TIBCO ActiveSpaces® Transactions

Quick Start Guide

Software Release 2.5.8
Published November 10, 2017

Two-Second Advantage®

Important Information

Contents

List of Figures

About this book

This guide describes how to quickly get started using the TIBCO ActiveSpaces® Transactions ™ software product.

This guide is part of a set of TIBCO ActiveSpaces® Transactions documentation, which also includes:

TIBCO ActiveSpaces® Transactions Architect's Guide — This guide provides a technical overview of TIBCO ActiveSpaces® Transactions .

TIBCO ActiveSpaces® Transactions Administration — This guide describes how to install, configure, and monitor a TIBCO ActiveSpaces® Transactions deployment.

TIBCO ActiveSpaces® Transactions Java Developer's Guide — This guide describes how to program a TIBCO ActiveSpaces® Transactions application.

TIBCO ActiveSpaces® Transactions Performance Tuning Guide — This guide describes the tools and techniques to tune TIBCO ActiveSpaces® Transactions applications.

TIBCO ActiveSpaces® Transactions System Sizing Guide — This guide describes how to size system resources for TIBCO ActiveSpaces® Transactions applications.

TIBCO ActiveSpaces® Transactions Javadoc — The reference documentation for all TIBCO ActiveSpaces® Transactions APIs.

Terminology

cluster — A TIBCO ActiveSpaces® Transactions cluster consists of one or more TIBCO ActiveSpaces® Transactions nodes, which may be configured for high availability.

node — A single instance of a TIBCO ActiveSpaces® Transactions software execution environment.

TIBCO ActiveSpaces® Transactions Administrator — A web browser user interface providing access to TIBCO ActiveSpaces® Transactions administration functions.

Domain manager — A specialized node providing management functions for TIBCO ActiveSpaces® Transactions nodes.

1

Introduction

This chapter provides an overview of TIBCO ActiveSpaces® Transactions development.

What is TIBCO ActiveSpaces® Transactions ?

TIBCO ActiveSpaces® Transactions is an in-memory transactional application server that provides scalable high-performance transaction processing with durable object management and replication. TIBCO ActiveSpaces® Transactions allows organizations to develop highly available, distributed, transactional applications using the standard Java POJO programming model.

TIBCO ActiveSpaces® Transactions provides these capabilities:

- Transactions - high performance, distributed "All-or-None" ACID work.
- In-Memory Durable Object Store - ultra low-latency transactional persistence.
- Transactional High Availability - transparent memory-to-memory replication with instant fail-over and fail-back.
- Distributed Computing - location transparent objects and method invocation allowing transparent horizontal scaling.
- Integrated Hotspot JVM - tightly integrated Java execution environment allowing transparent low latency feature execution.

The TIBCO ActiveSpaces® Transactions development model

Think of the TIBCO ActiveSpaces® Transactions server as a sophisticated Java™ Virtual Machine (JVM). Your application is compiled in the normal way on your workstation, and sent to the TIBCO ActiveSpaces® Transactions server for execution.

Your IDE will be configured to run your TIBCO ActiveSpaces® Transactions applications using `deploy.jar`, which transparently sends the compiled Java classes to the server for execution. `deploy.jar` also contains the TIBCO ActiveSpaces® Transactions software public interfaces.

Hardware and software requirements

TIBCO ActiveSpaces® Transactions software applications must be compiled for JDK 6 or greater.

See the TIBCO ActiveSpaces® Transactions release notes for hardware and software requirements to run the TIBCO ActiveSpaces® Transactions server.

The examples in this guide use the Eclipse IDE (http://www.eclipse.org).

2

Building a simple application in Eclipse

This chapter describes the basic process of configuring Eclipse for TIBCO ActiveSpaces® Transactions development. We'll build a simple TIBCO ActiveSpaces® Transactions application in Eclipse, configure Eclipse to run the application on an TIBCO ActiveSpaces® Transactions node, and run the application.

Install and Start TIBCO ActiveSpaces® Transactions

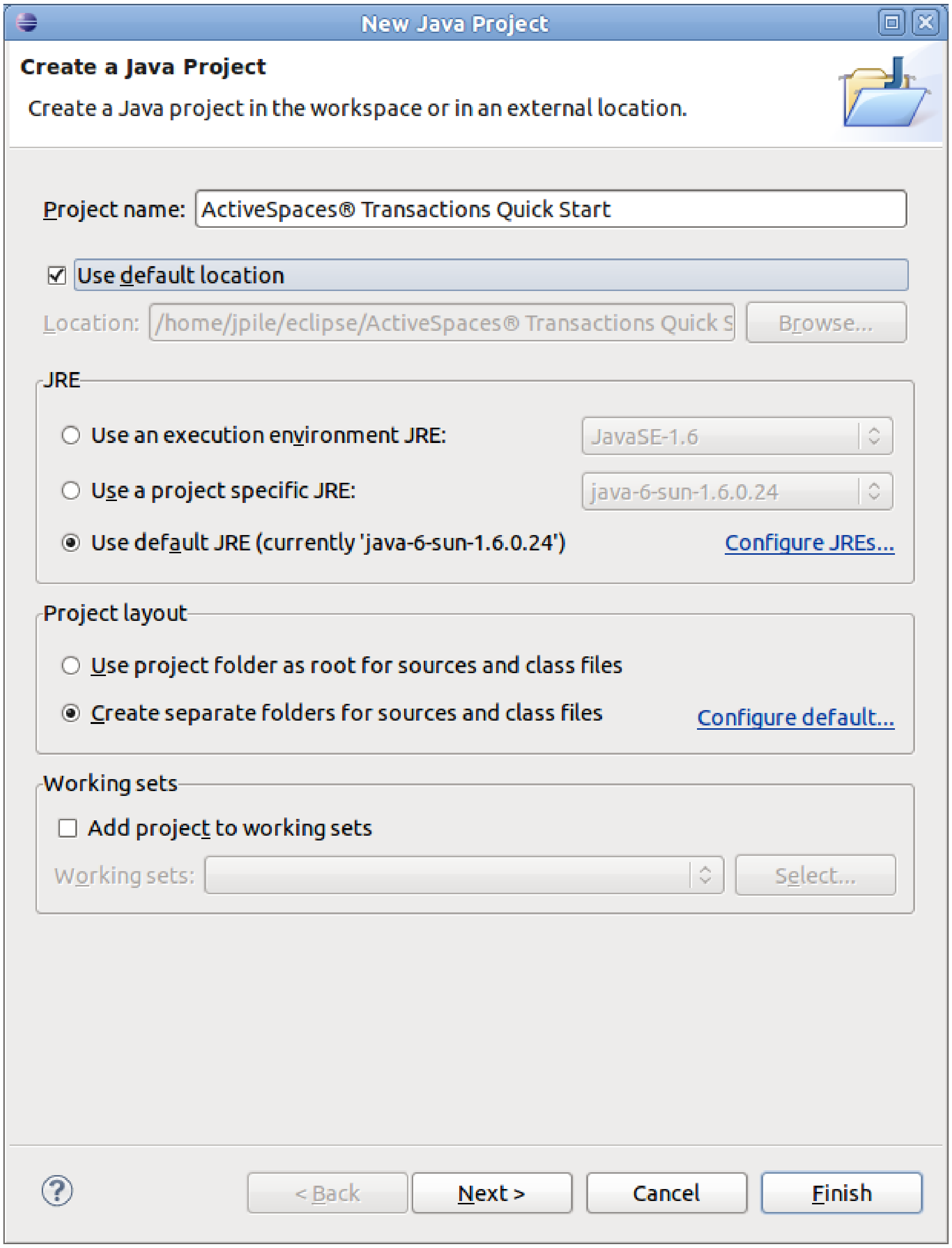
Before you can run the TIBCO ActiveSpaces® Transactions examples, you need to have installed and started a node as described in the TIBCO ActiveSpaces® Transactions **Installation Guide**. The following instructions assume that a Domain Manager node is running at port `2000` and it is managing a node named `A`.

Create a new Java project

First, we will create a new Java project, and include the TIBCO ActiveSpaces® Transactions Java SDK as a support library.

Start Eclipse and create a new Java project:

1. In the *File* menu, select *New ->Java Project*.
2. In the New Java Project dialog that pops up, set the Project name to "TIBCO ActiveSpaces® Transactions Quick Start"



New Java Project

Create a Java Project

Create a Java project in the workspace or in an external location.

