Run a structural analysis overview

This tutorial teaches you some of the structural analysis code review features. It is written for software architects.

Time required

To simply read through this tutorial you will need approximately **30 minutes**. To do the exercises using the supplied sample project, you will need approximately **1 hour**.

Prerequisites

In order to complete this tutorial, you should be familiar with designing Java software applications. It will also help if you understand how to use the perspectives and views in the IBM Rational Software Development Platform.

Learning objectives

This tutorial is divided into sections that you should take in sequence. You will learn how to perform the following tasks:

- Run a structural analysis code review
- Define an architectural control rule
- Apply a supplied quick fix to resolve a structural problem

When you are ready, begin “Overview of structural analysis.”

Overview of structural analysis

One type of code review is structural analysis, which allows you to perform the following tasks:

- Verify that the structural design for an application is followed during the implementation phase.
- Review dependencies between components in an application.
- Resolve code errors early when problems are easiest and cheapest to fix.

It is important to run structural analysis code reviews throughout the product development life cycle. The following scenario illustrates why in more detail.

It is possible that developers could change the design structure by introducing unplanned dependencies in the code. The software would work, so these changes would not be flagged during the testing phase as defects to fix. However, there could be serious business consequences in the future if you realize that the maintenance cost of the application is very high. As a result, business opportunities could be lost because the code cannot be modified in a reasonable timeframe.

Problems like the ones described in the previous scenario can be avoided if the architect acts proactively and takes the following steps:

- Run a structural analysis code review.
- Detect a problem early.
- Correct the problem before it can affect the application's performance, maintenance, or scalability.

Now you are ready to begin “Introduction to exercises.”

Introduction to exercises

There are four exercises in this tutorial. In the first exercise you import the required sample project called StructuralAnalysis. In the other exercises you perform code reviews. It is best to take the exercises in order.

Exercise 1.1: Importing the required resources

Exercise 1.2: Running a structural analysis code review

Exercise 1.3: Defining an architectural control rule

Exercise 1.4: Resolving a structural problem

Exercise 1.1: Importing the required resources

Before you can begin the exercises, you must first import the required sample project called StructuralAnalysis.

Unzipping the sample project

The sample project for this tutorial is included in a ZIP file. The following steps lead you through extracting files from that ZIP file into your Workspace folder.

1. Navigate to <*installdir*>\rsa\eclipse\plugins\com.ibm.r2a.rsa.tutorial.doc\resources where the ZIP file, StructuralAnalysis, is located.
2. Extract StructuralAnalysis to <*installdir*>\updater\eclipse\workspace. The sample project files are extracted in your Workspace folder so you can import them to do the exercises.

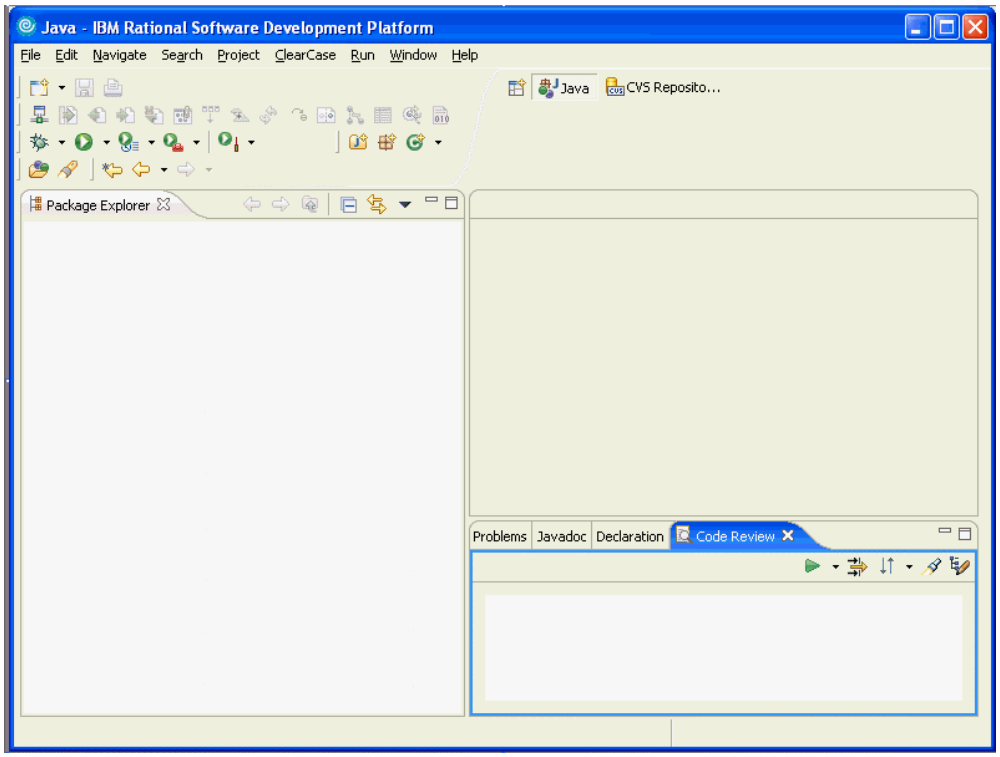
Opening the Code Review view

To open a perspective showing the Code Review view:

1. Start IBM Rational Software Development Platform 6.0.

2. Click **Window > Preferences**.
3. In the left pane expand **Workbench** and click **Capabilities**.
4. In the **Capabilities** list click **Java Developer**. Then click **OK**.
5. Click **Window > Open Perspective > Java**.
6. Click **Window > Show View > Other > Java > Code Review**.
7. Click **Window > Show View > Other > Java > Package Explorer**.

After you open the Java perspective and show the Code Review and Package Explorer views, the perspective shows the views in the following screen capture. Your layout might differ. That is, the perspective might show the views in different locations. The tutorial uses the layout in the screen capture.



Importing the sample project

To import the sample project to the workspace:

1. Right-click in the Package Explorer view to open the pop-up menu. Then click **Import** to open the Import wizard.
2. In the **Select** list click **Existing Project into Workspace**. Then click **Next**.
3. Next to the **Project contents** text box click **Browse** and select <*installdir*>\updater\eclipse\workspace\StructuralAnalysis.
4. Click **Finish**. The sample project and all its associated files are imported to Package Explorer.

You have completed “Exercise 1.1: Importing the required resources.”

Beginning the exercise

To begin click one of the following exercises:

Exercise 1.2: Running a structural analysis code review

Exercise 1.3: Defining an architectural control rule

Exercise 1.4: Resolving a structural problem

Exercise 1.2: Running a structural analysis code review

This exercise assumes you have completed “Exercise 1.1: Importing the required resources.” In Exercise 1.2 you read a user scenario first. Then you assume the role of the software architect described in the user scenario and use the project you imported in Exercise 1.1 to complete the exercise.

User scenario

To review newly written code to assess its quality, the architect wants to look for general anti-patterns. *Anti-patterns* are known problems that occur in code and do not follow best practices. While design patterns are good models to follow, anti-patterns are bad models that you should avoid. Some specific anti-patterns include the following types:

> Breakable: The object in the code has so many dependencies that it is likely to break when another object is changed.
>
> Cyclic dependency: A group of objects is so interconnected, often circular, that a change to any object could affect all of the others. Also referred to as a tangle.
>
> Hub: The object has both many dependencies and many dependents. It is affected when another object is changed. Likewise, when it is changed other objects are affected.

In the first exercise, the architect runs a code review to look for the anti-patterns described above.

Exercise

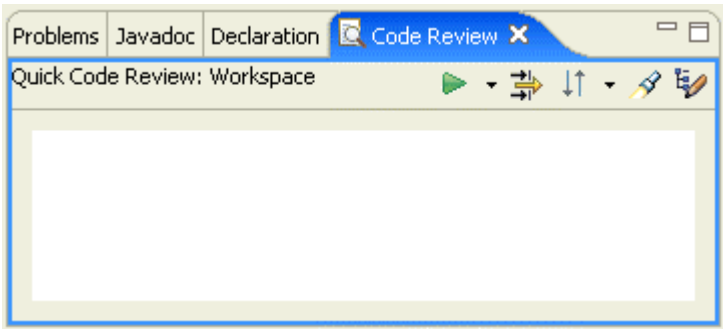
In this exercise you perform the following tasks:

1. Select a code review to run.
2. View the rules applied in the code review.
3. Choose what code to run the review on.
4. Run the code review.
5. View the findings of the code review.
6. Select a finding to see the following information for it:
    - Source code.
    - Description, examples, and solutions.

