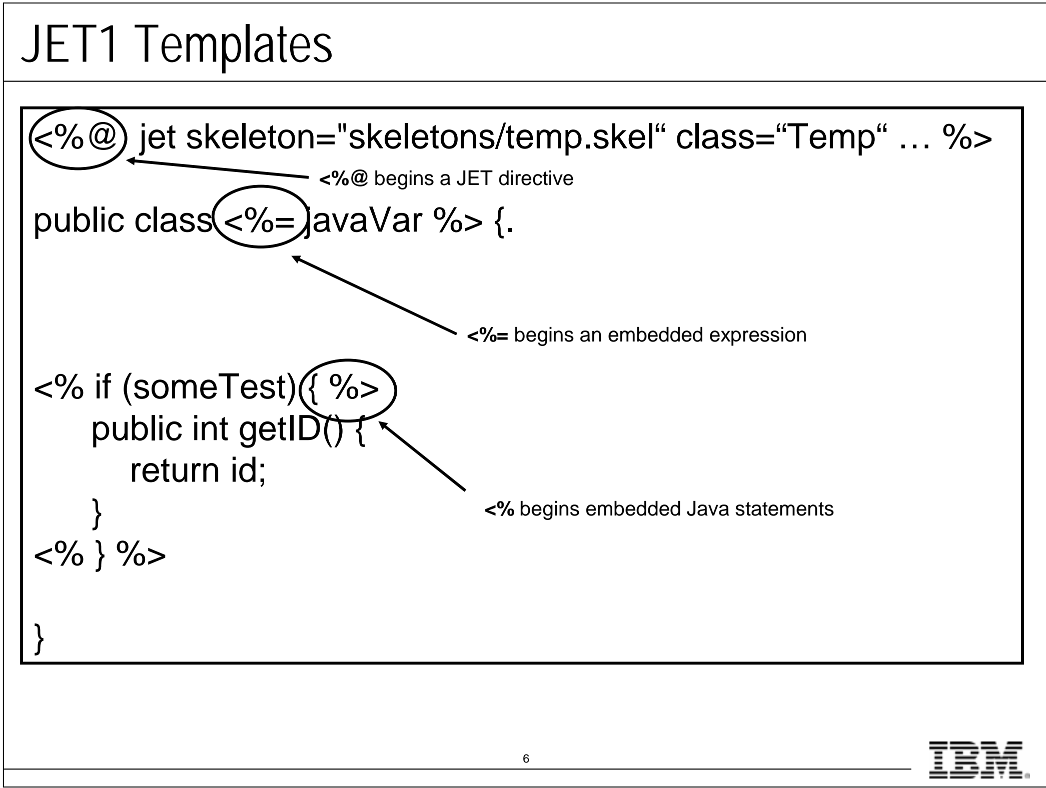
When you worked with the "original JET" you were always writing a Java application (usually an Eclipse tool) that needed to generate a large amount of text output. When you wrote your JET templates, you had to view them as helper classes that added a templating component to the larger application. The templates were marked up with imbedded Java expressions and code, and you had to be aware of the actual data model implementation (the business objects) that was referenced by that embedded Java.

In contrast, the new enhancements to JET allow you to build stand-alone transformations using nothing but templates. There is no Java required to invoke the templates, and the templates themselves do not require Java in order to access the data model. The Java has been effectively replaced by some 50 JET tags that encapsulate the common (and uncommon) templating behaviors.

Somebody almost always asks if the tags are really simpler to use than Java. That question is answered in the next few charts.

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

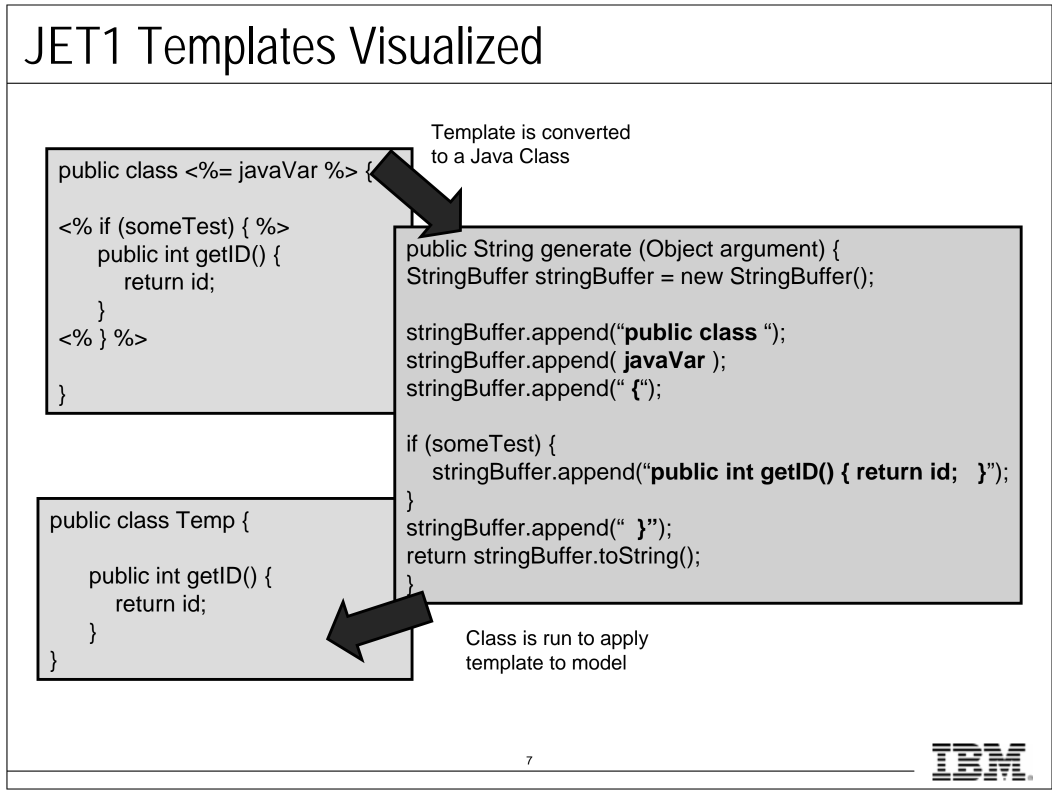


Start by looking at a representative JET (original) template. The template editor in Eclipse doesn't colorize the text, but we'll use blue to highlight the imbedded JET elements.

Each JET element begins with “<%” (open angle bracket percent sign), and the next character tells what kind of element it is. There are directives that essentially act as compile options, there are elements that contain Java expressions, and there are elements that contain Java code. Note that the Java code in any one element doesn't have to be syntactically correct, but the overall set of embedded Java does have to be syntactically correct.

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.



Take a look at what goes on "under the covers" with JET.

You should know that when you edit a Java source file and save, the Eclipse tooling will automatically compile that code and store the resulting .class file in another part of the project. The JET tooling works in much the same way. When you edit and save a JET template, a Java class is generated from the template source and is compiled. The resulting source and .class files are stored back into the same project.

The generated Java class has a generate method that takes an object, constructs a StringBuffer, appends a bunch of stuff to the StringBuffer, and finally returns the contents of the StringBuffer. The class is generated as follows:

1. Static text causes a line that appends the text to the StringBuffer
2. Imbedded expressions are resolved and appended to the StringBuffer
3. Java source is copied as-is

When the generate method is invoked, the logic in that method performs the templating intent of the JET template, and returns the string result.

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Where Are We?

- Original JET (JET1)
- **EMFT JET (JET2)**

8

IBM

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

EMFT JET Templates

- Use tags in addition to embedded Java
  - Common tasks
    - CRUD model data
    - Control template processing
    - Eclipse resource handling
    - Re-apply support
- Extensible architecture
  - Write your own tags, parsers, inspectors, functions

9

IBM

Internally, the new JET (what we'll call "JET" for the rest of this workshop) works much the same way as the old JET. In fact, embedded Java is still supported. The new JET has added a number of tags that support both common and uncommon templating behaviors. In addition, JET is extensible, so you can write your own tags.

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

JET2 Templates

```
public class <c:get select="$bean/impl/@name" /> {

<c:if select="$bean/impl/@type = 'keyed'" >
     public int getID() {
        return id;
     }
</c:if>

}
```

Tags can write data from the model

Tags can control template processing

10

This is the same template (as examined in original JET1) written with JET2 tags.

The `<c:get>` tag reads a piece of data (usually a string) out of the model and writes it inline with the template.

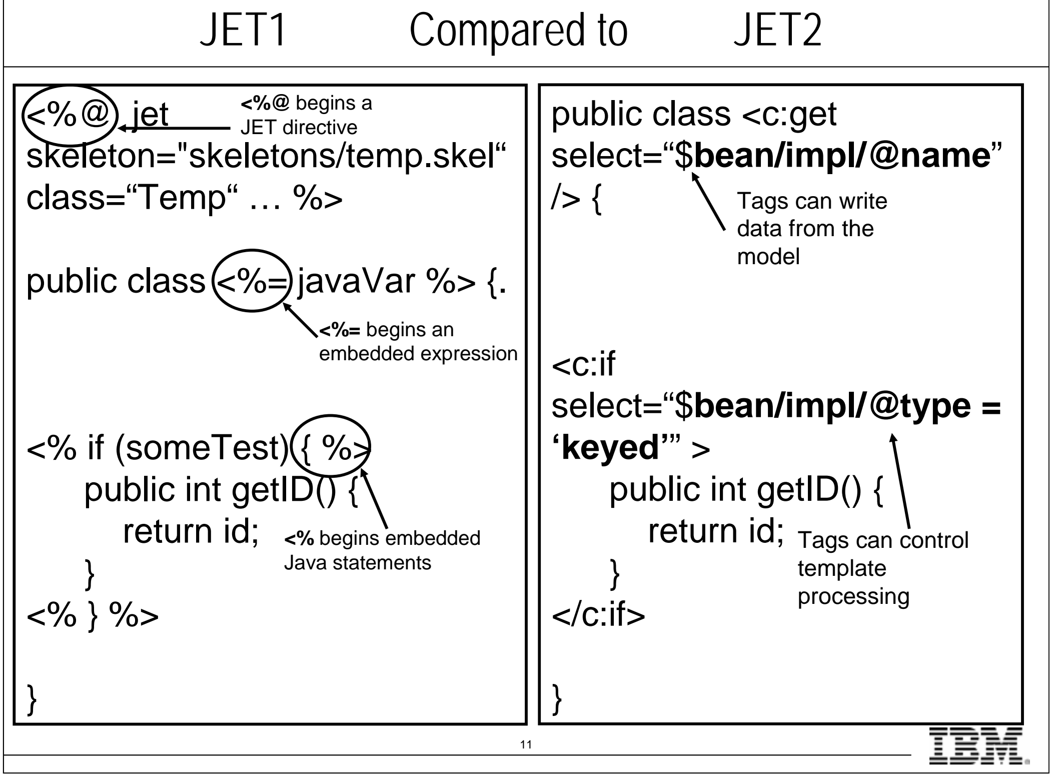
The `<c:if>` tag performs some test, usually using data in the model. JET will only process the contents of the `<c:if>` tag if that test resolves to `True`. In this case, processing the tag's content will result in the `getID` method source being written out as part of the template output.

While it may seem that these tags might be as complex (if not more so for Java programmers) as their embedded Java counterparts, there are a number of tags that are the equivalent of a great deal of embedded Java code. For example, the `<ws:file>` tag will:

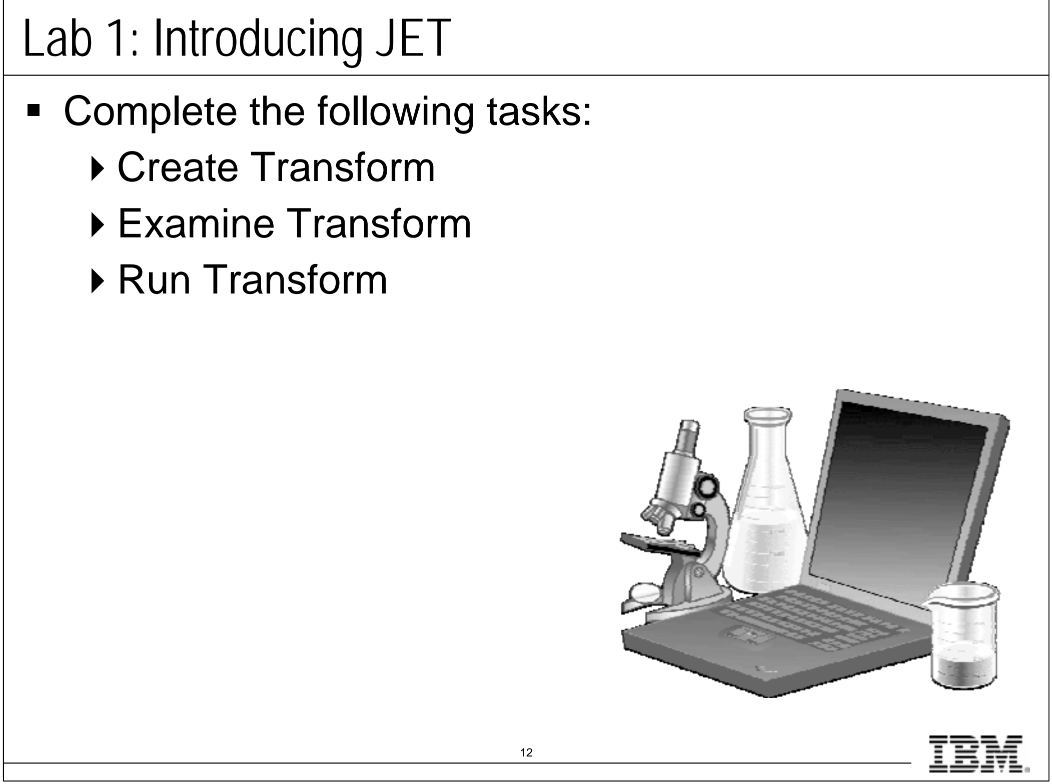
- Apply a template to the model
- Write the resulting string to a file in , but Eclipse only after
  - Collaborating with the CM plug-in to make sure that the file is checked out and otherwise modifiable
  - Collaborating with any editors that might have the file already open

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.



Course materials may not be reproduced in whole or in part without the prior written permission of IBM.



Complete Lab 1 in the student workbook.

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Review

- What are the differences between the original JET and EMFT JET?
- Should you use JET or EMFT JET? Why?

IBM

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Further Information

- Web resources
- Eclipse Help

14

Web Resources

Eclipse website (www.eclipse.org)

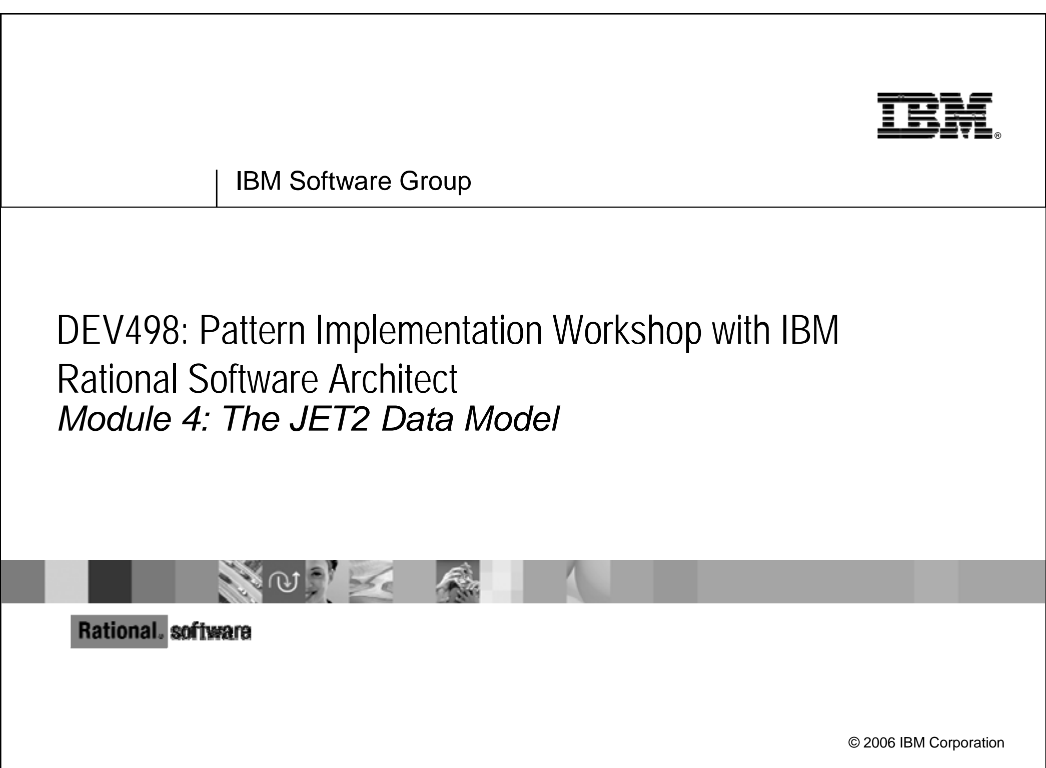
Eclipse Help

Eclipse Help > EMF Developer Guide > Tutorials > JET Tutorial Part 1

Eclipse Help > EMF Developer Guide > Tutorials > JET Tutorial Part 2

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.



Contents

Objectives	4-2
JET Data Model	4-4
XPath	4-8
Review	4-28
Further Information	4-29

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

The JET2 Data Model

- Objectives:
  - Describe the JET data model
  - Use XPath to query an XML model

IBM

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Where Are We?

- **JET Data Model**
- XPath
  - Query Expressions
  - Examples

3

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

JET Transforms

- JET Transforms are programs
  - Have a tag-based language
  - Have a data model
- JET Tags
  - Encapsulate common transformation behaviors
  - Some act against the model
  - Some perform templating actions

4

IBM

Although you do not need to write any Java to build a JET transform, each JET transform is still a full-blown program in its own right, with syntax and a data model.
This course has talked about the JET tags that perform various templating functions. Most of those tags act against the model in some way. This section is going to cover how a tag describes the part of the model that is to be the target of the tag's behavior.

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

JET Data Model

- Implementation level
  - Bunch of Java objects floating in a Java™ Virtual Machine (JVM) environment
  - Accessed via embedded Java

```
<%=   policy.getCustomerID()   %>
```

- Access via tags
  - Objects organized into DOM's
    - Optimal for templates: simple, easy access
    - Navigation and access via XPath
  - Tags use XPath to target their function

```
<c:get  select=" $policy/@customerID "  />
```

5

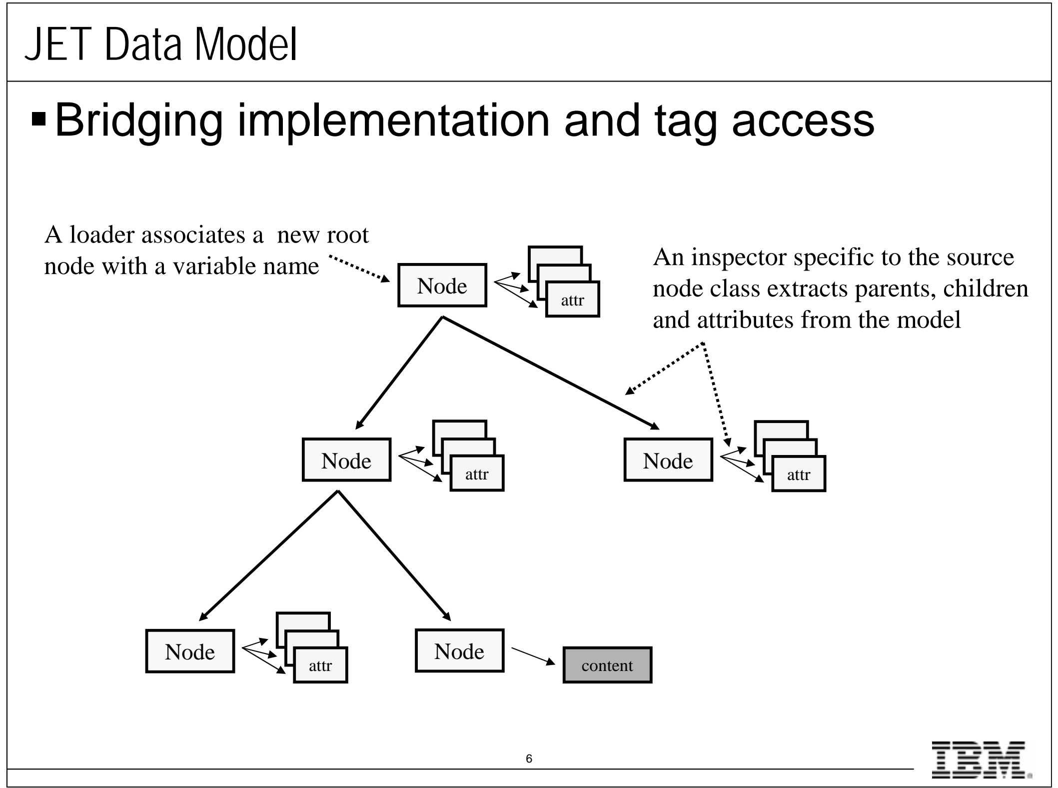
IBM

With JET, there are two ways to think of the data model:

1. The model is implemented as a set of Java objects, and if you can access them with imbedded Java elements.
2. However, you should view the data model as a set of objects organized in DOM's (tree structures of data objects). It turns out that no matter what the data model implementation is, templates almost always access the data in those models as if the data were in a DOM. Once the data is made available as a DOM (even if the implementation is otherwise), you can use XPath to access data in that DOM.

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.



To reiterate, the data model passed in to a JET transform can be in any shape or form, but the JET tags will access that data using XPath as if the data were in a DOM.

JET uses a loader to load a model into memory. The loader can load just the root node or the entire tree, but it returns the root node to JET. Inspectors help to access the data as if it were in a tree by answering questions like, given a model object:

- What is its parent?
- What are its children?
- What is its name?
- What are the names of its attributes?
- What is the name for the value of a given named attribute?
- What is the tag’s content?

Some loader and inspectors come with JET. They support:

- XML documents
- EMF documents (files and in-memory)
- UML
- Eclipse resources (IProject, IFolder, and IFile)

You can also write your own loaders and inspectors if you need to.

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Where Are We?

- JET Data Model
- **XPath**
  - **Query Expressions**
  - **Examples**

7

IBM

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

XPath

- Query Expression
  - Interpreted string that resolves into a value
    - String, List of objects, Object, integer, and so on
  - Can also perform calculations
  - Usually describes a model traversal
- Two forms
  - XPath verbose
  - XPath abbreviated
    - 90-10 rule: optional tokens defaulted
- www.w3.org/TR/1999/REC-xpath-19991116

8

An SQL statement is a string value that, when processed, returns data from a relational data base.

In a similar fashion, an XPath expression is a string that, when processed by an XPath processor, will resolve to some set of data from the model.

It is strongly recommended that you download this page: www.w3.org/TR/1999/REC-xpath-19991116. It contains the proposed XPath specification, and is an extremely handy document to be able to refer to when you have a question about XPath.

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Query Expressions

- A query expression has:
  - A description of where to begin the traversal
  - A series of steps, separated by slashes (“/”)
    - **step 1 / step 2 / ... / step n**
- Step
  - A simple traversal between related nodes
  - Consists of:
    - An axis
    - A node test
    - Predicates

9

IBM

Most of the time XPath query expressions are used to navigate the model, and return the result of that navigation. The result of a query expression is the set of objects that you arrive at when that navigation finishes. Each tag has its own way of further using the query expression result.

When a query expression is being used to navigate the model, that query expression will somehow indicate where that traversal will begin (which model object) and will have some number of steps, each of which describes a primitive navigation.

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Query Expressions: Start and Context Objects

- Each query expression has to start somewhere
- Three variants differ by beginning syntax
  - Expression starts with "/"
    - Start node is the document node (top element's parent)

      /page/link
  - Expression starts with "$"
    - Start node specified via variable name
    - Variable's value is the node from which to start

      $page/link
  - Otherwise, start node is the Context Object
    - For a "stand alone" expression, it's the document node
    - For a nested expression, it's the "current" node

      page/link

10

IBM

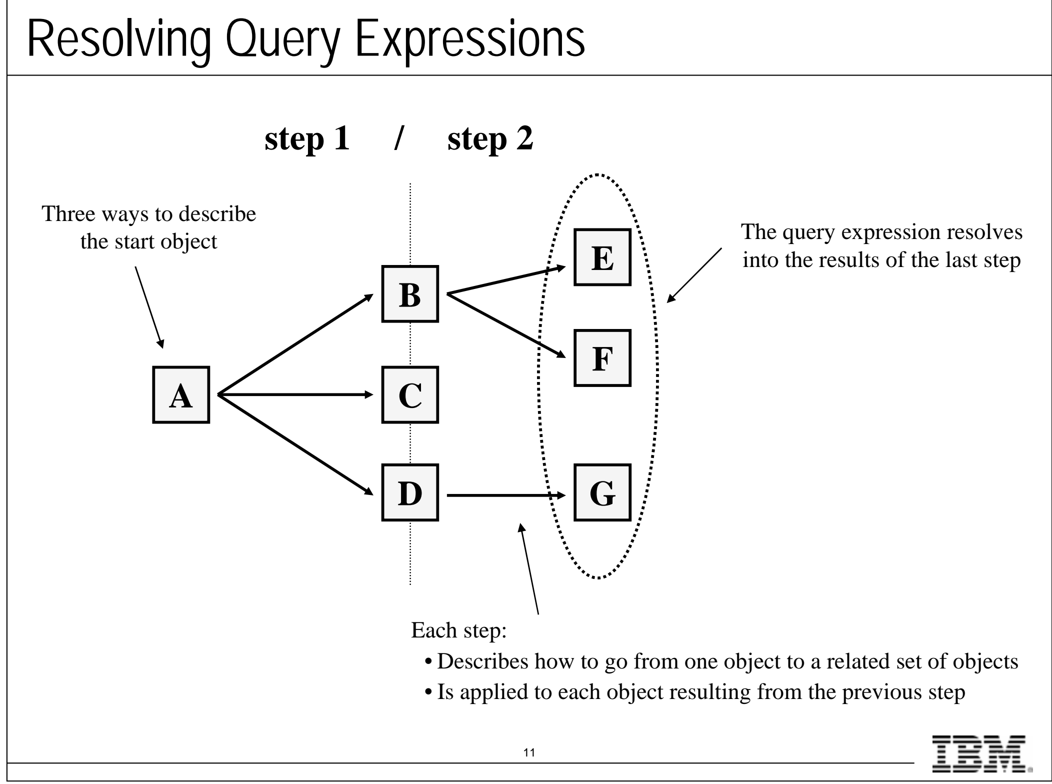
There are three ways that a query expression can indicate the start object for a model traversal:

1. If the expression begins with a forward slash, then the start object is the parent of the root. In XML terminology, this would be the document node – the parent of the high-order element.
2. If the expression begins with a dollar sign, then the following token is the name of a variable that should already be associated with a model object. That model object is the start object.
3. Otherwise, the start object is the context node. This usually happens when a query expression has a nested query expression for the purpose of performing a test on some node. The node being tested is the context object, and is where that nested expression would start (if that expression doesn't start with "$" or "/").

Note that the examples (boxed in blue) have two steps, one step, and two steps, respectively.

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.



Without the benefit of knowing exactly how a step is specified, let's look at an abstract example.

Here, a query expression starts at model object A. You do not care which of the three ways to specify the start object was used in this example. It is enough to know that you start at object A.

The first step (of two) in this query expression describes a simple navigation that, when followed, takes you from object A to objects B, C, and D. For example, the step might be from the source node (object A) to the source node's child nodes (B, C, and D in this case).

The second step also describes a simple navigation, and you follow that navigation from each of the objects resulting from the previous step (objects B, C, and D). From object B, that navigation takes you to objects E and F. There is no object that results from navigating from node C. When performing the step 2 navigation from node D, you get to node G.

Since there are only two steps in this example, the union of the nodes resulting from the last step (nodes E, F, and G) is the result of the query expression.

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Query Expression Steps: Axis

- Axis
  - Relationship between the source node and the target nodes
- XPath supports
  - Child, descendant, parent, and ancestor
  - Following-sibling and preceding-sibling
  - Following and preceding
  - Attribute and namespace
  - Self, descendant-or-self, and ancestor-or-self

12

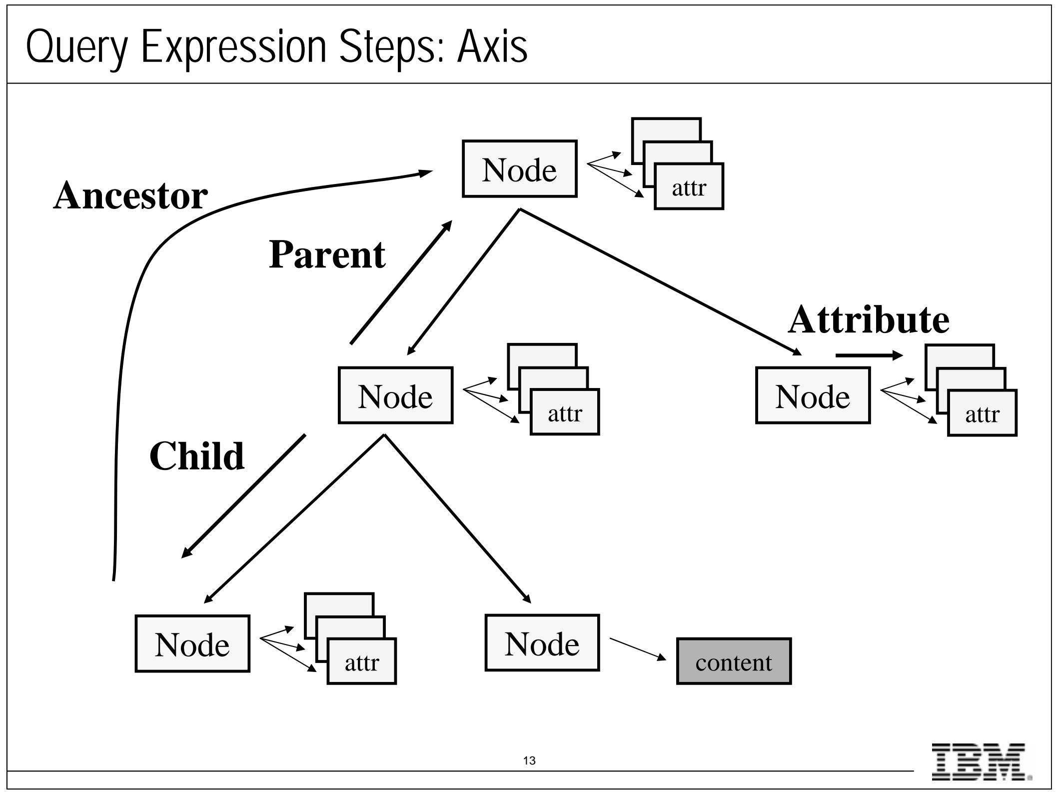
Each step is composed of three components, whether they are specified or defaulted. The first component is the axis, which describes the relationship between the source object and the target objects. Another way to think about the axis is as the direction of the simple traversal. Common axes include the child axis and the parent axis.

Since attributes are exposed as attribute nodes (regardless of the model implementation) another common axis is the attribute axis. In order to access the value of an object's attribute, you need to traverse from that model object to the attribute node representing that attribute. That traversal takes place along the attribute access.

There are other axes (listed on this chart). See the URL mentioned previously for a precise definition of each.

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

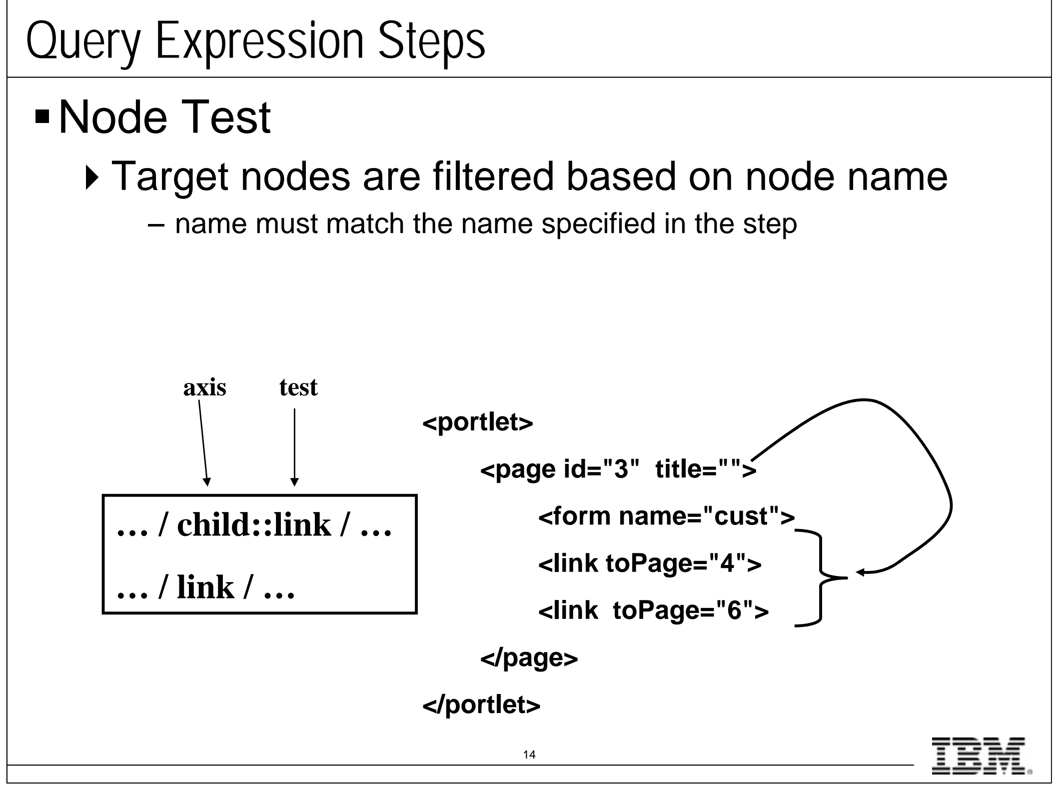


This visual demonstrates the traversals associated with common axis types.

Note that a singular axis ("Child" or "Ancestor") might reach multiple nodes. There may be zero or one parent, but there can be many children, ancestors, and attributes.

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.



The second component of a step is the node test. The axis is traversed to reach a set of target nodes, and those nodes are filtered based on their names.

In the example on this chart, the step `child::link` (or just “link” for short since child is the default axis) traverses the model to three child nodes, and two of those nodes are named “link.” The step results, therefore, in two nodes.

By the way, to bypass this node test and get all of the nodes reached by the axis, use the name “*” (asterisk). For example, “child::*” and the shorter “*” both result in traversing to all children of the source node.