Project name: ActiveSpaces® Transactions Quick Start

☑ Use default location

Location: /home/jpile/eclipse/ActiveSpaces® Transactions Quick S

JRE

○ Use an execution environment JRE: JavaSE-1.6

○ Use a project specific JRE: java-6-sun-1.6.0.24

● Use default JRE (currently 'java-6-sun-1.6.0.24')

Project layout

○ Use project folder as root for sources and class files

● Create separate folders for sources and class files

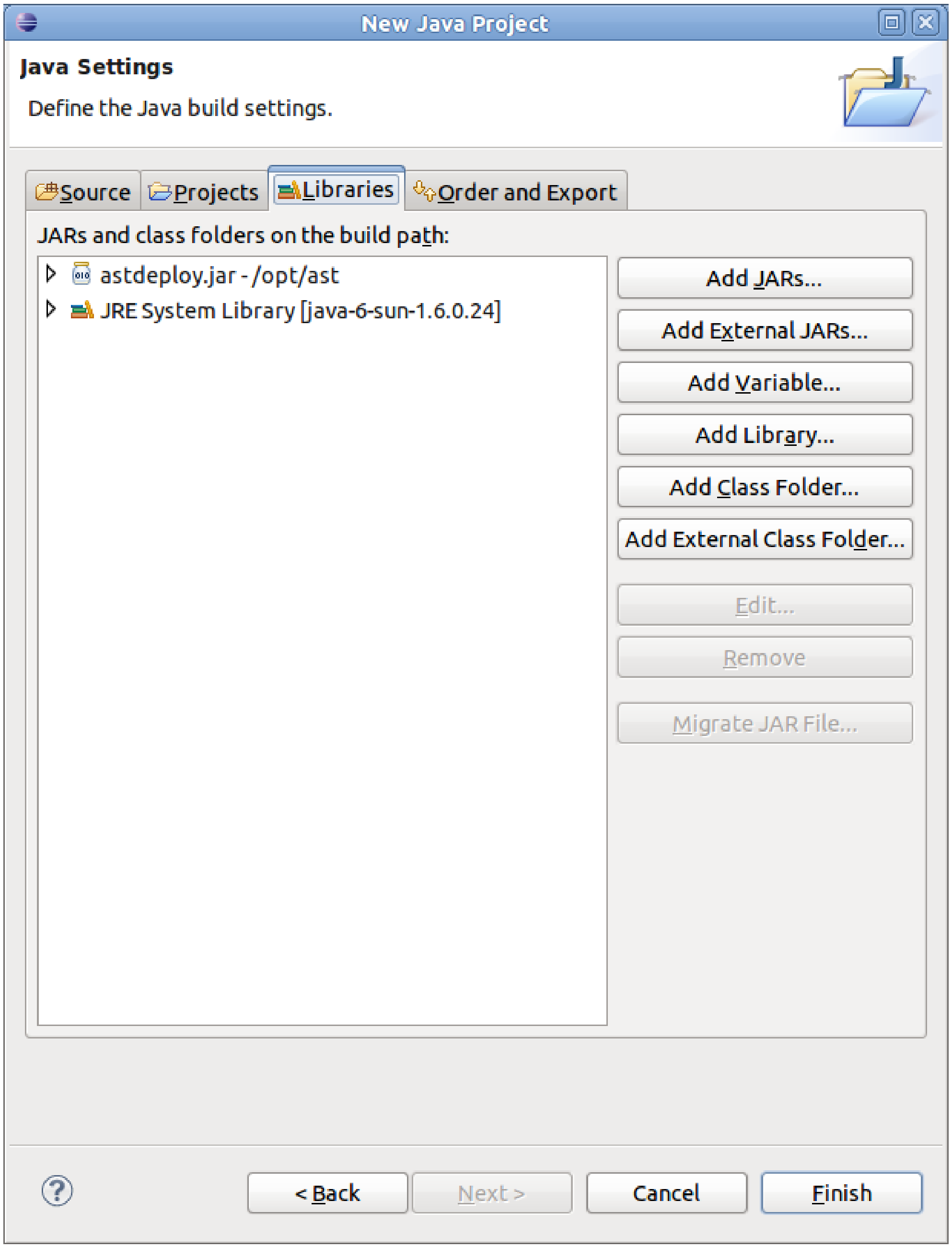
Working sets

☐ Add project to working sets

Working sets:

Figure 2.1. Creating the TIBCO ActiveSpaces® Transactions Quick Start project

3. Click "*Next*" to move to the Java Settings page.
4. Select the *Libraries* tab, and click on "*Add External JARs...*"
5. Browse to the folder where you installed the TIBCO ActiveSpaces® Transactions Java SDK, and find the file `deploy.jar`. Select it and click "*OK*".



JARs and class folders on the build path:

- astdeploy.jar - /opt/ast
- JRE System Library [java-6-sun-1.6.0.24]

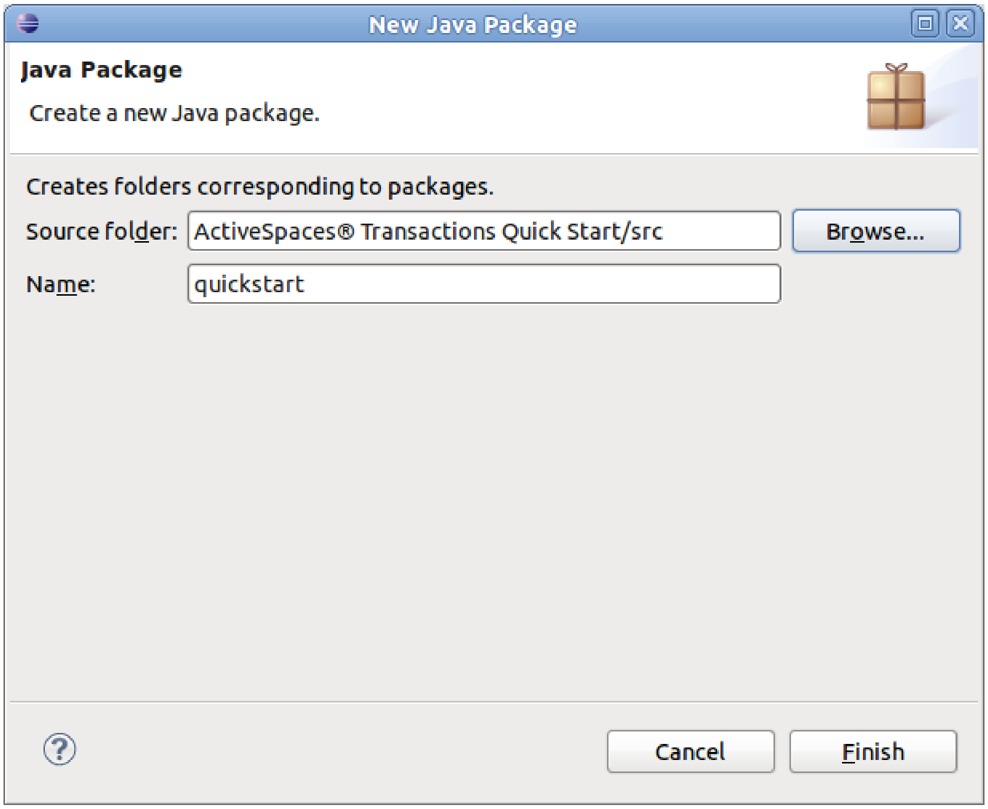
Figure 2.2. Adding deploy.jar to the project libraries

6. Select "*Finish*" to close the New Java Project window; Eclipse will generate the project structures in your default workspace.

Create a simple application

Now that we have an Eclipse project with the right dependencies, we'll create a simple transactional application.

1. In the *File* menu, select *New* -> *Package*. Name the new package `quickstart`, and then click the *Finish* button.



Java Package

Create a new Java package.