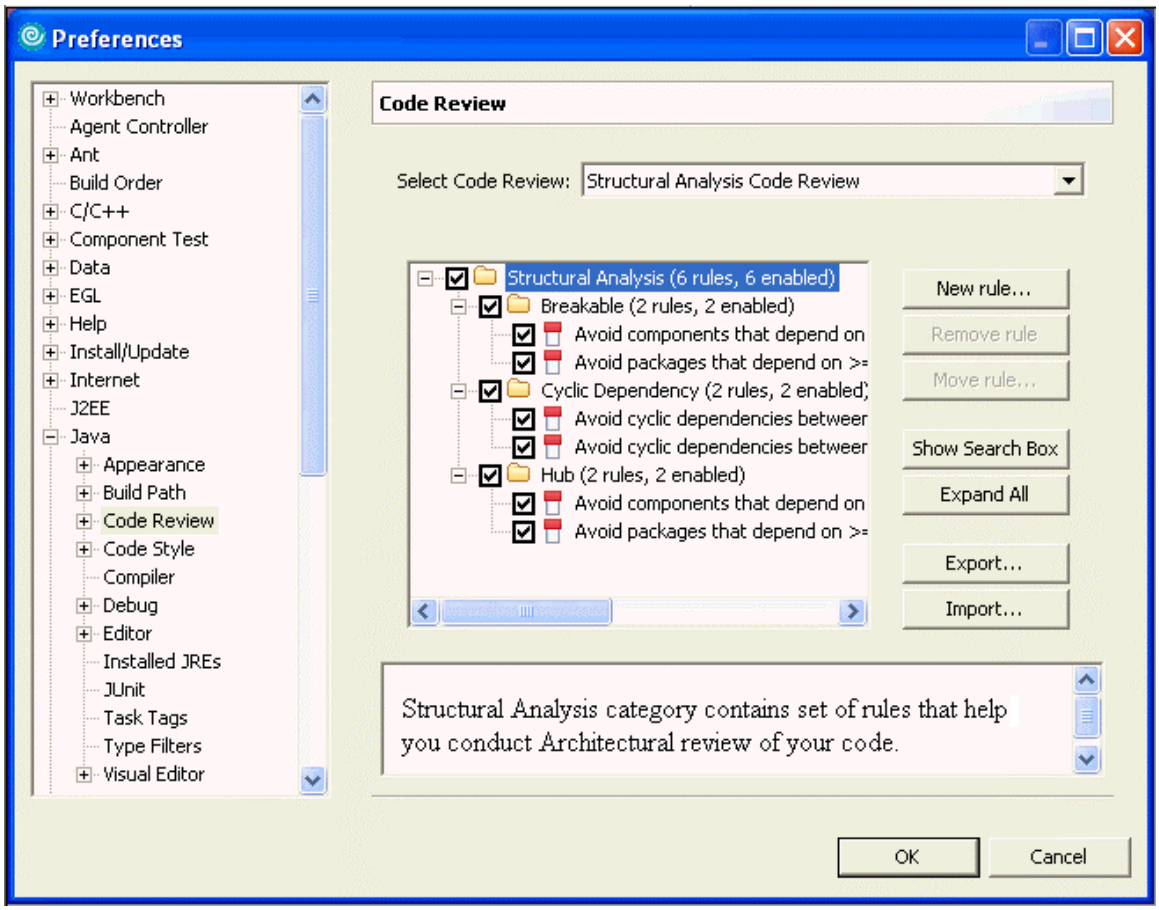
Selecting a code review

To select a structural analysis code review:

1. On the toolbar in the Code Review view click the **Manage Rules** icon,  .



2. In the **Select Code Review** list, click **Structural Analysis Code Review**.
3. Expand the **Structural Analysis** folder and subfolders to see the rules applied in the code review, as shown below. Click **OK**.



Selecting a code base to review

To select the workspace as the code base to review:

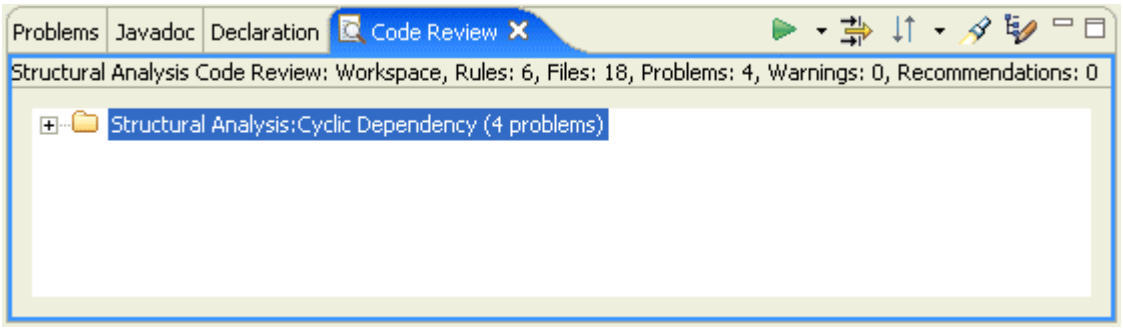
> On the toolbar in the Code Review view click the **Review** icon ( ) **> Review Workspace**.

Running the code review

Once you select the code base to review, the code review runs. You can track its status by checking the progress bar in the lower-right corner of the view.

Viewing the code review findings

When the code review is finished, the findings are shown in the Code Review view, as shown in the following screen capture:



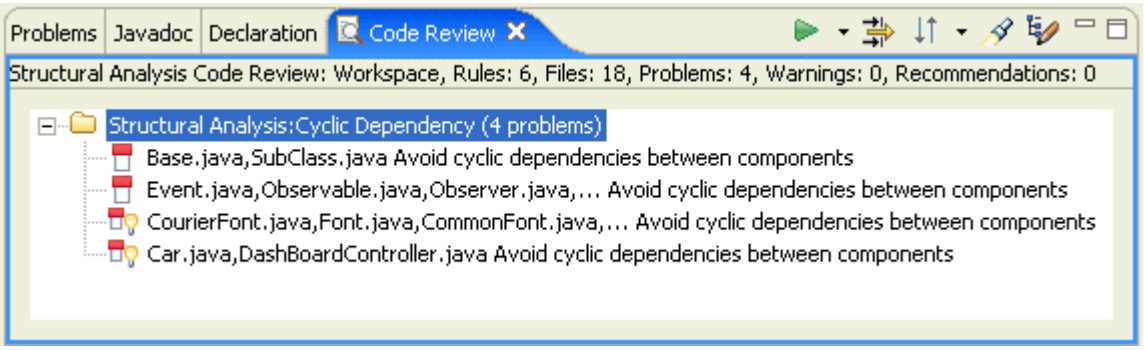
The following information is provided in the Code Review view.

> Code review statistics: The line above the findings displays information about the most recent code review: name, scope, number of rules and files included, and number and severity of findings.
>
> Code review findings: The findings in the code review are listed in the Code Review view, within folders. Each folder name tells you the code review name and the category and number of findings.

Getting more information on a code review finding

To get more information on a finding in the code review:

1. Expand the **Structural Analysis: Cyclic Dependency** folder. It contains four findings, as shown in the following screen capture:



   Each finding is preceeded by an icon that indicates its severity level.