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Query Expression Steps

- Predicate
  - A filter on a list of model objects
  - Form: [expression]
    - Expression can use and, or, ! (not), >, <, =, and ()
    - Values can be:
      - Variable references
      - Literals
      - Numbers
      - Function calls: position(), last(), count(), name(), namespace()
      - Query expressions
        - Node being tested is the expression's context object

15

The axis and node test are used to arrive at a set of nodes. Then, the third component, the predicate, is used to further filter that set of nodes.

The predicate consists of one or more expressions, each of which is enclosed in square brackets. Each predicate is used to filter the list of nodes before the next predicate is used (examples below). If the expression for a predicate contains a query expression, the context node for the query expression is the node being tested by that predicate.

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Query Expression Examples

page/link

– All of the children named “link” of all of the children named “page” of the expression’s context object

$page/link

– All of the children named “link” of the object associated with $page

/root/schema/@name

– The value of the name attribute of the high-level element’s child name “schema”, but only if the high-level element is named “root”

page [@id = ‘p001’]

– All of the children named “page” of the context object whose id attribute value is “p001”

– “@id” is a nested query expression

16

IBM

Some examples follow.

Note that the character “@” (the “at sign”) is short for the attribute axis. The string “@name” is a step whose node test is “name” and whose axis is the attribute axis.

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

JET Variables

- Variables indirectly reference nodes.
  - Variable names are passed as tag attributes

```
<c:setVariable select="/root/view"  var="view"   />
```

  - Are also passed in query expressions

```
<c:get select="$view/@label"  />
```

  - Can have a scope

```
<c:iterate select="/root/page"  var="page"   >
        <c:get   select="$page/@id"  />
</c:iterate>
```

17

We have shown how a “$” in front of a query expression means that the traversal defined by that query expression begins at a node associated with the given variable name.

This chart talks about variables more broadly, and about common ways to use them.

The first example shows how a variable can be set to a node, which in turn is the result of a query expression. From that point on, as in the second example, that variable name can be used to begin query expressions. The query expression traversal would begin at that original node (the one found by the query expression in the first example). Note that since all templates in a JET transform share the same data model, this association remains in affect in subsequent templates.

Another common use of variables is as a reference to the current node in an iteration over a set of nodes. In the third example, the `<c:iterate>` tag will use a query expression to get a collection of nodes. The tag will then iterate over that collection of nodes. For each node, the `<c:iterate>` tag will associate that node with the given variable name (“page” in this case), and will then process the `<c:iterate>` tag content. When the tag content has been processed for the last node in the collection, the variable name is disassociated with the last node, and does not have a value after the `</c:iterate>`.

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

JET XPath Functions

- Functions within an XPath expression
  - Write the value of the attribute named “package”

```
<c:get select=" $view/@package "    />
```

  - Write the corresponding directory

```
<c:get select="translate( $view/@package , '.' , '/' )"    />
```

18

XPath query expressions can also make use of functions.

In the first example, a `<c:get>` tag writes out a value that appears to be a package name – something in the form of a.b.c.d.

The second example shows the same `<c:get>` tag, except that the query expression in the first example is now an argument to the translate function. The translate function replaces all occurrences of one character with another character. This particular function example converts a package name (form: a.b.c.d) into the corresponding folder name (for example, a/b/c/d) by replacing all of the periods with forward slashes.

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Common functions you will likely use for simple formatting include:

- `lower-case` lowers the case of every character in a string to lower case
- `lowercaseFirst` lowers the first character in a string to lower case
- `upper-case` raises every character in a string to upper case
- `uppercaseFirst` raises the first character in a string to upper case

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

JET XPath Functions

- Calculations within an XPath expression
  - Set a variable to an integer value

```
<c:setVariable select=" 1 "  var="counter"    />
```

  - Increment the integer value of a variable

```
<c:setVariable select=" $counter + 1"  var="counter"    />
```


IBM

You can also use query expressions to perform calculations.

The first example sets variable “counter” to the integer value of 1. The second example adds 1 to the current value of variable “counter”, and stores the result back into variable “counter”.

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

XPath Examples

/root/library/book

```
<root>
  <library>
    <librarian name="Paige Turner" empno="123456" />
    <book id="001" pages="420">A Pattern's tale</book>
    <book id="002" pages="210" missing="true">The Seventh Sense</book>
    <book id="005" pages="293" missing="false">Patterns and You</book>
    <book id="021" pages="10"   missing="false">For the Love of patterns</book>
  </library>
</root>
```

1

21

IBM.

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

XPath Examples

/root//book

```
<root>
  <library>
    <librarian name="Paige Turner" empno="123456" />
    <book id="001" pages="420">A Pattern's tale</book>
    <book id="002" pages="210" missing="true">The Seventh Sense</book>
    <book id="005" pages="293" missing="false">Patterns and You</book>
    <book id="021" pages="10"   missing="false">For the Love of patterns</book>
  </library>
</root>
```

22

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

XPath Examples

/root/library/*

```
<root>
  <library>
    <librarian name="Paige Turner" empno="123456" />
    <book id="001" pages="420">A Pattern’s tale</book>
    <book id="002" pages="210" missing="true">The Seventh Sense</book>
    <book id="005" pages="293" missing="false">Patterns and You</book>
    <book id="021" pages="10"   missing="false">For the Love of patterns</book>
  </library>
</root>
```

1

23

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

XPath Examples

(1) **/root/library/book [2]**

(1) **/root/library/book [position() = 2]**

(2) **/root/library/book [last()]**

```
<root>
  <library>
    <librarian name="Paige Turner" empno="123456" />
    <book id="001" pages="420">A Pattern's tale</book>
(1) <book id="002" pages="210" missing="true">The Seventh Sense</book>
    <book id="005" pages="293" missing="false">Patterns and You</book>
(2) <book id="021" pages="10"   missing="false">For the Love of patterns</book>
  </library>
</root>
```

24

IBM

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

XPath Examples

1. **/root/library/book [@missing] [2]**
2. **/root/library/book [2] [@missing]**
3. **/root/library/book [1] [@missing]**

```
<root>
  <library>
    <librarian name="Paige Turner" empno="123456" />
    <book id="001" pages="420">A Pattern's tale</book>
(2) <book id="002" pages="210" missing="true">The Seventh Sense</book>
(1) <book id="005" pages="293" missing="false">Patterns and You</book>
    <book id="021" pages="10"   missing="false">For the Love of patterns</book>
  </library>
</root>
```

(3) False Boolean value

25

IBM

This subtlety is worth illustrating.

Example 1 is the 2nd book that has an attribute named “missing”.

Example 2 is the 2nd book if it has an attribute named “missing”.

Example 3 is the 1st book if it has an attribute named “missing”.

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

XPath Examples

1. **/root/library/book [3]**
2. **/root/library/book [3]/..**
3. **/root/library/book [3]/../librarian**

```
<root>
(2) <library>
(3) <librarian name="Paige Turner" empno="123456" />
    <book id="001" pages="420">A Pattern's tale</book>
    <book id="002" pages="210" missing="true">The Seventh Sense</book>
(1) <book id="005" pages="293" missing="false">Patterns and You</book>
    <book id="021" pages="10"   missing="false">For the Love of patterns</book>
  </library>
</root>
```

26

IBM.

The double period is short for the parent axis.

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

XPath Examples

1. **/root/library [book]**
2. **/root/library/book [author]**

```
<root>
(1) <library>
      <librarian name="Paige Turner" empno="123456" />
      <book id="001" pages="420">A Pattern's tale</book>
      <book id="002" pages="210" missing="true">The Seventh Sense</book>
      <book id="005" pages="293" missing="false">Patterns and You</book>
      <book id="021" pages="10"   missing="false">For the Love of patterns</book>
    </library>
</root>
```

(2) False Boolean value

27

IBM

Example 1 is the collection of library elements that have a child named “book”.

Example 2 is the collection of book elements that have a child named “author”.

In both cases, the string inside the predicate is not a numeric value or a quoted string. The value is assumed to be a query expression. Since the query expression does not start with “$” or “/”, the start node for the navigation is the node being tested (for instance, the library node in the first example). Since no axis is specified, it is assumed to be the Child axis, and the string is used as the node test for the step. If there are children by that name, the returned collection of objects is not empty, and it is converted by the predicate (remember, it’s a boolean expression) to true.   If there are no children by that name, the returned collection is empty. Empty collections are converted to a false boolean value.

© Copyright IBM Corp. 2007

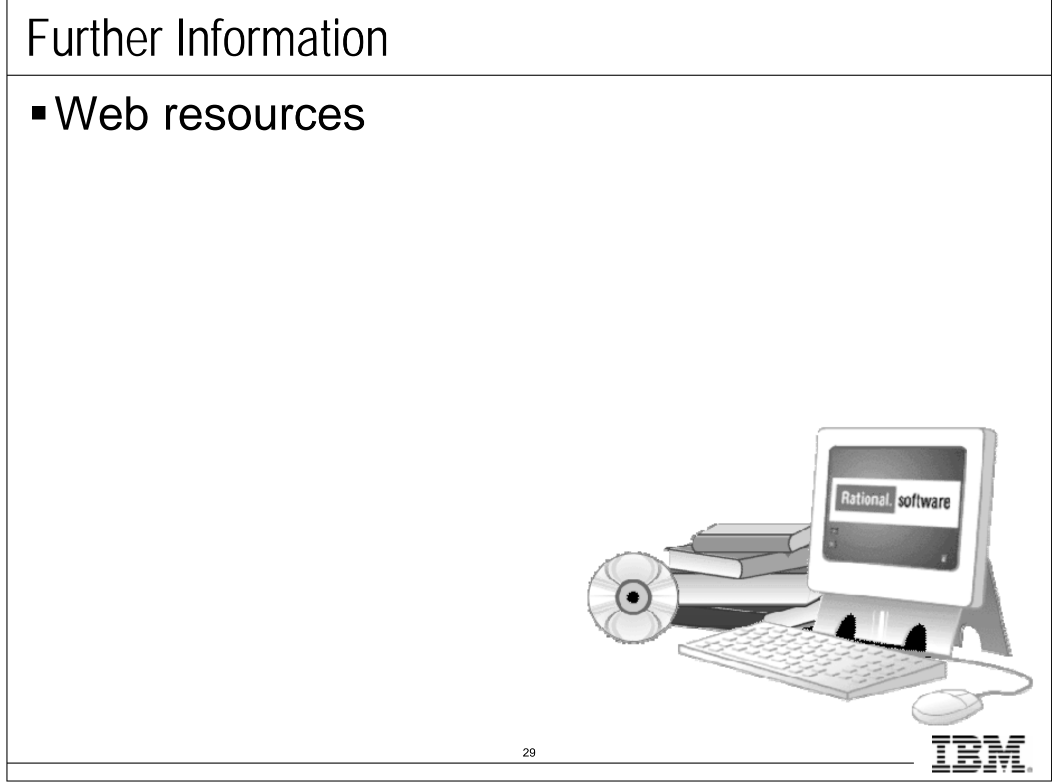
Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Review

- What is a query expression?
- How are variables used?
- What are the three components of a step?
- How do those three components work together?

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.



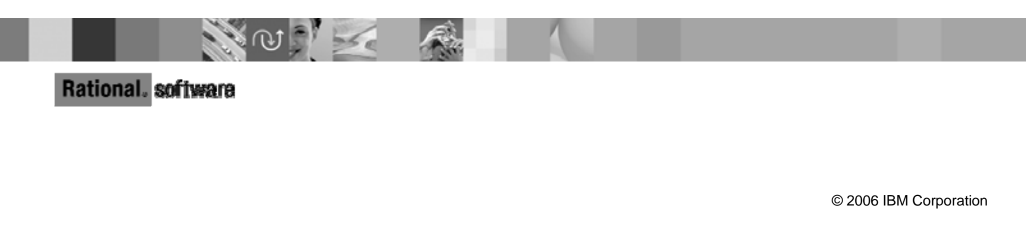
Web Resources

- XPath Specification:http://www.w3.org/TR/1999/REC-xpath-19991116
- Chris Aniszczyk and Nathan Marz. "Create more -- better -- code in Eclipse with JET." *IBM developerWorks*. http://www-128.ibm.com/developerworks/opensource/library/os-ecl-jet
- Adrian Powell. Model with the Eclipse Modeling Framework, Part 2: Generate code with Eclipse's Java Emitter Templates. *IBM developerWorks*. http://www-128.ibm.com/developerworks/opensource/library/os-ecemf2/

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

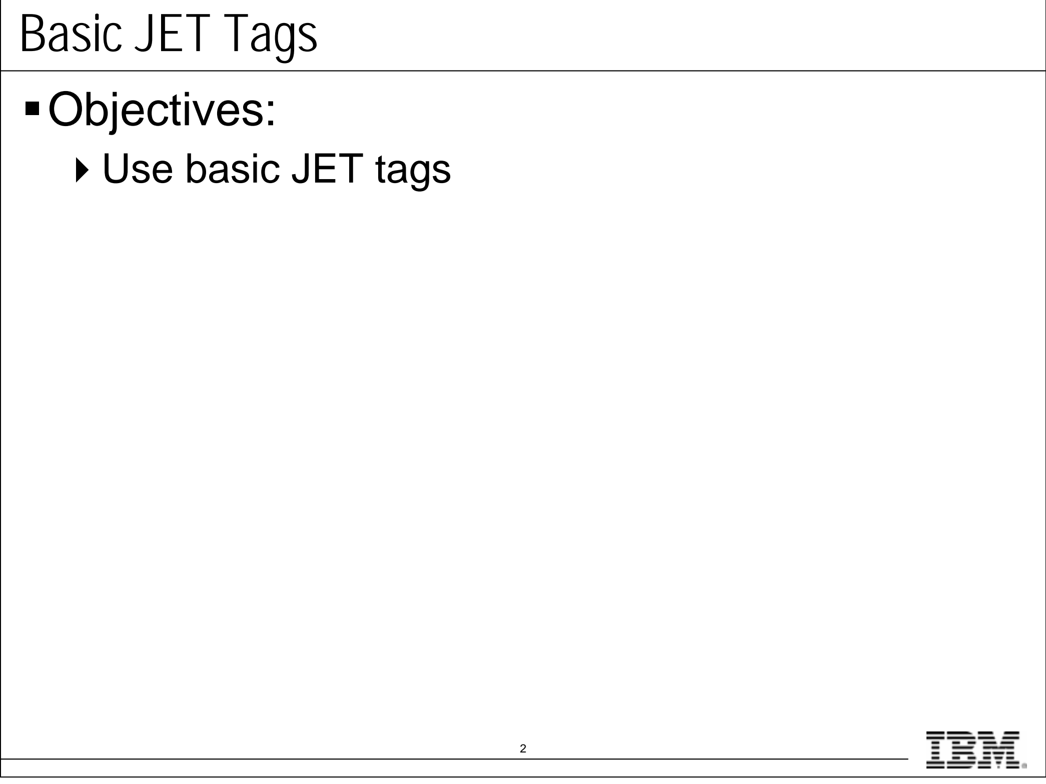
© Copyright IBM Corp. 2007
Course materials may not be reproduced in whole or in part without the prior written permission of IBM.



Contents

Objectives 5-2
The Basic JET Tags 5-3
Review 5-12

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.



Of the 50 or so tags that come with JET, you'll find yourself using only nine tags for most of your template authoring.

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

The Basic JET Tags

- ws:file
  - Applies a template to the model
  - Stores the result in a generated file
- c:get, c:set, and c:dump
  - Read and write model data
- c:iterate
  - Processes its body once for each node matching criteria
- c:choose, c:when, and c:otherwise
  - Process the body of exactly one case or default tag
- c:if
  - Conditionally processes template body

3

IBM

These are the nine most commonly used JET tags.

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

<ws:file>

- Applies a template to the model
- Stores the result in a generated file
- Attributes
  - template
    - Relative path name of the template to be applied
  - path
    - Full name of the file to be generated
    - Format: "/project/relativePath"
  - replace
    - Optional
    - Whether or not to replace file if it already exists

4

IBM

It's important to note that the entire data model (including all of the currently defined variables) is made available to the specified template when it is applied. Any changes made by that template to the data model, or to variables, will continue to be in effect after the tag completes.

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

<c:get>

- Reads from the model
- Writes the value
- Attributes
  - select
    - An XPath expression describing the value to be written

```
<c:get   select="$bean/@name" />
```

5

IBM

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

<c:dump>

- Writes out the subtree beneath an object
  - XML representation
  - Several formatting options
- Attributes
  - select
    - Query expression describing a single model object

IBM

It's important to note that the entire data model (including all of the currently defined variables) is made available to the specified template when it is applied. Any changes made by that template to the data model, or to variables, will continue to be in effect after the tag completes.

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

<c:set>

- Processes its template body
- Stores the result back into the model
- Attributes
  - select
    - An XPath expression describing the target model object
  - name
    - The name of the attribute whose value is to be set

```
<c:set   select="$bean"   name="impl" ><c:get   select="$bean/@name" />Impl</c:set>
```

7

IBM

Once all of the nested tags have been processed, the value of the `<c:set>` tag's content is the new value of the attribute being set.

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

<c:iterate>

- Queries a set of objects from the model
  - Processes its template content for each object
- Attributes
  - select
    - An XPath expression describing a collection of model objects
  - var
    - A variable name to be assigned to the current iteration object
  - delimiter
    - A string to be inserted after processing each object except the last

```
<c:iterate   select="$plugin/view"   var="currentView" >
        ....
    <c:get   select="$currentView/@label" />
        ....
</c:iterate>
```

8

IBM

The <c:iterate> tag will retrieve a List of model objects that are the result of the specified query expression. The <c:iterate> tag will then process each object in the List. For each object, the <c:iterate> tag will associate that node with the variable name specified in the var attribute, and the content of the <c:iterate> tag will be processed.

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

<c:choose>

- Processes the content of one of its `when` tags
  - Or it processes the `otherwise` tag by default

```
<c:choose select="$data/@widget" >

<c:when test=" 'text' " >
private Text   text<c:get select="$data/@name"/>;
private String entered<c:get select="$data/@name"/>;
</c:when>

<c:when test=" 'checkbox' ">
private Button  button<c:get select="$data/@name"/>;
private boolean <c:get select="$data/@name"/>Setting;
</c:when>

<c:otherwise>
private Object  object<c:get select="$data/@name"/>;
</c:otherwise>

</c:choose>
```

9

IBM

Note that the select and test attributes are both query expressions. That means that if you want to compare a constant string, like 'checkbox' above, you need to enclose it within single quotes. In other words, the value of the test attribute, specified between double quotes, is a string surrounded by single quotes.

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

<c:if>

- Evaluates the given XPath expression
  - Converts results to a Boolean value
    - One or more objects → true
    - No objects → false
    - Integer 0 → false
  - Processes template content if test is true
- Attributes
  - test – XPath expression
  - var – the value before being converted to boolean

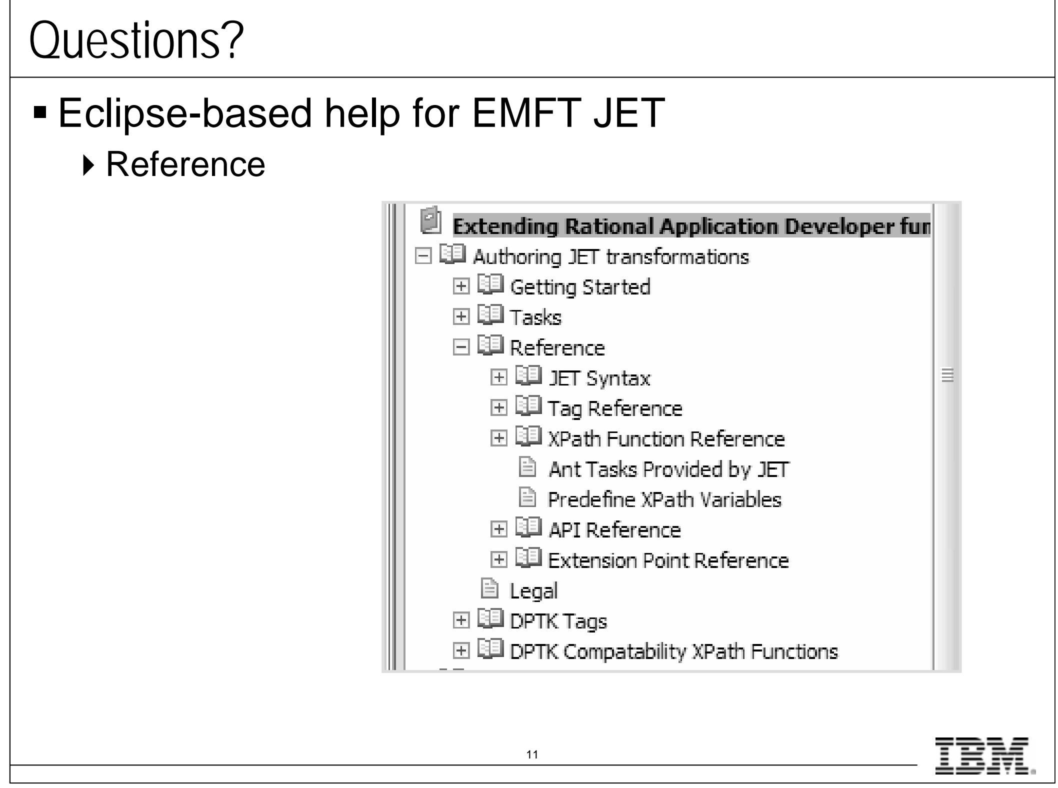
```
<c:if  test="$action [@doubleclick = 'true']"  var="dcAction" >
        list.setDoubleClickAction(<c:get select="$dcAction/@id" />);
</c:if>
```

10

The var attribute is useful if you are testing for the existence of an object. The variable specified by the var attribute is set to the located object, if one exists. This saves you from having to perform the query expression again inside the `<c:if>` contents.

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.



© Copyright IBM Corp. 2007
Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Review

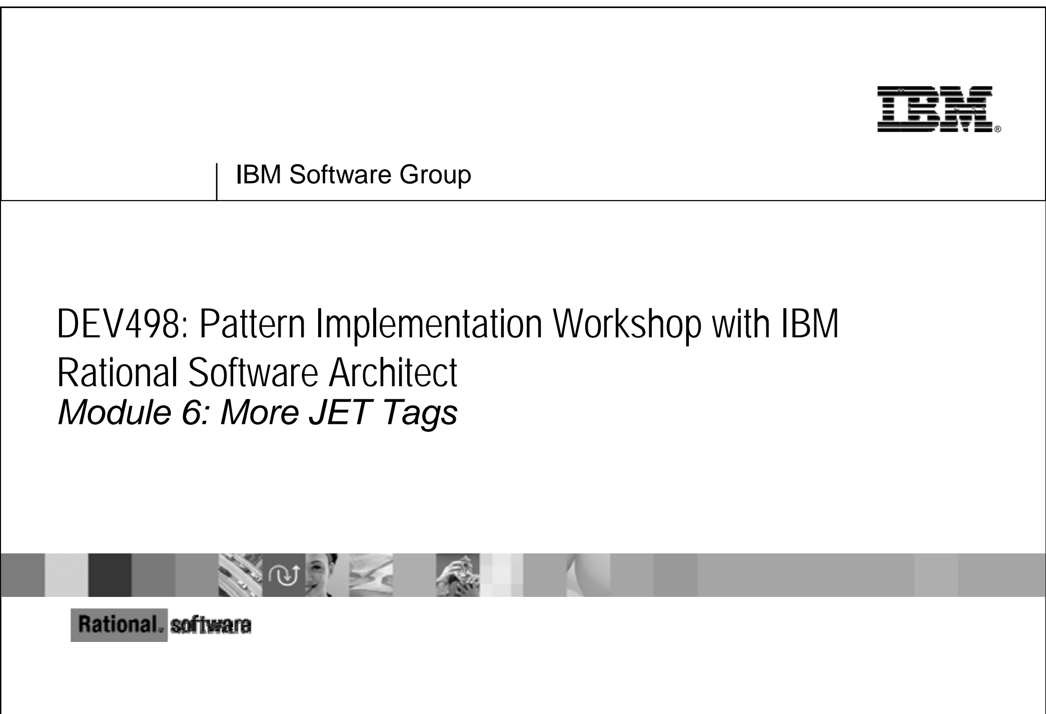
- What tag creates files?
- What tags write model data?
- What are the basic conditional tags?

12

IBM

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.



Contents

Objectives	6-2
Tags and Tag Libraries	6-4
JET2 Control Tags	6-8
Simple Tag Combinations	6-14
Lab 2: Using XPath	6-18
Review	6-19

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

More JET Tags

- Objectives:
  - Use tag libraries
  - Use tags in each library
  - Use simple tag combinations

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Where Are We?

- **Tag Libraries**
- JET Tags
- Simple Tag Combinations

3

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Tags and Tag Libraries

- Tags are configured into tag libraries
  - Standard JET2 control tags
    - Flow control; Pattern-level; Model CRUD
  - Standard JET2 workspace tags
    - Eclipse resource creation
  - Standard JET2 Java tags
    - Java files, resources, or packages; Merge
  - Standard JET2 format tags
    - Unique values
  - DPTK tags
    - DPTK compatibility layer

4

IBM

Tags (whether shipped as part of JET, or written by other JET authors) are packaged together into tag libraries. When you want to use a tag you have to point JET at the tag library containing that tag. You can then use the original tag that you wanted to use, as well as any other tag in the tag's tag library.

There are four standard tag libraries shipped with JET: Control Tags, Workspace Tags, Java Tags, and Format Tags. The tag libraries are available in Eclipse.

A fifth tag library, the DPTK Compatibility Layer, contains tags that look exactly like the DPTK tags, but which are implemented on JET. Using this tag library lets you run DPTK patterns as JET transformations without modifying the templates.

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Using a Tag Library

- Define the library in the transform's plugin.xml

```
<plugin>
   <extension
         point="org.eclipse.jet.transform">
      <transform
            startTemplate="dptk.pattern.test.suite.001a/controller/control.pat"
            templateLoaderClass="org.eclipse.jet.compiled._jet_transformation" >
         <description>dptk.pattern.test.suite.001a</description>

         <tagLibraries>
             <importLibrary id="com.ibm.xtools.jet.dptk.dptk" usePrefix="" autoImport="true"/>
         </tagLibraries>

      </transform>
   </extension>
</plugin>
```

- importLibrary
  - `id` refers to the taglibrary extension defining the tag library
    - Value is defining plugin id + "." + tag library name
  - `usePrefix` defines the namespace for all tags in the library
  - `autoImport` indicates whether templates need to include a tag library directive

5

There are two ways to declare that you want to use the tags in a tag library in your templates.

The first way is to add a bit of XML configuration to the transform's plugin.xml. This makes tags in that tag library available to all templates in the transform, but only if the `autoImport` attribute is set to `true`.

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Using a Tag Library

- Use a taglib directive in each template

```
<%@taglib prefix="ws" id="org.eclipse.jet.workspaceTags" %>
```

- taglib
  - `prefix` is the namespace for all tags in the library
    - For this template only
  - `id` refers to the taglibrary extension defining the tag library
    - Value is defining plugin id + "." + tag library name

- Either use the directive or the plugin reference

6

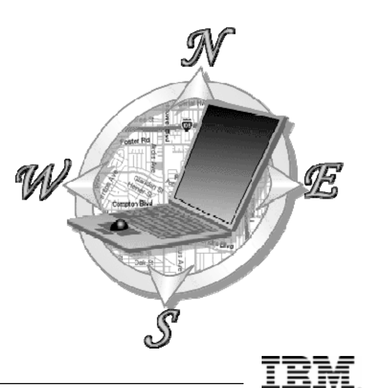
The other way to declare that you want to use the tags in a tag library is to use a taglib directive in each template that needs to use the tags.

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Where Are We?

- Tag Libraries
- **JET Tags**
- Simple Tag Combinations



7

IBM

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Standard JET2 Control Tags Library

```
<%@taglib prefix="c" id="org.eclipse.jet.controlTags" %>
```

- iterate (select, var, delimiter)
- if (test, var)
- get (select, default)
- choose (select)
- when (test)
- otherwise ()
- visitor (select, var)
- visit (test)
- userRegion
- initialCode
- Include (template, passVariables)
- log (severity)
- setVariable (select, var)

8

IBM

These next few charts list the tags in each of the four standard tag libraries, as well as the attributes defined for each tag.

The `<c:userRegion>` and `<c:initialCode>` tags are used to identify areas of generated content that can be modified by the user. If the transform is re-applied, those user changes will be moved to the new versions of the generated content.

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Standard JET2 Control Tags Library (cont.)

<%@taglib prefix="c" id="org.eclipse.jet.controlTags" %>

- addElement (select, name, var)
- addTextElement (select, name, var, cdata)
- removeElement (select)
- copyElement (select, toSelect, name, recursive, var)
- load (url, var, urlContext, loader, type)
- set (select, name)
- marker (description)
- invokeTransform (transformId, passVariables)
- dump (select, format, entities)
- loadContent (var, type, loader)
- nodeAttributes (node, name, delimiter)
- replaceStrings (replace, with)
- stringTokens (string, delimitedBy, name, delimiter, reverse, tokenLength)

9

IBM

The *Element tags let you create, copy, and delete entire model objects within the model.

The `<c:load>` tag is useful for dealing with multiple input models.

The `<c:loadContent>` tag is useful for simple model-to-model transformations from one DOM to another.

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Standard JET2 Workspace Tags Library

<%@taglib prefix="ws" id="org.eclipse.jet.workspaceTags" %>

- project (name)
- folder (path)
- file (template, path, encoding, replace, derived)
- copyFile (binary, replace, srcEncoding, targetEncoding, src, srcContext, target)
- rebuildWorkspace

10

IBM

The tag `<ws:project>` creates a project with the specified name for you, if one does not already exist.

`<ws:folder>` does the same with folders.

`<ws:copyFile>` is useful for copying binary files like JARs, or image files from the transform itself, to generated projects.

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Standard JET2 Java Tags Library

<%@taglib prefix="java" id="org.eclipse.jet.javaTags" %>

- importsLocation (package)
- impliedImport (name)
- import
- package (name, srcFolder, project)
- resource (name, package, srcFolder, template, replace, encoding, derived)
- class (name, package, srcFolder, template, project, replace, encoding, derived)
- merge (rules, rulesContext)

11

IBM

The tags `<java:package>`, `<java:resource>`, and `<java:class>` are like `<ws:folder>`, `<ws:file>` and `<ws:file>`, respectively. They take naming attributes in the Java style (for example, class and package names) instead of the traditional Eclipse URL format.

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Standard JET2 Format Tags Library

<%@taglib prefix="f" id="org.eclipse.jet.formatTags" %>

- replaceAll (value, replacement, regex)
- uc (offset, length)
- lc (offset, length)
- formatNow (pattern)
- milliseconds
- unique
- uuid

12

The tag `<f:formatNow>` writes out the current time, formatted using a pattern just like the SimpleDateFormat class in Java.

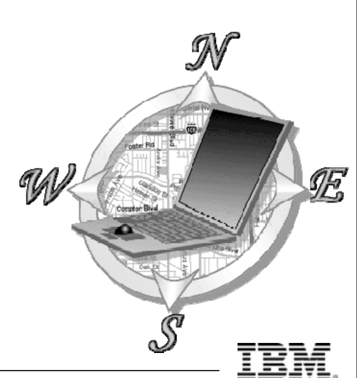
`<f:milliseconds>`, `<f:unique>`, and `<f:uuid>` each write out a unique value every time they're used (even by the same tag in an iterate loop). They are useful in writing many kinds of persisted data.

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Where Are We?

- Tag Libraries
- JET Tags
- **Simple Tag Combinations**



13

IBM

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Simple Tag Combinations

Write the id attribute for all the books in library

```
<root>
  <library>
    <book id="001" pages=230" />
    <book id="002" pages="410" />
  </library>
</root>
```

```
<c:iterate select="/root/library/book" var="b" >
The value of attribute id is <c:get select="$b/@id" />.
</c:iterate>
```

14

IBM

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Simple Tag Combinations

If there's a book element with id = "002", then write that element to the log

```
<root>
  <library>
    <book id="001" pages="230" />
    <book id="002" pages="410" />
  </library>
</root>
```

```
<c:if test="//book [@id = '002'] " var="b" >
<c:log severity="info"><c:dump select="$b" /></c:log>
</c:if>
```

15

IBM

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Simple Tag Combinations

Write the id of the book whose title is “Patterns”

```
<library>
   <book id="001" pages="230" >
      <title>Patterns</title>
   </book>
   <book id="002" pages="410" >
      <title>More Patterns</title>
   </book>
</library>
```

```
<c:get  select="/library/book [title = ‘Patterns’]/@id"  />
```


IBM

The solution will not appear on this slide until the instructor hits RETURN.

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Simple Tag Combinations

Write the title of the book whose id is “002”

```
<library>
  <book id="001" pages="230" >
    <title>Patterns</title>
  </book>
  <book id="002" pages="410" >
    <title>More Patterns</title>
  </book>
</library>
```

```
<c:get  select="/library/book [@id = ‘002’]/title"  />
```

17

IBM

The solution will not appear on this slide until the instructor hits RETURN.

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Lab 2: Using XPath

- Complete the following tasks:
  - Modify 15 templates according to the instructions in the templates

18

Complete Lab 2 in the student workbook.

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Review

- What are the 4 JET tag libraries?
- What must you do to reference a tag from a tag library?
- How is the tag library prefix used?

19

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

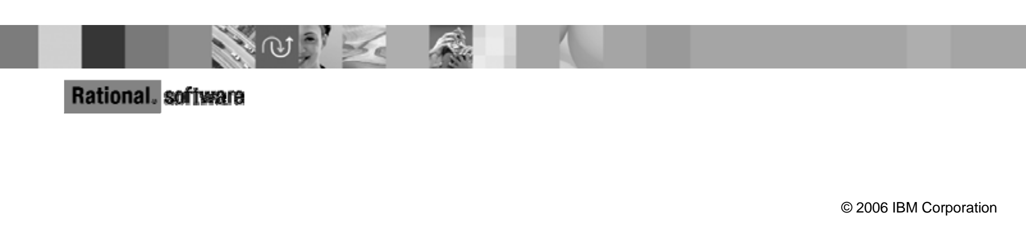
© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

IBM Software Group

DEV498: Pattern Implementation Workshop with IBM Rational Software Architect

Module 7: JET Examples



© 2006 IBM Corporation

Contents

Objectives	7-2
Writing an Arbitrary List	7-4
Generating an Arbitrary Number of Files	7-8
Attributes and Derived Attributes	7-13
Lookups and De-Normalizations	7-22
Getter Names	7-27
Comma-Separated Lists	7-30
Lab 3: Authoring Transforms Manually	7-32
Review	7-33

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

JET Examples

- Objectives:
  - Discuss common JET scenarios
  - Discuss JET best practices

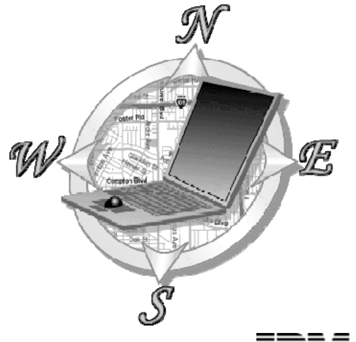
IBM

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Where Are We?