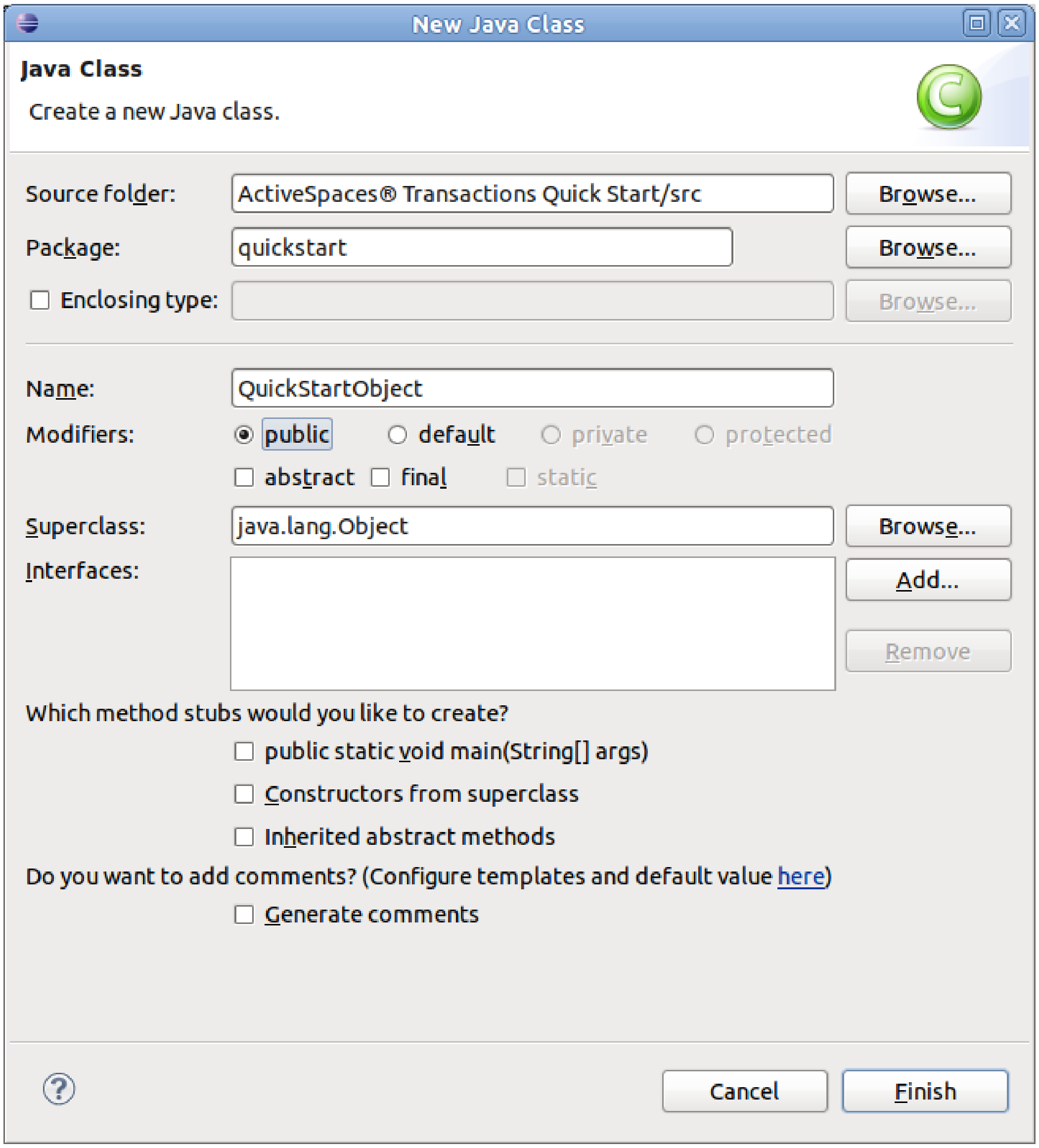
Creates folders corresponding to packages.

Source folder: ActiveSpaces® Transactions Quick Start/src

Name: quickstart

Figure 2.3. Create quickstart package

2. In the Package Explorer, right-click on the new `quickstart` package and select *New* -> *Class*. Set the class name to `QuickStartObject`. This will be our TIBCO ActiveSpaces® Transactions managed type. Click the Finish button.



New Java Class

Java Class

Create a new Java class.

Source folder: ActiveSpaces® Transactions Quick Start/src

Package: quickstart

☐ Enclosing type:

Name: QuickStartObject

Modifiers: ● public ○ default ○ private ○ protected

☐ abstract ☐ final ☐ static

Superclass: java.lang.Object

Interfaces:

Which method stubs would you like to create?

☐ public static void main(String[] args)

☐ Constructors from superclass

☐ Inherited abstract methods

Do you want to add comments? (Configure templates and default value here)

☐ Generate comments

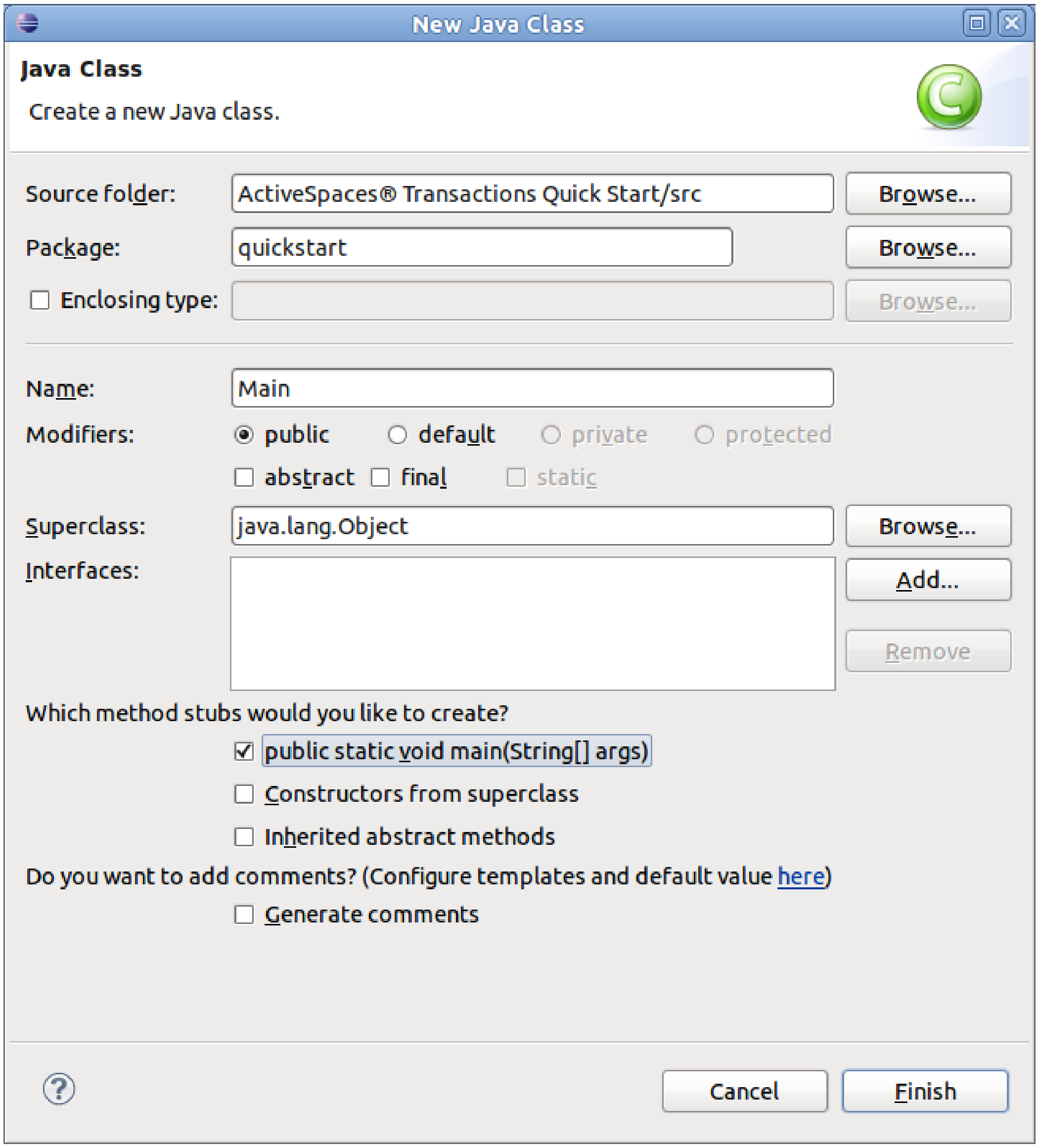
Figure 2.4. Create QuickStartObject class

3. Now edit `QuickStartObject.java` in the edit pane. We will add an `@Managed` annotation to the class, and define a package-private String field named `message`. The class should look like the following:

```
package quickstart;

import com.kabira.platform.annotation.Managed;

@Managed
public class QuickStartObject
{
    String message;
}
```

As a Managed type, any instance of this object we create will be stored in TIBCO ActiveSpaces® Transactions shared memory.

4. Now we'll create another public class with a `main()` method. As before, right-click on the `quickstart` package and select *New* -> *Class*. Set the class name to Main, and let Eclipse generate the method stub for us:



Java Class

Create a new Java class.

Source folder: ActiveSpaces® Transactions Quick Start/src

Package: quickstart

☐ Enclosing type:

Name: Main

Modifiers: ● public ○ default ○ private ○ protected

☐ abstract ☐ final ☐ static

Superclass: java.lang.Object

Interfaces:

Which method stubs would you like to create?