Icon	Severity Level
	Problem
	Warning
	Recommendation

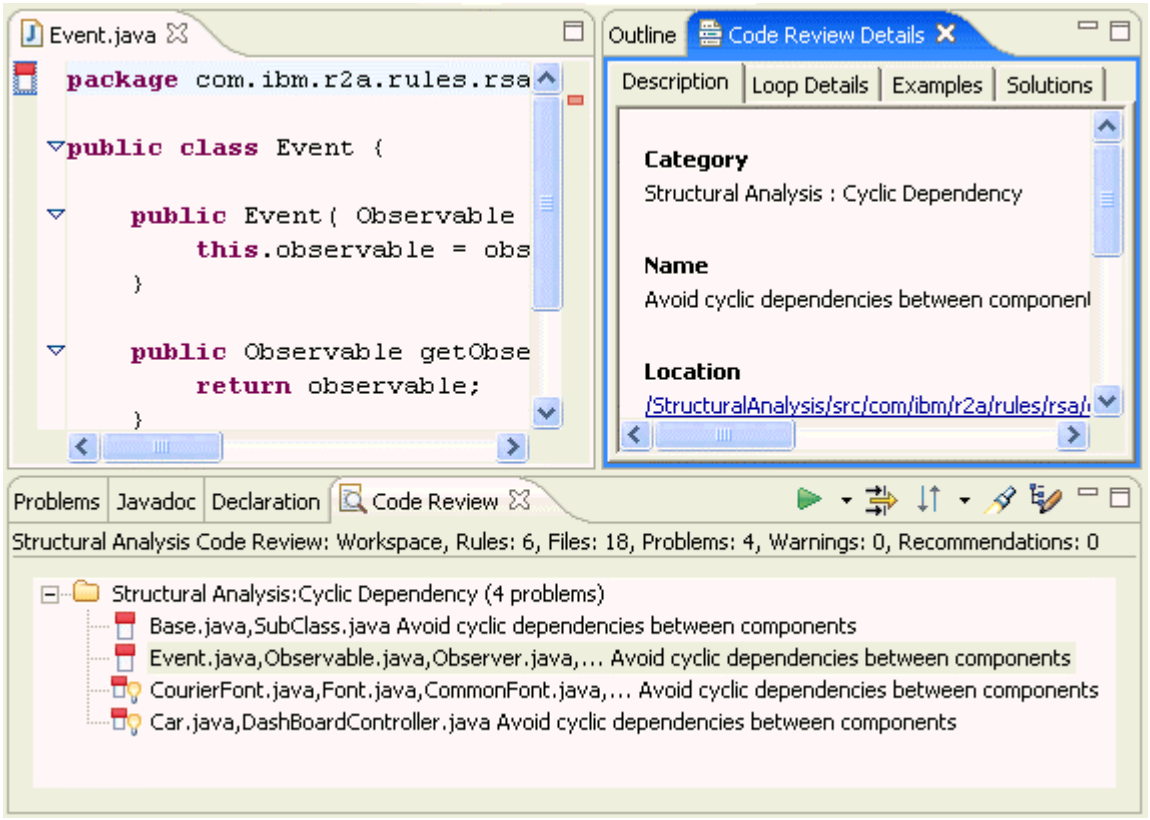
   If an icon has a lightbulb next to it ( ), that means that a quick fix exists for the finding. A quick fix is an automated, supplied solution for a specific finding. The quick fix icons are shown in the following illustration:

Quick Fix Available

2. Double-click the finding that begins with Event.java. Details about it appear in two places, as outlined in the following points and screen capture:

   - Source code: Displays the code where the finding occurs and highlights the exact location of it.

   - Code Review Details view: Describes the finding in more detail and provides examples and solutions to correct it. If the finding is a cyclic dependency, there is also a section on loop details.



You have completed Exercise 1.2: Running a structural analysis code review.

Leveraging the power of a code review

By proactively running a code review you are able to spot problems early so you can also correct them early, before they cause the following problems:

- Affect your application's performance, maintenance, or scalability.
- Cost your company money, time, and resources.

In Exercise 1.4, you build on the work you did in Exercise 1.2 by fixing a code review finding.

Wrapping up Exercise 1.2

You have completed Exercise 1.2: Running a structural analysis code review. In it you performed the following tasks:

1. Selected a code review to run.
2. Viewed rules applied in the code review.
3. Chose a body of code to run the review on.
4. Ran the code review.
5. Viewed the findings of the code review.
6. Selected a finding to see the following information for it:
    - Source code.
    - Description, examples, and solutions.

Now you are ready to begin “Exercise 1.3: Defining an architectural control rule.”

Exercise 1.3: Defining an architectural control rule

This exercise assumes you have completed “Exercise 1.1: Importing the required resources.” In Exercise 1.3 you read a user scenario first. Then you assume the role of the software architect described in the user scenario and use the project you imported in Exercise 1.1 to complete the exercise.

User scenario

To prevent extraneous dependencies, the architect wants to put a safeguard in place to ensure that none are introduced into the application. To do this, he creates a rule from a supplied wizard. The rule is to alert him if the utility package becomes dependent on the application package.

After creating the rule, the architect runs a code review by applying it to a code base. The findings will show any extraneous dependencies in the application.

Exercise

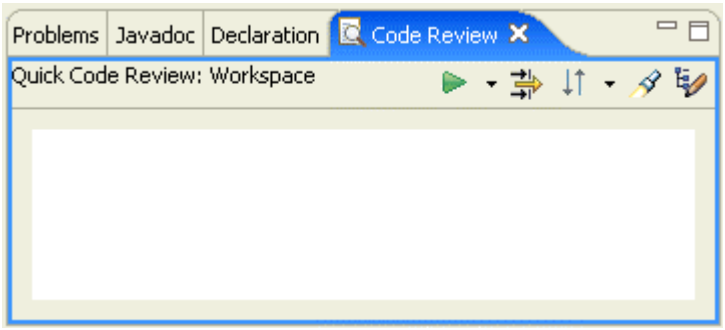
In this exercise you perform the following tasks:

1. Define a rule based on a supplied wizard.
2. Verify that your user-defined rule is added to the Structural Analysis code review.
3. Run the Structural Analysis code review.
4. View the code review findings that do not adhere to the criteria of your rule.

Defining a rule

To define an architectural control rule based on a supplied wizard:

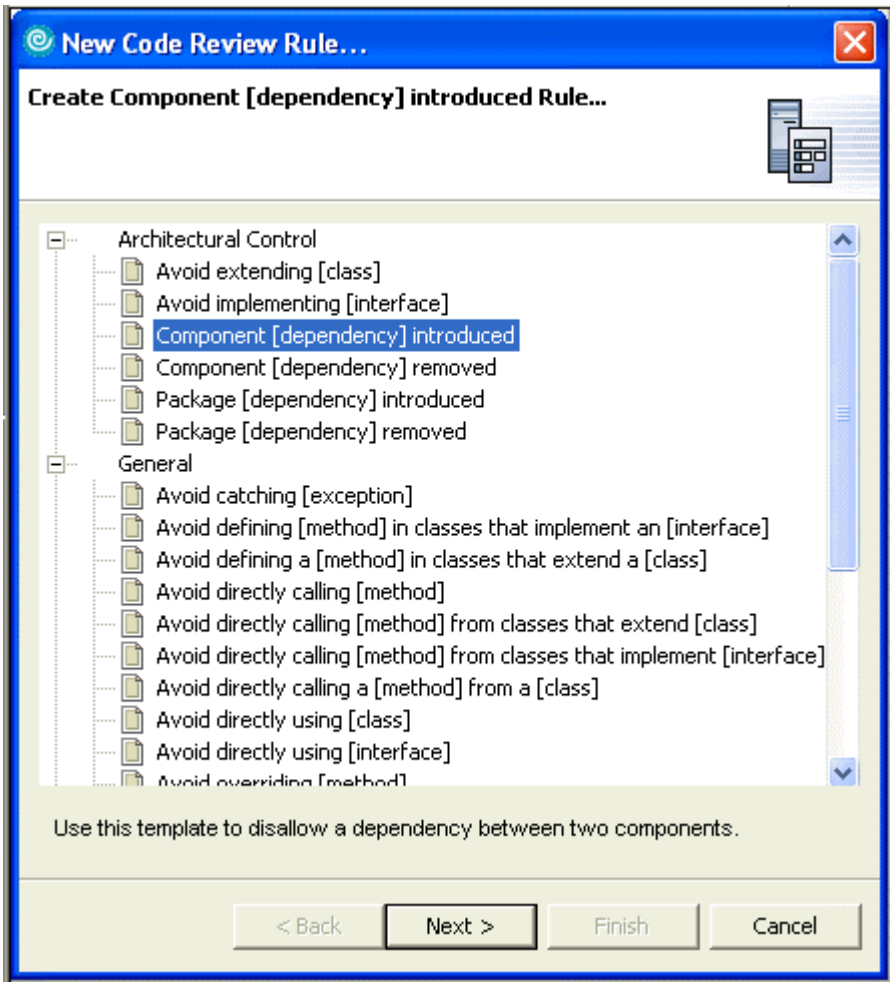
1. On the toolbar in the Code Review view click the **Manage Rules** icon,  .