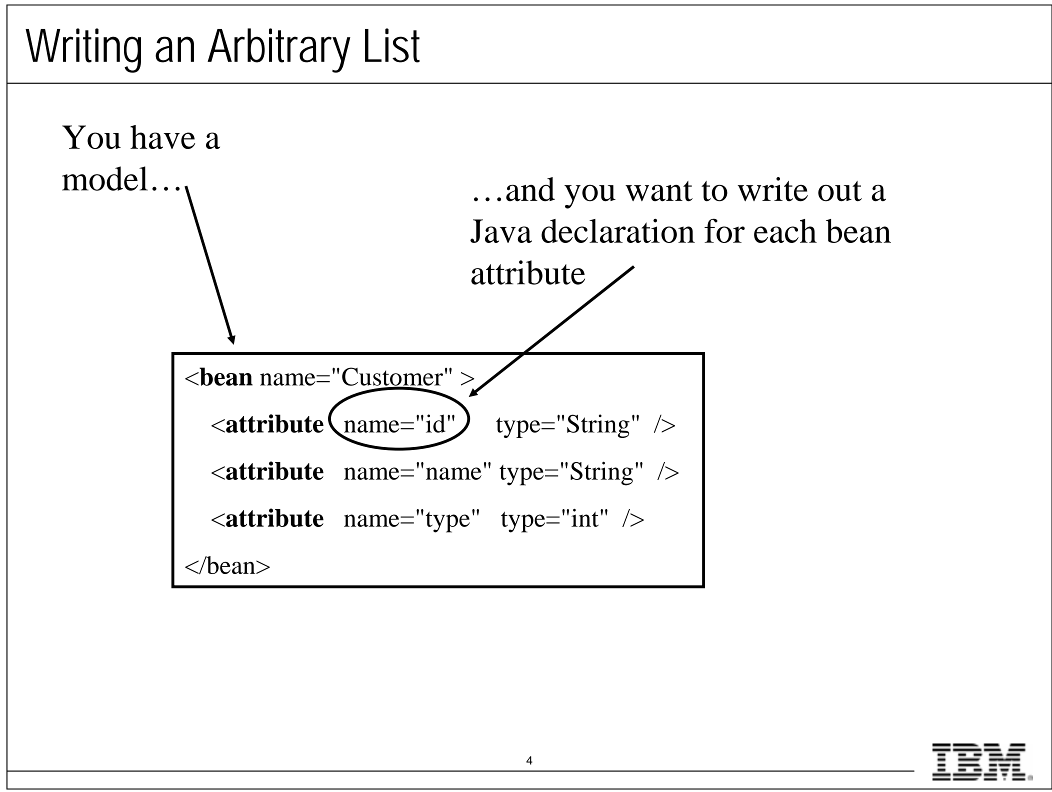
- **Writing an Arbitrary List**
- Generating an Arbitrary Number of Files
- Attributes and Derived Attributes
- Lookups and De-Normalizations
- Getter Names
- Comma-separated Lists



3

IBM

© Copyright IBM Corp. 2007
Course materials may not be reproduced in whole or in part without the prior written permission of IBM.



© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Writing an Arbitrary List (cont.)

So you:

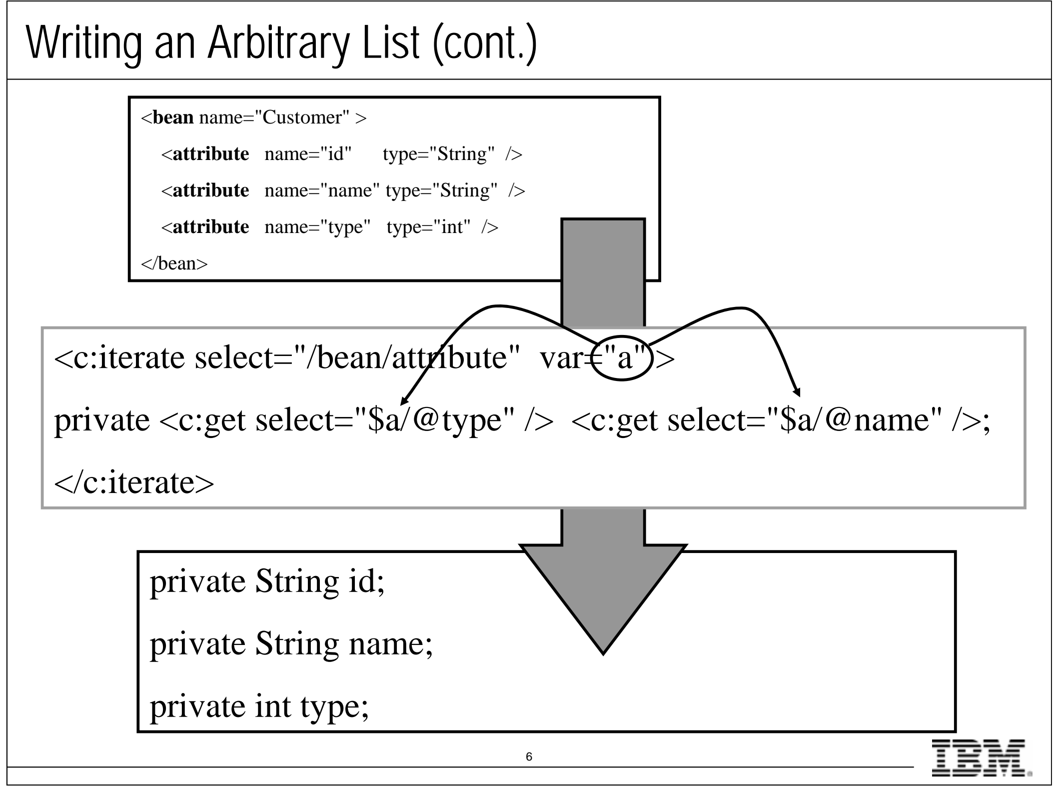
- Collect all of the elements
- Iterate over that collection
- Write the Java source that declares the attribute

```
<bean name="Customer" >
   <attribute  name="id"     type="String"  />
   <attribute  name="name" type="String"  />
   <attribute  name="type"   type="int"  />
</bean>
```

"/bean/attribute" describes the set of elements over which to iterate.

5

© Copyright IBM Corp. 2007
Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

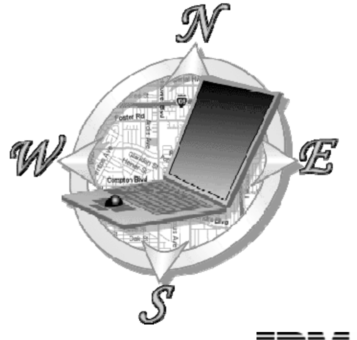


© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Where Are We?

- Writing an Arbitrary List
- **Generating an Arbitrary Number of Files**
- Attributes and Derived Attributes
- Lookups and De-Normalizations
- Getter Names
- Comma-separated Lists



IBM

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Generating an Arbitrary Number of Files

You have a model…

…and you want to write out a Java class and a Java interface for each object element

```
<root  pkg="org.commerce"  dir="org/commerce"  project="Commerce">
  <object  class="CustomerImpl"  interface="ICustomer" >
    <method   name="markSpecial"  />
    <method   name="archiveHistory"  />
  </object>
  <object  class="OrderImpl"  interface="IOrder" >
    <method   name="fulfill"  />
  </object>
</root>
```


IBM

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Generating an Arbitrary Number of Files (cont.)

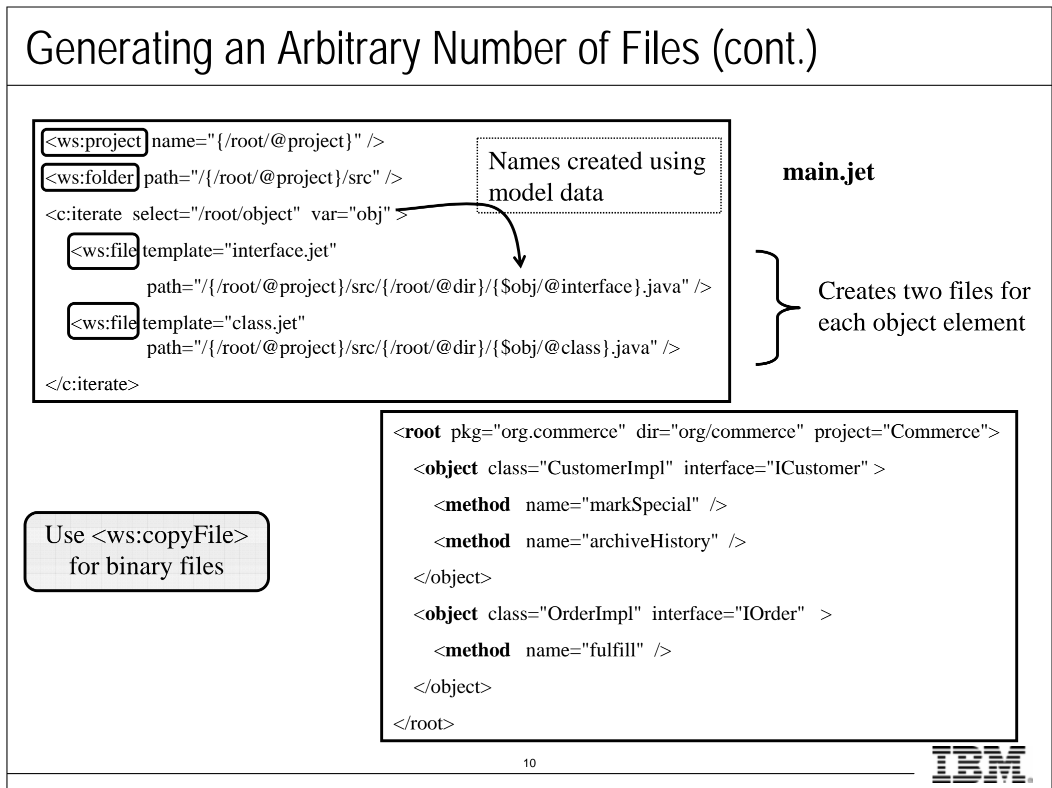
```
<root pkg="org.commerce" dir="org/commerce" project="Commerce">
  <object class="CustomerImpl" interface="ICustomer" >
    <method name="markSpecial" />
    <method name="archiveHistory" />
  </object>
  <object class="OrderImpl" interface="IOrder" >
    <method name="fulfill" />
  </object>
</root>
```

Note how each <object> subtree contains information for one set of files.

9

IBM

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

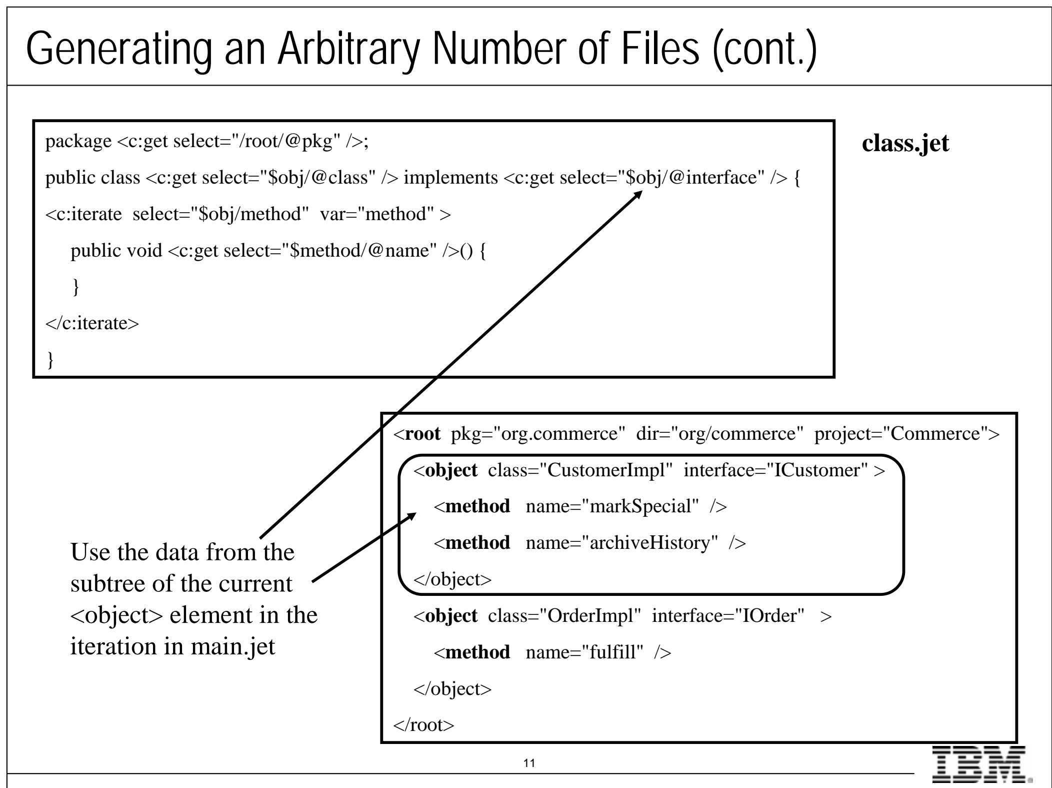


Some things were left out of the above example because of space restrictions. In addition to the <ws:project> and <ws:folder> tags, there should also be two <ws:file> tags – one for the .project file and one for the .classpath file. When you create new projects, you also have to create any metadata files or folders that are needed by any of the project's natures.

Also note that you really did not need the <ws:folder> tag, because the source folder would have been created automatically when the first Java class was created (all folders containing a file are created automatically if they don't already exist). In the case where there are no `<object>` elements in the model, though, no files would have been created, and the source folder wouldn't have been created either. That is why you include a <ws:folder> tag here.

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.



Just to reinforce: templates all share the same data model and variable values. Use the passVariables attribute to restrict variable access.

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Where Are We?

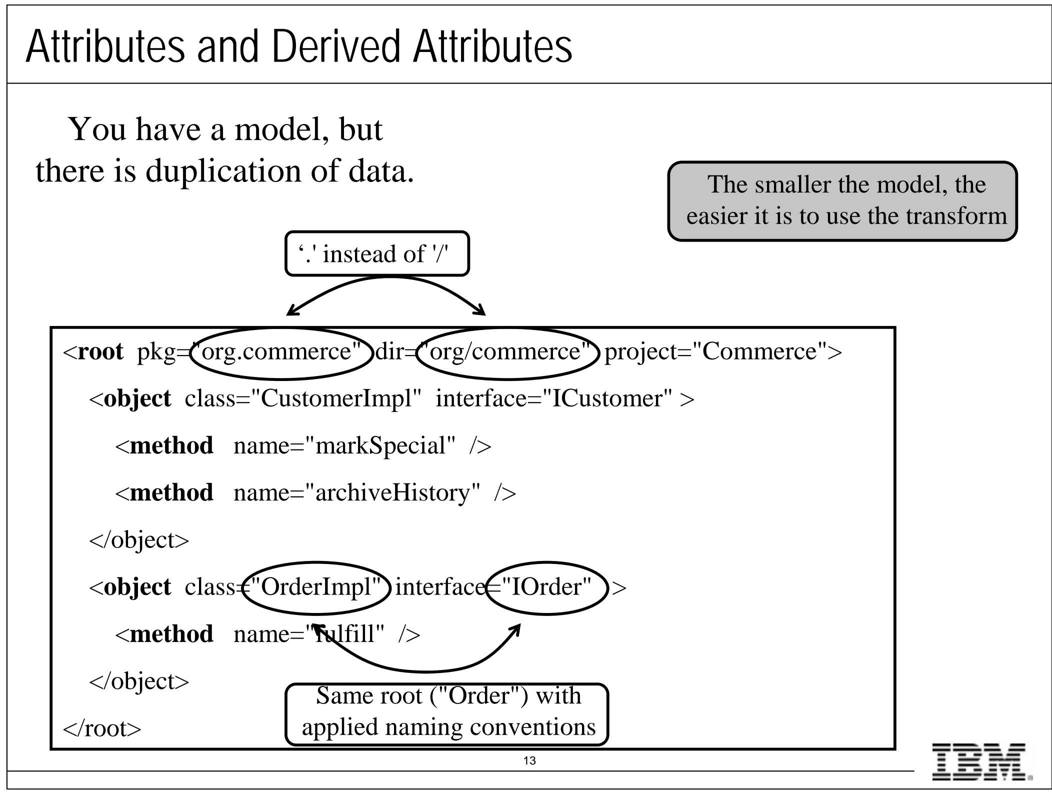
- Writing an Arbitrary List
- Generating an Arbitrary Number of Files
- **Attributes and Derived Attributes**
- Lookups and De-Normalizations
- Getter Names
- Comma-separated Lists

12

IBM

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.



Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Attributes and Derived Attributes (cont.)

- Attributes
  - Present in the model when the transform starts
- Derived Attributes
  - Not present when the transform starts
  - Added to the model by the transform

IBM

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Attributes and Derived Attributes (cont.)

- When attribute A is a function of attribute B
  - Make attribute A a derived attribute
  - Remove it from the input model
  - Add logic in main.jet to calculate its value

Replace all "." with "/"

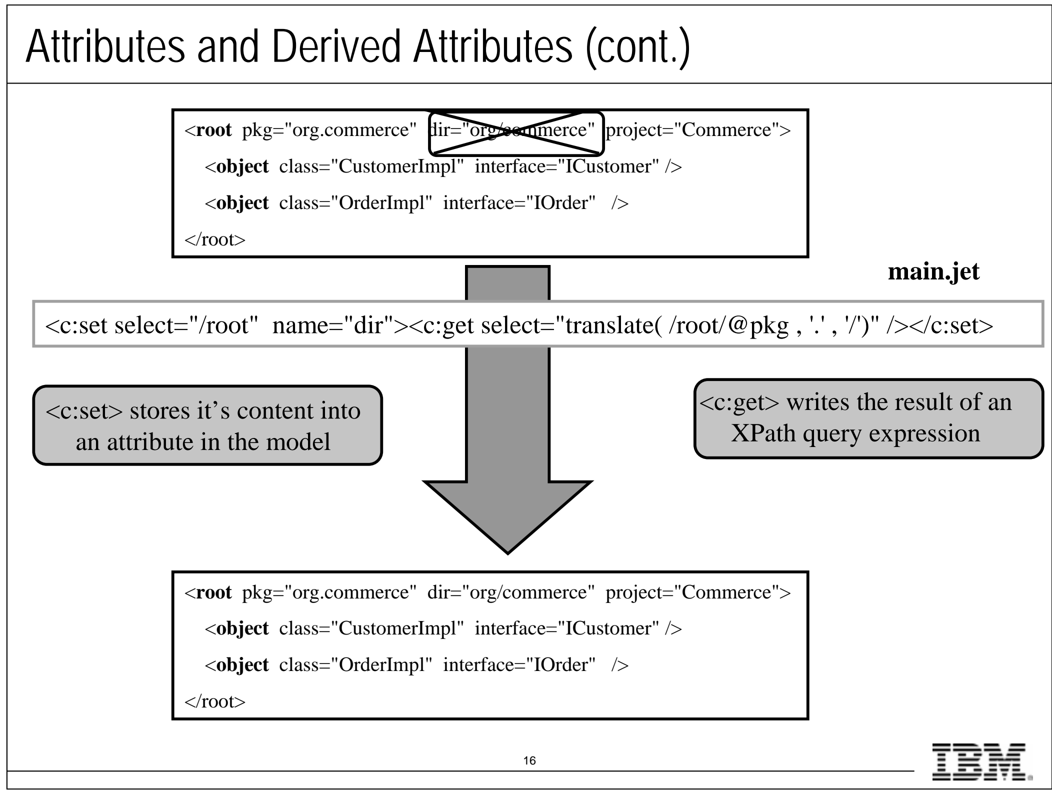
```
<root pkg="org.commerce" dir="org/commerce" project="Commerce">
  <object class="CustomerImpl" interface="ICustomer" >
    <method name="markSpecial" />
    <method name="archiveHistory" />
  </object>
  <object class="OrderImpl" interface="IOrder" >
    <method name="fulfill" />
  </object>
</root>
```

15

IBM

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.



Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Attributes and Derived Attributes (cont.)

- When attributes A and B share a root value
  - Make attributes A and B derived attributes
  - Remove them from the input model
  - Add a new attribute to the model to hold the root
  - Add logic in main.jet to calculate A and B

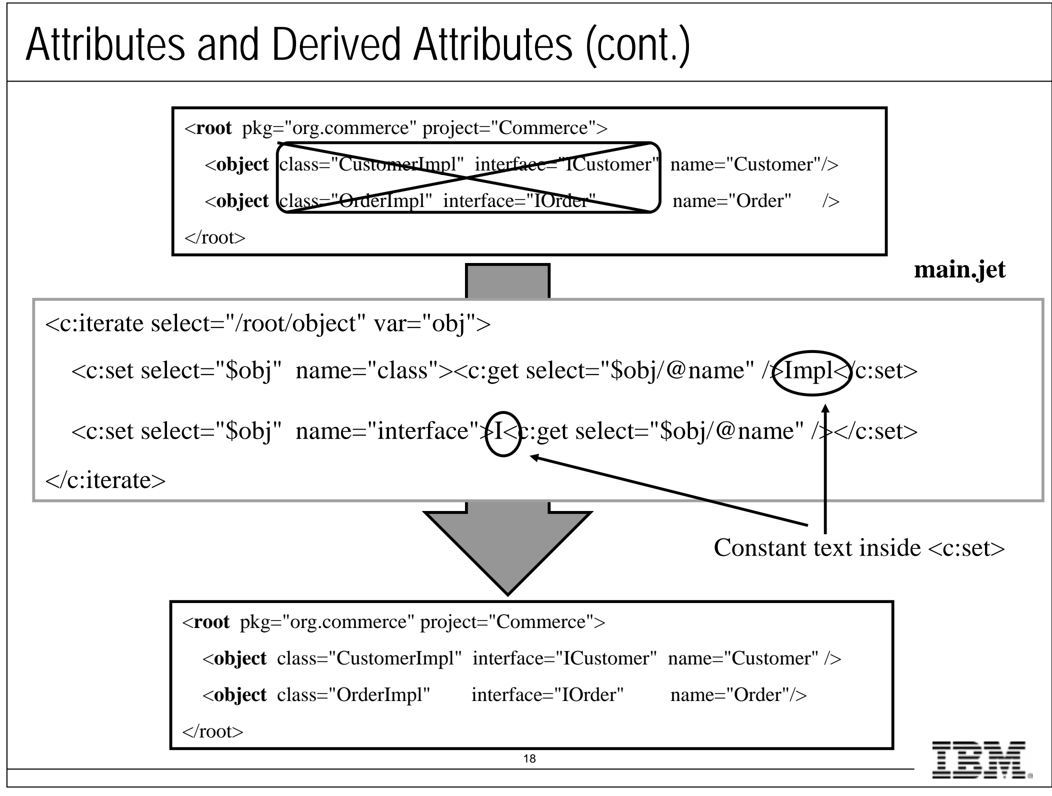
```
<root pkg="org.commerce" dir="org/commerce" project="Commerce">
  <object class="CustomerImpl" interface="ICustomer" >
    <method name="markSpecial" />
    <method name="archiveHistory" />
  </object>
  <object class="OrderImpl" interface="IOrder" >
    <method name="fulfill" />
  </object>
</root>
```

17

IBM

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.



Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Attributes and Derived Attributes Best Practices

- Names of things should be in the model
  - Stored as derived attributes
  - Calculated once, used many times
    - Project names
    - Repeated file names
    - Java class and interface names
    - Method, variable, and property names
- main.jet has three parts
  - A model traversal that calculates derived attributes
    - The only time you write to the model
  - A model traversal that generates artifacts
    - Only reads from the model
  - Optional dump of the model (using <c:dump/>)
    - Make a habit of looking at this for debugging

19

IBM

These are some of the most important best practices.

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Attributes and Derived Attributes

main.jet

1
```
<c:set select="/root"  name="dir"><c:get select="translate( /root/@pkg , '.' , '/')" /></c:set>
<c:iterate select="/root/object" var="obj">
   <c:set select="$obj"  name="class"><c:get select="$obj/@name" />Impl</c:set>
   <c:set select="$obj"  name="interface">I<c:get select="$obj/@name" /></c:set>
</c:iterate>
```

2
```
<ws:project  name="{/root/@project}" />
<ws:folder  path="/{/root/@project}/src" />
<c:iterate  select="/root/object"  var="obj" >
    <ws:file template="interface.jet"  path="/{/root/@project}/src/{/root/@dir}/{$obj/@interface}.java" />
    <ws:file template="class.jet"       path="/{/root/@project}/src/{/root/@dir}/{$obj/@class}.java" />
</c:iterate>
```

3
```
<ws:file template="dump.jet"  path="/{/root/@project}/dump.xml" />
```

```
<root  pkg="org.commerce" dir="org/commerce" project="Commerce">
   <object  class="CustomerImpl" interface="ICustomer"  name="Customer" >
      <method   name="markSpecial"  />
      <method   name="archiveHistory"  />
   </object>
   <object  class="OrderImpl" interface="IOrder"   name="Order"   >
      <method   name="fulfill"  />
   </object>
</root>
```

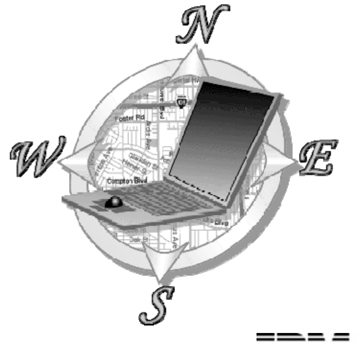
20

IBM

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Where Are We?

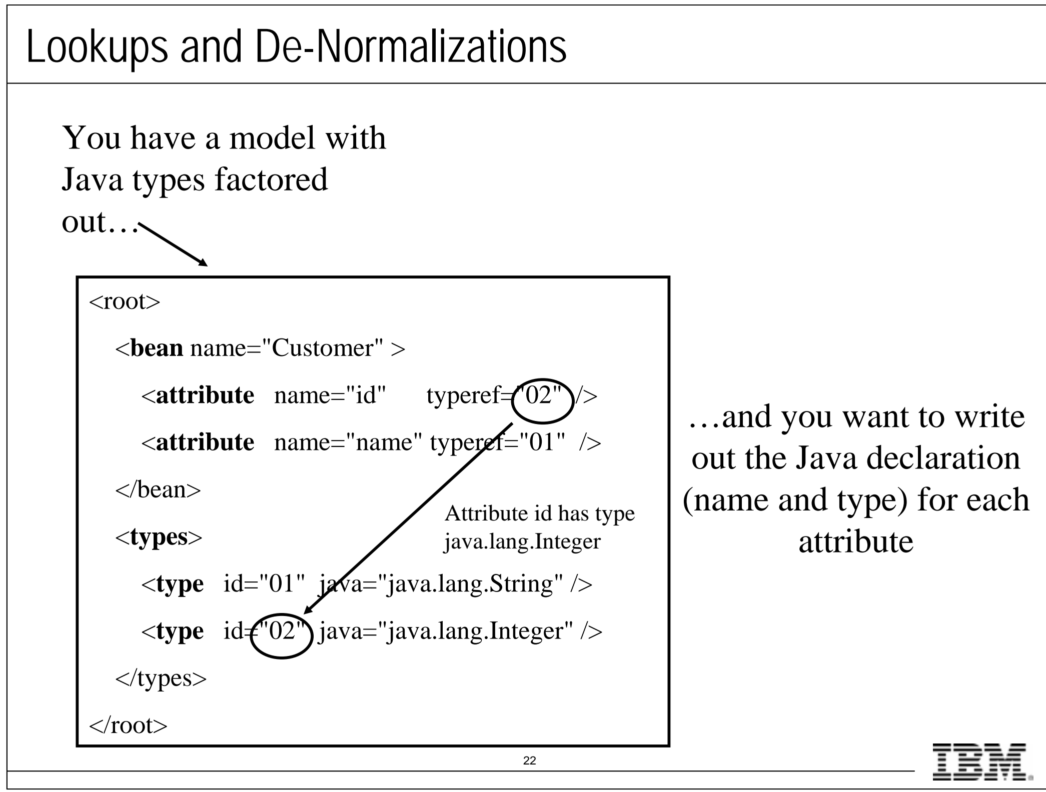
- Writing an Arbitrary List
- Generating an Arbitrary Number of Files
- Attributes and Derived Attributes
- **Lookups and De-Normalizations**
- Getter Names
- Comma-separated Lists



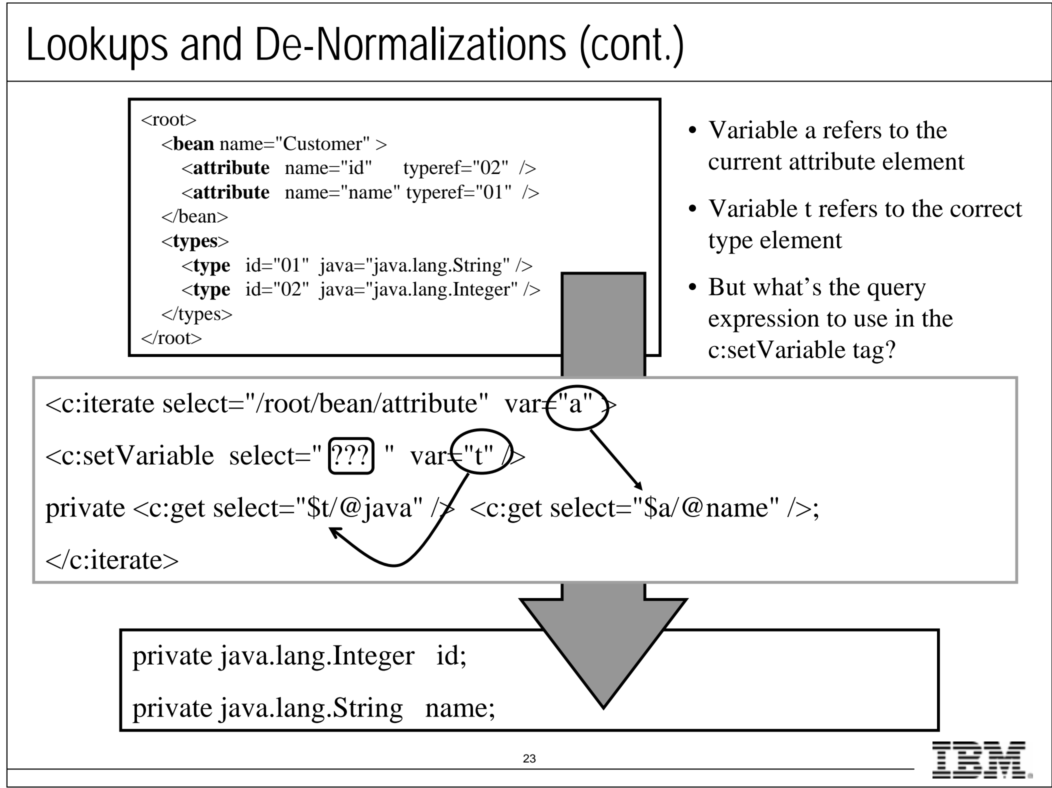
21

IBM

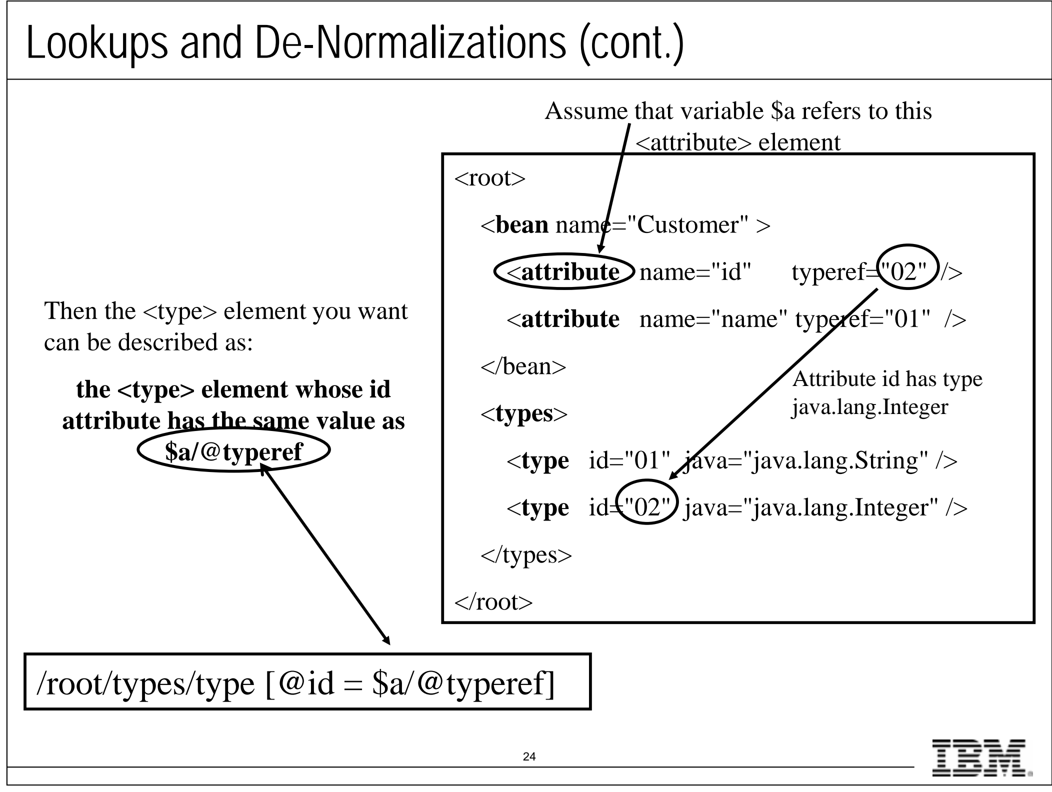
Course materials may not be reproduced in whole or in part without the prior written permission of IBM.



Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

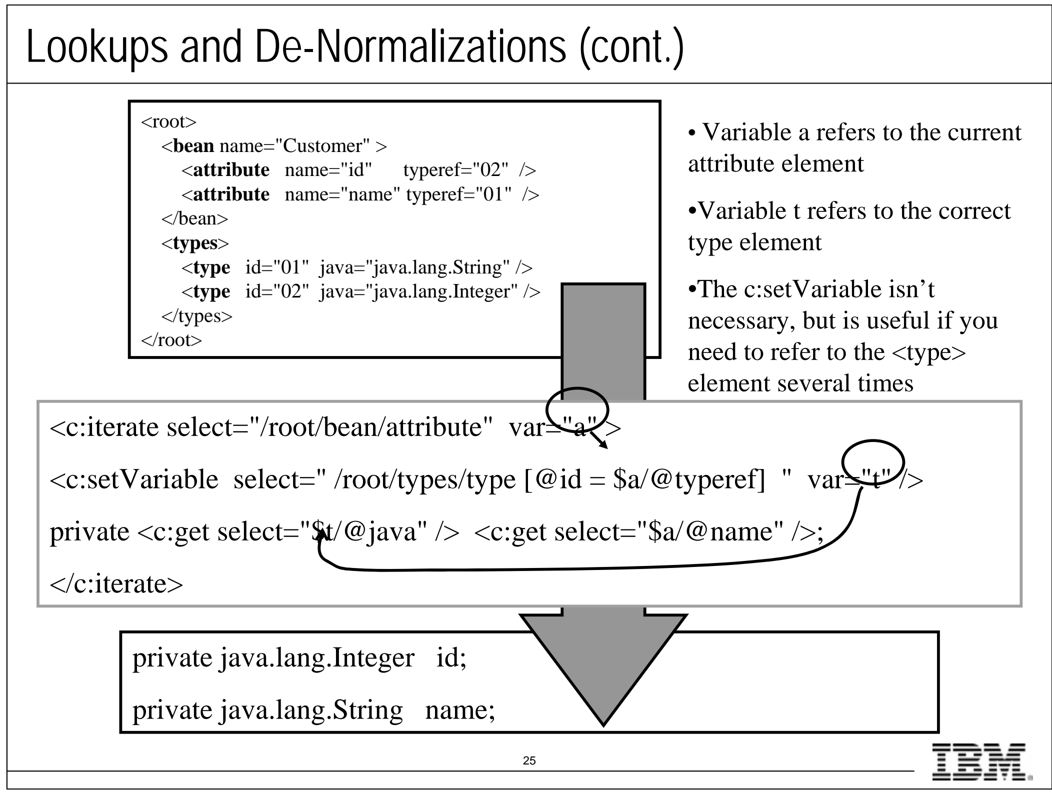


Course materials may not be reproduced in whole or in part without the prior written permission of IBM.



© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.



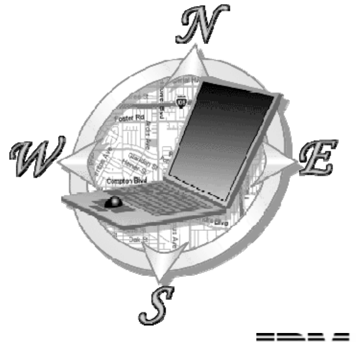
The <c:setVariable> tag is not really needed here. If you take it out, you will need to combine the select expressions in the <c:setVariable> and the <c:get> tags. The <c:setVariable> is often used to perform a common model traversal, and cache the result.

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Where Are We?

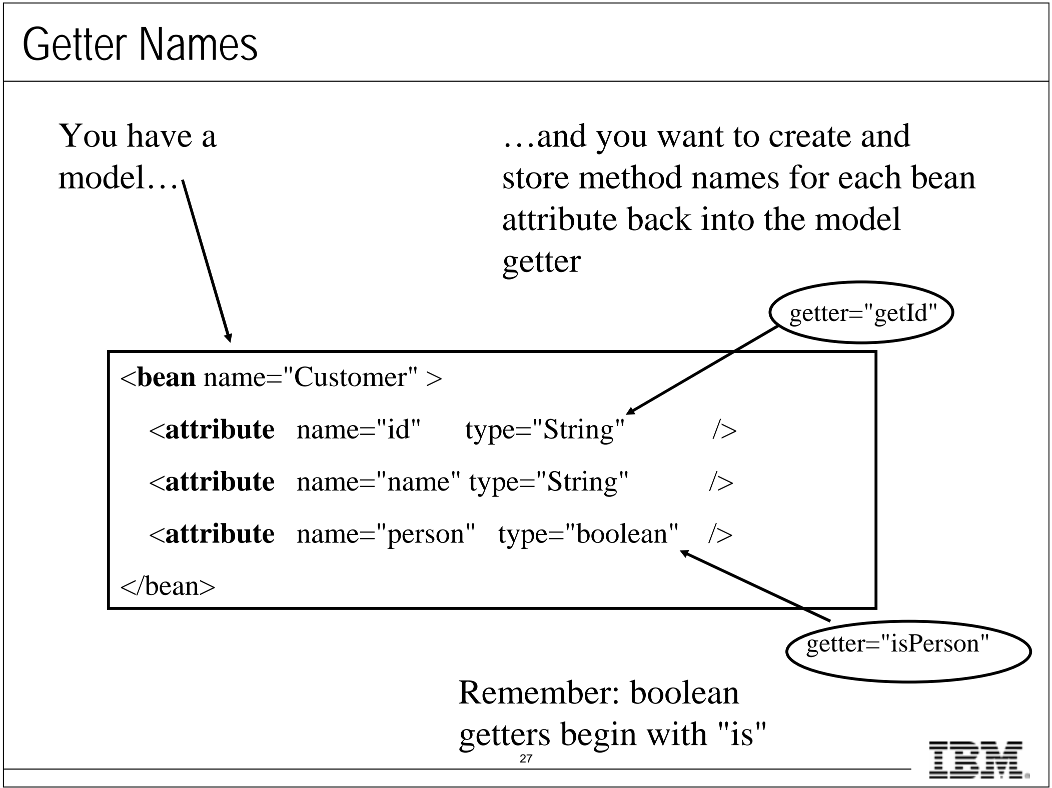
- Writing an Arbitrary List
- Generating an Arbitrary Number of Files
- Attributes and Derived Attributes
- Lookups and De-Normalizations
- **Getter Names**
- Comma-separated Lists



IBM

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.



Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Getter Names (cont.)

```
<bean name="Customer" >
   <attribute  name="id"     type="String"  />
   <attribute  name="name" type="String"  />
   <attribute  name="person"   type="boolean"  />
</bean>
```

```
<c:iterate select="/bean/attribute"  var="a" >
   <c:choose select=" $a/@type" >
      <c:when  test="  'boolean' " >
         <c:set  select="$a"  name="getter">is<c:get select=" uppercaseFirst( $a/@name)" /></c:set>
      </c:when>
      <c:otherwise>
         <c:set select="$a" name="getter">get<c:get select=" uppercaseFirst( $a/@name)" /></c:set>
      </c:otherwise>
   </c:choose>
</c:iterate>
```


© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Where Are We?

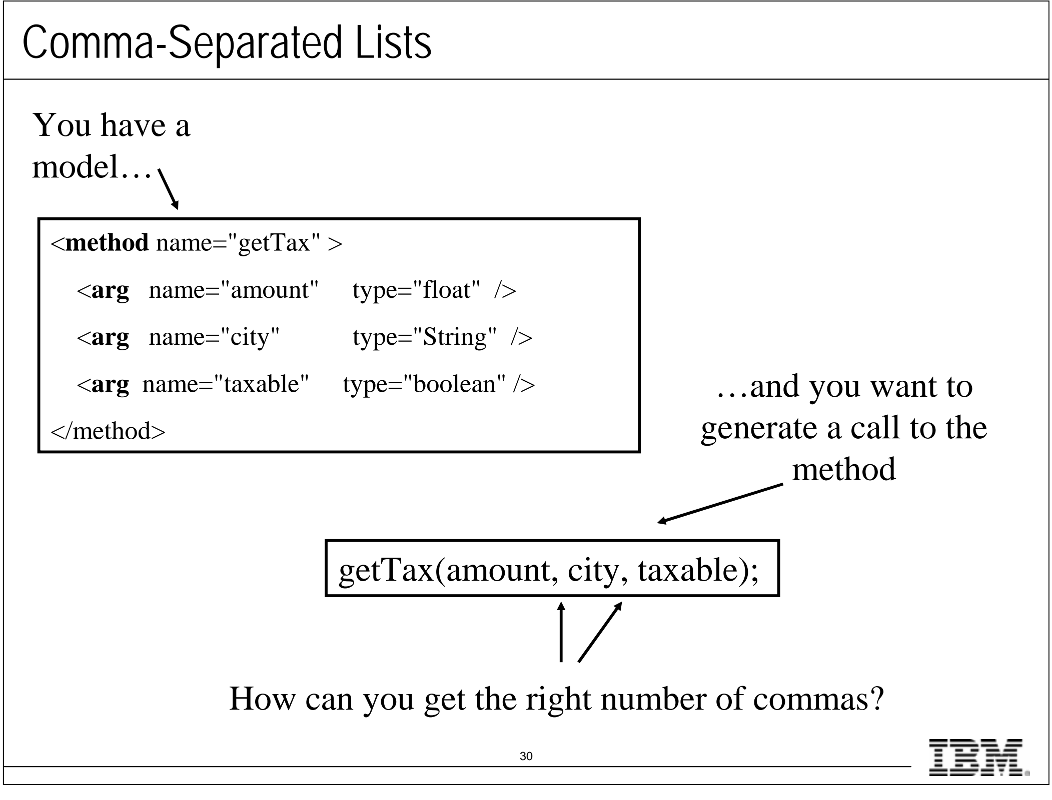
- Writing an Arbitrary List
- Generating an Arbitrary Number of Files
- Attributes and Derived Attributes
- Lookups and De-Normalizations
- Getter Names
- **Comma-separated Lists**

29

IBM

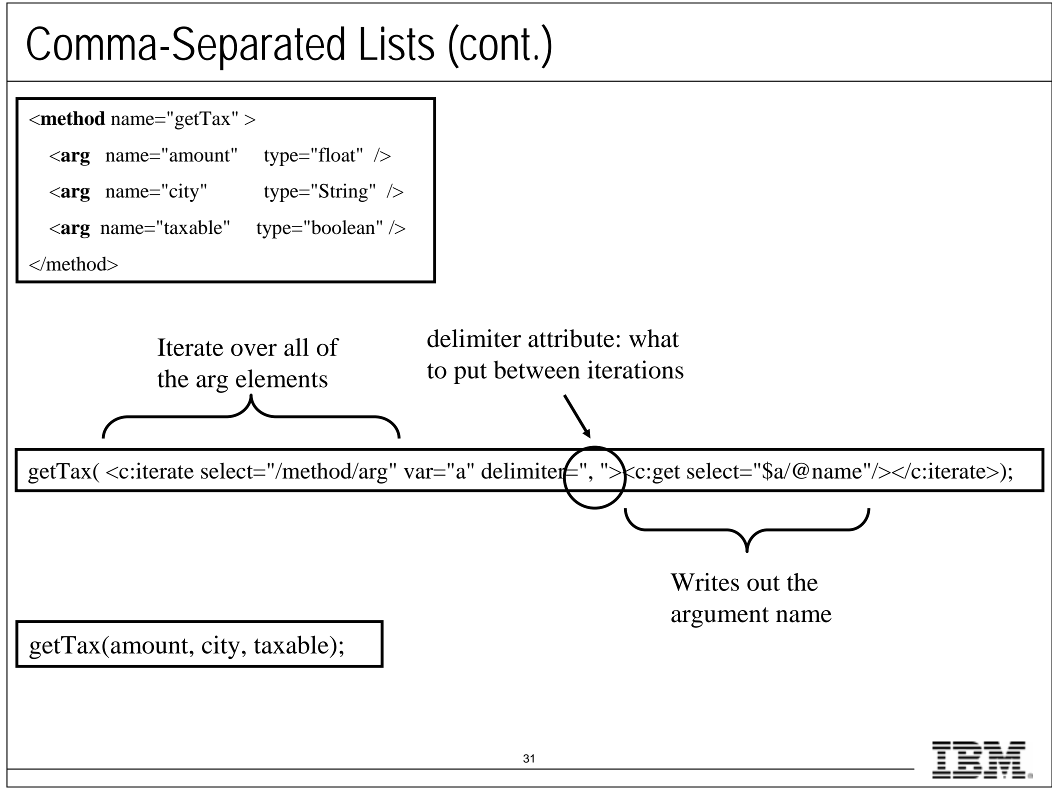
© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.



© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.



The "getTax" in the template snippet above should really be a `<c:get>` tag pulling the value of the name attribute of the method object. Unfortunately there wasn't enough room on the chart without using a font that was too small.

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Lab 3: Authoring Transforms Manually

- Given:
  - A partial transformation
  - An example of what the transformation is to generate
- Complete the following tasks:
  - Add tags and templates to the transformation to generate required output

32

IBM

Complete Lab 3 in the student workbook.

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

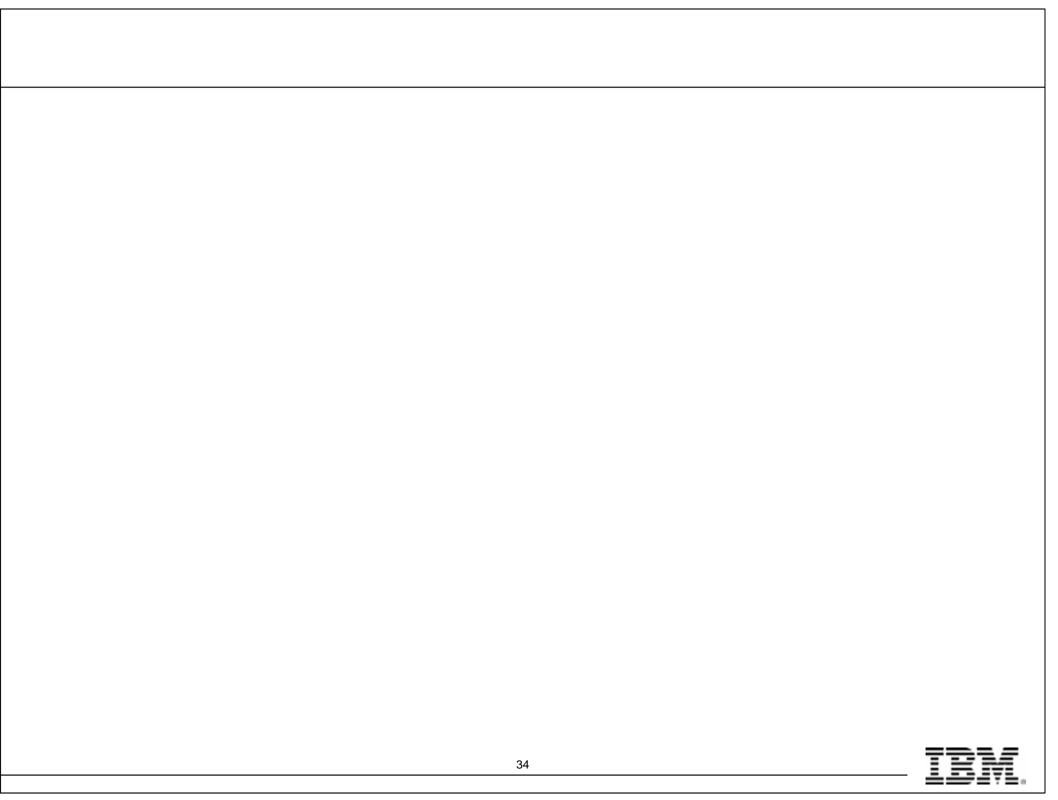
Review

- What is main.jet and what does it do?
- What are the three sections in main.jet?
- What is the difference between an attribute and a derived attribute?

33

IBM

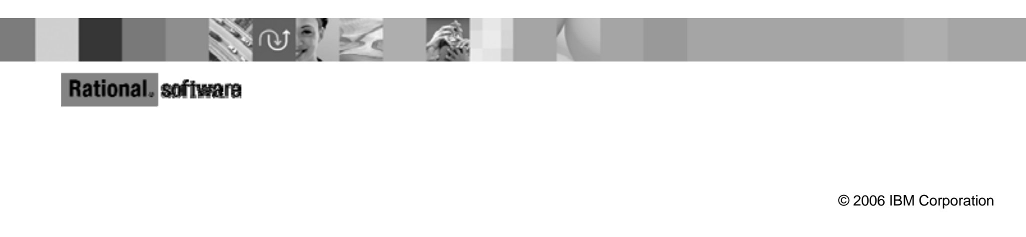
Course materials may not be reproduced in whole or in part without the prior written permission of IBM.



© Copyright IBM Corp. 2007
Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

IBM Software Group

DEV498: Pattern Implementation Workshop with IBM Rational Software Architect
Module 8: Exemplar Analysis



© 2006 IBM Corporation

Contents

Objectives 8-2
Finding a Pattern to Implement 8-4
Preparing to Author a Model-to-Text Transform 8-14
Authoring the Model and Templates 8-18
Lab 4.1: Exemplar Authoring 8-32
Lab 4.2: Exemplar Authoring 8-33
Lab 5: Console Transform 8-34
Review 8-35
Further Information 8-36

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Exemplar Analysis

- Objectives:
  - Describe:
    - Where to look for patterns
    - How to find patterns
    - The model-to-text transform authoring roadmap
  - Perform exemplar analysis
  - Author a model-to-text transform

2

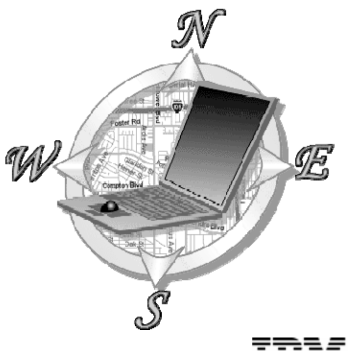
IBM

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Where Are We?

- **Finding a Pattern to Implement**
- Preparing to Author a Model-to-Text Transform
- Authoring the Input Model and Templates



3

IBM

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Finding a Pattern to Implement

- Look at existing reusable assets
  - Each asset class has unique reuse attributes
  - Is there a better implementation?
- Think twice before creating a reusable asset
  - Consider authoring a model-to-text transform
- Déjà vu
  - If you think you've solved the same problem before

4

IBM

If you want to author a model-to-text transformation, but are not quite sure what the pattern should be, there are several techniques you can use to identify a potential pattern.

A great place to start is with existing reusable assets. Many development organizations use reusable assets as a way to communicate information that is used over and over while building applications. These reusable assets point to pattern authoring opportunities. It turns out that often model-to-text transformations are actually more consumable that other reusable asset classes. More on this later.

Another way to identify a potential pattern is to look at when you might create your own reusable assets. For the reasons listed above, you might want to try authoring a pattern instead of creating another reusable asset.

Finally, most experienced transformation authors know to look for situations when they solve the same problem several times (and are likely to solve that same kind of problem in the future).

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Re-Assessing Reusable Assets

- Reusable Asset
  - Valued for time saved per reuse
  - But there is a cost associated with each reuse
    - Learning curve
    - Customization required
- Typical classes of reusable assets
  - Libraries
  - Best practices papers
  - Frameworks
  - Pattern descriptions (GoF, Enterprise)
  - Code samples
  - Wizards

5

IBM

Usually, attention is paid to the time, cost, and resource savings from using reusable assets. As authors of reusable assets, though, we need to understand that there are costs associated with reusable assets, too. Not only is there a learning curve, but often the solution provided by the reusable asset requires customization before it solves the problem that you are trying to solve.

For example, a best practices document requires the user to read the document and understand it well enough to mentally apply the best practices to the problem at hand. The user then has to manually create the solution using the IC in the best practices document. Note that there is a good deal of learning curve required as users read and self-educate themselves on the IC.

That's not to say that the different classes of reusable assets should be avoided. You just need to understand the total cost and benefit of reuse for your specific IC when implemented as a particular form of reusable asset.

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Patterns and Reusable Assets

- Key reusable asset question:
  - The reusable asset is a solution to a recurring problem.
  - How much of that solution has to be customized?
- Different customization requirements
  - Libraries
    - No customization at all
  - Frameworks
    - Heavy customization of relatively small number of files
      - But those files have exactly the same shape across applications
  - Best Practices, Design Docs, and so on
    - All files manually created

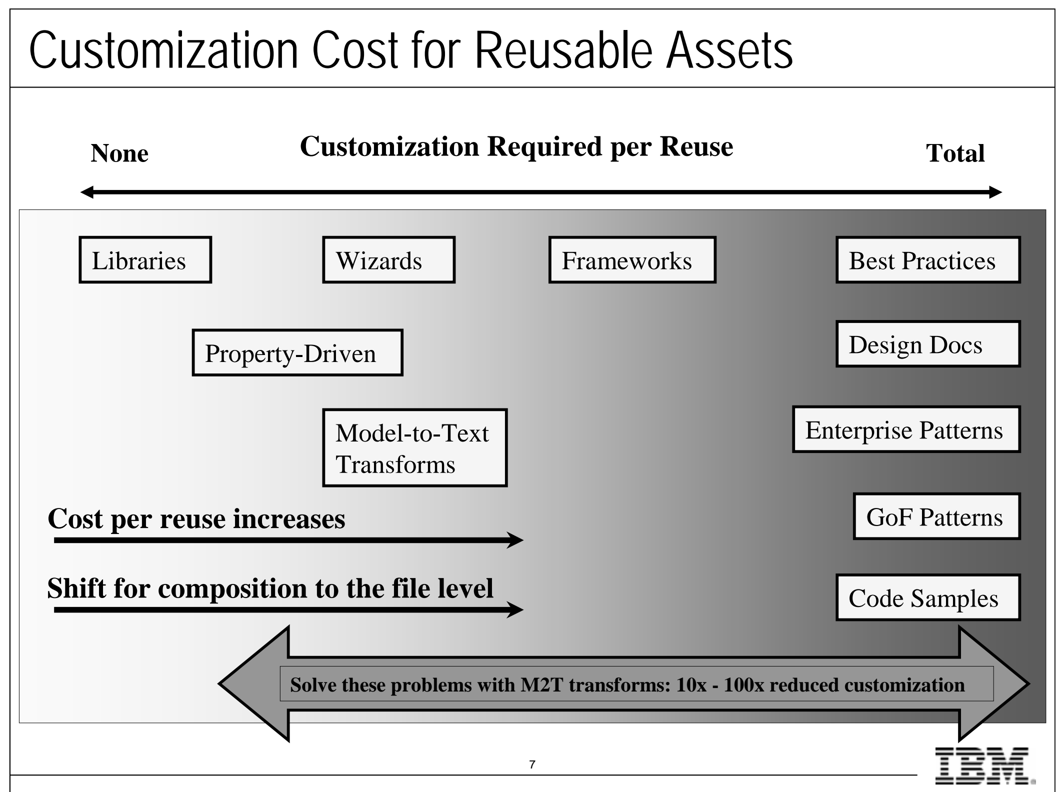
6

IBM

Each reusable asset class requires some form of learning curve, but that's a fixed cost. No matter how many times you reuse an asset, you still only have to learn how to apply that asset only once.
The more important cost for reusable asset use is the time it takes to customize the solution provided by that reusable asset.
So what is that cost?
Libraries generally require no customization. Meanwhile, frameworks provide most of a solution and only require the user to create only relatively few files that sit on top of that framework. As mentioned above, documents tend to require the user to create the complete solution manually.

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.



This chart shows where on the “customization-required” spectrum each reusable asset class falls. The blue arrow shows that part of the spectrum for which model-to-text transforms might be the best option in terms of customization required after each application of the intellectual capital.

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Deriving Transforms from Reusable Assets

- Start with some reusable asset or intellectual capital
  - Best Practices or Design Document
- Reuse that intellectual capital to create a number of files
  - This may take some time
    - Every reuse of this asset requires this much work
    - One of several hidden costs
  - Reusable Asset solution probably needs some customization

8

IBM

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Deriving Transforms from Reusable Assets

- Model-to-text (M2T) transforms generate to one of several scopes
  - Single file
    - Eclipse dialog box
  - Small set of collaborating files
    - ISSW Exception Framework
  - Component
    - WAS-optimized Java™ DataBase Connectivity (“ JDBC”) beans
  - Deployable application
    - In one or more Eclipse projects

- Assess the artifacts created with the Reusable Asset
  - Choose the smallest of the above scopes containing those artifacts

9

IBM

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Deriving Transforms from Reusable Assets

- Determine what additional assets are needed:
  - To fill out the scope
  - By the assets you've created
  - By other applicable reusable assets
    - Internationalization, packaging guidelines, naming conventions
- Make sure that:
  - All necessary artifacts are identified
  - All applicable reusable assets have been applied
- This gives you a closed set of artifacts
  - A M2T transform should be used to generate these

10

IBM

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Example: Portlet Best Practices

- A whitepaper described core collaborations
  - Between portlet and action classes
  - Between action and state classes
  - Between state, cargo beans, and JavaServer Pages™ (JSPs)
- Built a portlet as described by the whitepaper
  - Added Eclipse project and meta-data
  - Added portal deployment descriptor
- Scope was an Eclipse project
- Authored M2T transform to generate portlet projects

11

IBM

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

A Potential Reusable Solution

- Reusable assets as a guide
  - When applied, a rough pattern emerges in the code resulting from the application
  - You have a solution to a specific problem
  - Customization can solve similar problems

- Déjà vu can result in a rough pattern, too
  - You solve the same kind of problem over and over
  - Pick any of those implementations
    - Later is probably better that earlier

12

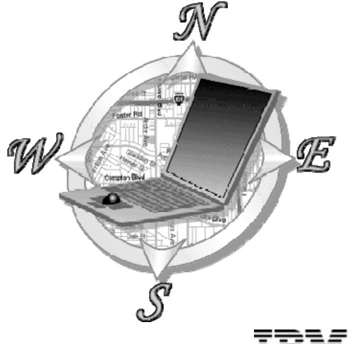
IBM

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Where Are We?

- Finding a Pattern to Implement
- **Preparing to Author a Model-to-Text Transform**
- Authoring the Input Model and Templates



13

IBM

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Exemplar Analysis Overview

- Methodology for the authoring of model-to-text transforms
  - Scalable to arbitrarily large and complex transforms
  - Applicable to any model-to-text transform
  - Must be relatively fast
- Interview-style approach between two roles
  - Domain SME understands the pattern to be authored
  - Pattern SME understands the methodology
  - Analogous to how patent attorneys work
- Requires as input an Exemplar
  - Representative example of what the pattern is to produce
  - Well and consistently written
  - Might take 18 months to write
  - Stop and go home if you don't have one

14

IBM

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Preparing to Author a Pattern

- "A pattern is a solution to a recurring problem"
  – Grady Booch

- What's your recurring problem?
  - Don't know? Review the "Where to look" charts.

- Keep solving the problem

- The solution will eventually stabilize
  - You'll eventually stop improving on it

15

IBM

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Find a Pattern to Implement

- Things to do to your solution as it stabilizes
  - Refactor classes
  - Expand the pattern to cover more artifacts
    - Adding j-unit tests or Help pages
  - Refine your naming conventions
  - Debug and optimize your code

16

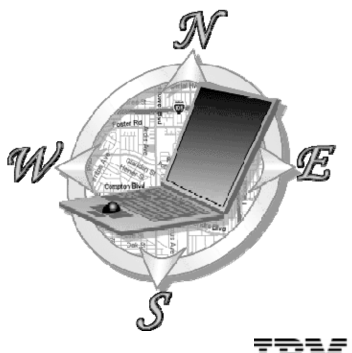
IBM

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Where Are We?

- Finding a Pattern to Implement
- Preparing to Author a Model-to-Text Transform
- **Authoring the Input Model and Templates**

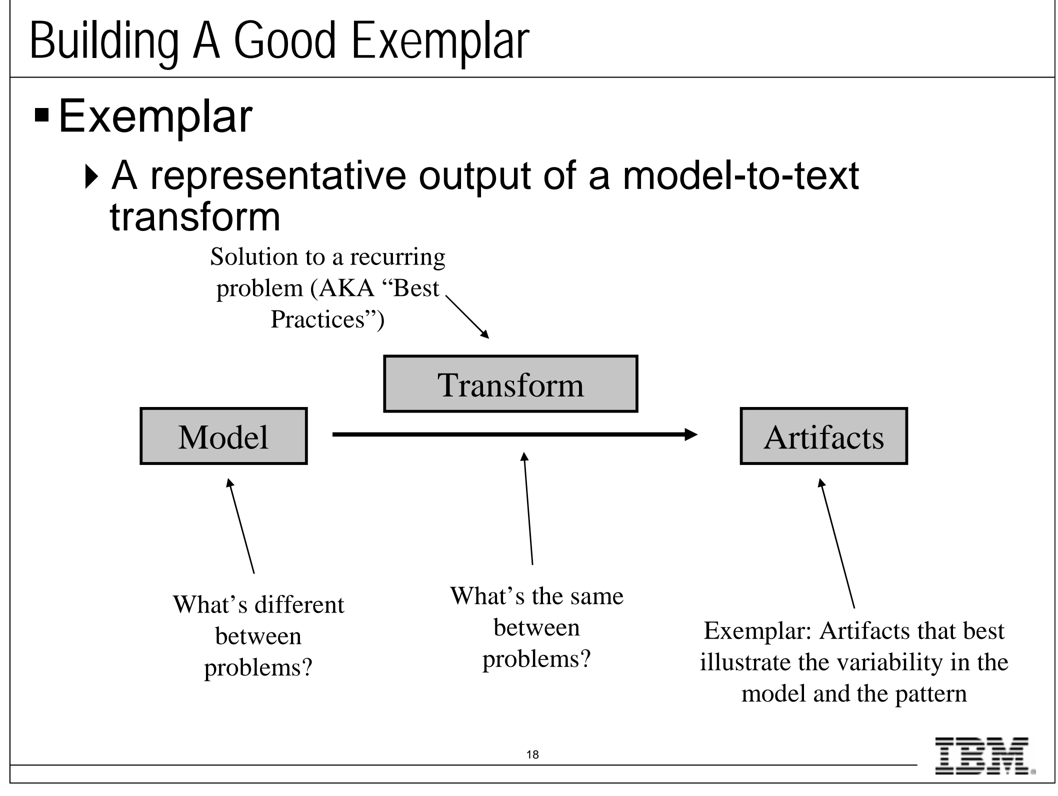


17

IBM

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.



© Copyright IBM Corp. 2007
Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Good Exemplars

- A good exemplar demonstrates variability
  - In the model
  - In the pattern
- Example: The JavaBeans™ Pattern
  - Beans can have any number of properties
    - A good exemplar has more than one property
  - The getter for a boolean property starts with "is"
    - Implement both boolean and non-boolean properties
- Not even a perfect exemplar replaces the SME

19

IBM

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Exemplar Analysis

- Exemplar Analysis is a methodology
  - Supported by the Exemplar Authoring tool
  - Based on a set of Best Practices
    - Transform design
    - Input model
    - JET tag usage
- A working knowledge of Exemplar Analysis requires a working knowledge of these Best Practices

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

A Monologue on the Model

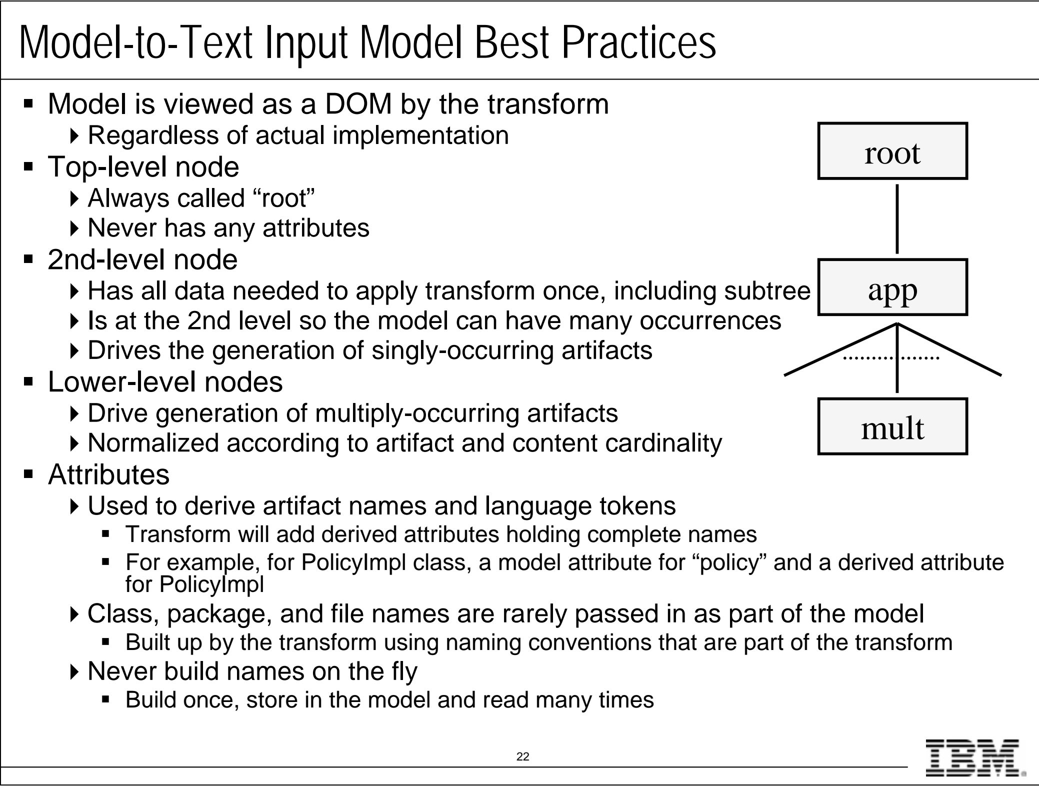
- For model-to-text, there are only two model design goals. The model design:
  - Must contain all required dynamic values
  - Must be optimized to make template access simple
- In practice, there is a pattern to the model

21

IBM.

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.



© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Model-to-Text Input Model Notes

- The model
  - Does **not** reflect the user's view of the problem
  - Does **not** contain terminology familiar to the user
  - Is **not** the model originally populated by the user
  - Is **not** an input to Exemplar Analysis, but an output

23

IBM

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Exemplar Analysis

- Given an Exemplar
  - Extract the input model schema
  - Extract the transformation logic (templates)

- Steps
  - Identify all dynamic content in the exemplar
  - Normalize that content
    - Build a schema to describe that normalization
  - Create templates from artifacts
    - Replacing dynamic content with tags

24

IBM

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

The Simple Approach

- Only consider artifacts that are unique
- Start by identifying artifact roles
  - Artifact Role: Why is that artifact in the exemplar?
  - Same role ➔ generation by the same template
- Model the unique roles and their cardinality
  - Group according to cardinality
  - Name the groups
  - Create a template for each role

25

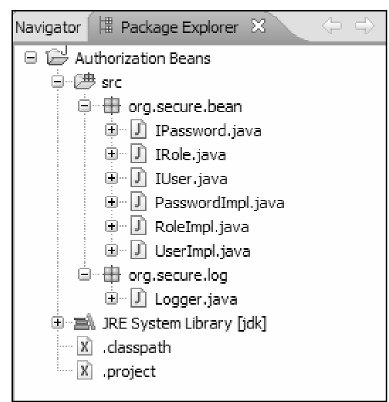
IBM

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Listing the Roles

- Sample exemplar
  - Project (Authorization Beans)
  - Classpath file (.classpath)
  - Project description (.project)
  - Logger (org.secure.log.Logger)
  - Interface (IRole, IUser,..)
  - Bean (RoleImpl, UserImpl,…)



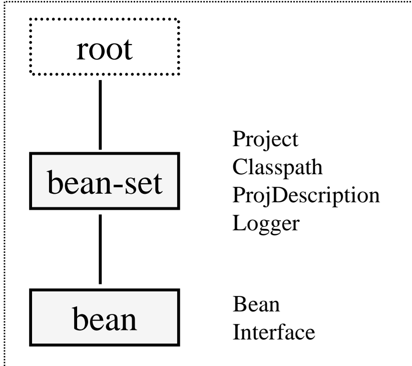
26

IBM

© Copyright IBM Corp. 2007
Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

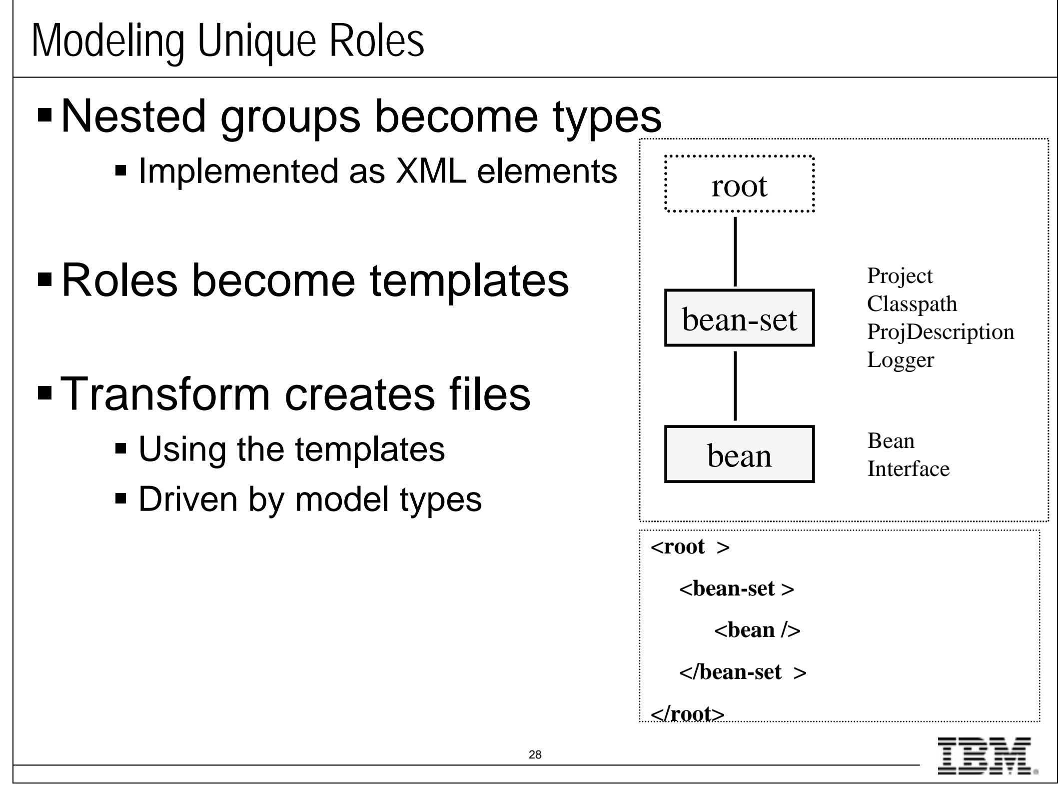
Modeling Unique Roles

- First, name the entire group
  - Bean-set
    - Keep it to one word
- Identify the one-time roles
    - Project (Authorization Beans)
    - Classpath file (.classpath)
    - Project description (.project)
    - Logger (org.secure.log.Logger)
- Create nested sub-groups for repeating roles
  - Bean contains Interface and Bean roles



27

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.



© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Exemplar Analysis

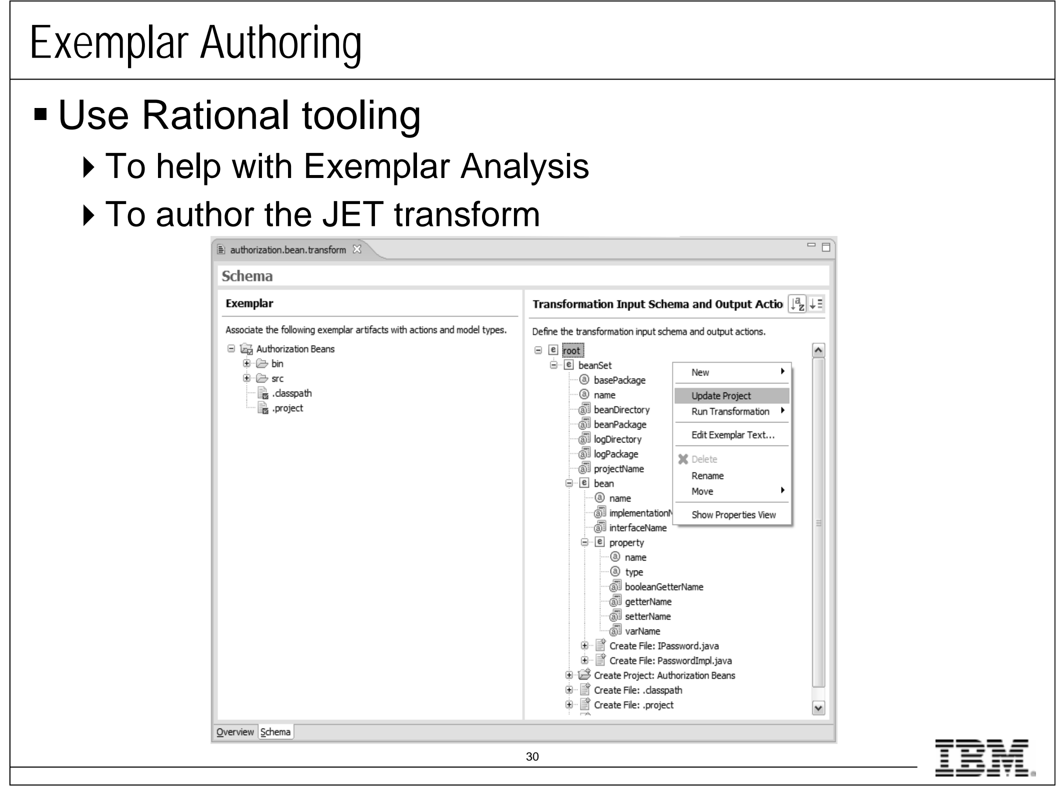
- Model types drive the creation of artifacts
  - But they lack the information required to generate content
- Need to add attributes to model types
  - Naming
  - Language tokens
    - Class, variable, method names
- Distinguish between input and derived
  - Names of Eclipse artifacts are usually derived
    - Naming conventions
  - Input attributes are usually simple and atomic

29

IBM

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.



© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Demonstration: Authoring a Transformation

- The instructor will now show you how to:
  - Author a Transformation
  - Perform Exemplar Analysis
  - Run the Transformation

31

IBM

The instructor will show you how to author a transformation.

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Lab 4.1: Exemplar Authoring

- Given:
  - Authorization Bean exemplar
- Complete the following tasks:
  - Author a transform for the exemplar

32

IBM

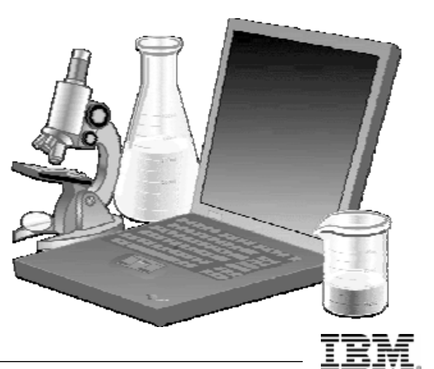
Complete Lab 4.1 in the student workbook.

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Lab 4.2: Exemplar Authoring

- Given:
  - Update Site and Feature Projects Exemplar
- Complete the following tasks:
  - Author a transform for the exemplar



33

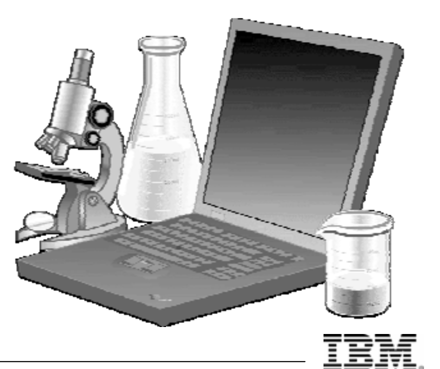
IBM

Complete Lab 4.2 in the student workbook.

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Lab 5: Console Transform

- Given:
  - Exemplar
- Complete the following tasks:
  - Author a transform for the exemplar



34

IBM

Complete Lab 5 in the student workbook.

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

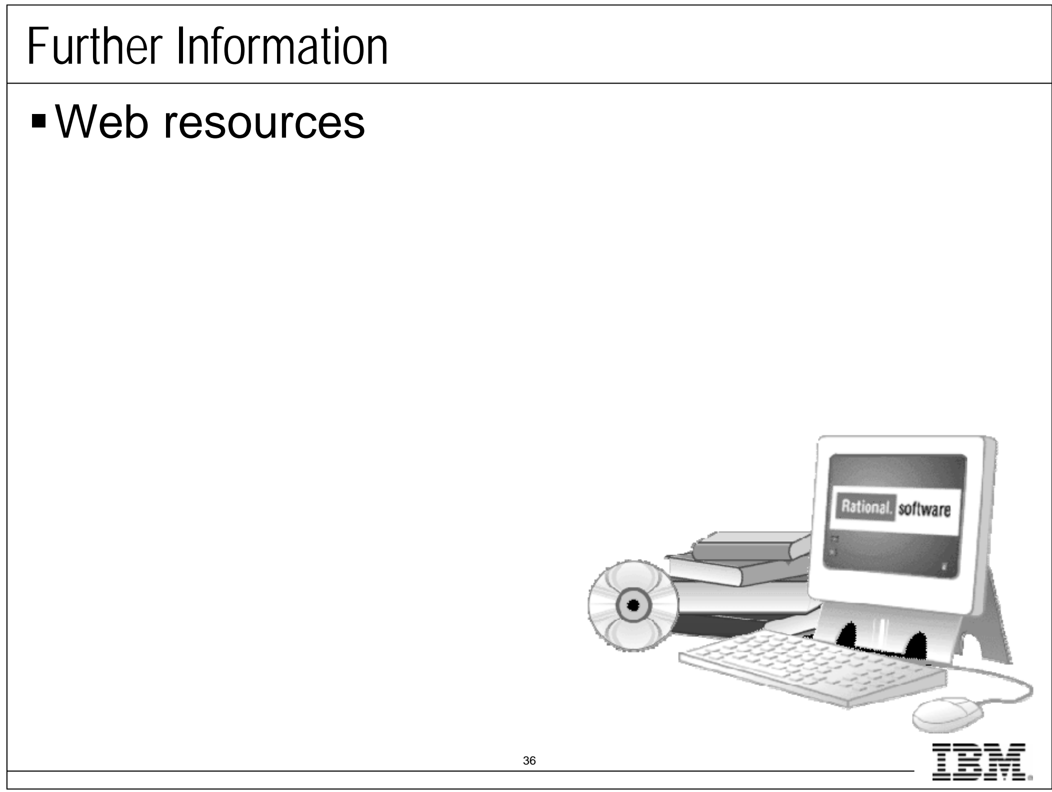
Review

- Why does the root node in the model never have attributes and actions?
- What is the purpose of the second-level node in the model?
- What are the three kinds of Eclipse resources that a transform can generate?

IBM

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

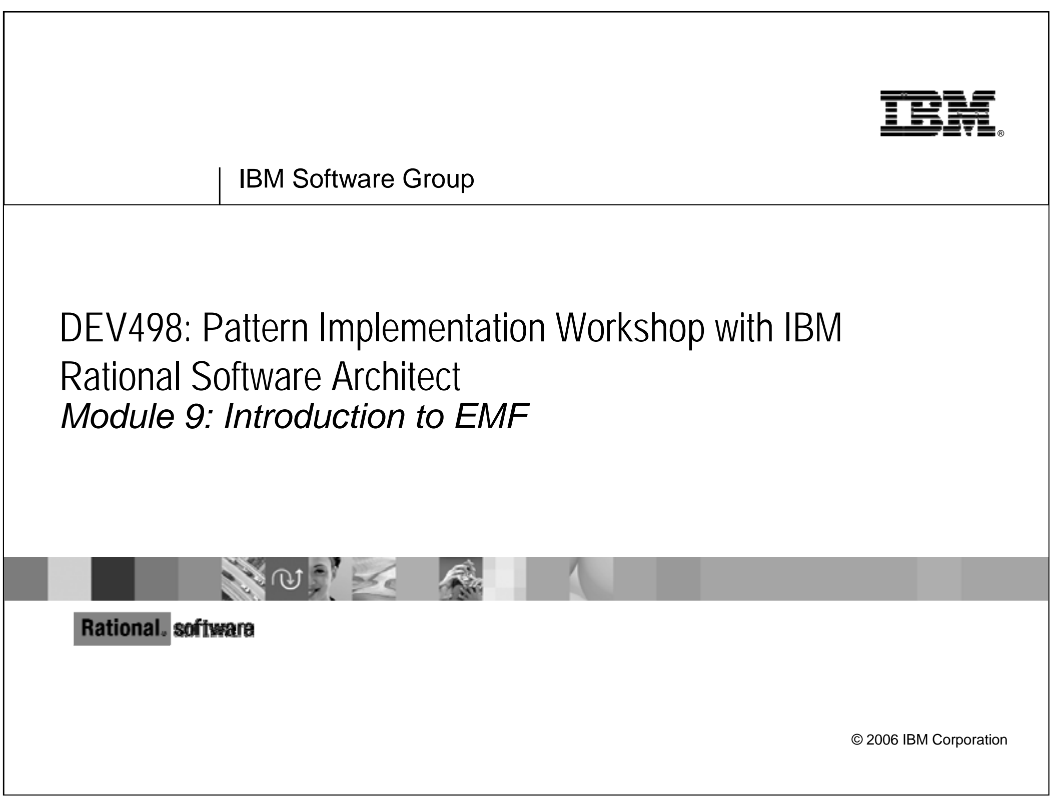


Web Resources

- Pattern Solutions: Use patterns to drive productivity in software design and development: http://www-128.ibm.com/developerworks/rational/products/patternsolutions/

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.



Contents

Objectives 9-2

What is EMF? 9-3

Labs 9-13

Further Information 9-15

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Introduction to EMF

- Objectives:
  - Describe EMF (Eclipse Modeling Framework)
  - Understand how you can use EMF along with JET2
  - Understand how to create a simple data editor using EMF

2

IBM

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

What is EMF

- EMF = Eclipse Modeling Framework
- Eclipse standard mechanism to manage and store structured data
  - Eclipse-based products and plug-ins need to manage structured data
- Lets you define a data structure (model), generate the runtime code, use the runtime code, and map to persist data stores (like XML)
- For example, use EMF to create a simple API for an XML file
- EMF designer also includes the ability to create a simple editor automatically

3

IBM

In other words, EMF let's you define your Eclipse data structures (models), generate the runtime code, use the runtime code, and map to persistence data stores, (like XML).

You can use the EMF capabilities to create custom XML file editors. The EMF designer even includes the ability to create a simple editor for XML (or EMF) files.

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Importance of EMF

- Standard data definition and management framework
- Used by most non-trivial extensions to Eclipse
- Other technologies and frameworks (such as JET2, UML2, and GMF) take EMF and extend it

4

IBM

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Key capabilities

- Can define (model) data structures
- Can generate runtime data classes
- Includes runtime framework to support the data classes
- Can map runtime data classes to persistent storage(like XML)
  - Lets you create XML schema-specific API
- Unifies UML models, Java, and XML
  - Start with a model of the data, annotated Java code, or XML structure, and generate the other two
  - The three forms become interchangeable

5

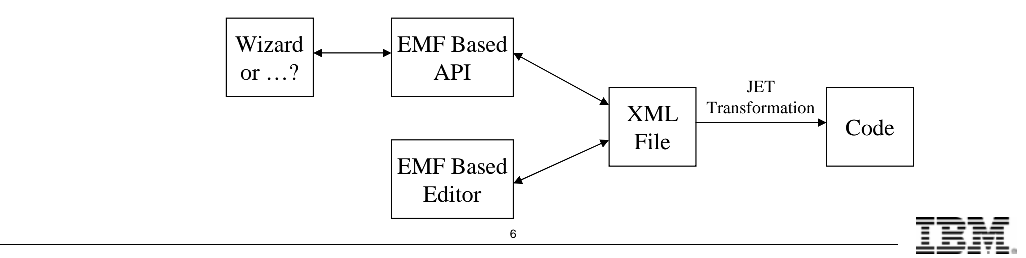
IBM

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

EMF can enhance JET

- Remember that JET transformations take XML (or EMF) files as input
- You can use EMF-based API to manipulate JET input files
  - For example, create a custom wizard that uses the EMF API
- You can use EMF-based editors to manipulate JET input files



6

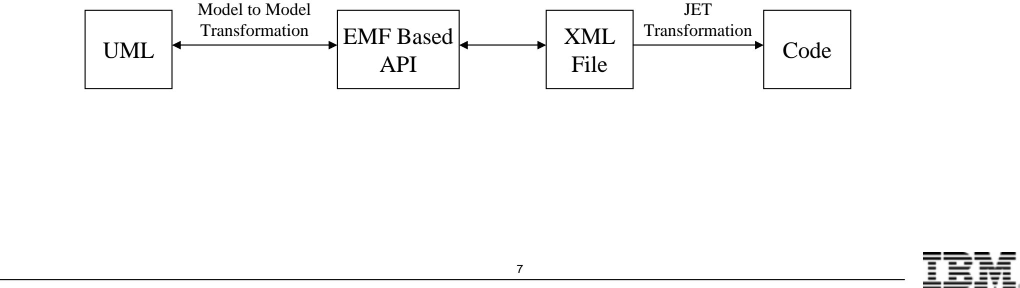
You can use EMF to provide different ways to create and maintain the input files for JET transformations, such as an EMF-based editor.

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

EMF used in UML-to-code transformations

- JET2 generates an input.ecore model representing the input model
- In another module, you will use this EMF model to map between a UML front end and JET2



7

© Copyright IBM Corp. 2007
Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

EMF Files

- ECore
  - Contains the data Schema
  - Even if you import the definition from another source, like XML Schema, EMF stores it in ECore
  - ECore files are used at Design Time and for Code Generation
  - ECore files are not used at run time
    - Generated Java files used at run time
- GenModel
  - Augments an ECore file with code generation settings
  - Maintains a link back to the corresponding ECore file
  - Used at Design Time and for Code Generation

8

IBM

ECore = schema of the data model

GenModel = code generation options

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Creating EMF definition

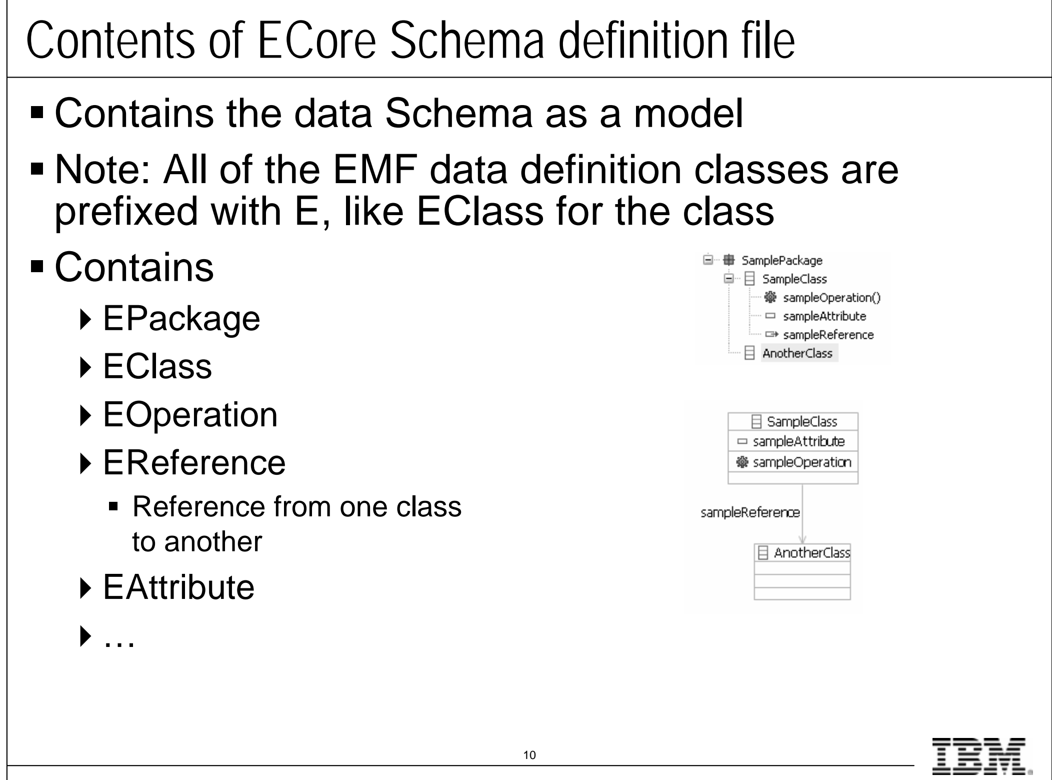
- Can be imported
  - Annotated Java
  - IBM® Rational Rose®
  - Rational Software Architect or Rational Software Modeler model
  - Generic UML2 model
  - XML Schema
- Can be generated from scratch
  - Create a new ECore file and populate it

9

IBM.

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.



The examples on the right show the same example twice. The top right is the ECore file in the ECore editor. The lower right is the ECore file as a diagram.

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Java Code Generation

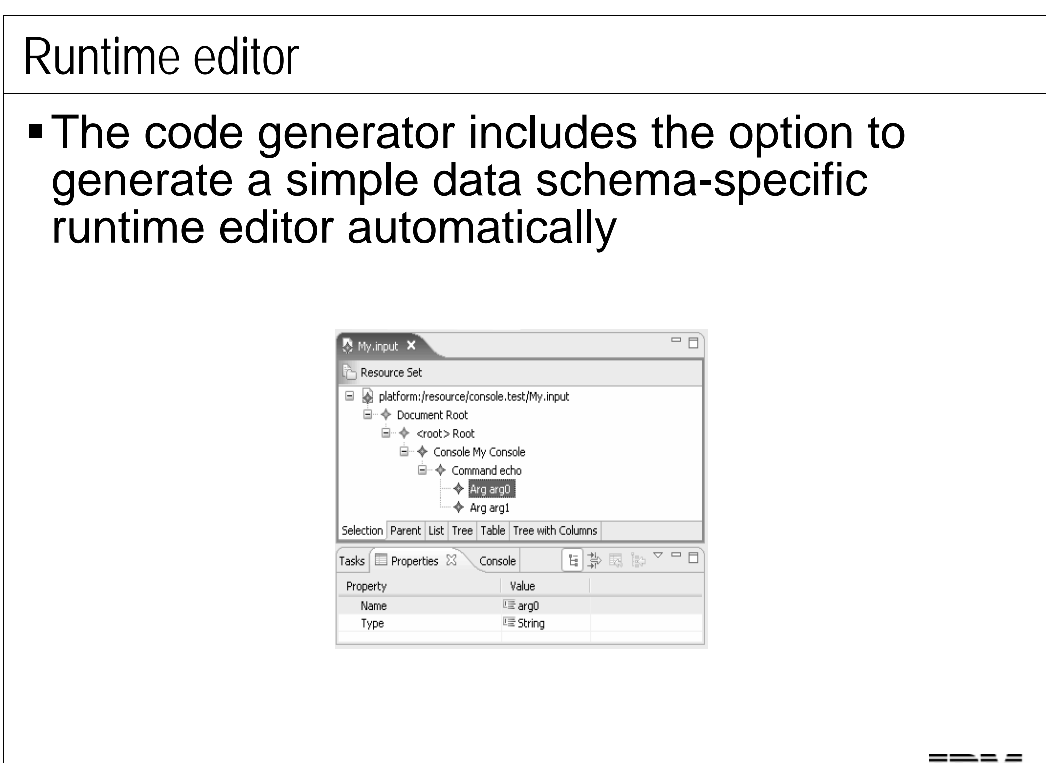
- The Genmodel file is used to customize Java Code Generation
  - Very customizable
- Each EMF data class (EClass) maps to a Java Interface and a Java implementation Class
- The EClass defines getters and setters for attributes
  - Example: aSampleClass.getName()
- It also generates the Eclipse Plugin configuration files
- The result is an easy-to-use API for the data files

11

IBM

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.



© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Lab 6.1: Introducing EMF

- Given:
  - The JET Console Transformation
- After completing this lab, you will be able to:
  - Import an XML Schema Definition into EMF
  - Generate EMF Framework-based code
  - Create an EMF-based Editor which acts as a front-end to a JET transformation

13

IBM

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

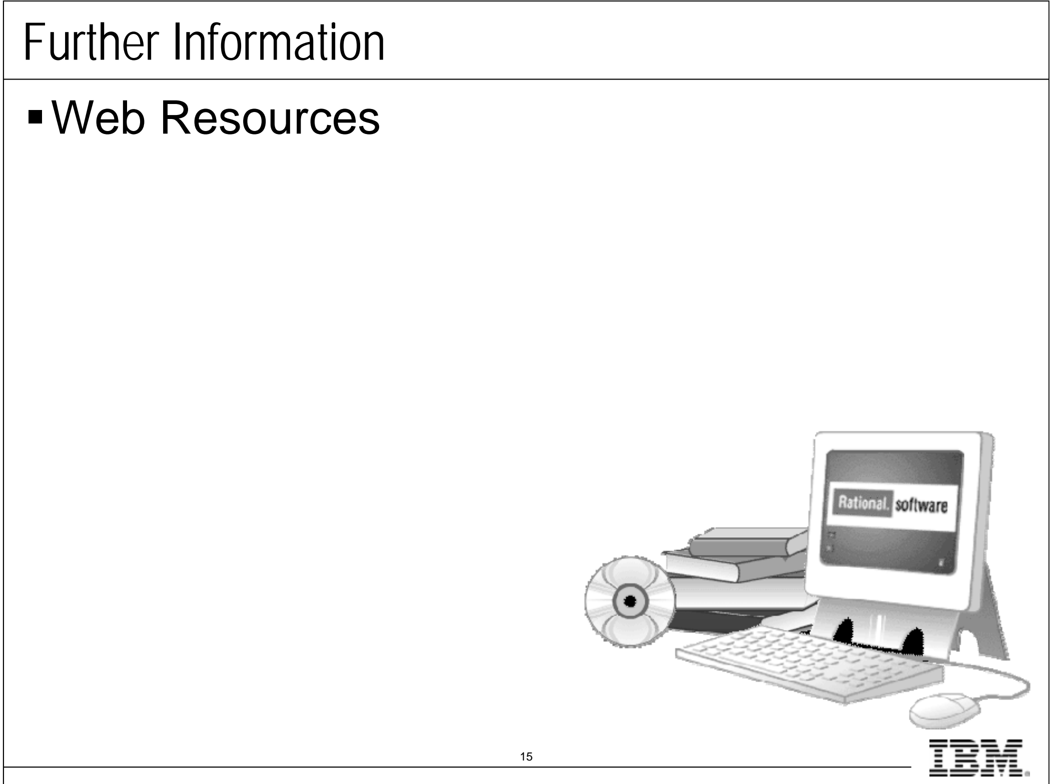
Lab 6.2: Optional EMF Lab

- This optional lab builds a simple EMF model and an EMF editor by hand
- It does not start with any existing data schema
- It does not link into JET

14

IBM

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.



Web Resources

- www.eclipse.org/emf (Eclipse page for EMF)
- www.eclipse.org/emft (EMFT is a set of technologies that extend the base EMF framework.)

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

16

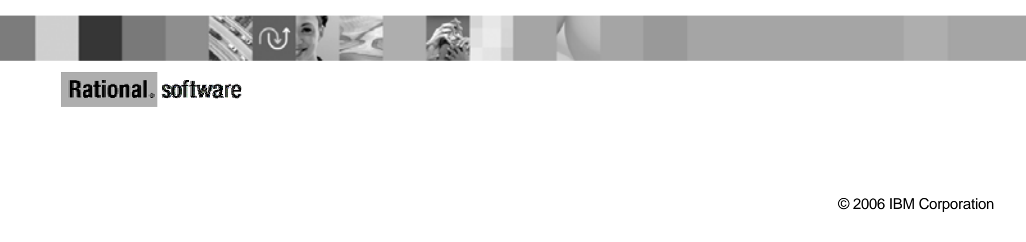
IBM

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

IBM Software Group

DEV498: Pattern Implementation Workshop with IBM Rational Software Architect

Module 10: Introduction to Transformations



Rational. software

© 2006 IBM Corporation

Contents

Objectives 10-2
Configuring and Running Transformations 10-7
Lab 7: Customize a Transformation 10-12
Creating a Model-to-Text Transformation 10-14
Lab 8: Create a Model to JET Transformation 10-38
Review 10-39
Further Information 10-40

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Introduction to Transformations

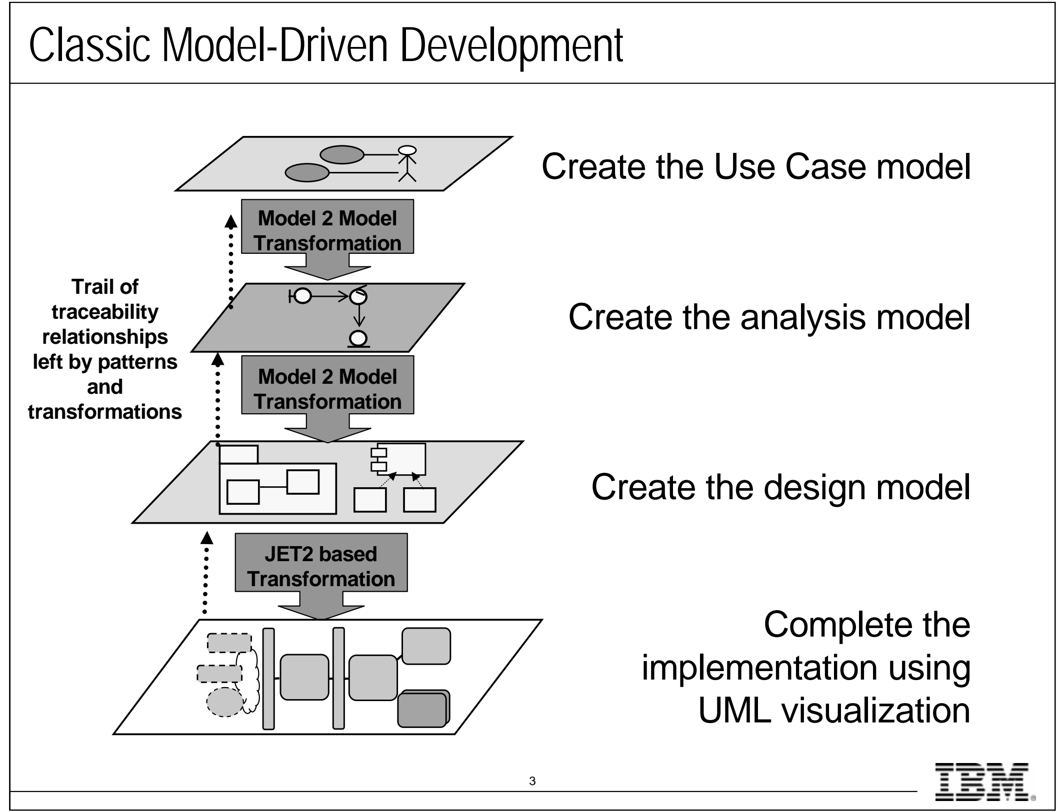
- Objectives:
  - Describe the role of model transformation in the Model-Driven Development approach to software development
  - Connect a UML model to an EMFT JET based transformation using Model Mapping
  - Apply a transformation in Rational Software Architect

IBM

This module introduces model transformations in Rational Software Architect. After briefly introducing the role of transformations in Model-Driven Development, the module discusses how to configure a transformation and then moves on to how to connect a UML model to an EMFT JET based transformation using Model Mapping.

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

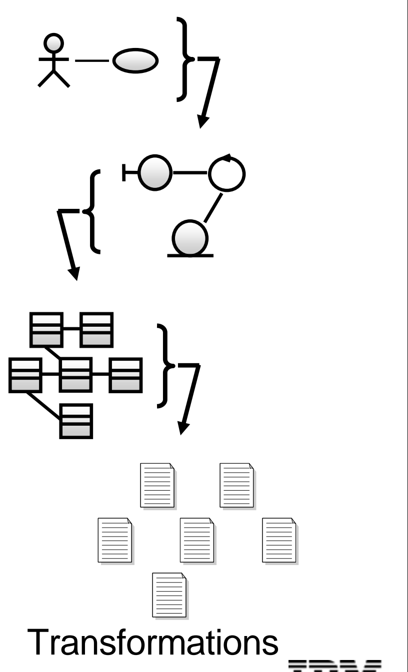


© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Transformations

- Transformations create elements in a target model (domain) based on elements from a source model
- Often, the source domain is more abstract than the target domain
- Examples:
  - Based on a use-case model, create an analysis model containing analysis classes, sequence diagrams, and so on, that realize the use cases following company standards
  - Based on the analysis model, create a design model(containing the appropriate design classes) that incorporates elements of the company's security and persistence frameworks, and that follows the company standards
  - Starting with a UML model, apply Rational Software Architect's standard "UML to EJB" transformation to create EJB code elements



IBM

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Model Transformation Uses

- There is flexibility in choosing transformation sources and targets:
  - Transform a model to a model of the same type
  - Transform a model across levels of abstraction
  - Transform one type of model to another
  - Extend another transformation

Transformation Definition

Model → Transformation Tool → Model

5

The following transformations are possible:

- **Across models of the same type:** When adding levels of refinement, you may transform from a PSM to another PSM. More details are added, but the type of model remains the same.
- **Across levels of abstraction:** Move from a PIM model to a PSM model as you add in details about the platform and get closer to the implementation.
- **From one type of model to another:** With transformations you can transform UML to code. This is the most common transformation available in Rational Software Architect.
- **Extend another transformation:** In Rational Software Architect transformations can be built on top of existing transformations.

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Where Are We?

- **Configuring and Running Transformations**
- Creating a Model to Text Transformation

6

IBM

This section introduces the role of transformations in model development with Rational Software Architect.