☑ public static void main(String[] args)

☐ Constructors from superclass

☐ Inherited abstract methods

Do you want to add comments? (Configure templates and default value here)

☐ Generate comments

Figure 2.5. Create the Main class

5. Replace the empty main() method body with some transactional code. We'll create and update an instance of the Managed QuickStartObject class we defined:

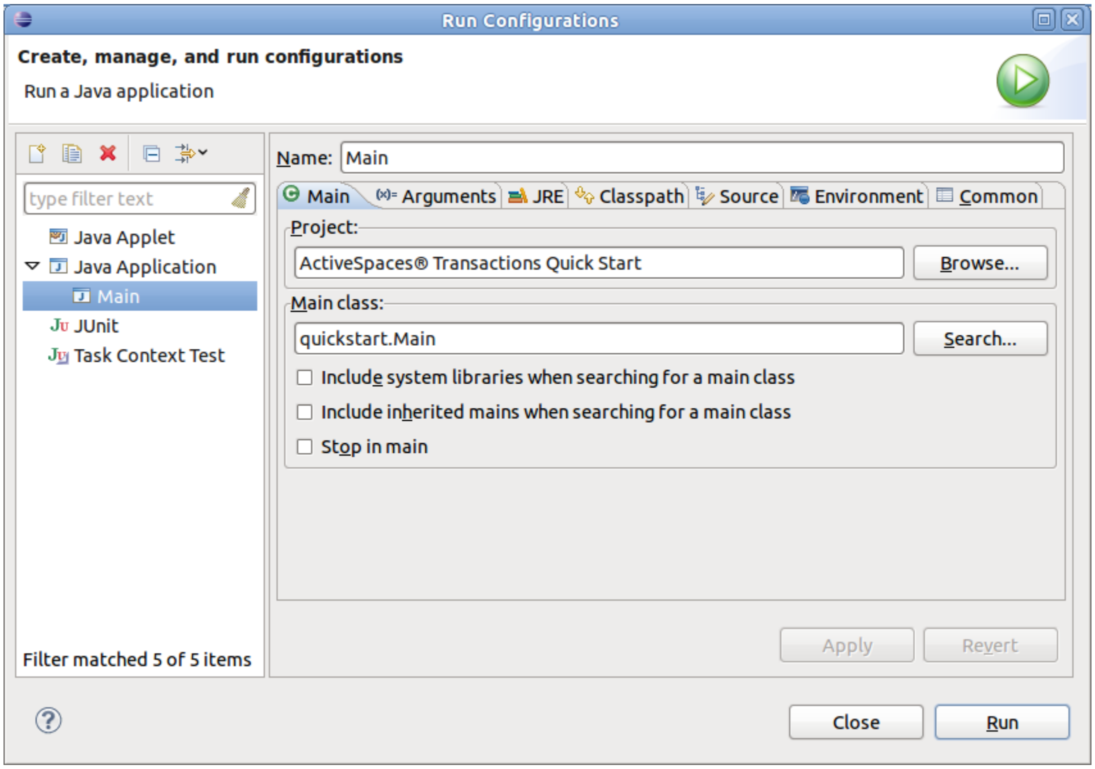
```
package quickstart;

import com.kabira.platform.Transaction;

public class Main
{
    public static void main(String[] args)
    {
        new Transaction()
        {
            @Override
            public void run() throws Rollback
            {
                String message = "Welcome to TIBCO ActiveSpaces® Transactions
!";
                System.out.println(message);

                QuickStartObject quickStartObject = new QuickStartObject();
                quickStartObject.message = message;
            }
        }.execute();
    }
}
```

6. Take a moment to check for any source errors identified by Eclipse. Under the *File* menu, select *Save all*.

Create a run configuration

Next we'll configure Eclipse to run our application on the installed nodes.

1. Under the *Run* menu, select *Run Configurations...*
2. In the left-hand pane of the Run Configurations window, right-click "*Java Application*". This will define a new run configuration.
3. In the *Main* tab, make sure that the "*Main class:*" is set to `quickstart.Main`. Eclipse should have identified this for you; if not, click *Search...* and select the class.



Create, manage, and run configurations

Run a Java application

Name: Main

Project: ActiveSpaces® Transactions Quick Start

Main class: quickstart.Main

☐ Include system libraries when searching for a main class

☐ Include inherited mains when searching for a main class

☐ Stop in main

Filter matched 5 of 5 items

Figure 2.6. Creating a run configuration for quickstart

4. Next, select the "*Arguments*" tab. Here we will configure Eclipse to run the deploy client tool for each invocation of the project.

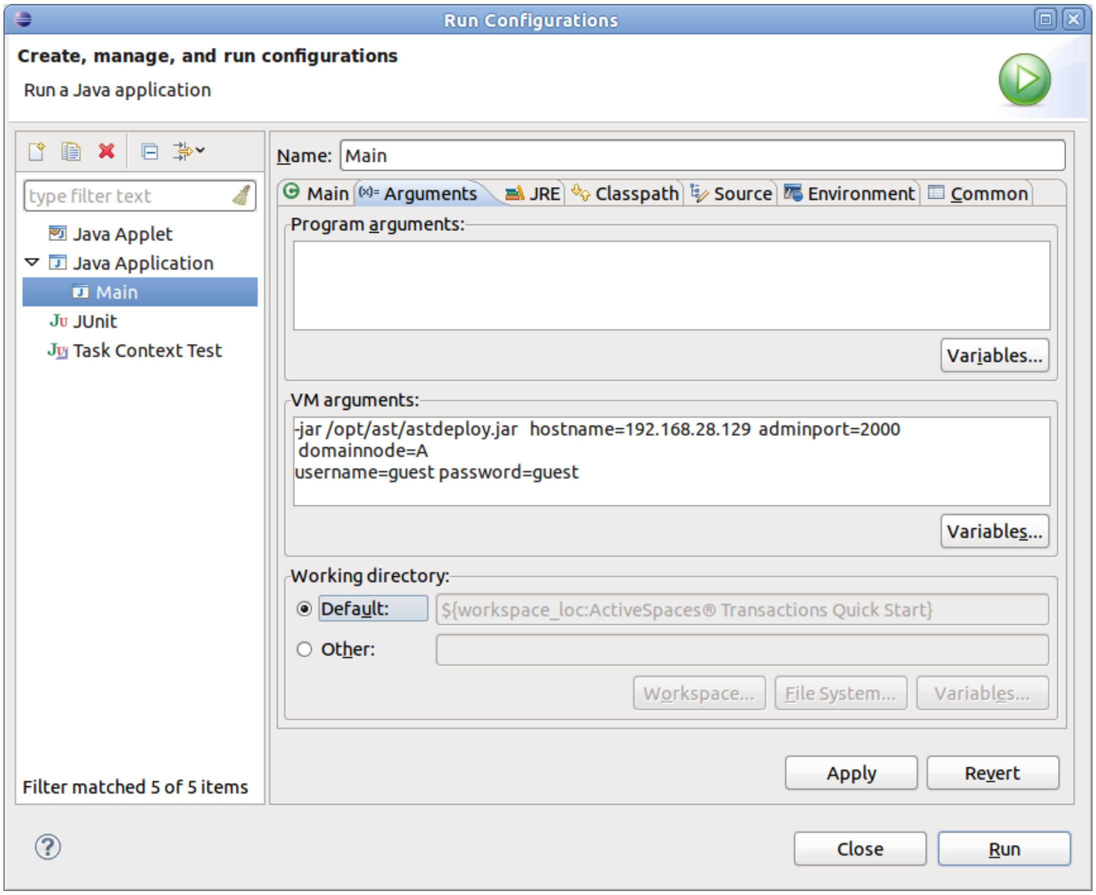
   In the "*VM arguments*" text box, enter the deploy tool command required for the installed nodes.

   For example, if the product was installed in `/opt/ast`, we would use the following argument line:

   ```
   -jar /opt/ast/kis/distrib/kabira/ast/java/deploy.jar hostname=192.168.28.129 adminport=2000 domainnode=A username=guest password=guest
   ```

   This command line tells eclipse to wrap your project invocation with the development client in `deploy.jar`. The client takes parameters that tell it about the network location of the nodes (the `hostname` and `adminport` values) as well as credentials to attach to the server (the `username` and `password` parameters).

   The `domainnode` parameter selects the A node in the domain to execute this application.



Name: Main

VM arguments:

```
-jar /opt/ast/astdeploy.jar  hostname=192.168.28.129 adminport=2000
 domainnode=A
username=guest password=guest
```

● Default: ${workspace_loc:ActiveSpaces® Transactions Quick Start}

○ Other:

Figure 2.7. Configuring the VM arguments and working directory

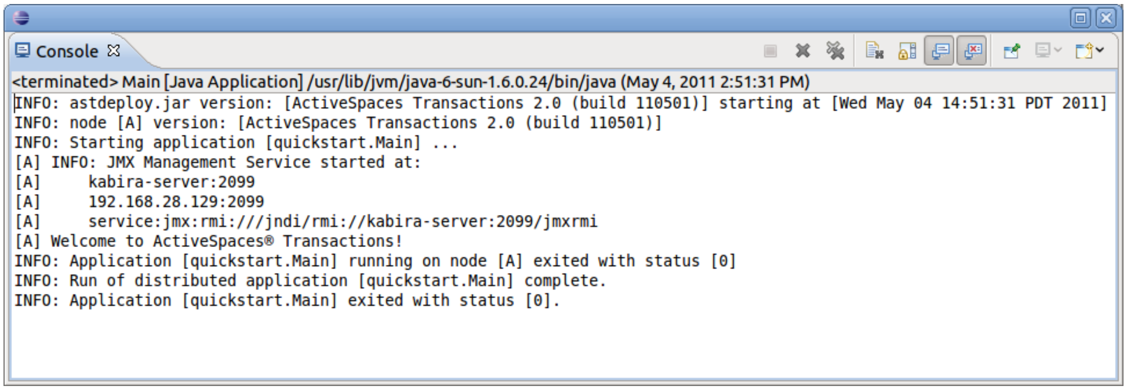
5. select *Apply*, then *Close*.

Run it!

We're ready to run our test application.