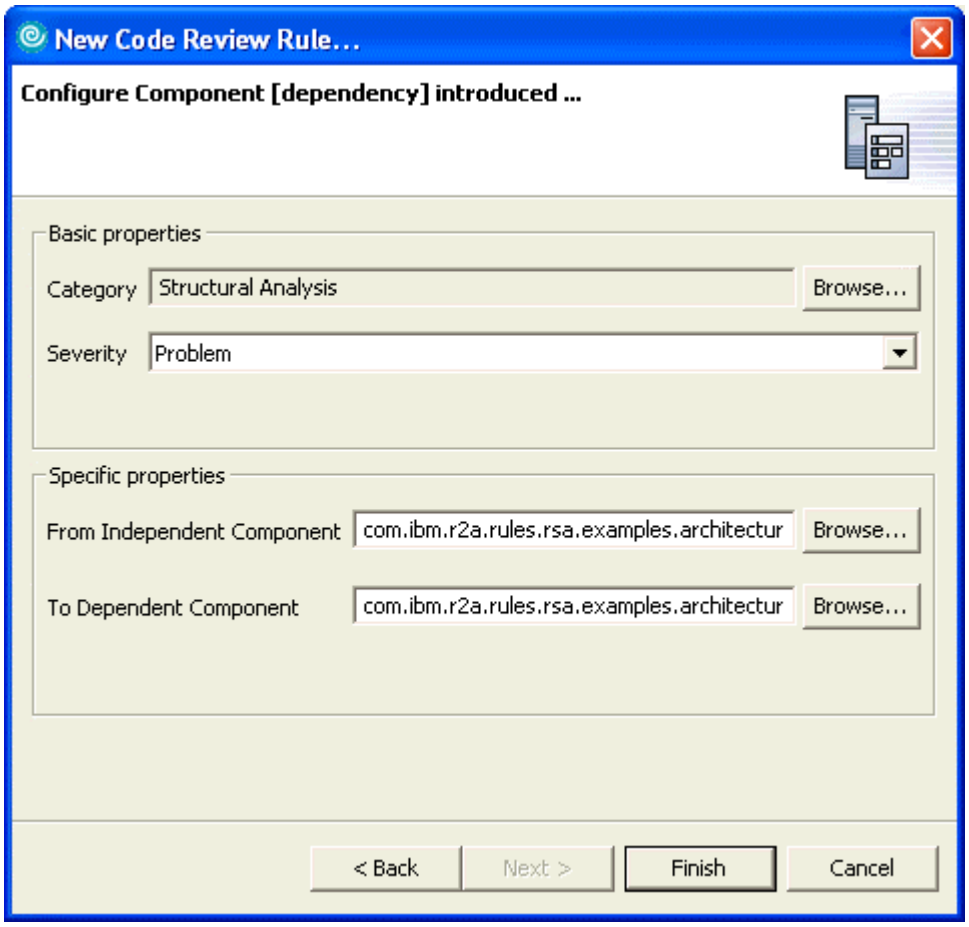
2. In the Preferences window click **New rule**. The New Code Review Rule wizard opens.

   The New Code Review Rule wizard, as shown in the next screen capture, takes you through a few steps to define your own rule. In this exercise, you will design an architectural control rule to alert you if a dependency between two components is introduced.

3. In the Architectural Control list of rules, click **Component [dependency] introduced**. Notice that when you select this choice, the text below the list tells you to use this template to disallow a dependency between two components. Click **Next**.



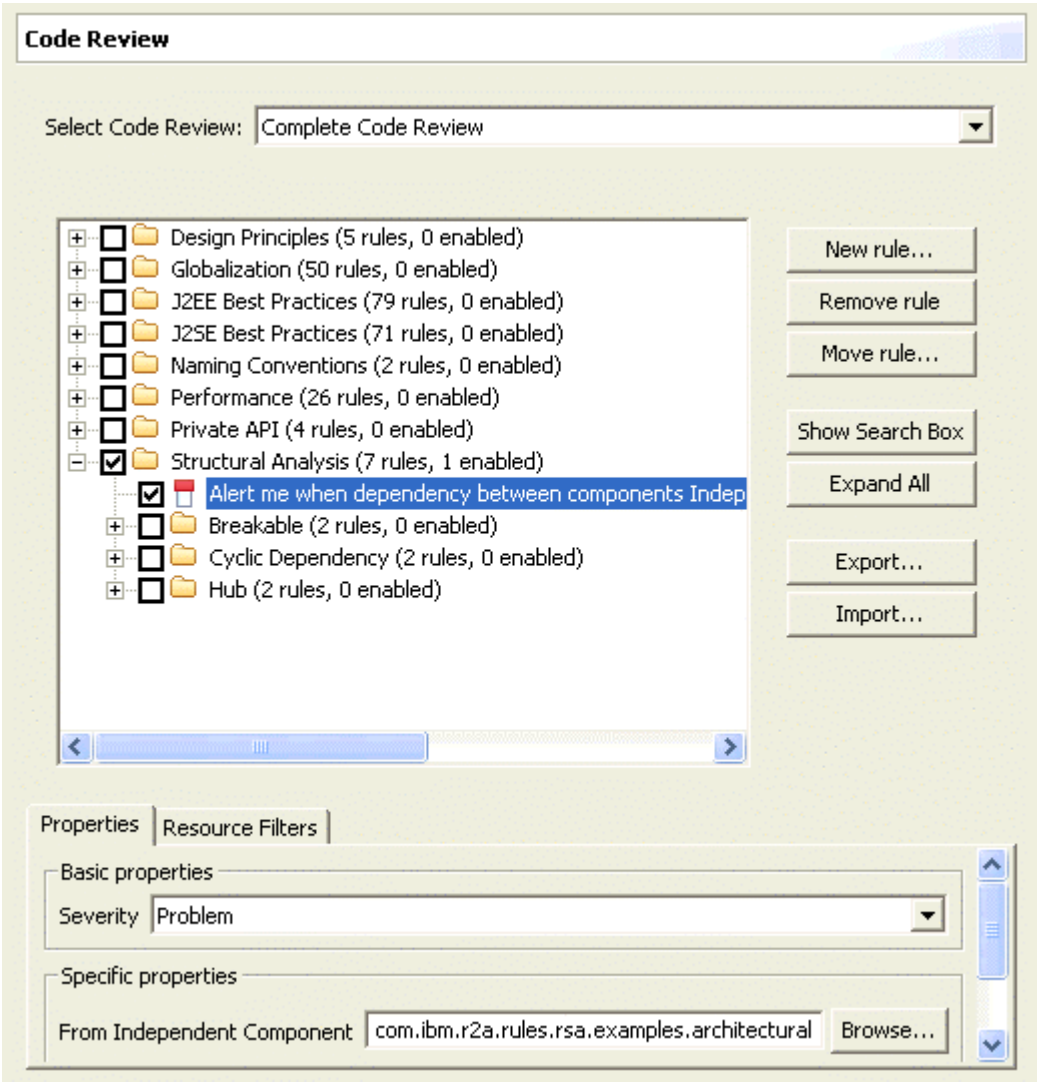
Under **Basic properties**, accept the defaults that put the rule in the structural analysis category, with a problem severity level.



4. Under **Specific properties**, specify the independent and dependent components for the rule:
   - Independent Component: workspace\StructuralAnalysis\src\com.ibm.r2a.rules.rsa.examples.architecturalcontrol\IndependentComponent.java
   - Dependent Component: workspace\StructuralAnalysis\src\com.ibm.r2a.rules.rsa.examples.architecturalcontrol\DependentComponent.java.
5. Click **Finish**.