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Applying a Transformation

- Create a new instance of the transformation
- Set up a mapping model (optional)
- Choose an appropriate project target
- Create source to target relationships

Transformation Profile
Design Model
Markup
Model File
Transformation
Transformation Configuration
Mapping Model (Optional)
Model (L)

7

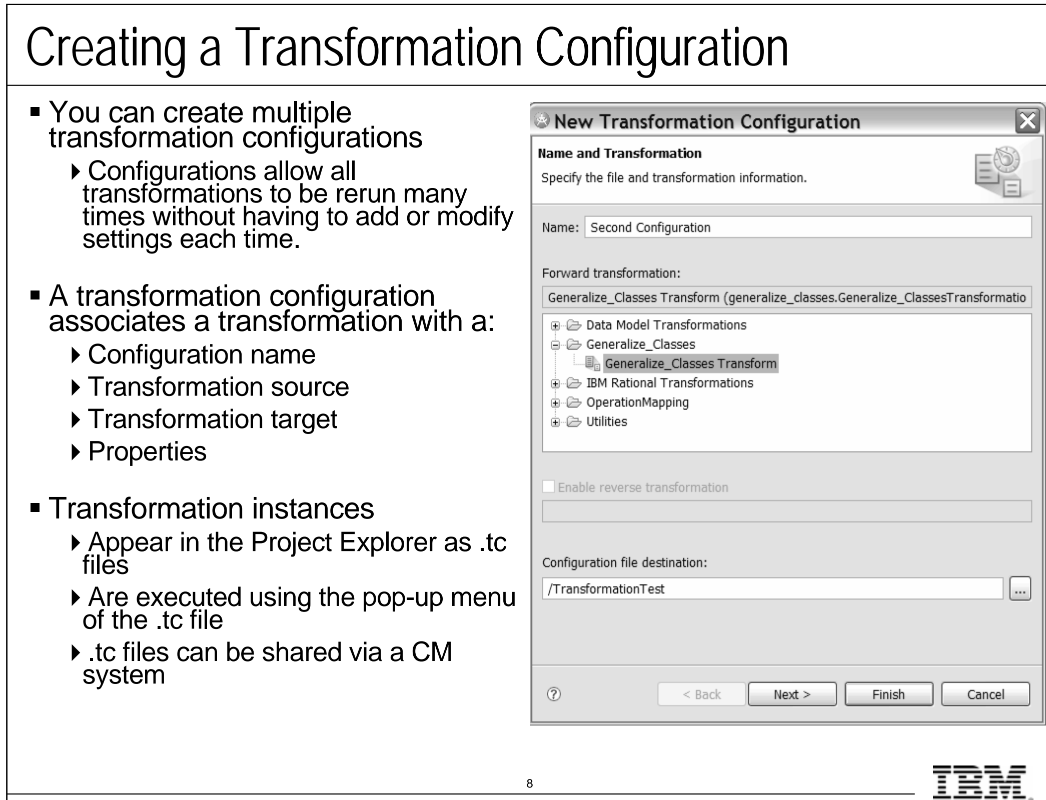
To apply a transformation, you must configure it by specifying properties. Transformation configurations define how a specific transformation will be applied. You can define multiple transformation configurations for the same transformation. The model to be transformed can include markup, such as keyword applications (often from UML patterns applied to the model) that get used in the transformation. The transformation can apply stereotypes from any profiles created for the more platform-specific target model.

As an optional step you can also use a mapping model. Mapping models describe how the transformed elements will be created in your target; what is going to be the the name of the created artifact going to be going to be, under which package will they reside, and so on.

The last step is applying the transformation configuration to generate the transform elements in the target you specified.

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

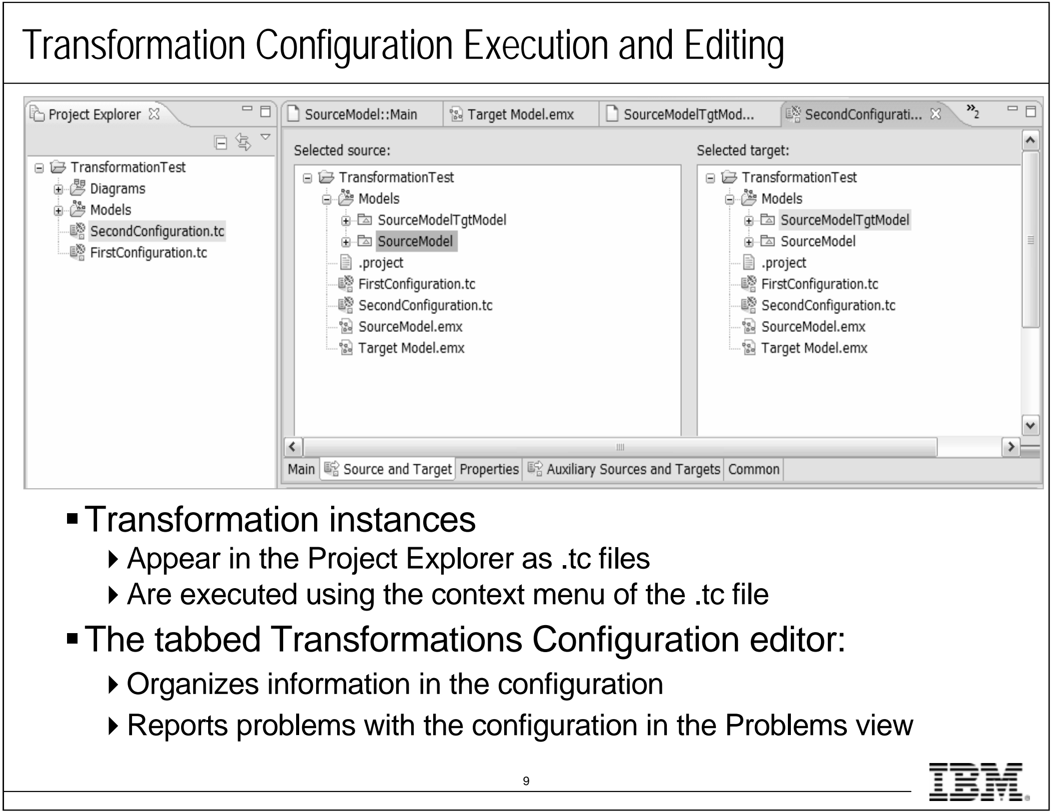


Before you can apply a transformation to a source model, you must first create a transformation configuration. A transformation configuration is an instance of a transformation that contains the information that the transformation uses to generate the output that you expect, such as the specific transformation source and target, and its properties.

The Configure Transformations dialog shows what transformations are installed and which configurations are based on them, with the instance shown under the transformation. Clicking the transformation or instance in the left pane brings up the properties of the item in the right side of the dialog.

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.



Before you can apply a transformation to a source model, you must first create a transformation configuration. A transformation configuration is an instance of a transformation that contains the information that the transformation uses to generate the output that you expect, such as the specific transformation source and target, and its properties.

The Configure Transformations dialog shows what transformations are installed and which configurations are based on them, with the instance shown under the transformation. Clicking the transformation or instance in the left pane brings up the properties of the item in the right side of the dialog.

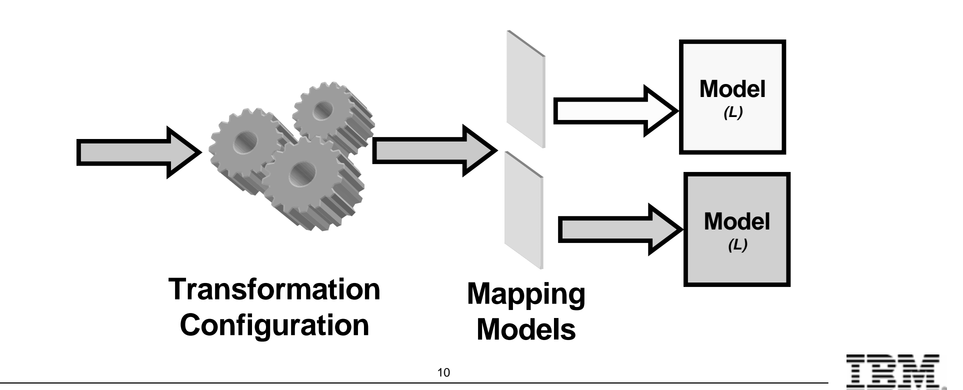
© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Using a Mapping Model

A mapping model allows you to rename elements and rearrange the structure of transformed elements in the target model.

- Generate the mapping model from the Transform Configuration wizard, or the multi-page editor.
- Specify the file property of the artifacts.
- Set the transform to use the mapping model.



10

A transformation assigns default file names to the files and folders that it generates based on the logical element names and structure of the source model. You can use a mapping model to specify an alternate file name for files and folders that a transformation generates. You can also use a mapping model to specify the file structure of files that a transformation generates.

A mapping model contains an artifact for each element selected in the source model. Each artifact refers to, and has the same name as, the corresponding source model element. You can specify an alternate file name by changing the file name property of an artifact. The next time you run a transformation, you can select the mapping model that you edited. The transformation assigns the file name (that you specified in the file name property of each artifact) to the corresponding target element.

You must create a mapping model in the same workspace and project as the selected model elements.

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Uses for Mapping Models

Use a mapping model in cases where:

- You need to create different file names and structures in the target model
- It is impossible or impractical to change the names of elements in the source model

Project Explorer
Generalize Classes
MyJava
my.proj
otherStuff.java
stuff.java
JRE System Library [jdk]
operationMapping
SimpleModel
Diagrams
Models
MyMapping
(UMLPrimitiveTypes)
Thing1
(Thing1)
Thing2
Simple Model
(UMLPrimitiveTypes)
Main
Thing1
Thing2

Properties
General
Stereotypes
Documentation
Constraints
Advanced
<Artifact> MyMapping::Thing1
Name: Thing1
File Name: my.proj.stuff
Visibility: Public Private Protected
Abstract:
Leaf:

11

In many cases, the structure of the model produced by the transformation will have to be different from the source model. When you need to seed the code based on a design model, the names and structure of design packages might not make sense in the target coding environment, and they might need to be mapped to a different structure of physical packages. A mapping model can assist you in handling this transition, so that you do not have to make any temporary changes to the design model just to perform the transformation.

In some cases, it is not desirable just to change the structure of the source model, such as when you might need to transform the same model to many different transformation targets, with different structures. Developing sets of mapping models for different target types is the best solution for these cases.

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Lab 7: Customize a Transformation

Complete the following tasks:

- Create the Workspace
- Create the Source and Target Projects
- Populate the Source Project
- Apply a UML-to-Java Transformation
- Use a Mapping Model

12

IBM.

Complete Lab 7 in the student workbook.

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Where Are We?

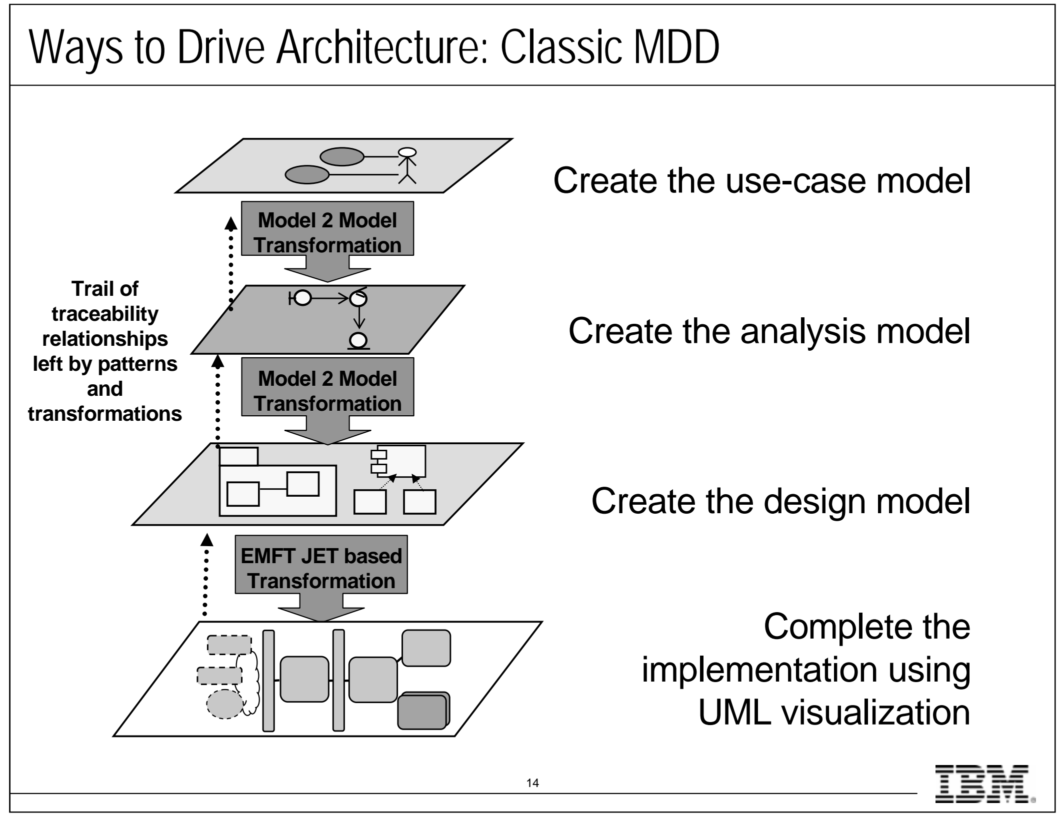
- Configuring and Running Transformations
- **Creating a Model to Text Transformation**

13

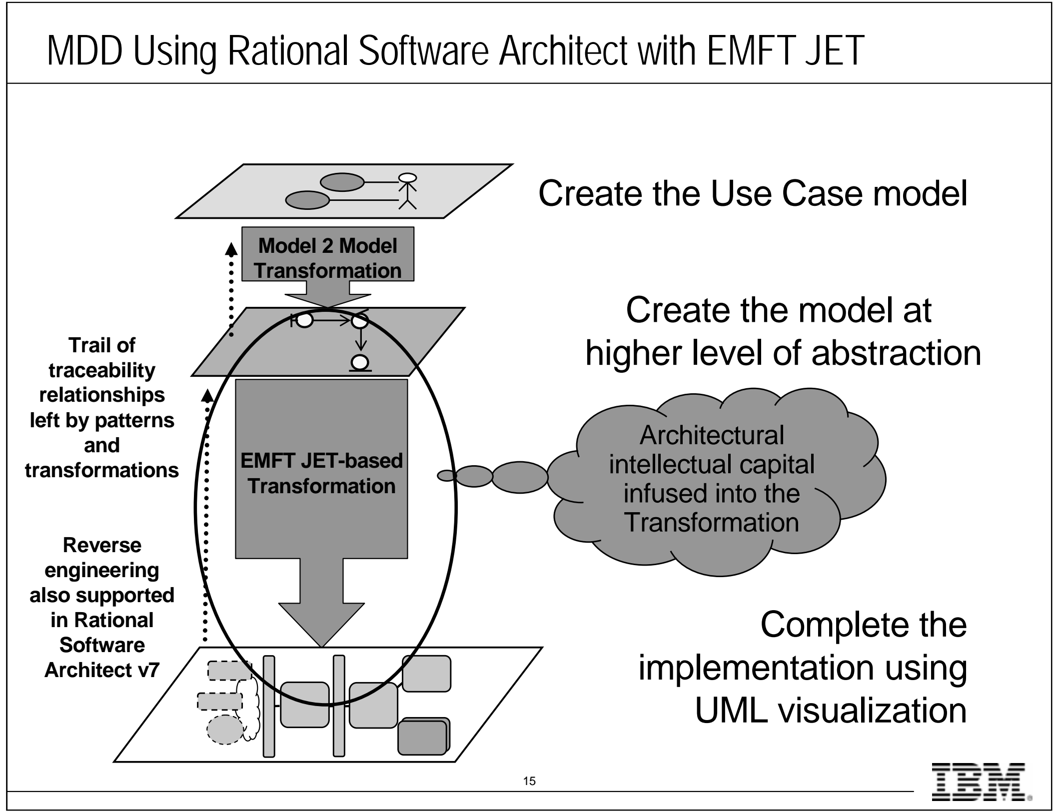
This section introduces the role of transformations in connecting a UML model to an EMFT JET based transformation using Model Mapping within Rational Software Architect.

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.



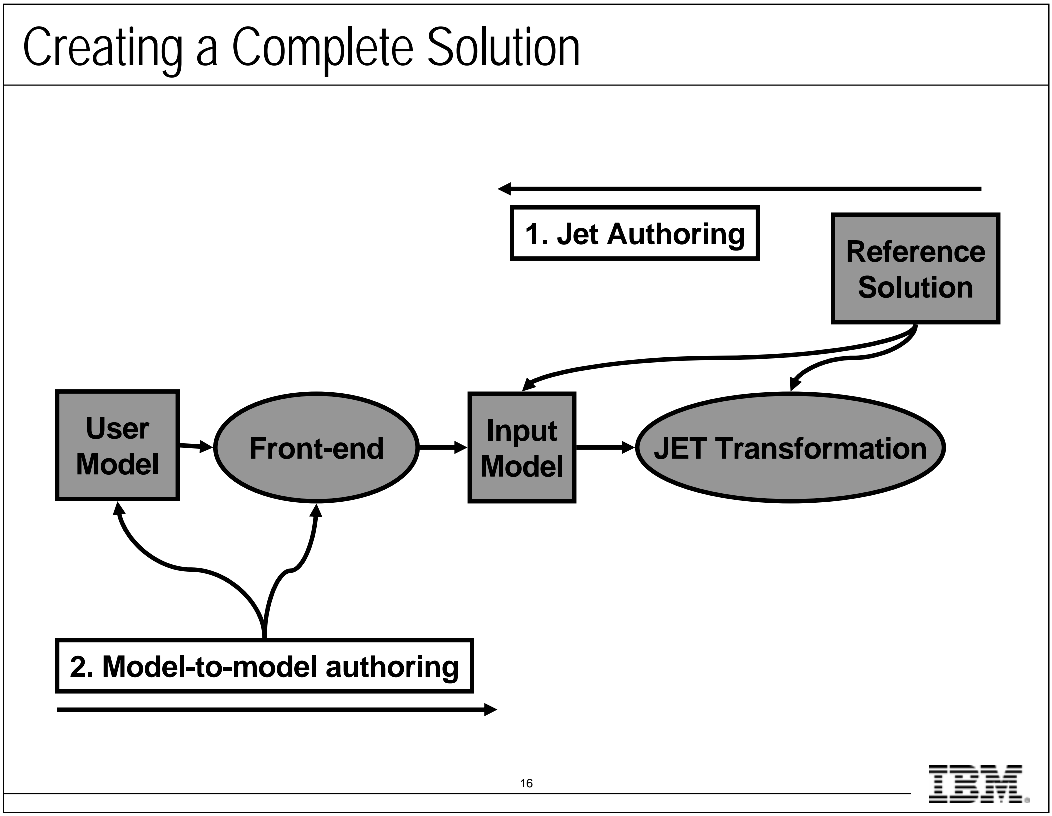
Course materials may not be reproduced in whole or in part without the prior written permission of IBM.



This represents one of various approaches customers have taken to reduce the amount of modeling (thus limiting variability) while infusing consistent architecture in the form of a pattern based transformation.

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.



Using Rational Software Architect transformations, you can create EMFT JET input from UML models.

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Steps to Create Model to JET Transform

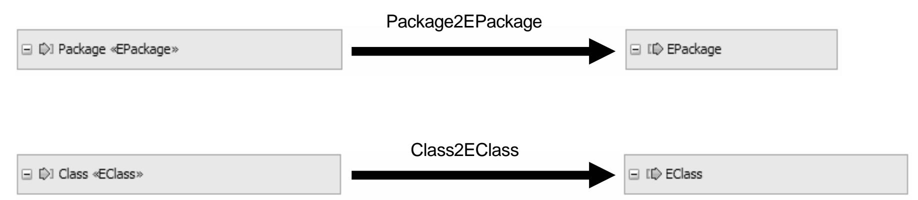
1. Determine what (and how) UML elements will map to the input model of the JET transformation
2. Create an EMF Project from the ECore model of the JET Transformation
3. Generate code for the EMF model
4. Create a mapping transformation from the UML ECore model to the JET transformation ECore model
5. Generate the mapping transformation code
6. Add the JETRule code to the mapping transformation
7. Test and run the mapping transformation

17

IBM

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Mapping Models Contain Mapping Declarations



- Mapping models are ECore models
- Mapping models contain references to the ECore models that are being mapped. For example:
  - UML.ecore – input model
  - Ecore.ecore – output model (input into JET2)
- Mapping models are persisted like other ECore Models; they are serialized as XML files

18

IBM

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Mapping Model in XML Editor

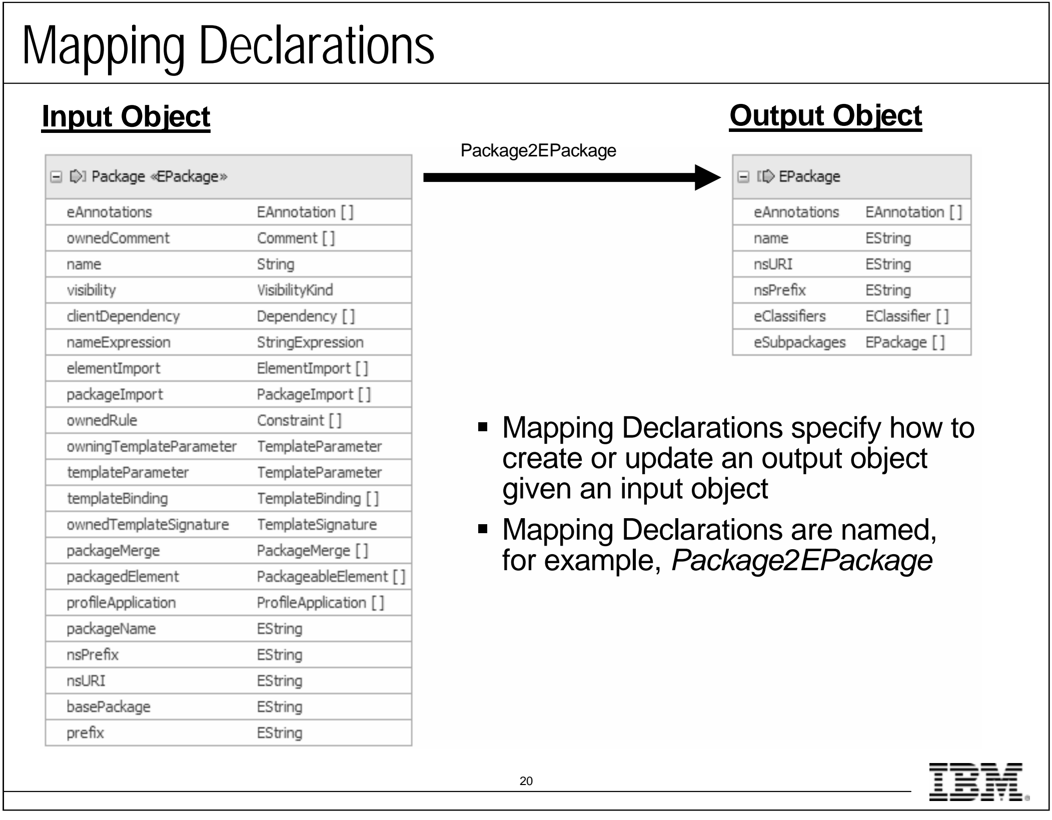
```
<?xml version="1.0" encoding="UTF-8"?>
<mappingRoot xmlns="http://www.ibm.com/2006/ccl/Mapping" xmlns:map="http://lab.console.tra
    <input path="/plugin/org.eclipse.uml2.uml/model/UML.ecore" var="src"/>
    <output path="/resource/lab.console.project.model/model/input.ecore" var="tgt"/>
    <mappingDeclaration name="ModelToRoot">
        <input path="type('Model')" var="Model_src"/>
        <output path="type('Root')" var="Root_tgt"/>
        <mapping>
            <input path="packagedElement" var="packagedElement_src"/>
            <output path="console" var="console_tgt"/>
            <submap ref="map:ConsoleClassToConsole"/>
        </mapping>
    </mappingDeclaration>
    <mappingDeclaration name="ConsoleClassToConsole">
        <input path="type('Class')" var="Class_src"/>
        <output path="type('Console')" var="Console_tgt"/>
        <mapping>
            <input path="name" var="name_src"/>
            <output path="name" var="name_tgt"/>
        </mapping>
        <mapping>
            <input path="package/name" var="name_src"/>
            <output path="package" var="package_tgt"/>
        </mapping>
        <mapping>
            <input path="ownedOperation" var="ownedOperation_src"/>
            <output path="command" var="command_tgt"/>
            <submap ref="map:OperationToCommand"/>
        </mapping>
    </mappingDeclaration>
    <mappingDeclaration name="OperationToCommand">
```

19

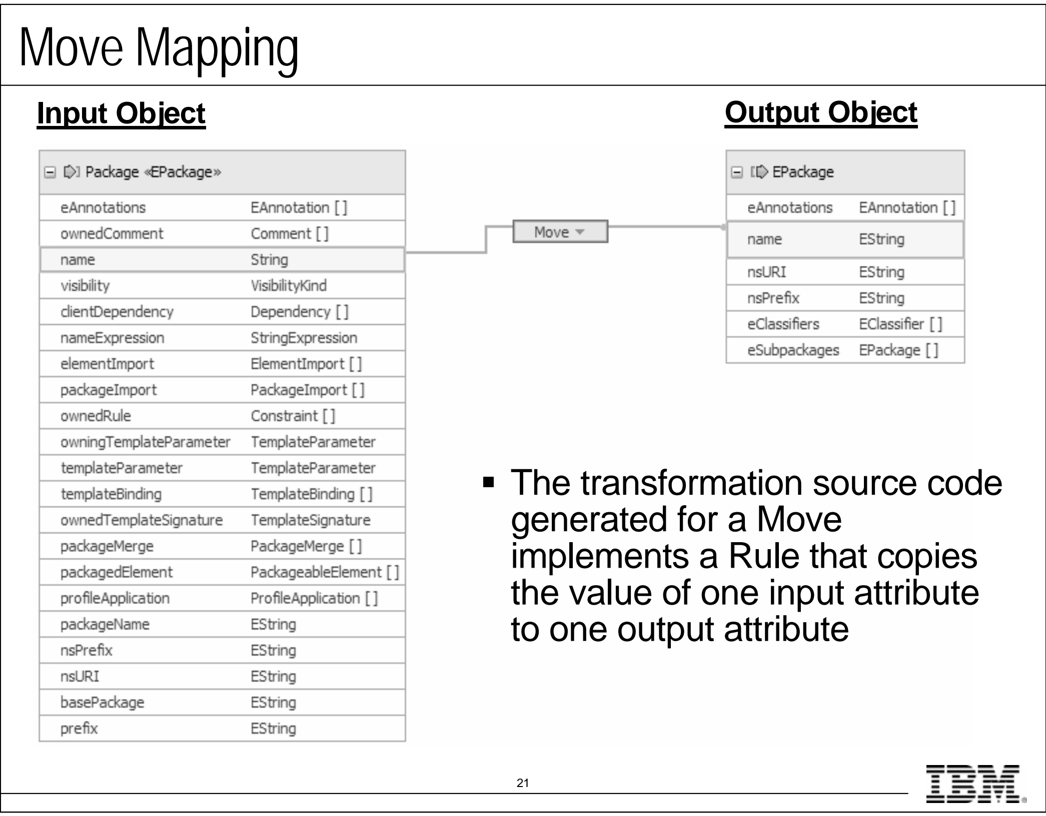
IBM

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

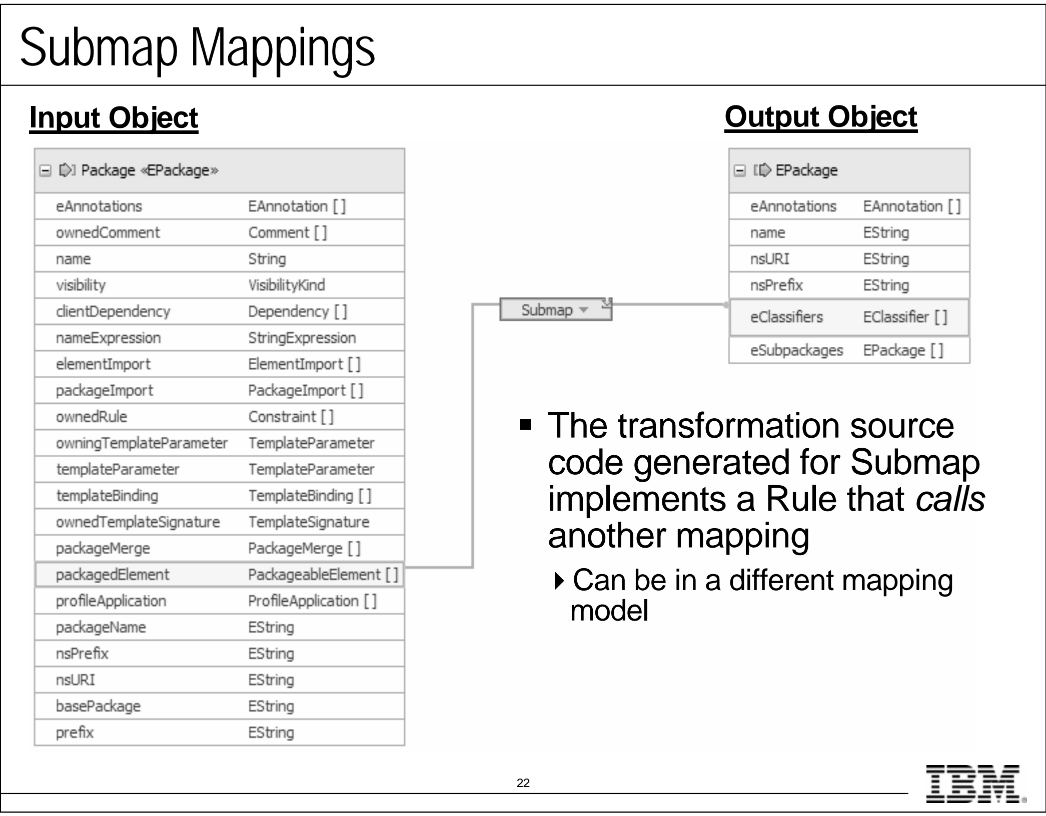


Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

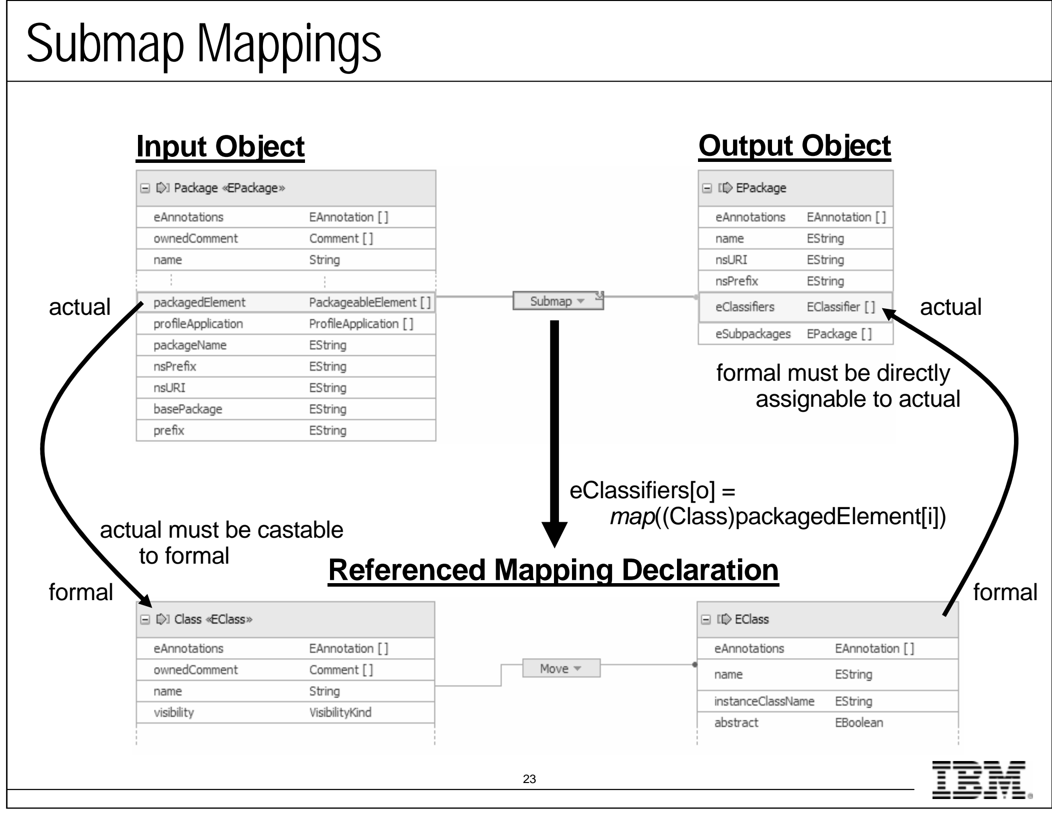


© Copyright IBM Corp. 2007

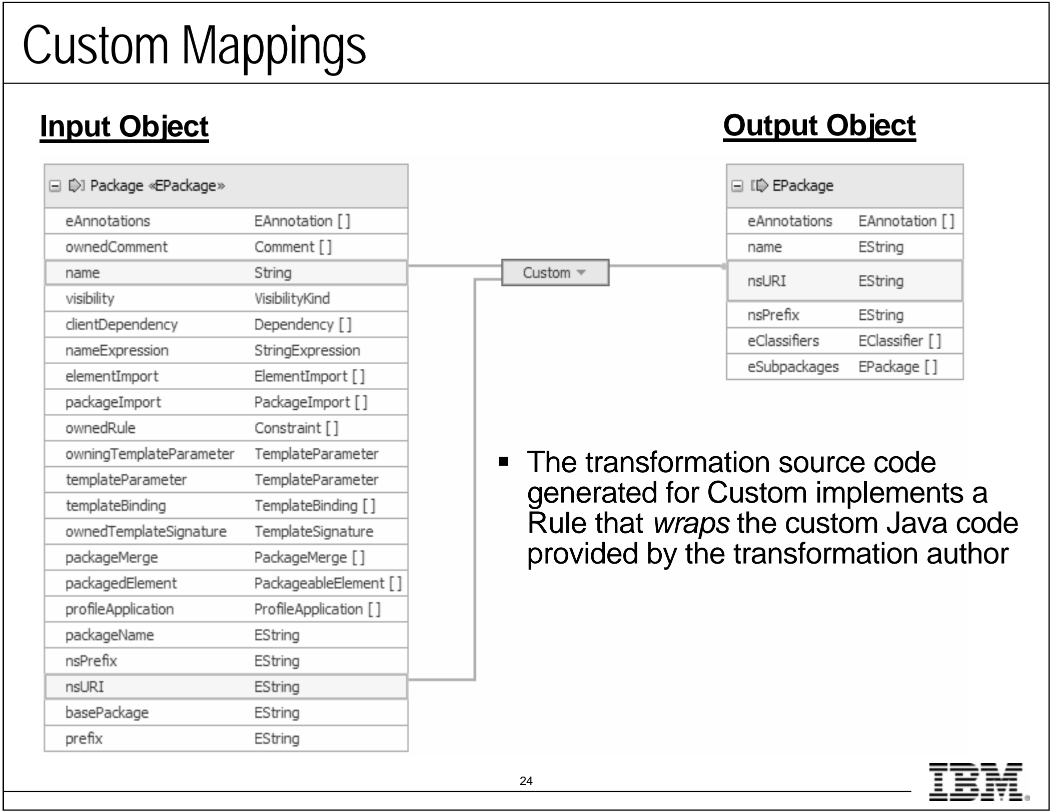
Course materials may not be reproduced in whole or in part without the prior written permission of IBM.