Make sure the the TIBCO ActiveSpaces® Transactions Quick Start project is selected in the Package Explorer pane.

Under the "*Run*" menu, select "Run" (or click the green arrow "Run" icon in the toolbar). This will compile the source, deploy the compiled class files and dependent classes to the TIBCO ActiveSpaces® Transactions node and execute the application there. The output of the example is displayed in the IDE console window as shown below:



<terminated> Main [Java Application] /usr/lib/jvm/java-6-sun-1.6.0.24/bin/java (May 4, 2011 2:51:31 PM)

```
INFO: astdeploy.jar version: [ActiveSpaces Transactions 2.0 (build 110501)] starting at [Wed May 04 14:51:31 PDT 2011]
INFO: node [A] version: [ActiveSpaces Transactions 2.0 (build 110501)]
INFO: Starting application [quickstart.Main] ...
[A] INFO: JMX Management Service started at:
[A]     kabira-server:2099
[A]     192.168.28.129:2099
[A]     service:jmx:rmi:///jndi/rmi://kabira-server:2099/jmxrmi
[A] Welcome to ActiveSpaces® Transactions!
INFO: Application [quickstart.Main] running on node [A] exited with status [0]
INFO: Run of distributed application [quickstart.Main] complete.
INFO: Application [quickstart.Main] exited with status [0].
```

Figure 2.8. The output of the TIBCO ActiveSpaces® Transactions Quick Start test application

The build identifier and date may differ in your installation.

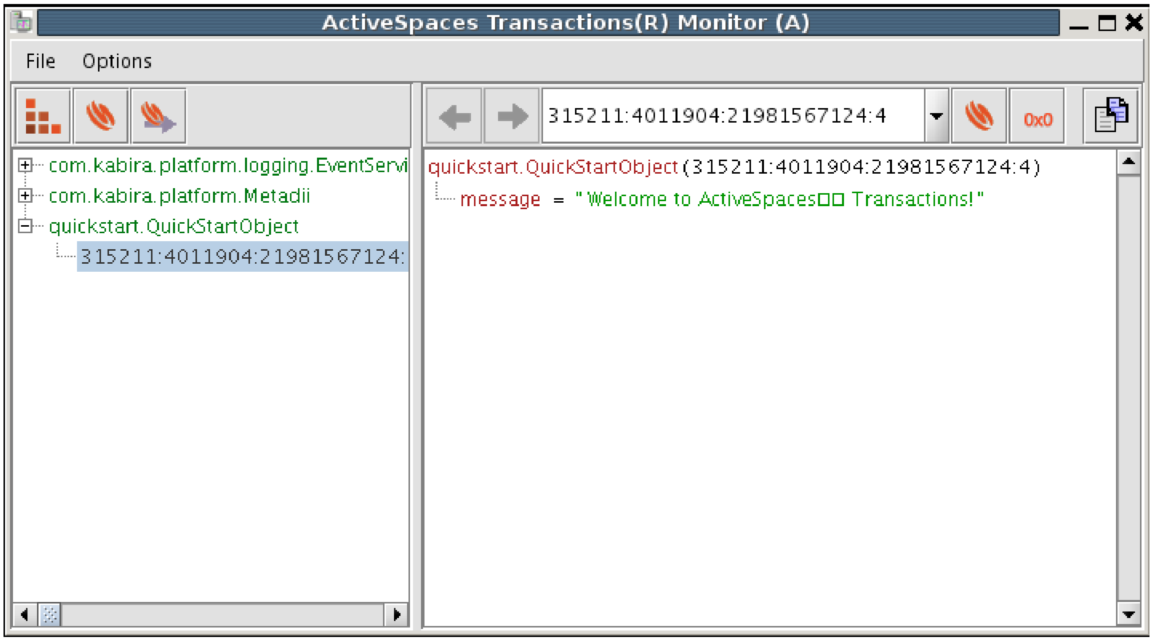
Using the monitor

Now that our application has created a Managed object, we can use the TIBCO ActiveSpaces® Transactions monitor to see that object in persistent shared memory.

1. Start the monitor by using the monitor command line tool and specifying the path to the shared memory file for the application node, for example:

   ```
   monitor /opt/ast/nodes/A/ossm
   ```

2. On the left-hand side of the screen (under the File menu) there are 3 icons. Select the middle one (the tool tip says "Model View").
3. The left-hand pane will now show a listing of all Java types in shared memory. Expand `quickstart.QuickStartObject`, and click the single object reference for that type. In the right-hand pane, you will see the details of the `QuickStartObject` instance that the application created.



```
com.kabira.platform.logging.EventServi
com.kabira.platform.Metadii
quickstart.QuickStartObject
    315211:4011904:21981567124:
```

```
315211:4011904:21981567124:4
quickstart.QuickStartObject(315211:4011904:21981567124:4)
  message = "Welcome to ActiveSpaces□□ Transactions!"
```

Figure 2.9. Displaying the QuickStartObject in the TIBCO ActiveSpaces® Transactions monitor

4. Each time you run the application, another instance of `quickstart.QuickStartObject` will be created in the monitor. Experiment with adjusting the message or defining additional Managed types.

3

Debugging applications in Eclipse

This chapter describes how to debug TIBCO ActiveSpaces® Transactions applications in Eclipse. The configuration for other Java IDEs is similar.

The process consists of the following general steps:

1. Configure Eclipse to run the application on a node in a debug-enabled mode.
2. Configure the Eclipse debugger to attach remotely to the application node.
3. Run the application in debug mode, then attach the remote debugger.

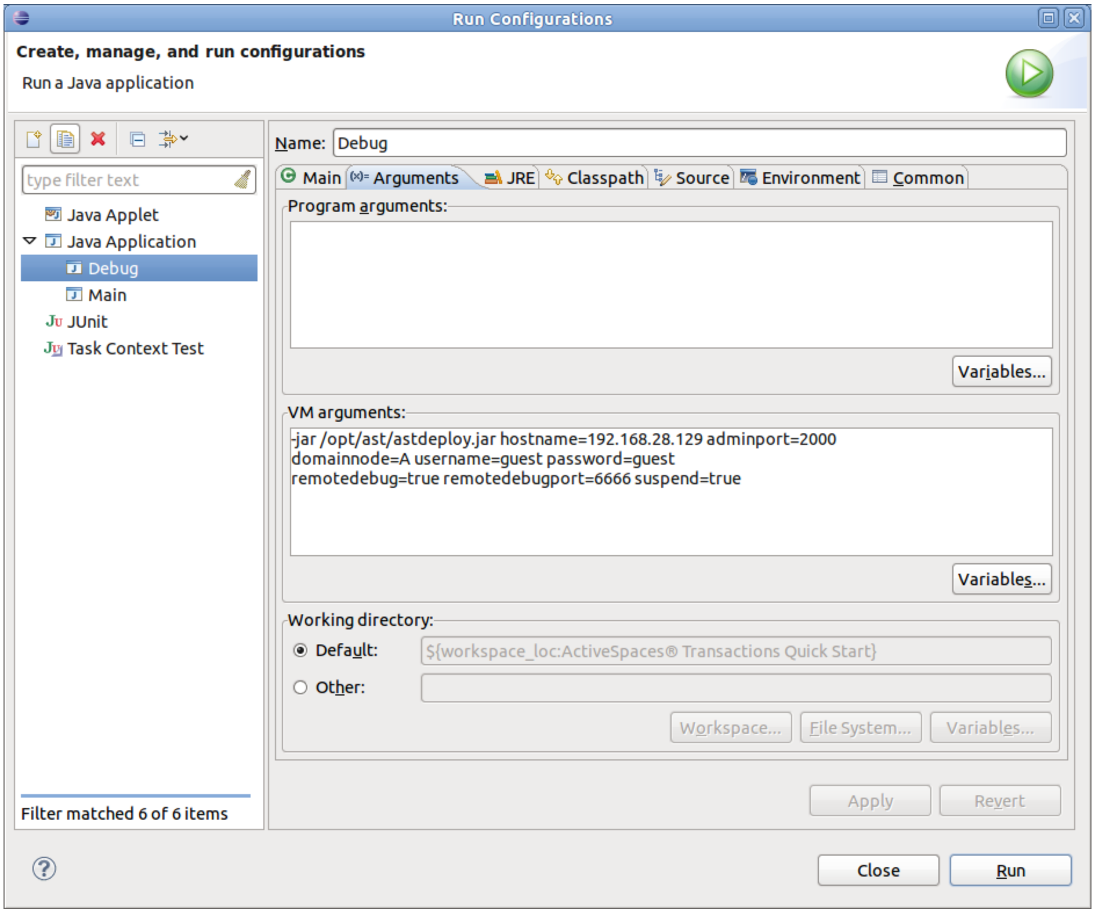
Create a debug-enabled run configuration

This chapter uses the example application that was described in Chapter 2. We will use a copy of the Run Configuration, with remote debugging services enabled.