Seeing your rule added to a code review

1. After you define a rule, the **Preferences** window appears. In the **Select Code Review** list, click **Complete Code Review** if it is not already the selected code review. This shows all code review categories.
2. Expand the **Structural Analysis** folder to see the rule you just created.
3. Click the rule to see the properties you set for it, as shown in the following screen capture:



Selecting a code review that applies your rule only

To run a code review for your rule only, clear all folders in the list except for the **Structural Analysis** folder, as shown in the previous screen capture. Click **OK**.

Selecting a code base to review

To select the workspace as the code base to review:

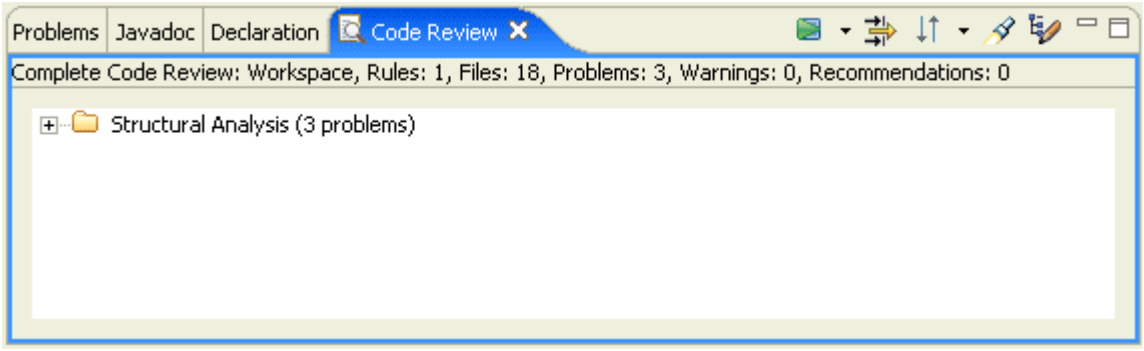
> On the toolbar in the Code Review view click the **Review** icon ( ) **> Review Workspace**.

Running the code review

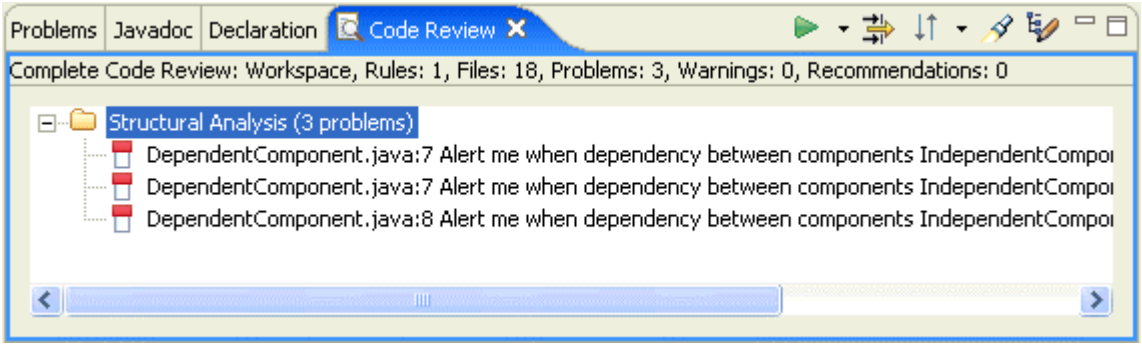
Once you select the code base to review, the code review runs. You can track its status by checking the progress bar in the lower-right corner of the view.

Viewing the code review findings

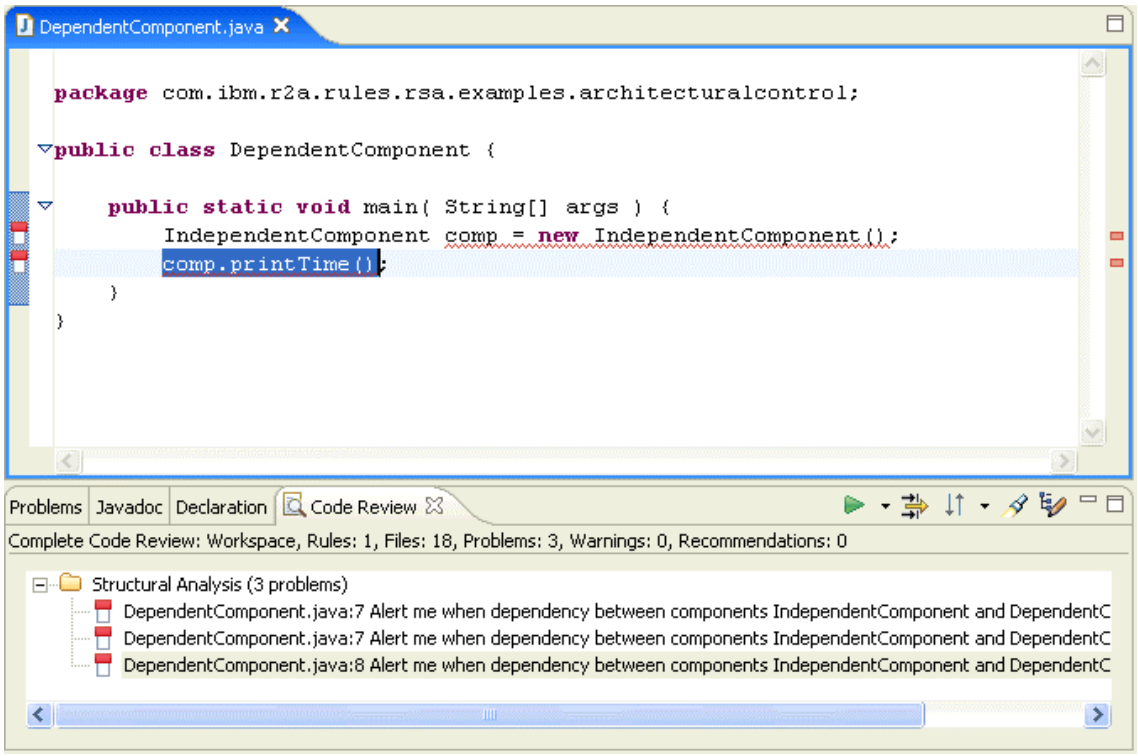
When the code review is finished, the findings are shown in the Code Review view, as shown in the following screen capture. Below the tab the statistics line summarizes the review.



1. Expand the **Structural Analysis** folder to see the findings in it.



2. Double-click the third finding to see the source code for it in the editor, as shown in the following screen capture:



You have completed “Exercise 1.3: Defining an architectural control rule.”

Leveraging the power of user-defined rules

By creating your own rules, you put custom safeguards in place to monitor the implementation of your design. You can take the following measures:

Specify criteria for a rule.

Assign a severity level to the rule: problem, warning, or recommendation.

Run a code review on your rule or rules only.

Wrapping up Exercise 1.3

You have completed Exercise 2: Defining an architectural control rule. In it you performed the following tasks:

1. Defined a rule based on a supplied wizard.
2. Verified that your user-defined rule was added to the Structural Analysis code review.
3. Ran the Structural Analysis code review.
4. Viewed the code review findings that do not adhere to the criteria of your rule.

Now you are ready to begin “Exercise 1.4: Resolving a structural problem.”

Exercise 1.4: Resolving a structural problem

This exercise assumes you have completed “Exercise 1.1: Importing the required resources.” In Exercise 1.4 you read a user scenario first. Then you assume the role of the software architect described in the user scenario and use the project you imported in Exercise 1.1 to complete the exercise.

User scenario

To check code specifically for cyclic dependencies, the architect runs a structural analysis code review that looks only for such dependencies. The code review does find some cyclic dependencies and the architect notices that quick fixes exist for a couple of them. A quick fix is a supplied automated way to repair a common finding. To refactor the code to be free of cycles, the architect applies the quick fix to one cyclic dependency.

In the final exercise, the architect runs a code review and fixes one of the findings.