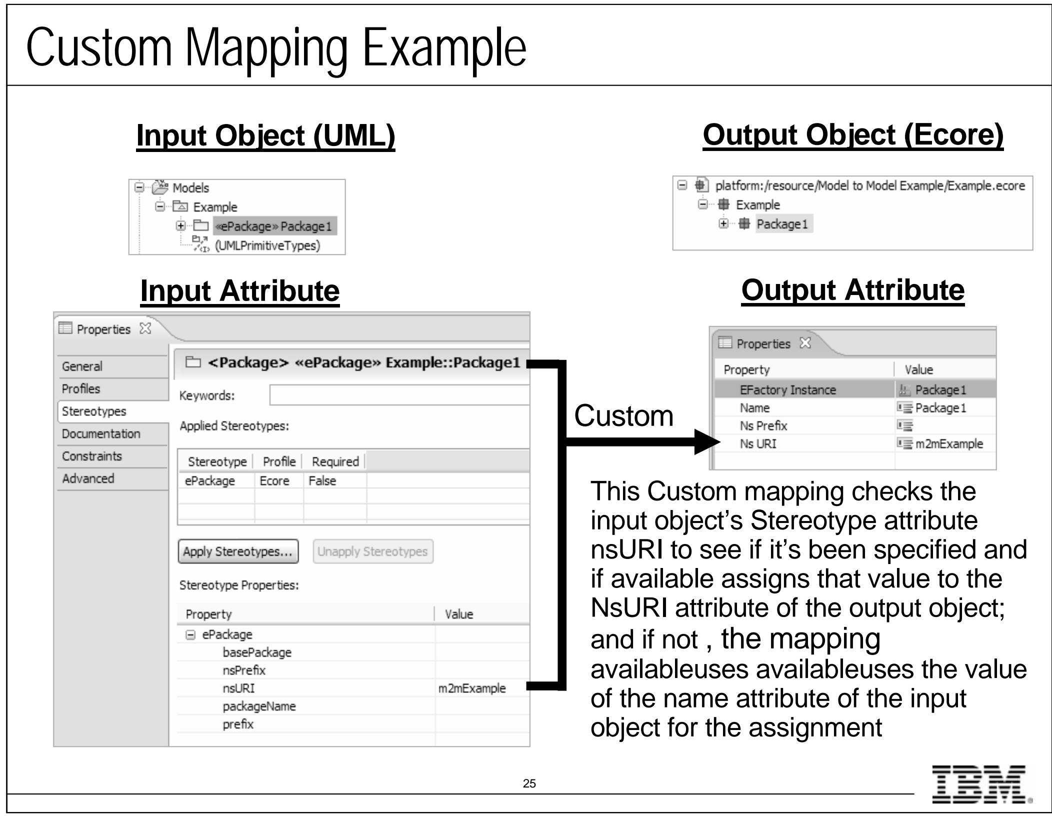
Course materials may not be reproduced in whole or in part without the prior written permission of IBM.



Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

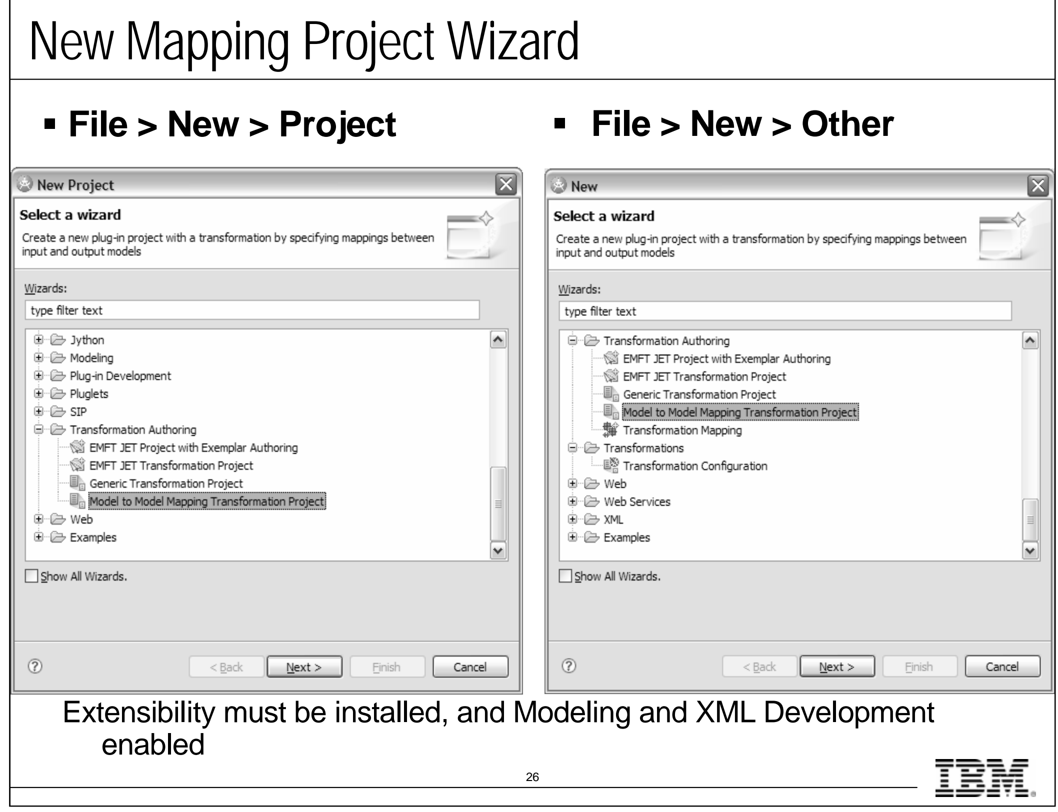


Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

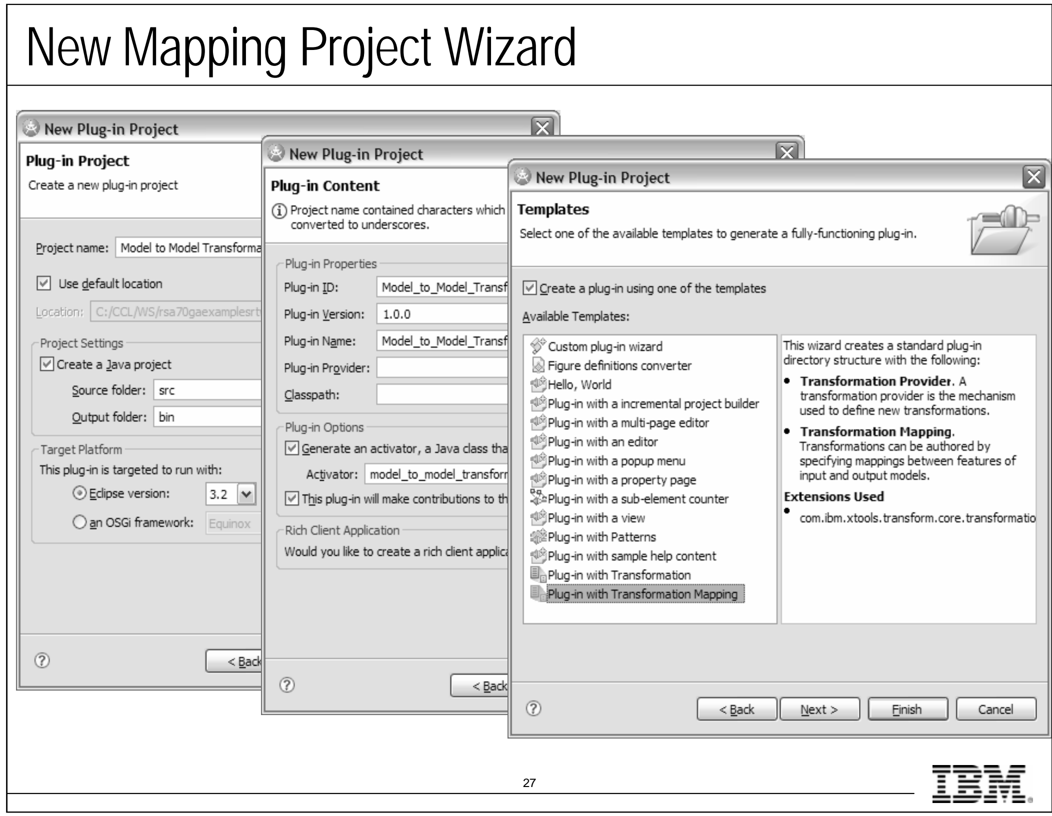


© Copyright IBM Corp. 2007

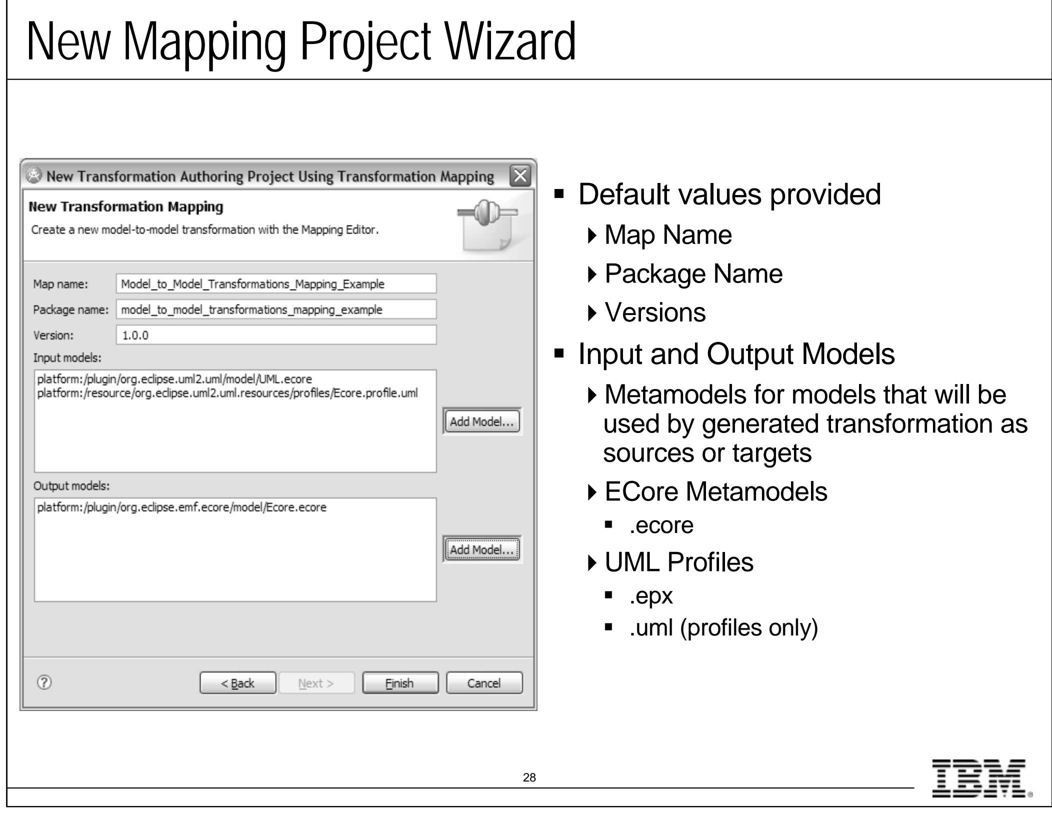
Course materials may not be reproduced in whole or in part without the prior written permission of IBM.



Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

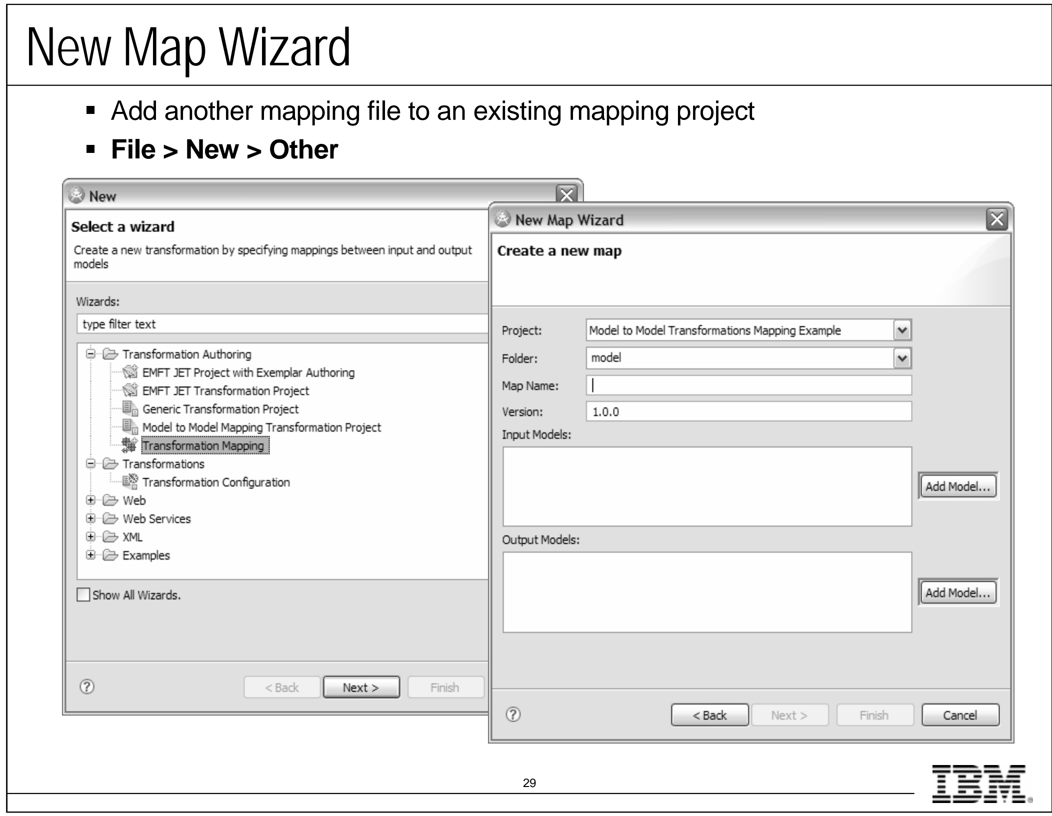


Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

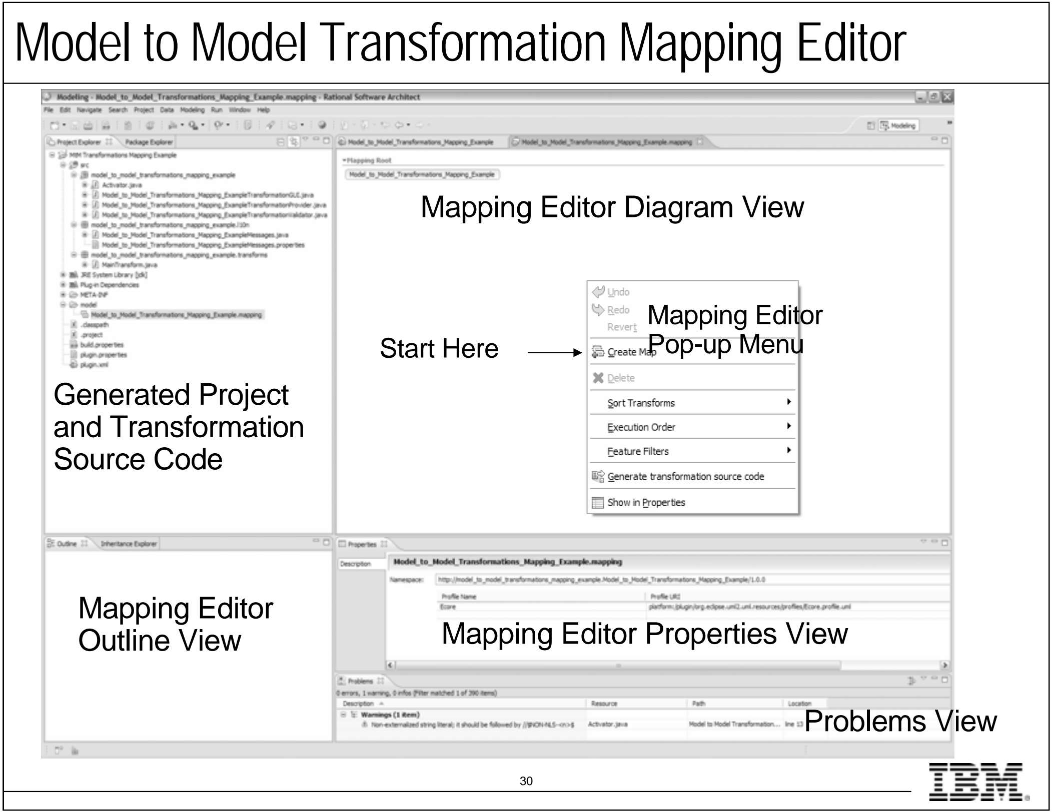


© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

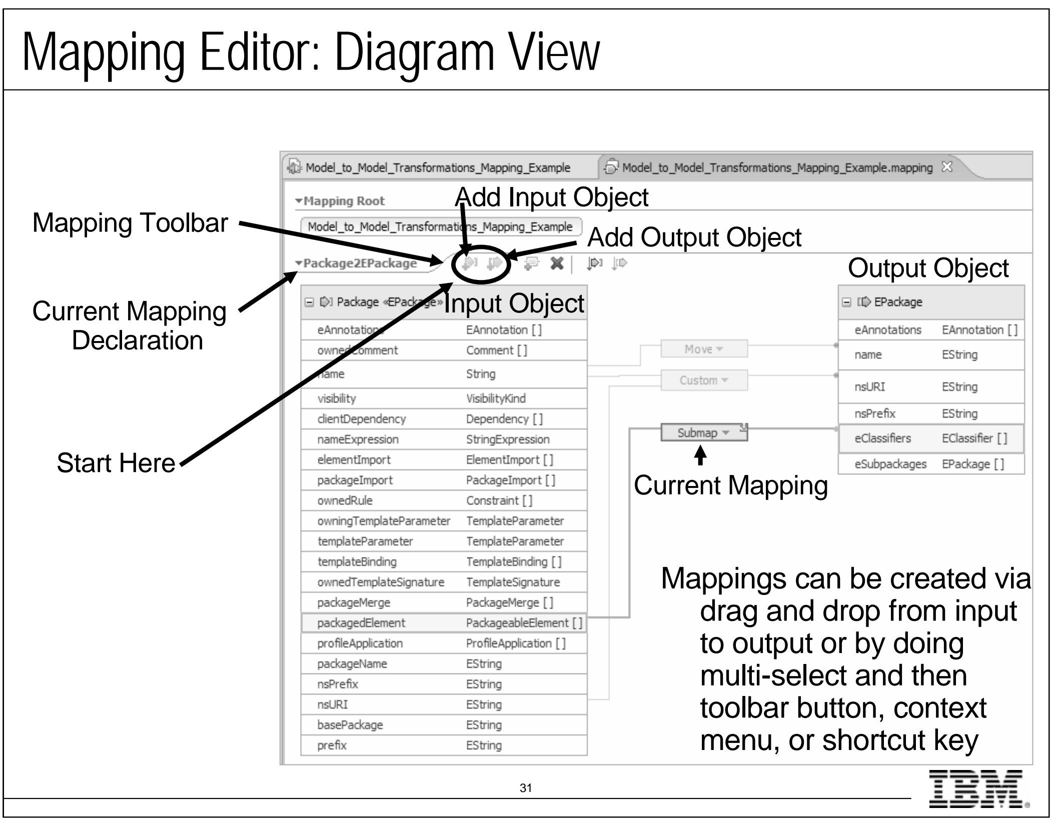


© Copyright IBM Corp. 2007
Course materials may not be reproduced in whole or in part without the prior written permission of IBM.



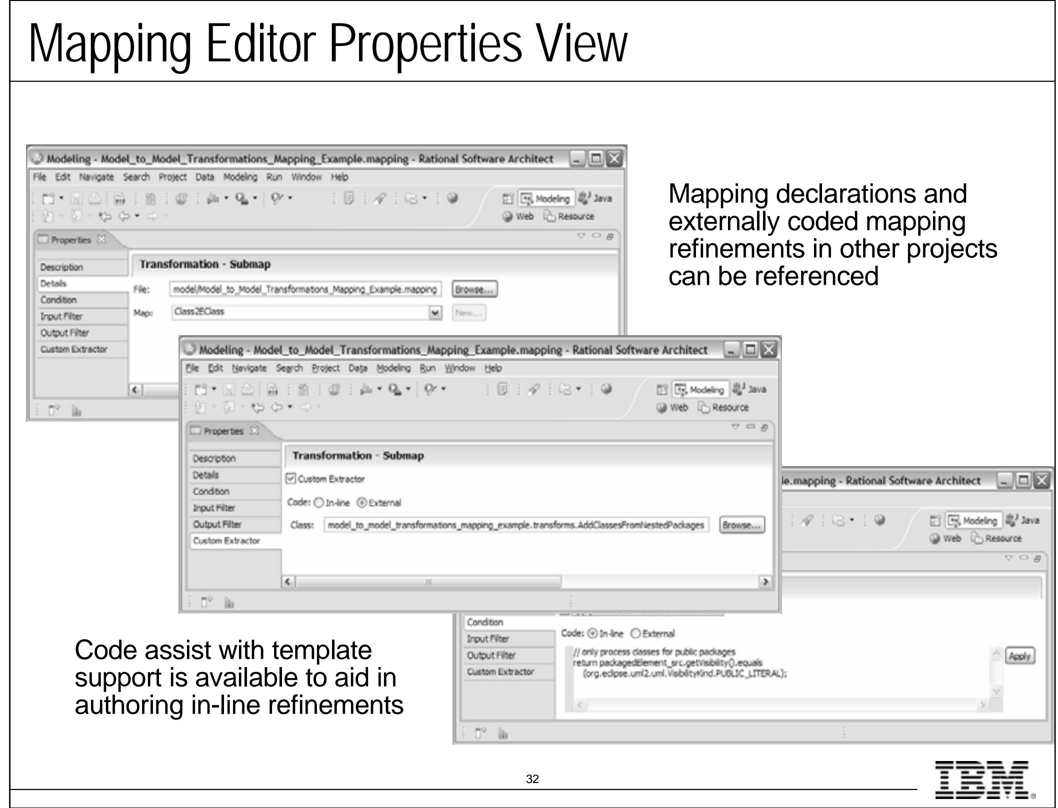
© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.



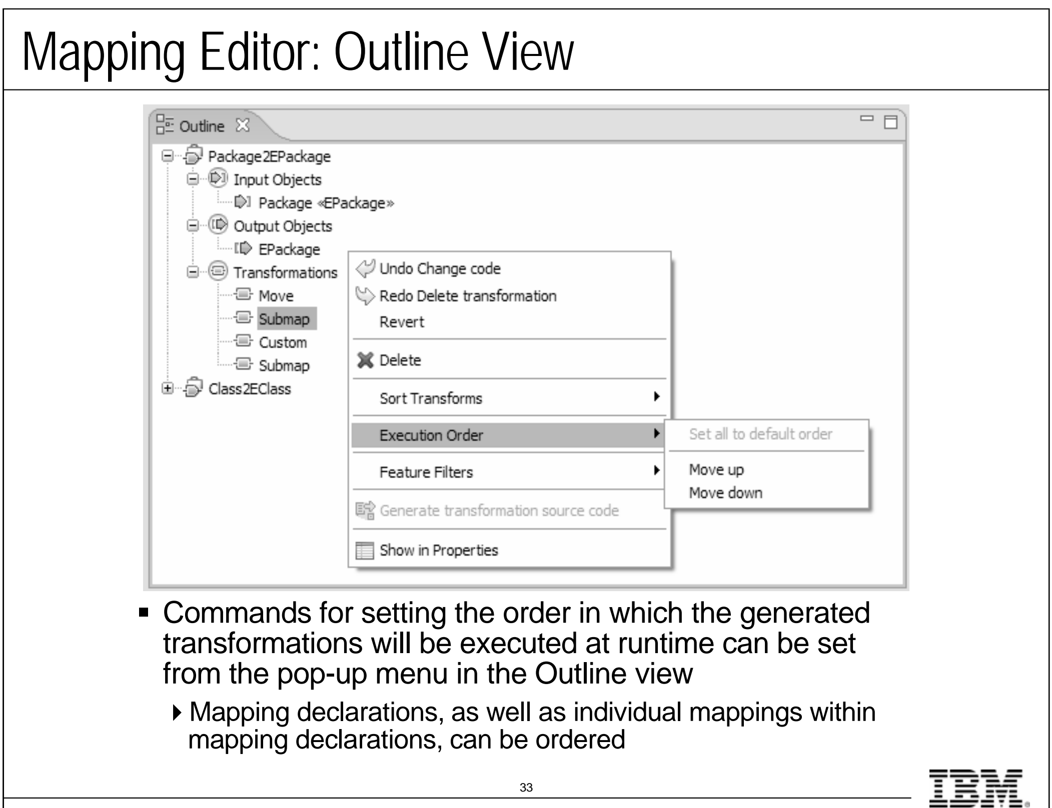
© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.



Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Mapping Editor: Outline View

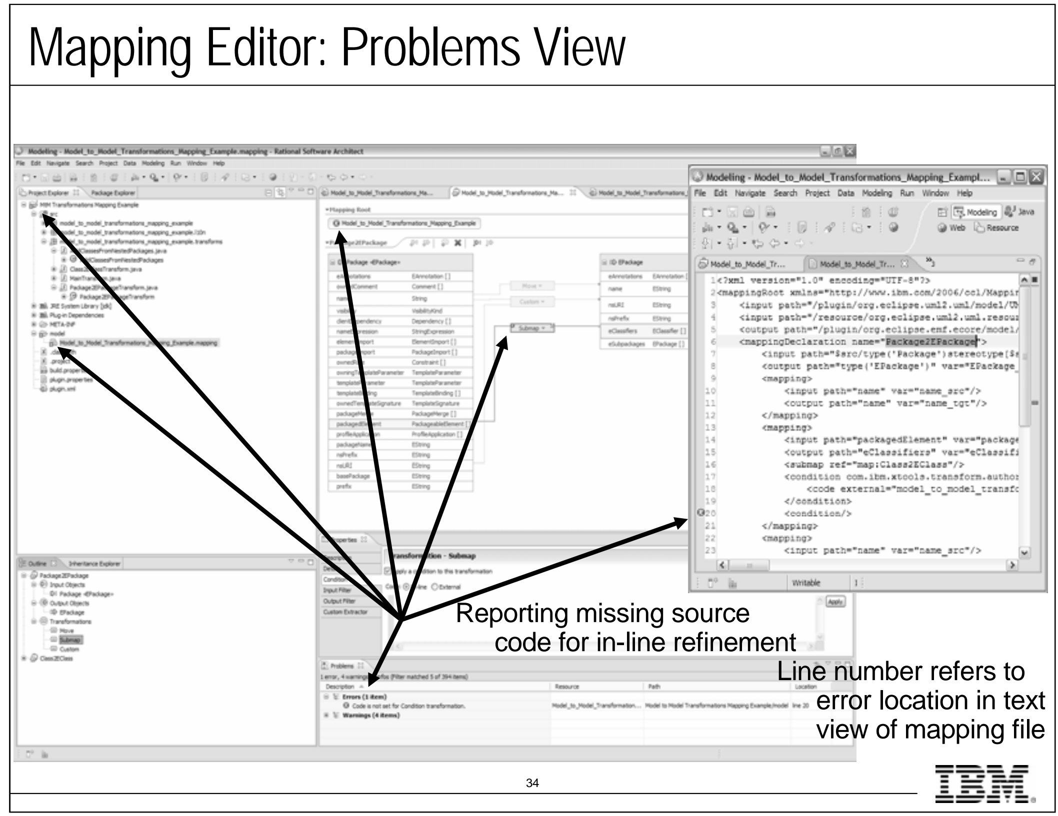


- Commands for setting the order in which the generated transformations will be executed at runtime can be set from the pop-up menu in the Outline view
  - Mapping declarations, as well as individual mappings within mapping declarations, can be ordered

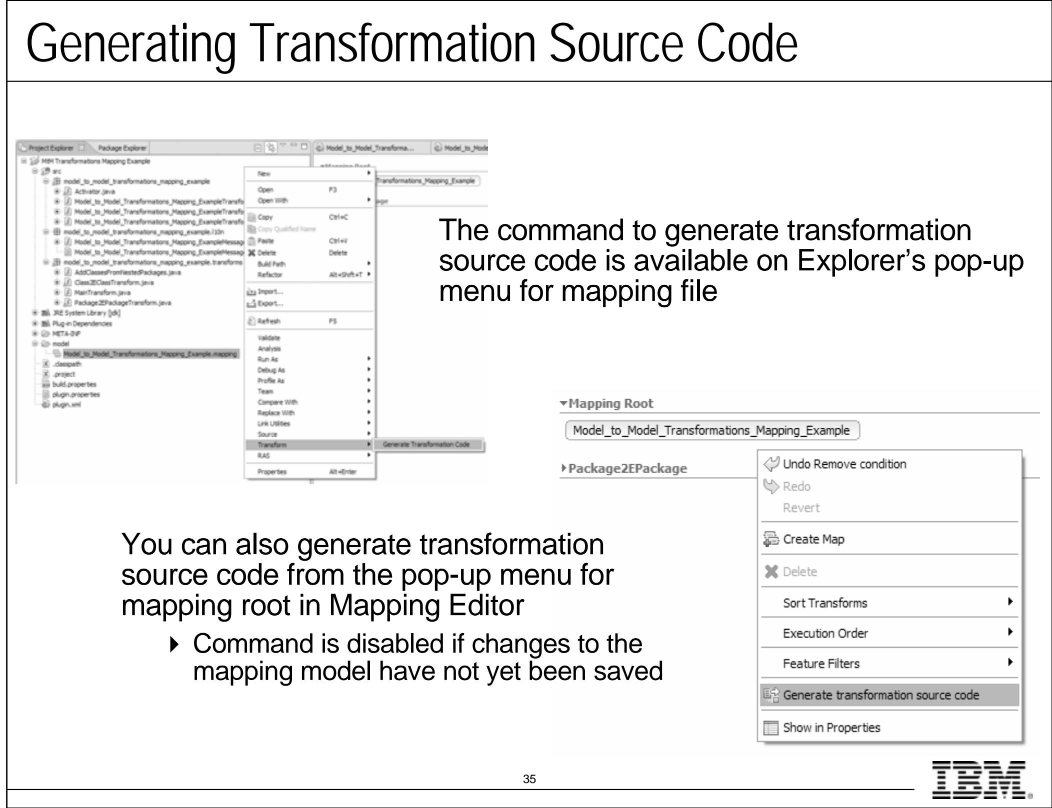
33

© Copyright IBM Corp. 2007

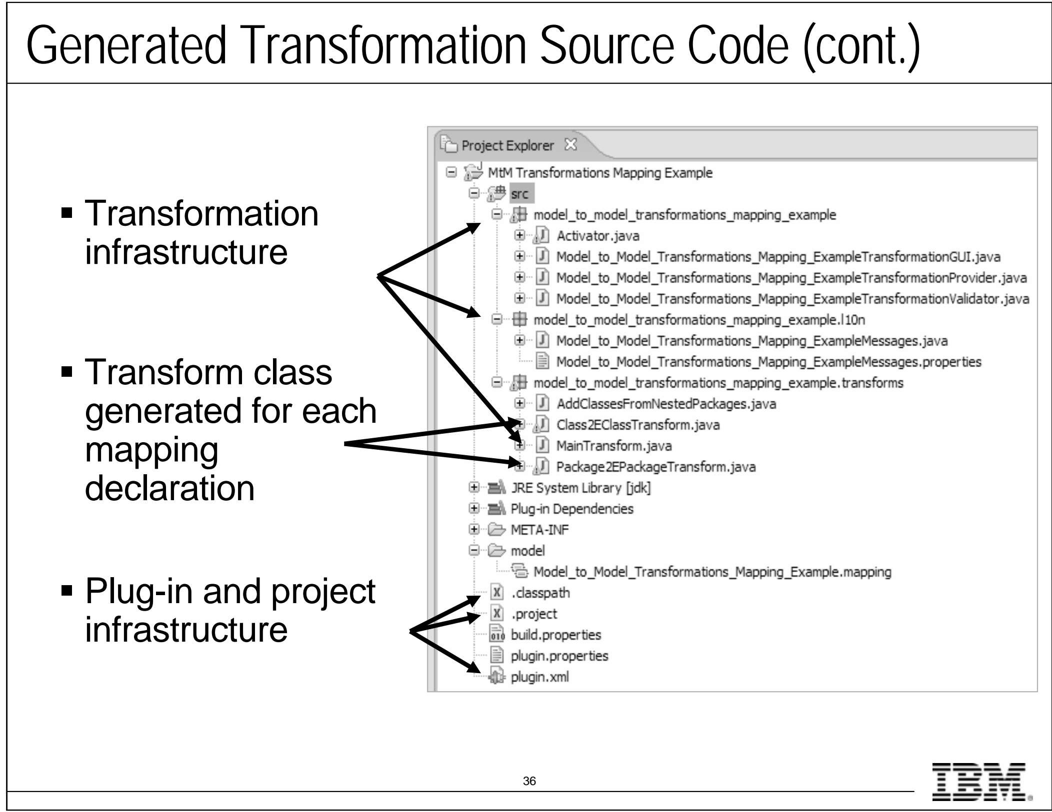
Course materials may not be reproduced in whole or in part without the prior written permission of IBM.



Course materials may not be reproduced in whole or in part without the prior written permission of IBM.



Course materials may not be reproduced in whole or in part without the prior written permission of IBM.



Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Typical Extension: Chain to JET Transformation

- Transformations can be chained together
- A model-to-model transformation can be chained to a JET model-to-text transformation
  - The intermediate model need not be persisted

```
/**
    * Creates a root transformation. You may add more rules to the transformation here
    * <!-- begin-user-doc -->
    * <!-- end-user-doc -->
    * @param transform The root transformation
    * @generated NOT
    */
   protected RootTransformation createRootTransformation(ITransformationDescriptor descriptor)
   {
       return new RootTransformation(descriptor, new MainTransform()) {
           protected void addPostProcessingRules() {
               add(new JETRule("MyJetTransformation")); //$NON-NLS-1$
           }
       };
   }
```

- Add post-processing rule to createRootTransformation method in generated TransformationProvider class
  - Override @generated tag
  - Specify JET transformation

37

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Lab 8: Create a Model to JET Transformation

- Given
  - JET Transformation
  - Code Snippets
  - Test Model
- Complete the following tasks:
  - Create a UML model-to-text transformation

38

IBM

Complete Lab 8 in the student workbook.

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Review

- What is a transformation configuration?
- Describe potential uses for custom transformations.
- Describe possible uses for mapping models.
- How does JET2 work with Rational Software Architect transformations?
- How do you select the right transformation technology?