1. Make sure the TIBCO ActiveSpaces® Transactions Quick Start project is selected in the Package Explorer.
2. Under the "*Run*" menu, select "*Run Configurations...*" to display the run configuration dialog.
3. In the left pane of the dialog, under the Java Application type, click to select the *Main* configuration.
4. Click the "*Duplicate*" button to make a copy.
5. Name the new configuration `"Debug"`.
6. Click the *Arguments* tab. Add the following to the "VM arguments" field:

```
remotedebug=true remotedebugport=6666 suspend=true
```

These arguments instruct the runtime environment to listen for remote debuggers on port 6666. The `suspend` argument instructs the runtime to pause and wait for a debugger to attach before invoking the `main` method of the application.



Name: Debug

VM arguments:

```
-jar /opt/ast/astdeploy.jar hostname=192.168.28.129 adminport=2000
domainnode=A username=guest password=guest
remotedebug=true remotedebugport=6666 suspend=true
```

Working directory:

● Default: ${workspace_loc:ActiveSpaces® Transactions Quick Start}

○ Other:

Figure 3.1. Creating a debug run configuration

7. Click Apply and then Close.

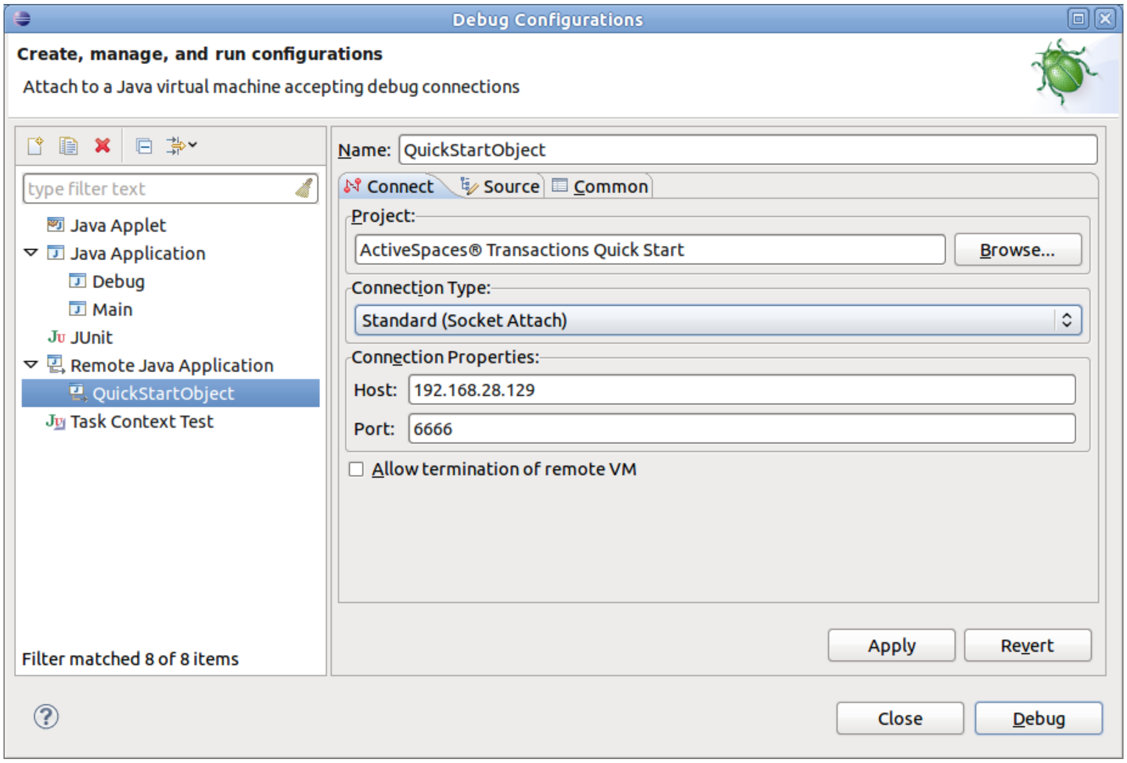
Create a remote debugger configuration

The "Debug" run configuration will execute the Quick Start application in a remote-debug mode. Now we must configure the Eclipse debugger to attach to a remote Java Virtual Machine.

1. Make sure the TIBCO ActiveSpaces® Transactions Quick Start project is selected in the Package Explorer.
2. Under the "*Run*" menu, select "*Debug Configurations...*" to display the debug configuration dialog.
3. In the left pane, select the "*Remote Java Application*" config type, then click the New button.
4. In the "*Connect*" tab, provide remote address information for the "*Connection Properties*" section:

- Set "*Host*" to the address of the running application node (This is the same as the address used in the Run Configuration).
- Set "*Port*" to match the `remotedebugport` that was defined in the run configuration - 6666.

5. Click "*Apply*" to save the configuration, and then close the dialog.



Debug Configurations

Create, manage, and run configurations

Attach to a Java virtual machine accepting debug connections

Name: QuickStartObject

Project: ActiveSpaces® Transactions Quick Start

Connection Type: Standard (Socket Attach)

Connection Properties:

Host: 192.168.28.129

Port: 6666

☐ Allow termination of remote VM

Filter matched 8 of 8 items

Figure 3.2. Creating a debug configuration

Run and debug the application

Debugging the application is a two-part process. First you run the application using the `Debug` run configuration, which will start the debug listener on the application node. Then you launch the Eclipse remote debugger and connect to the Virtual Machine.

1. Set a breakpoint in your application: select a source line in the `main` method. Under the "*Run*" menu, select "*Toggle line breakpoint*".
2. Under the "*Run*" menu, select "*Run Configurations...*". Select the Debug configuration.
3. Click the *Run* button to start the application. You'll see output in the console window ending with:

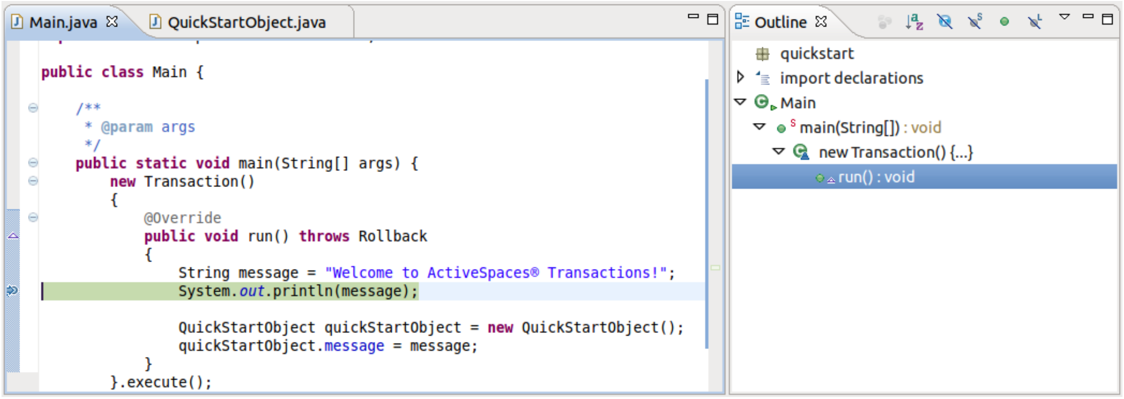
   ```
   [A] Listening for transport dt_socket at address: 6666
   ```

4. Now, choose the Debug perspective (the selector is in the top right tab, or look under the "*Window*" menu and select "*Open Perspective*").
5. Under the "*Run*" menu, select "*Debug Configurations*".

6. In the left pane, select the debug configuration created earlier - the default name is QuickStartObject.

7. Click the Debug button.

Eclipse returns to the debug perspective, and the Eclipse remote debugger connects to the application node. The application will start and run until the breakpoint you set is reached.



```
public class Main {

    /**
     * @param args
     */
    public static void main(String[] args) {
        new Transaction()
        {
            @Override
            public void run() throws Rollback
            {
                String message = "Welcome to ActiveSpaces® Transactions!";
                System.out.println(message);

                QuickStartObject quickStartObject = new QuickStartObject();
                quickStartObject.message = message;
            }
        }.execute();
```

Figure 3.3. An excerpt of the Debug perspective - execution stopped at breakpoint

Now you can:

- Set more breakpoints;
- Examine variables in the "Variables" (top-right) pane;
- Step through program execution a line at a time;
- Display thread stacks by drilling down into the Remote Java Application entry in the Debug pane (top-left).

As you debug the application, its console output continues to be displayed in the debug perspective Console tab.

4

Running the JMS example in Eclipse (with Maven)

This chapter shows how to run an TIBCO ActiveSpaces® Transactions example using Eclipse. To run the examples in Eclipse, you will need a version of Eclipse that supports Maven. Version *3.7.1.x* is known to work.

Install and Start TIBCO ActiveSpaces® Transactions

Before you can run the TIBCO ActiveSpaces® Transactions examples, you need to have installed and started a node as described in the TIBCO ActiveSpaces® Transactions **Installation Guide**. The following instructions assume that a Domain Manager node is running at port `2000` and it is managing a node named `A`.