Exercise

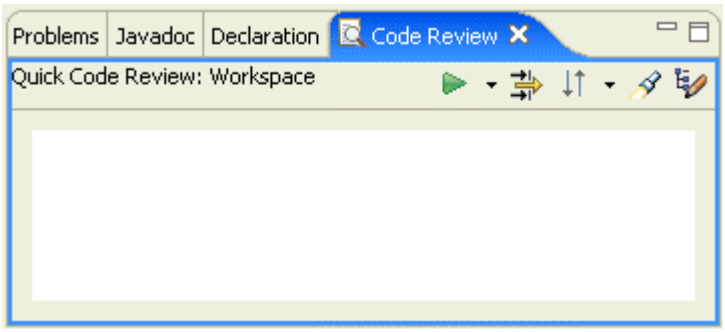
In this exercise you perform the following tasks:

1. Run a code review to find cyclic dependencies.
2. Recognize when a cyclic dependency has a supplied quick fix.
3. Apply a quick fix to resolve a cyclic dependency:
    - See a list of changes to be made.
    - View the existing and refactored code for each change.
4. Get a confirmation that the quick fix has been applied.

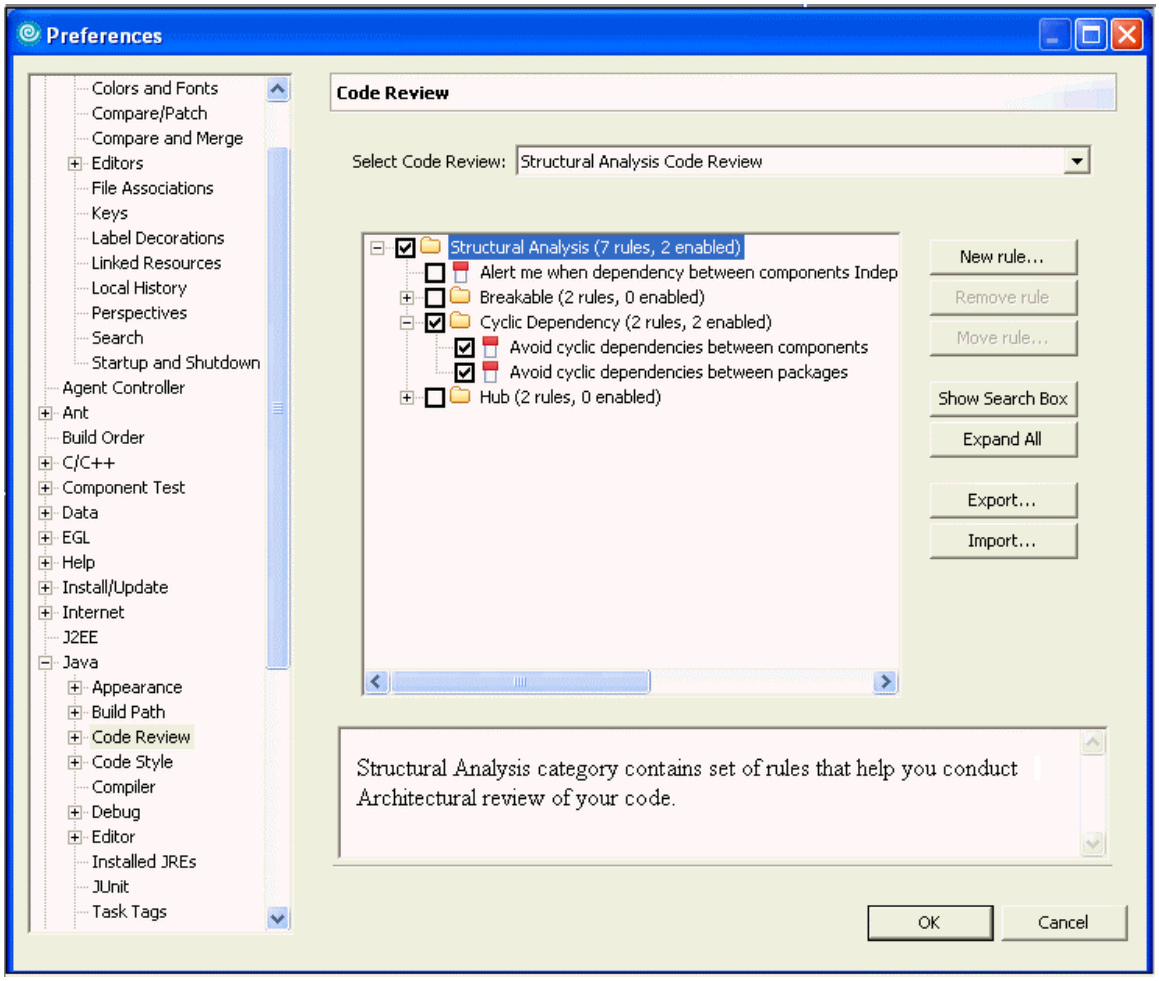
Selecting a code review to check for cyclic dependencies

To select a code review that checks for cyclic dependencies:

1. On the toolbar in the Code Review view click the **Manage Rules** icon,  .



2. In the **Select Code Review** list, click **Structural Analysis Code Review**.
3. Expand the **Structural Analysis** folder and clear everything but the **Cyclic Dependency** subfolder.
4. Expand the **Cyclic Dependency** subfolder to see the rules that will be applied in the code review, as shown in the following screen capture. Click **OK**.



Selecting a code base to review

To select the workspace as the code base to review:

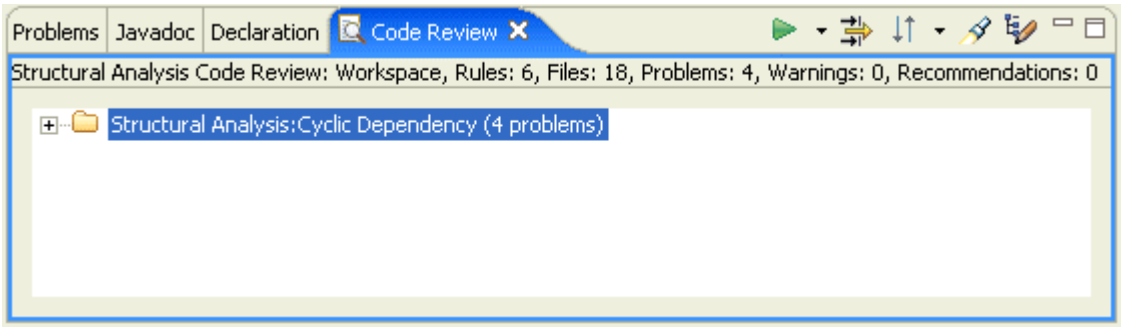
> On the toolbar in the Code Review view click the **Review** icon ( ) >**Review Workspace**.

Running the code review

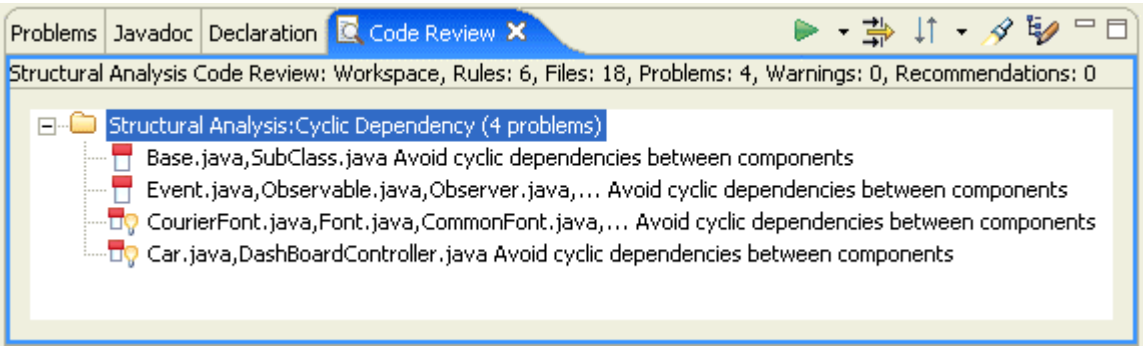
Once you select the code base to review, the code review runs. You can track its status by checking the progress bar in the lower-right corner of the view.