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Further Information

- Rational Software Architect Help
- Web Resources
- Literature

40

IBM

Rational Software Architect Help Topics

- IBM Rational Software Modeler API

Web Resources

- Alan Brown, "An introduction to Model Driven Architecture Part I: MDA and Today's Systems." http://www-128.ibm.com/developerworks/rational/library/3100.html
- Alan Brown, "An Introduction to Model-Driven Architecture Part III: How MDA affects the iterative development process" http://www-128.ibm.com/developerworks/rational/library/apr05/brown/

Literature

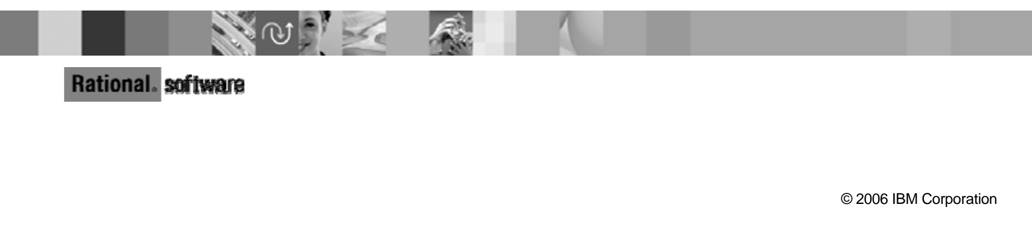
- Frankel, David S. *Model-Driven Architecture: Applying MDA to Enterprise Computing.* Indianapolis, IN: Wiley, 2003.

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

IBM Software Group

DEV498: Pattern Implementation Workshop with IBM Rational Software Architect
Module 11: Designing Reusable Assets



Rational. software

© 2006 IBM Corporation

Contents

Objectives	11-2
Model-Driven Development (MDD)	11-4
Summary	11-25
Review	11-26

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Designing Reusable Assets

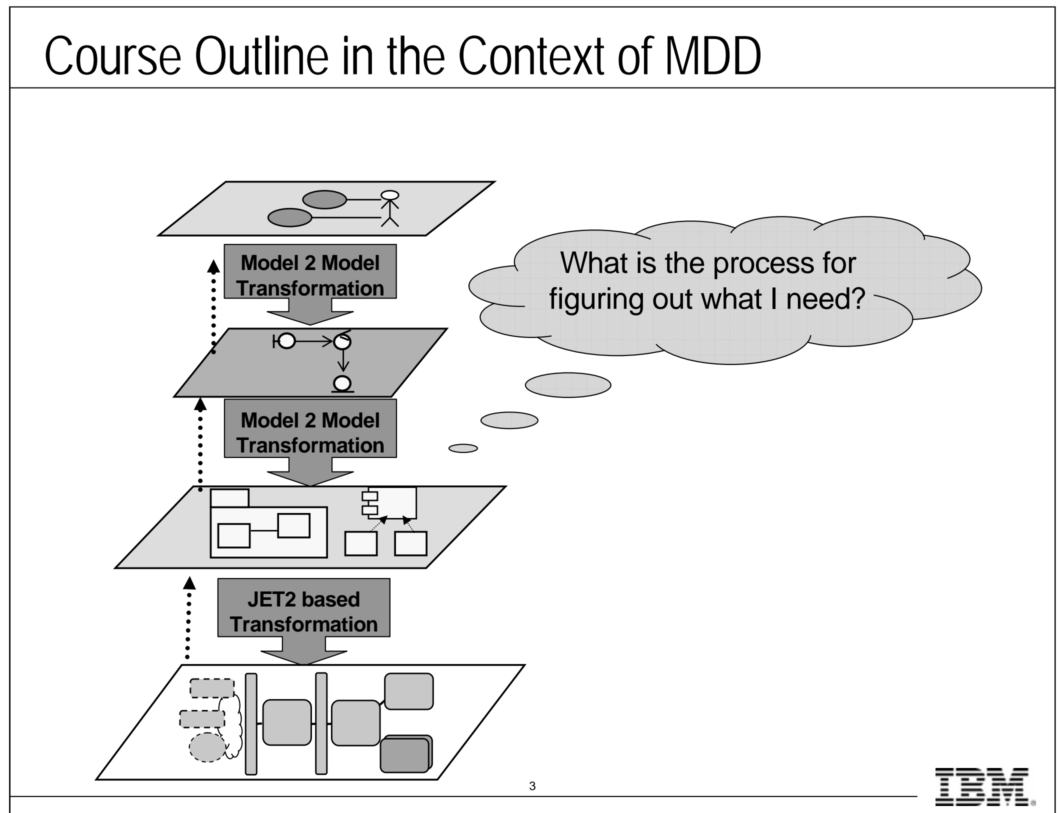
- Objectives:
  - Describe the role of assets in a Model-Driven Development process
  - Describe the component parts of a transformation-based solution
  - Describe the steps in designing an asset in Rational Software Architect

2

IBM

© Copyright IBM Corp. 2007

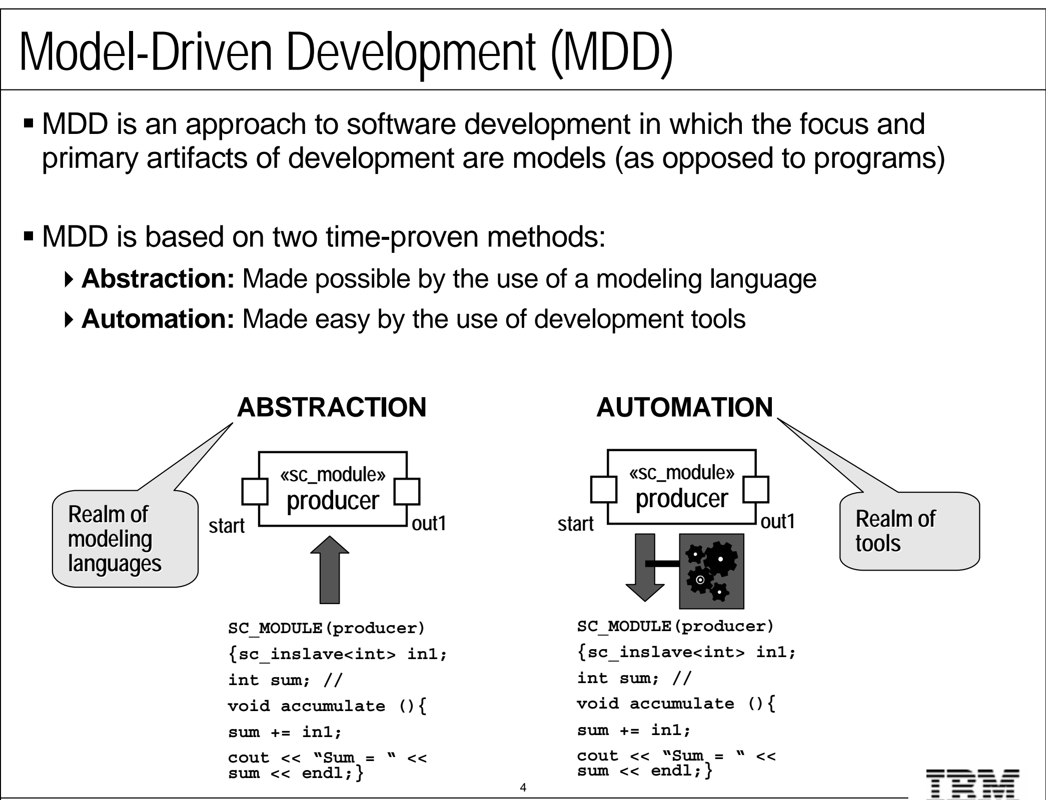
Course materials may not be reproduced in whole or in part without the prior written permission of IBM.



You will see this slide several times throughout the workshop. It will serve as a visual guide to the skills that you are learning, and to how they fit into MDD Model Driven Development.

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

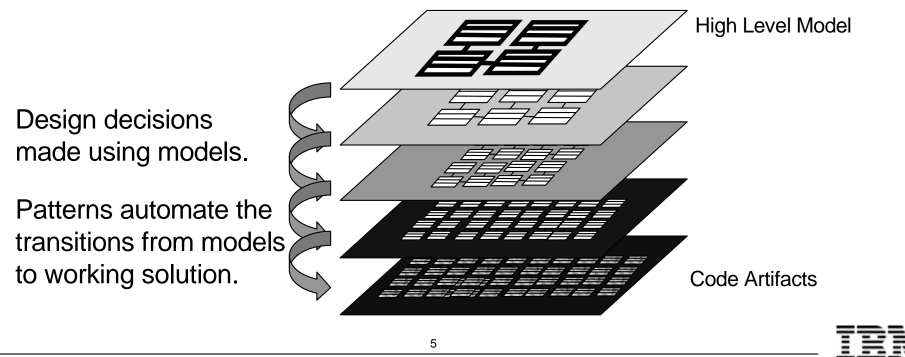


© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Assets in Model-Driven Development

- Rational Software Architect enables MDD by providing modeling and automated model transformation
  - **UML models** used to specify the solution
    - Separate the key information from the details of the target environment or platform
  - **UML Patterns and transformations** handle the details of transforming each input model into a target that is closer to the final artifacts

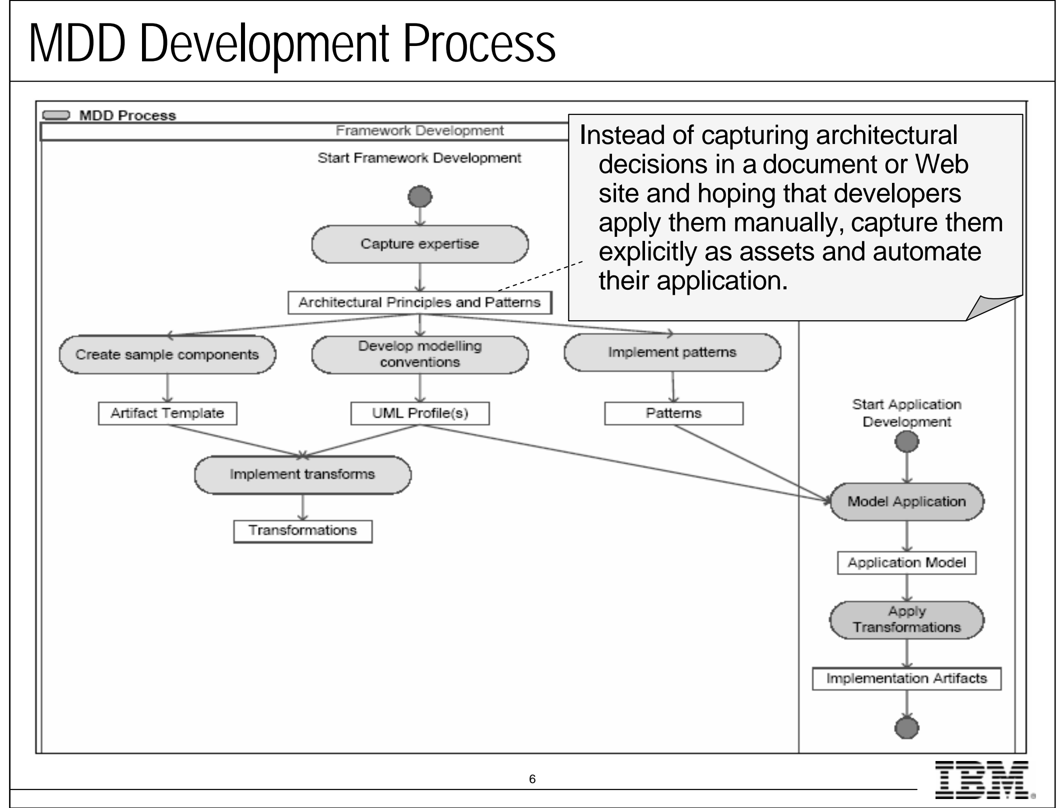


5

IBM

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.



There the two distinct activities in the MDD process:

- **Expertise Capture and Automation:** Build the MDD framework that partially automates the development of software that follows a particular architectural style.
- **Application Development:** Apply your MDD framework to build software components, applications, and solutions. These activities are performed by different groups of people and require different skills. You should use Rational Software Architect to build UML profiles, patterns, and transformations.

People must create modeling conventions and develop transformations to automate code generation. The key dependencies between modeling conventions and transformation development are:

- UML profiles and patterns must be available for application modeling. Sometimes, this dependency is managed in an iterative manner.
- To generate implementation artifacts, transformations must be available. Often, the target platform and the transformations are selected first. In others, this decision is deferred.

8.1.1 Framework development

MDD framework development is concerned with:

- Capturing expertise in the form of architectural principles and patterns
- Implementing sample components and defining the technical architecture
- Designing and implementing UML profiles and Rational Software Architect patterns and transformations

8.1.2 Application development

- Uses an MDD framework to rapidly build well architected applications and components.
- Includes modeling the application using UML and applying transformations.

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Asset Design Process

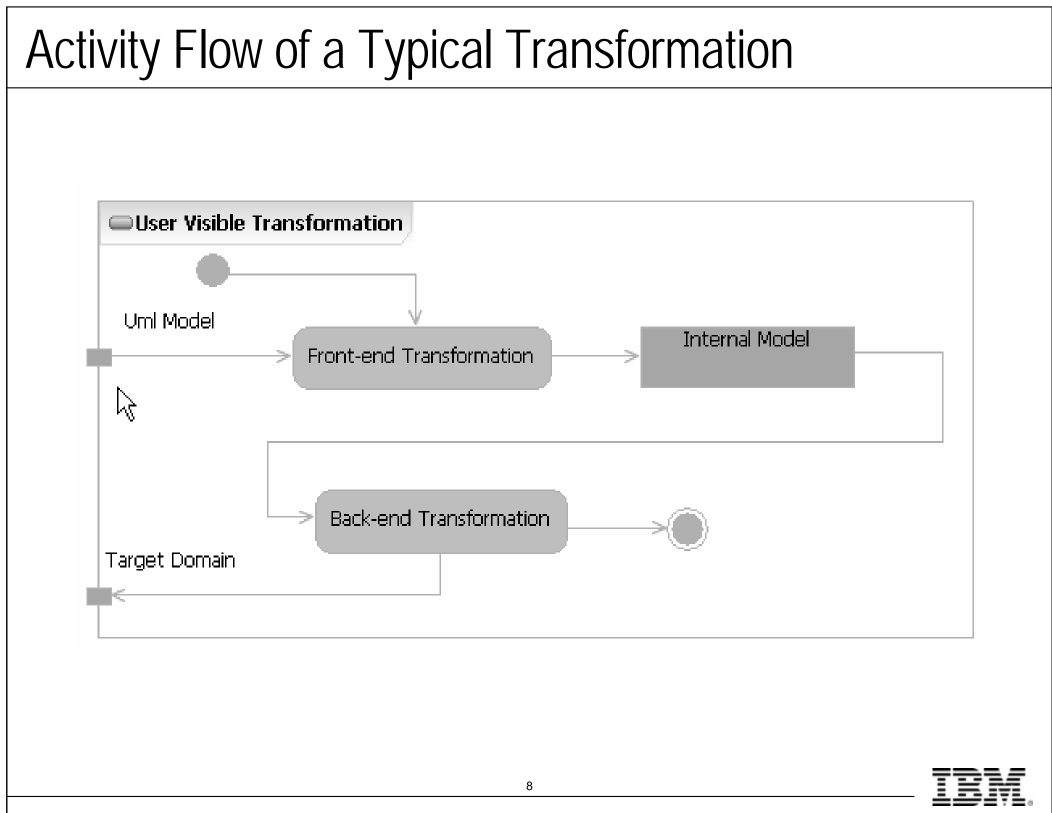
- Transformations drive the process of creating assets with Rational Software Architect extensibility artifacts
- Designing a transformation involves creating:
  - **Internal Model:** An abstraction of the target domain represented as UML
  - **Front-end Transformation:** From representation to abstraction
  - **Back-end Transformation:** From abstraction to target domain
- These elements are invisible to the user
  - The transformations are chained together, and the user runs them with one gesture

7

IBM

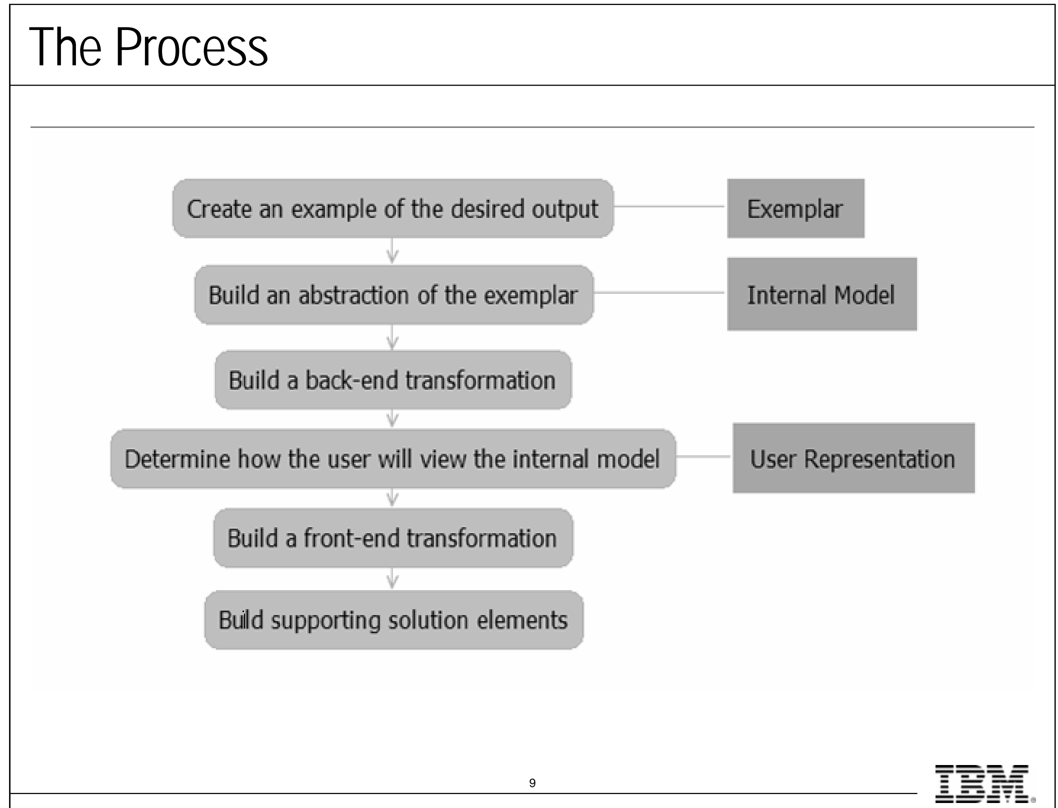
© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.



The top-level activity represents the transformation as seen by the user.

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.



© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Create an Example of the Desired Output

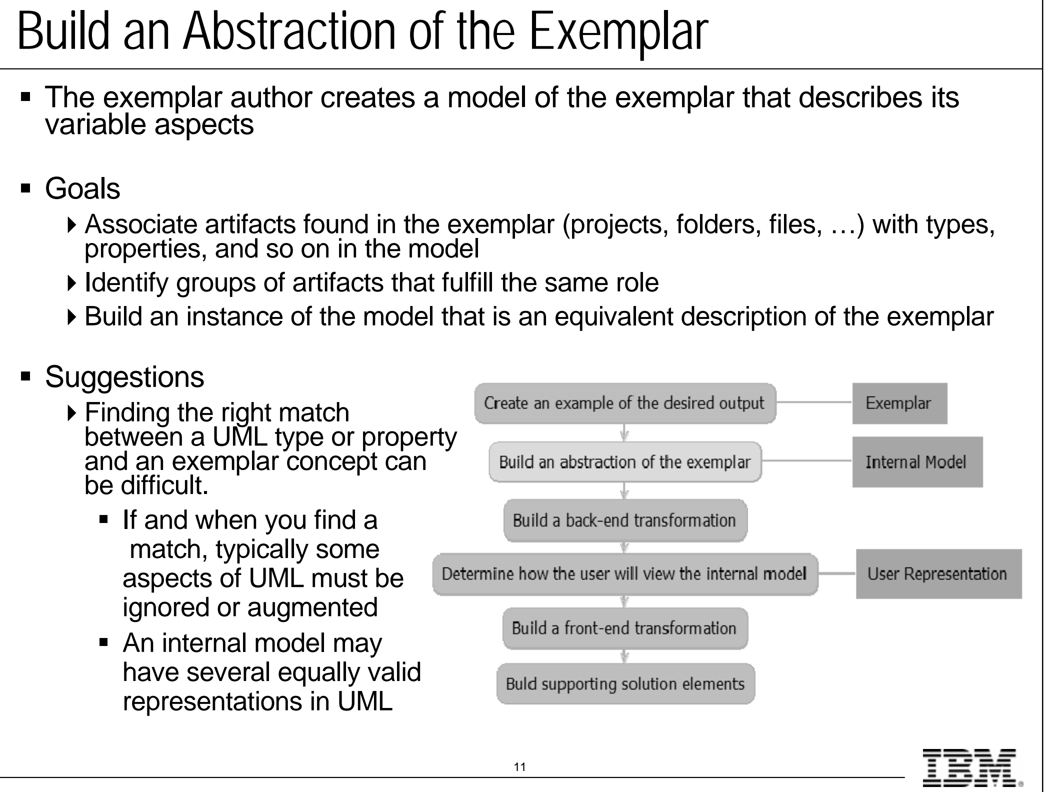
- A domain expert creates an exemplar, which represents the best solution the author can write
- Goals:
  - Constrain the rest of your development to a known problem
  - Provide a model of best practices output to drive creation of the back-end transformation
  - Include as much variability as can be imagined
- Suggestions:
  - Allow enough time for care to be taken in creating an exemplar
  - The exemplar must work; you may include unit tests as part of the exemplar.
  - Expect to make changes to the exemplar during development

Create an example of the desired output — Exemplar
Build an abstraction of the exemplar — Internal Model
Build a back-end transformation
Determine how the user will view the internal model — User Representation
Build a front-end transformation
Buld supporting solution elements

IBM.

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.



The Solution Author is the person who creates the transformation and all the other associated bits. It is important to note that this person may be different from the Exemplar Creator, who is the expert in the transformation's output domain.

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Tool Tips

- Document the Internal Model (input.core) using UML class diagramming concepts
- Represent found artifacts (things in the exemplar) as UML Artifacts, potentially including properties on the UML Artifacts
- Represent the artifact types or roles as UML Classes with a «template» keyword
  - These roles represent transform processes that we will create
- Map artifacts to «template» Classes with Manifestations
  - These mappings define the outputs from the transform processes
- Map «template» Classes to Internal Model types with Dependencies
  - These mappings define the inputs to the transform processes

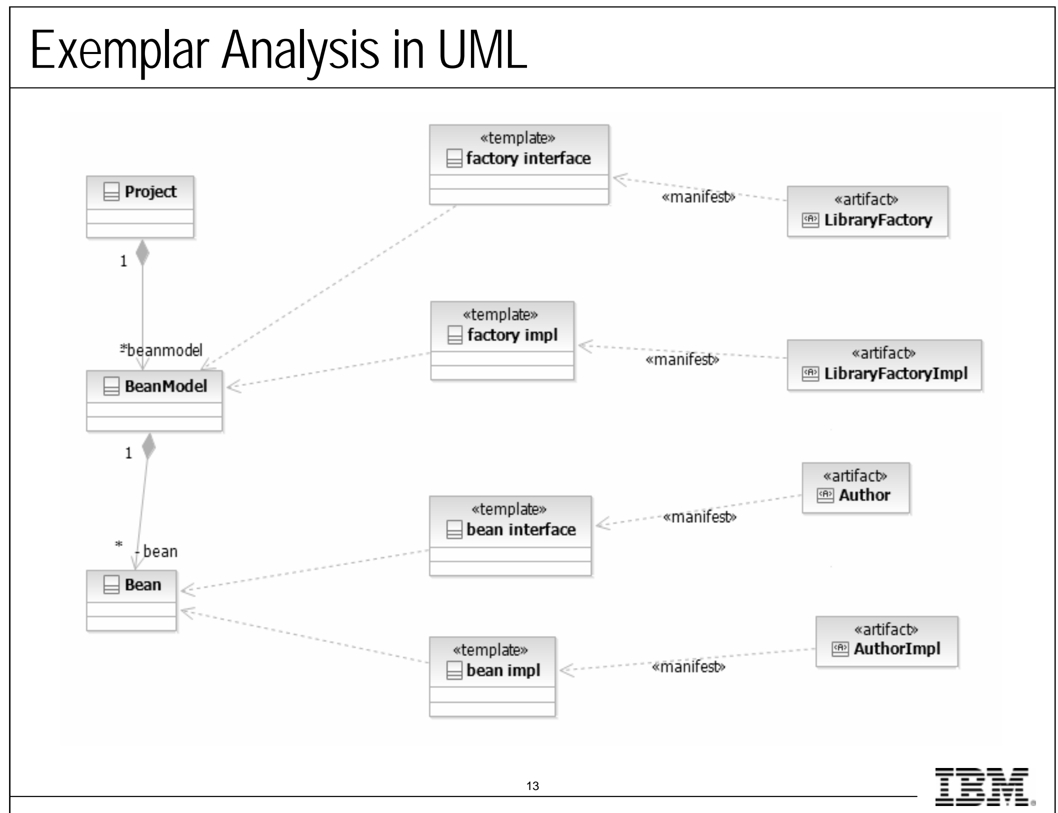
12

IBM.

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Exemplar Analysis in UML



To do an Exemplar Analysis using a UML Model:

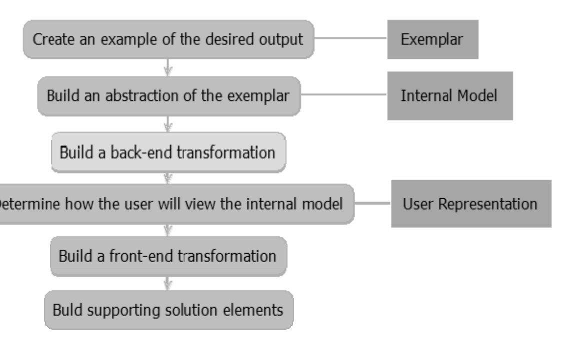
1. Represent files/folders/projects that must be generated as UML Artifacts.
2. Start building an Internal Model to represent an abstraction of these artifacts.
3. Bind artifacts model elements with a “ manifestation”.
4. Once all of the artifacts are associated with a model type, revisit them. Often several artifacts fulfill the same role. Create a Class stereotyped «template» to represent this role, and move the manifestations to point to this «template». Create a dependency from the new «template» to the original model type.

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Build a Back-End Transformation

- The solution author implements the internal model, and builds templates that generate desired artifacts from its objects
- Goals:
  - The back-end transformation traverses the Internal Model, executing the templates and writing artifacts to the workspace
- Suggestions:
  - Internal Model could be implemented as bean-like Java classes, EMF, or XML
  - Templates could be implemented with Velocity or JET2
  - Available enabling tools include JET2 and JMerge



14

IBM

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Options for Deriving Values from the Internal Model

- **Option 1:** Do the calculation in the template itself
  - **Pro:** simple
  - **Con:** duplicate code, pollutes the template with calculation
- **Option 2:** Declare the derived methods in the Internal Model
  - **Pro:** avoids polluting templates with calculations
  - **Con:** pollutes the Internal Model interface
  - The example used this option
- **Option 3:** Derive a secondary model that wraps the Internal Model in the back-end
  - **Pro:** avoids polluting templates and Internal Model interface
  - **Con:** More complex coding

15

IBM

A variation on doing the template calculation is to build a helper class that wraps calculations inside a method. This way, only the template calculation is calling these helper methods.

Who cares about pollution?

A major goal of a template should be to resemble the ultimate output as much as possible. Putting excess calculations in a template generally works against this goal because it can pollute the template.

Internal Model pollution: There are two uses of the Internal Model: templates and model creation. Templates benefit from the addition of derived methods – they need the extra information. On the other hand, model creation code becomes more complex to create if many derived methods are included – the extra methods add to the “weight” of the interfaces.

Choosing a method:

- If only a few derived methods are required, choose option 2.
- If many derived methods are required, choose option 3.

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Kinds of Transformation Output

- There are three kinds of transformation output:
  - **Transformation Owned:** no user modification is allowed
  - **User Modifiable:** transform will continue to write default versions unless the user specifies a custom version
  - **Seeded:** transformation will write this element only once
- In addition, the transformation may encounter elements in the target domain that are none of these. That is, they are user-generated (or generated by another transformation)

Best Practice: Separate *Transformation Owned* output from other elements

16

IBM

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Recognizing Transformation Output

- In order to identify kinds of transformation output, it must be either:
  - Marked with some form of annotation, OR
  - Placed in a specific location that is declared transformation owned
- Examples:
  - The Java compiler owns the **bin** directory, and feels free to overwrite its contents at any time
  - The Rational Software Architect Java transformation uses special Javadoc tags to indicate ownership. (The Java transform has a Re-apply contract stating what it will preserve and what it will overwrite.)

IBM.

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Re-Running a Transformation

- When rerun, a transformation may have to write a file that already exists.

- Establish a reapply contract between the transformation and its users, clearly identifying:
  - Which files the transformation will always overwrite
  - Which files the transformation will never overwrite
  - Which files are shared between transformation and user (and how the sharing works)

18

IBM

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Recommended Transformation Re-apply Actions

	Transform output kind / Re-apply action		
Existing Element	**Transformation Owned Output**	**User Modifiable Output**	**User Seeded Output**
Does not exist in target	Create	Create	Create
Found, has same output kind	Update	Update, if no user modifications	Do not update
Found, different output kind	Error, output may be inconsistent		
Element is in target, but not in output	Remove	Remove, if no user modifications	Do not remove

19

IBM.

This table shows recommended actions. They are not automatically enforced.
There are two approaches to implementing a re-apply strategy:

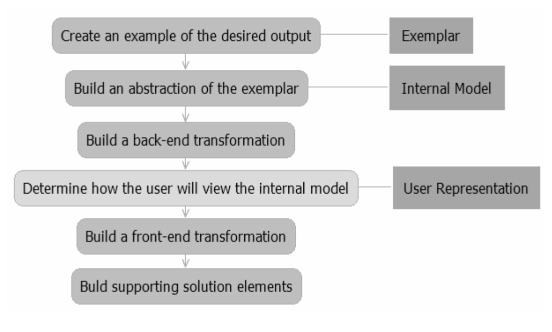
- At each point in the transformation where an element is being created, check to see if it already exists, and perform the appropriate action
  - **Pros:** It is straight forward to implement
  - **Cons:** The re-apply code gets dispersed throughout the transformation, making it harder to correctly modify the re-apply strategy later
  - This works when the re-apply actions are simple.
- The transformation assumes there are no existing elements. Just prior to writing the generated elements, a reconciliation is performed to merge the generated elements with any existing elements.
  - **Pros:** Centralizes re-apply code in a single location; simplifies generation logic; re-apply tooling can be re-used (like JMerge)
  - **Cons:** There are a limited number of tools available: JMerge for Java, but little else. Creating other merge tools is not a trivial activity.

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Determine How User Will View the Internal Model

- Choose an appropriate UML diagramming metaphor
  - Class Diagrams, State Chart Diagrams, Activity Diagrams, and so on
- Create a mapping between Internal Model types and the UML types used in the diagram
- Determine how Internal Model attributes will be calculated from the UML types or attributes
- If necessary, create a UML profile with stereotypes and constraints to represent specializations of UML types
- If UML is not a fit consider EMF\GMF



20

IBM

Note that keywords that are not programmatically applied are prone to failure. As such, if the user is expected to apply this type of differentiator, a profile would be preferable.

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Creating UML Profiles

- Create stereotype properties to represent Internal Model attributes that have no natural UML equivalent
  - UML keywords can be used instead of stereotypes. However , keywords cannot have properties, and entering them is more prone to error
- When a transform requires a profile and stereotypes:
  - Minimize the number of elements to which stereotypes must be applied
  - Use stereotypes to denote non-default characteristics, and lack of a stereotype to imply documented defaults
  - Consider creating template UML models with the profile already applied

21

IBM.

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

UML Profile Tool Tips

- Represent UML meta-types (Class, Property, and so on) as UML classes with a «uml» keyword
- Create association classes between UML meta-types and Internal Model types. The association class is a mapping rule.
- Rules often contain rules
  - Document this containment with UML composition associations.
  - Name the association for the UML collection attribute that is used to navigate from parent source to target source
- Some rules are not bound to the input. Document these as UML classes with an «init-rule» or «final-rule» keyword.

22

IBM

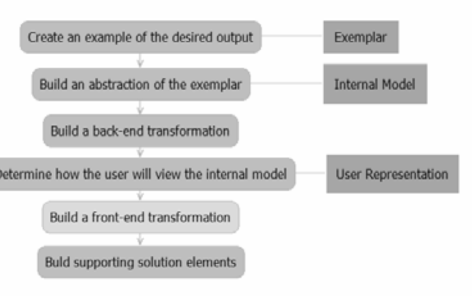
With respect to the «uml» keyword, note that other people and companies have used different keywords for this. The important aspect is to be consistent.

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Build the Font-End Transformation

- Map the User Representation Model to a Rational Software Architect Transformation
  - First choice should be to start with Model Mapping
    - Second choice should be Manual creation
  - Group rules that operate on the same elements
  - Initial cut at the number of mappings needed in the transformation is equal to the number of rule groups
- Other considerations:
  - How many levels of abstraction are needed?
  - If this is a case of meet-in-the-middle, does the input need to be filtered before performing the transformation?



23

IBM

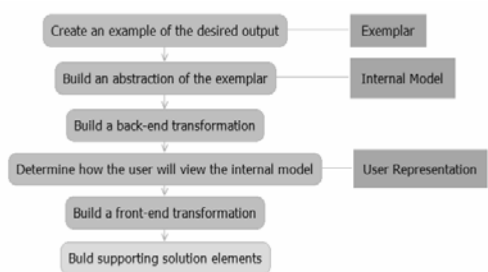
© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Build Supporting Solution Elements

Use Rational Software Architect to support your solution

- UML Template Models, to help guide users to the proper configuration of their UML Model
- Rational Software Architect UML Patterns, to configure the UML model in a more automated way
- Transformation Documentation and Help, important to describe:
  - The processes to follow in creating the source model
  - The transformation contract: elements transformed, outputs, markup needed for the source model
  - The "re-apply" contract: which solution elements to modify in the source model versus the generated output
  - How to configure the transformation



24

IBM

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Summary

- The process for creating transformations:
  - Generally begins with the end result in mind
  - Works backward to establish the form of intermediate and initial input models
  - Can use low-level Eclipse and higher-level Rational Software Architect APIs to manipulate initial and internal models
  - Can use various code generation template technologies to accelerate the creation of code-generating transformations
- The Internal Model is separated from UML Representation
  - Allows evolution of representation without rebuilding the back-end
  - Separates UML knowledge from output domain knowledge

25

IBM

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Review

- Explain why transformations drive the process of Rational Software Architect asset creation.
- Why create a transformation solution composed of an internal model with front-end and back-end transformations?
- Describe the three types of transformation output.

© Copyright IBM Corp. 2007

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.