Check out the examples source from the public CVS server

1. From the `File` menu, select `Import` -> `CVS` -> `Projects from CVS` and click `Next`.

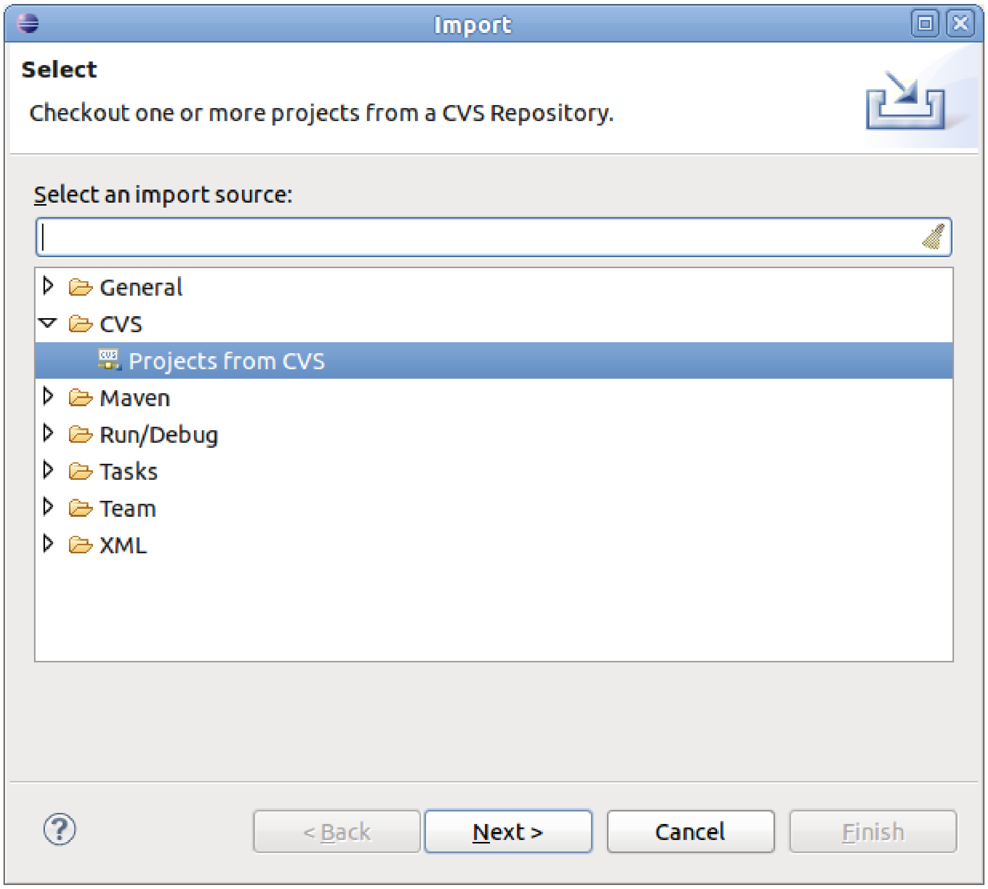
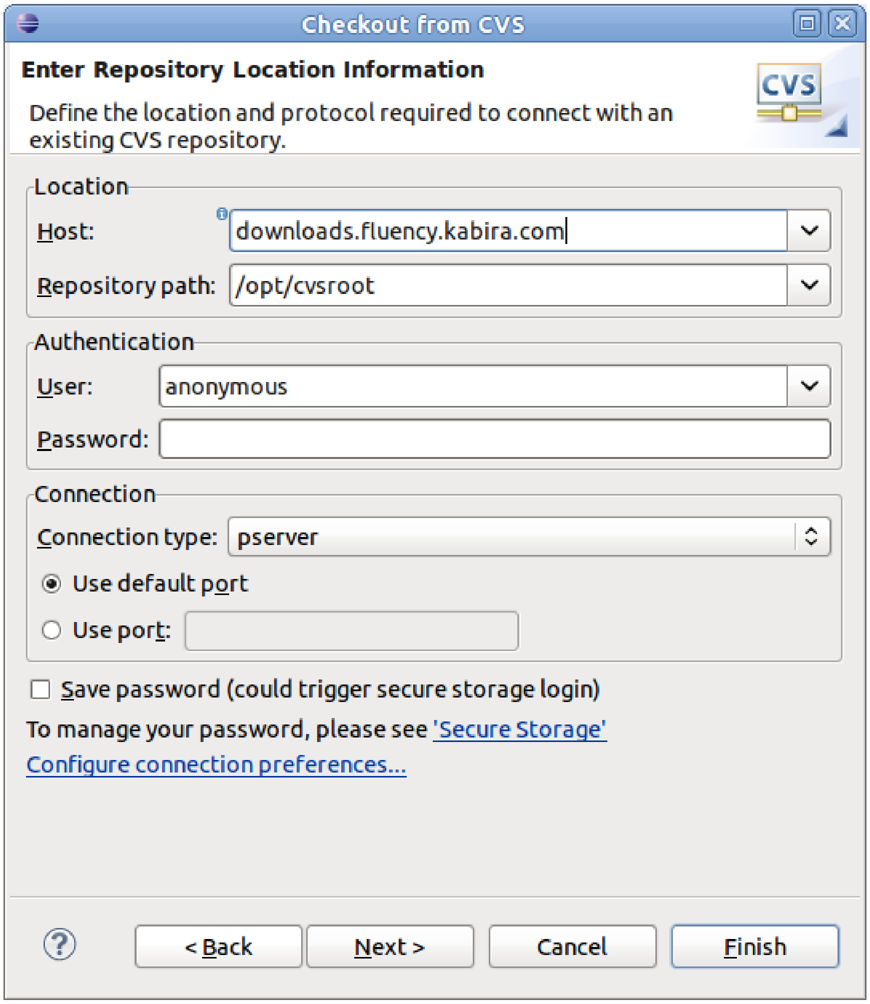


Figure 4.1. CVS Import

2. In the `Checkout from CVS` dialog, enter the following:

Host	downloads.fluency.kabira.com
Repository path	/opt/cvsroot
User	anonymous
Password	(leave blank)
Connection type:	pserver

Click `Finish`.



Checkout from CVS

Enter Repository Location Information

Define the location and protocol required to connect with an existing CVS repository.

Location

Host: downloads.fluency.kabira.com

Repository path: /opt/cvsroot

Authentication

User: anonymous

Password:

Connection

Connection type: pserver

● Use default port

○ Use port:

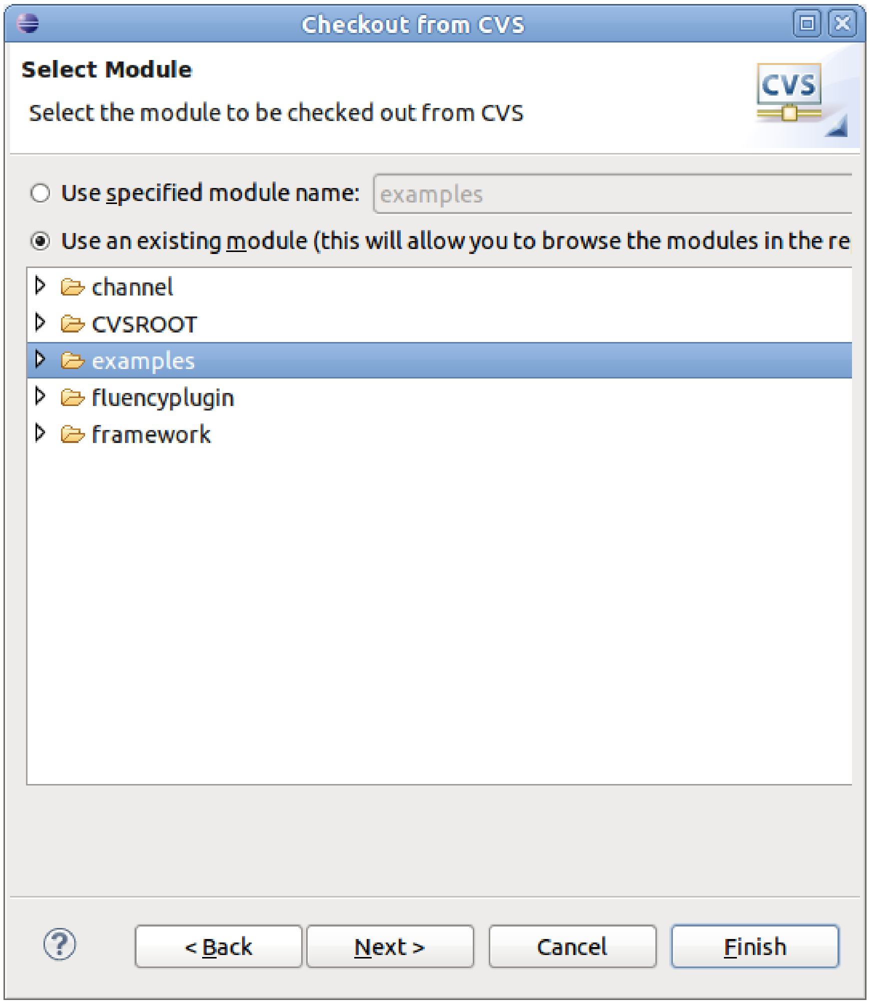
☐ Save password (could trigger secure storage login)

To manage your password, please see 'Secure Storage'

Configure connection preferences...