Viewing the list of cyclic dependencies found

The code review found four cyclic dependencies, as shown in the next screen capture. A cyclic dependency is an undesirable anti-pattern that should be avoided. In a cyclic dependency, a group of objects is so interconnected that a change to any object could affect all of the others.



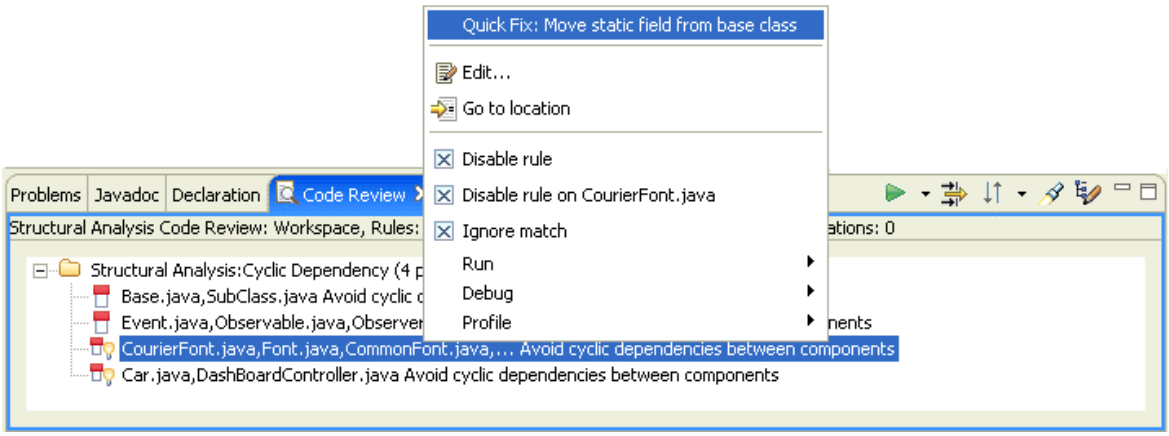
1. Expand the **Structural Analysis: Cyclic Dependency** folder and note the following findings in it, as shown in the next screen capture:



   - Each cyclic dependency has the highest severity level of problem ( ) assigned to it.
   - Two of the cyclic dependencies have a quick fix ( ) to eliminate the dependency.

2. Right-click the third finding in the list. The **Quick Fix** pop-up menu choice varies depending upon the solution. For the cyclic dependency you selected, the fix is to move the static field from the base class to another class, thereby eliminating the cyclic dependency.

3. Click **Quick Fix: Move static field from base class**.

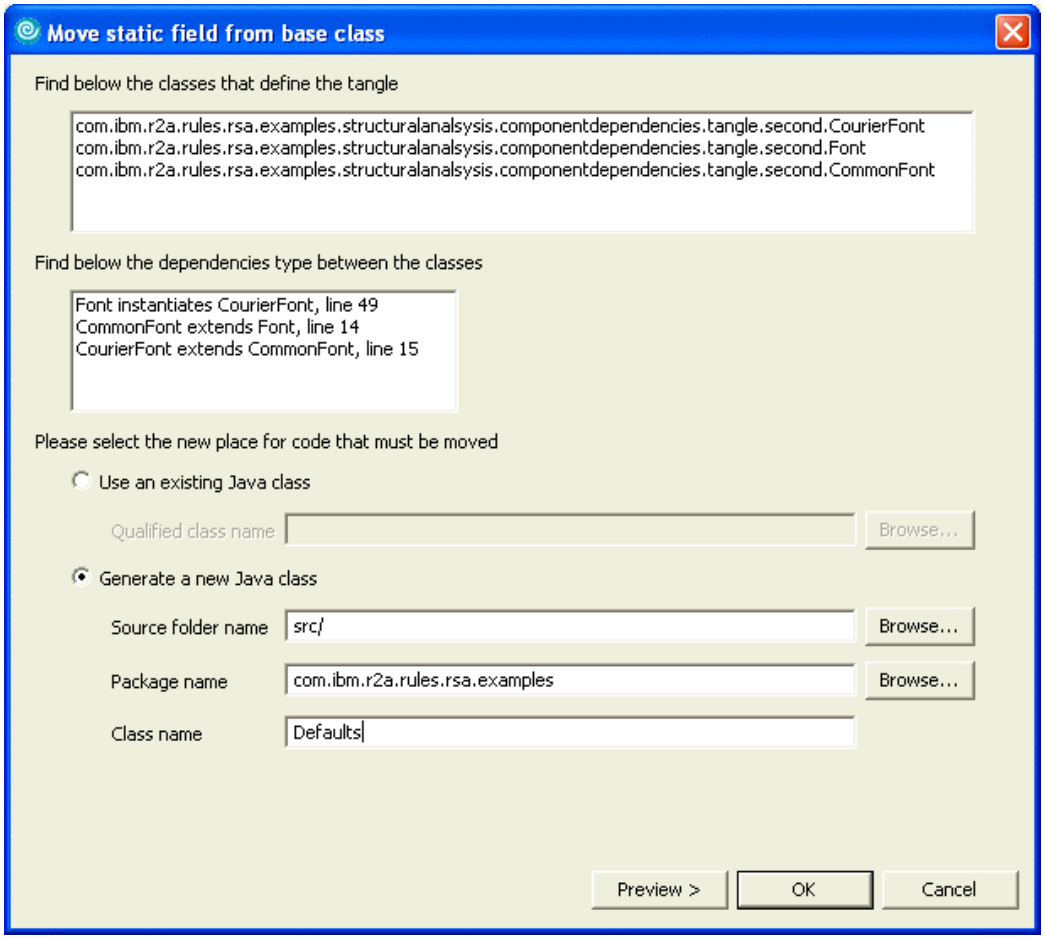


Applying the quick fix

The quick fix for the cyclical dependency you selected is to move the static field from the base class to another class. You can choose to move the field to an existing class or to a new class.

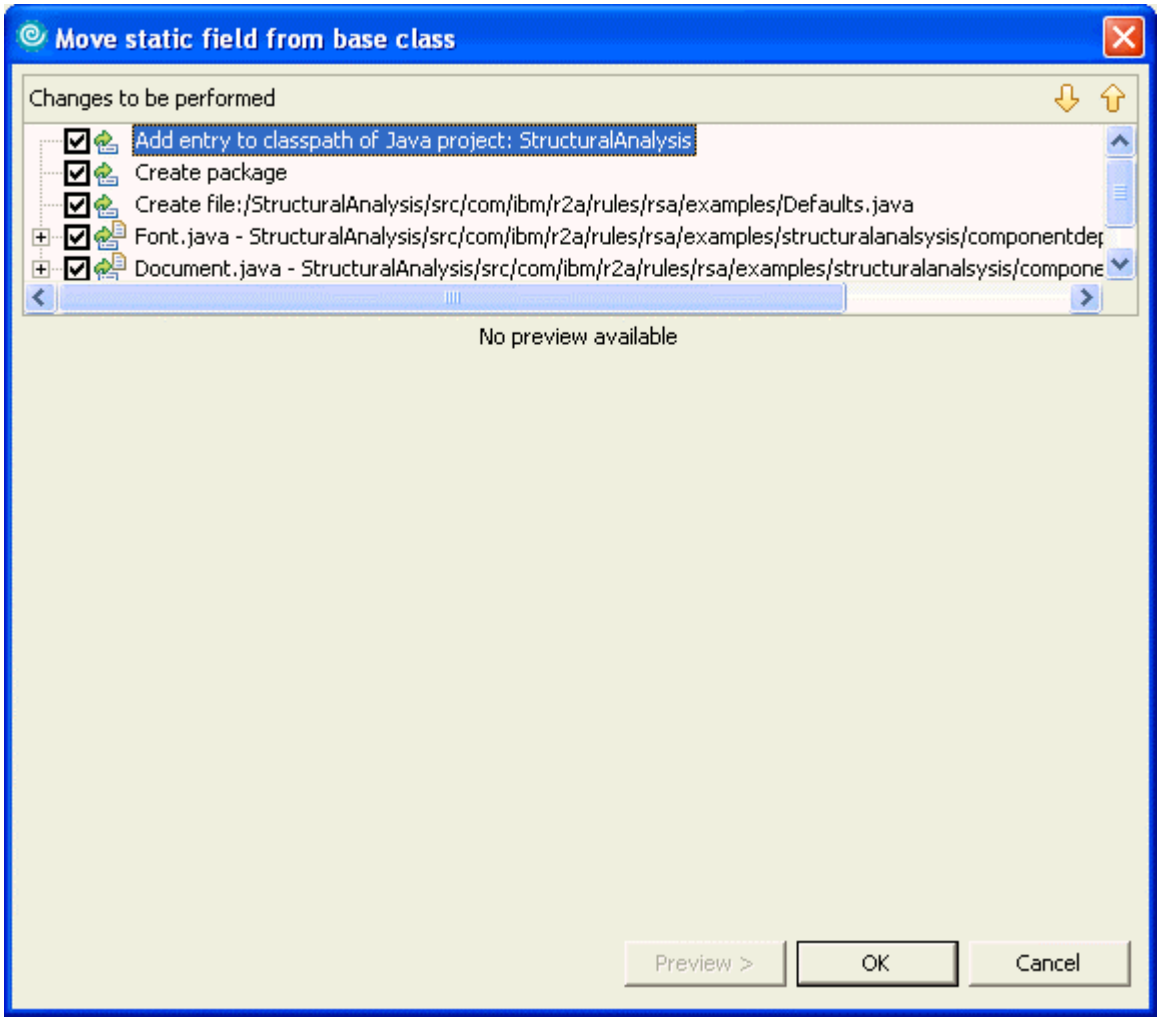
To move the field to a new class:

1. In the next screen capture, review the read-only information about the cyclic dependency, also referred to as a tangle, that you are going to fix:
    - Classes that are part of the tangle
    - Dependencies between classes in the tangle
2. Click **Generate a new Java class** and type:
    - `src/` for the source folder name
    - `com.ibm.r2a.rules.rsa.examples` for the package name
    - `Defaults` for the class name

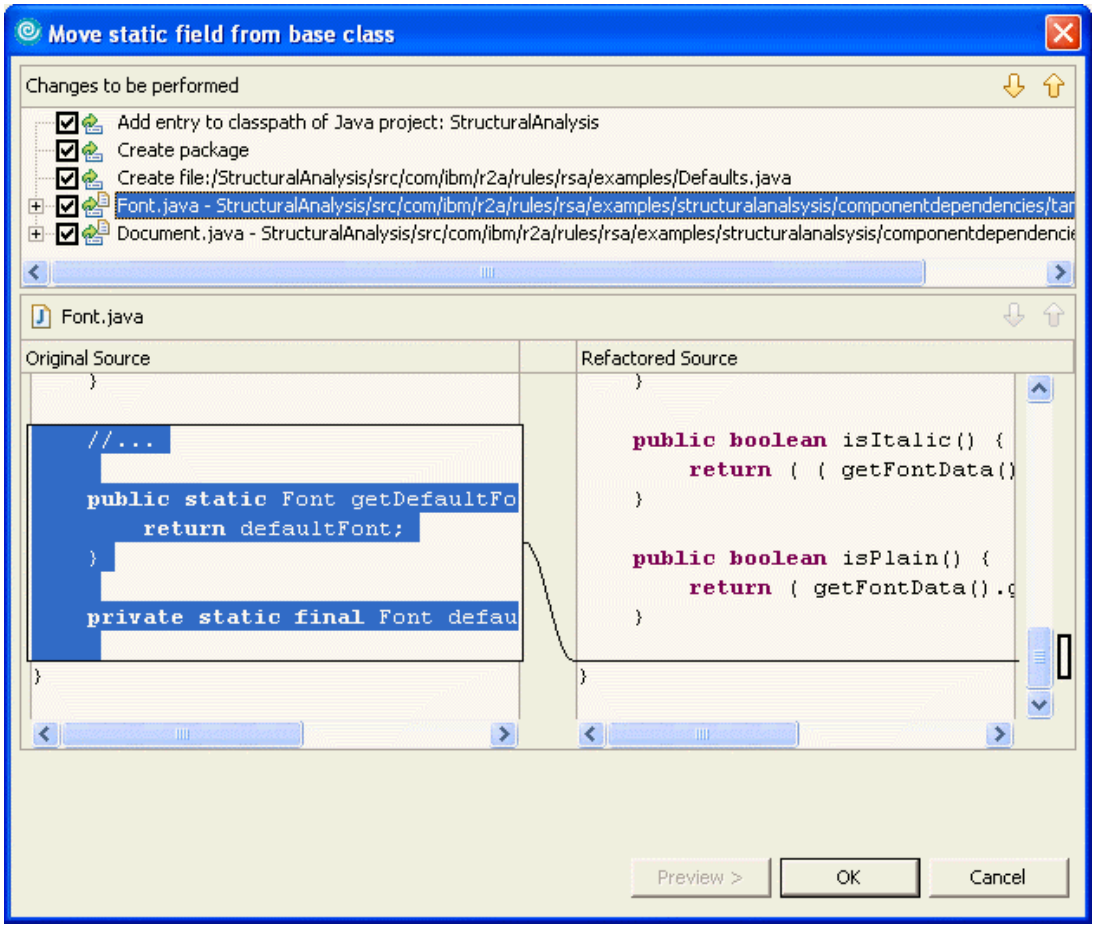


3. Click **Preview** to see the code that the quick fix will change.

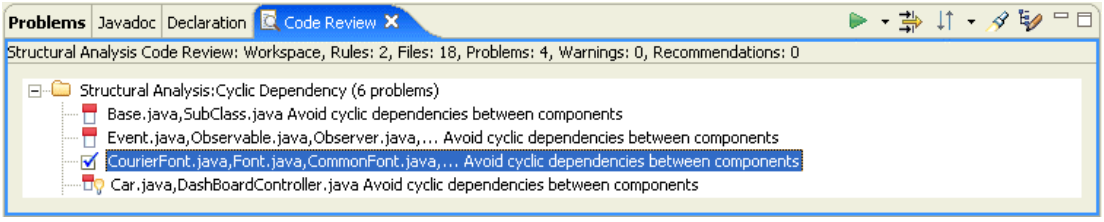
4. Expand the **Changes to be performed** list, as shown in the following screen capture, to see exactly what changes the quick fix will make to the code when it moves the static field to the new class.



5. Click the fourth change in the list, that begins with Font.java, to see a side-by-side view of the code. The original code is on the left and the refactored code that will be created by the quick fix is on the right.



6. Click **OK** to apply the quick fix to all of the selected changes in the list.
7. After the quick fix has been applied, you see a checkmark as confirmation so you know that the problem is resolved.



You have completed Exercise 1.4: Resolving a structural problem.

Leveraging the power of quick fixes

Quick fixes are supplied for some common findings in code reviews. By applying a supplied quick fix, you have an automated way to resolve a cyclic dependency. You can:

Identify and evaluate cyclic dependencies.

Eliminate the dependency quickly with an automated quick fix.

See a list of exactly what changes the quick fix would make to your code.

Fix the cyclic dependency consistently each time.

Wrapping up Exercise 1.4

You have completed Exercise 1.4: Resolving a structural problem. In it you performed the following tasks:

1. Ran a code review to find cyclic dependencies.
2. Recognized when a cyclic dependency has a supplied quick fix.
3. Applied a quick fix to resolve a cyclic dependency:
    - Saw a list of changes to be made.
    - Viewed the existing and refactored code for each change.
4. Got a confirmation that the quick fix has been applied.

Finish the tutorial by reviewing the learning objectives in the “Summary.”

Summary: Run a structural analysis code view

This tutorial showed you how to run a structural analysis code review.

Completed learning objectives

If you completed all of the exercises, you should now be able to do the following tasks:

Run a structural analysis code review.

Define an architectural control rule.

Apply a supplied quick fix to resolve a structural problem.

More information

If you want to learn more about the topics covered in this tutorial, please refer to the online Help for structural analysis.