Figure 4.2. Checkout from CVS

3. In the next screen, select `Use an existing module [...]`, then select `examples` and click on `Next`.



Checkout from CVS

Select Module

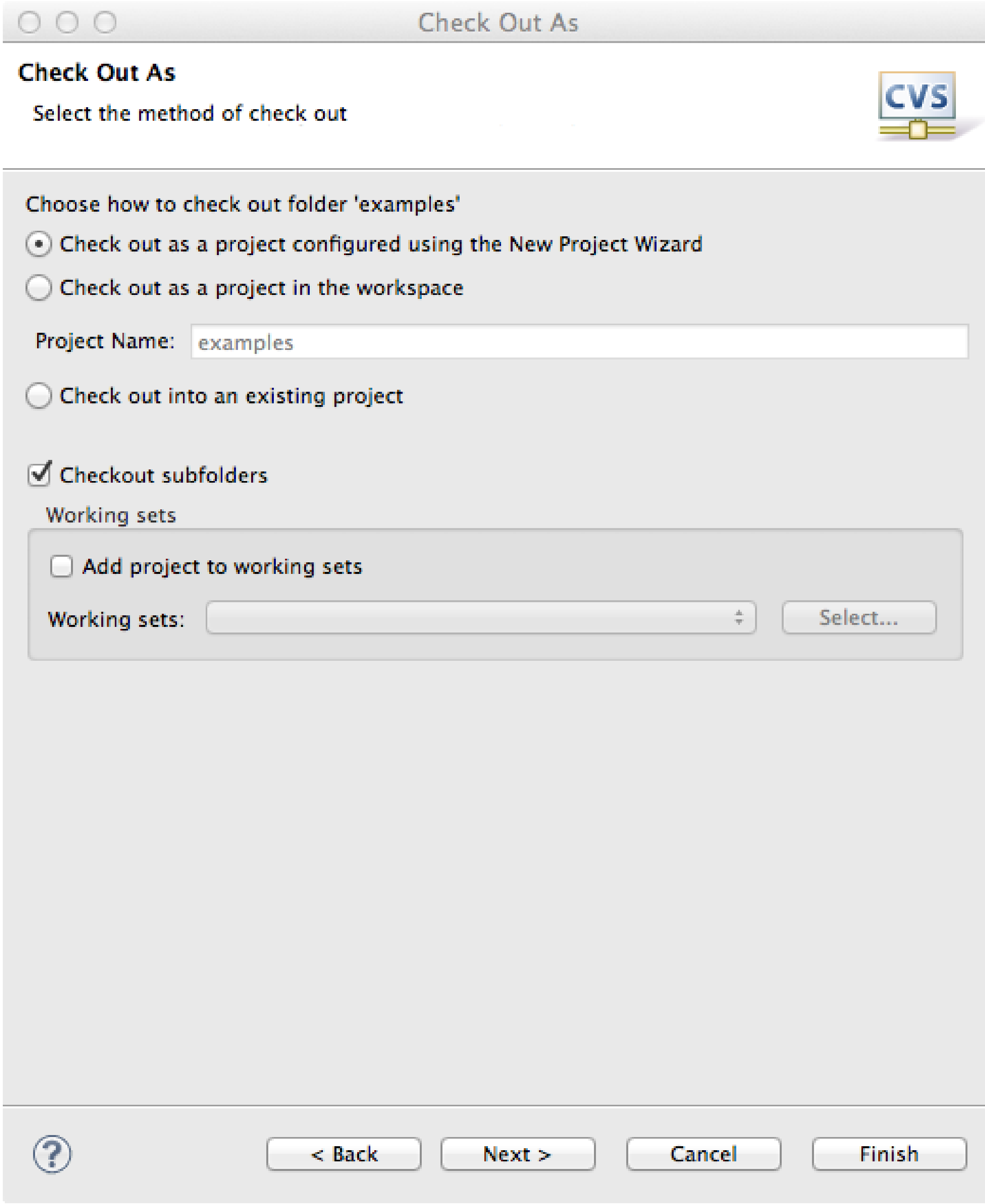
Select the module to be checked out from CVS

○ Use specified module name: examples

● Use an existing module (this will allow you to browse the modules in the re

Figure 4.3. Select Module

4. In the `Check Out As` screen, select `Check out as a project configured using the New Project Wizard`. Click `Next`.



Check Out As

Select the method of check out

Choose how to check out folder 'examples'

● Check out as a project configured using the New Project Wizard

○ Check out as a project in the workspace

Project Name: examples

○ Check out into an existing project

☑ Checkout subfolders

Working sets

☐ Add project to working sets

Working sets:

Figure 4.4. Check Out As

5. From the `Select Tag` dialog select `Versions` and click on the expand triangle to see the available versions. Select the examples version that should be checked out. Click on `Finish`.

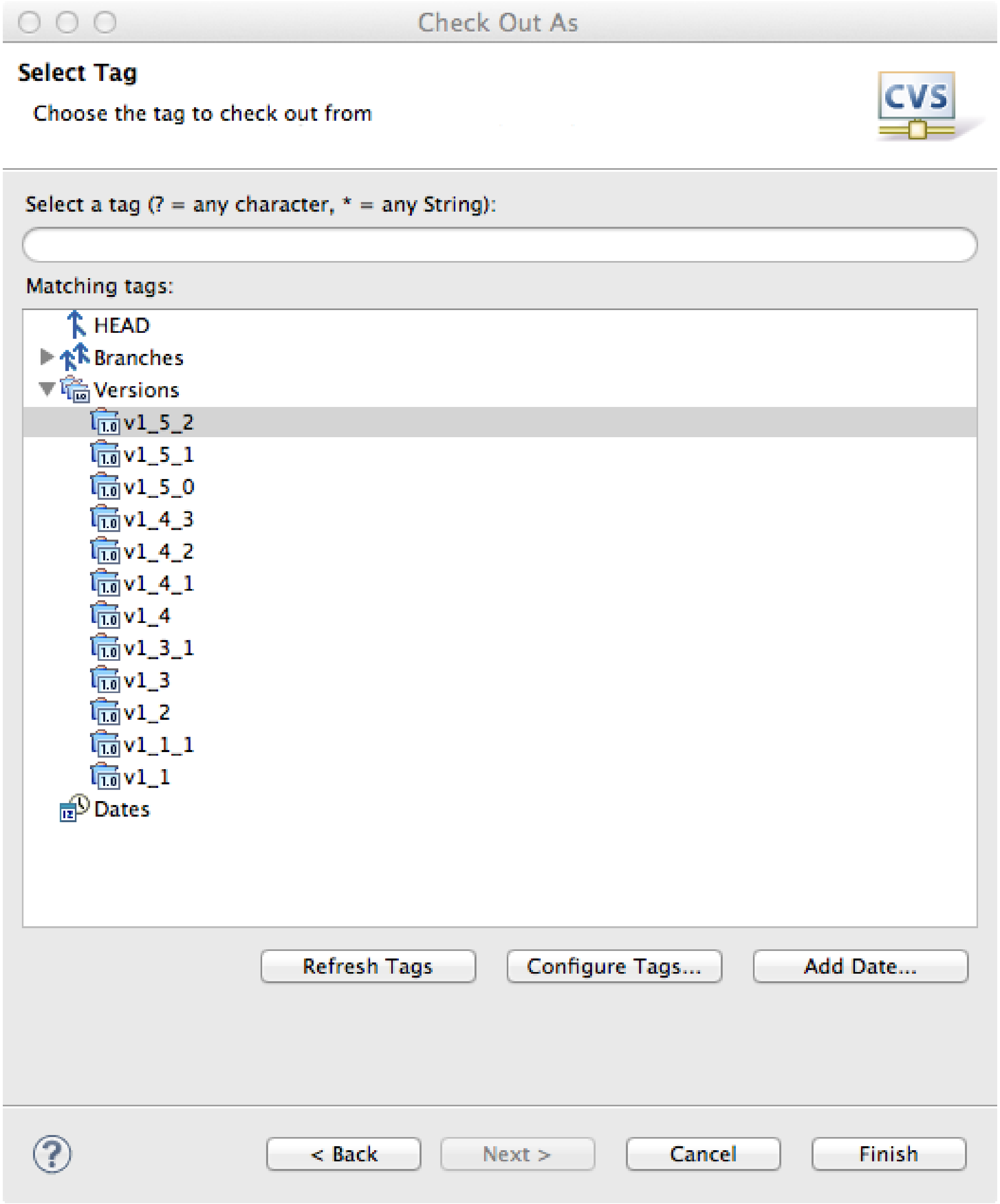


Figure 4.5. Select Tag

6. From the `Select a Wizard` dialog, select `Java Project`. Click `Next`.

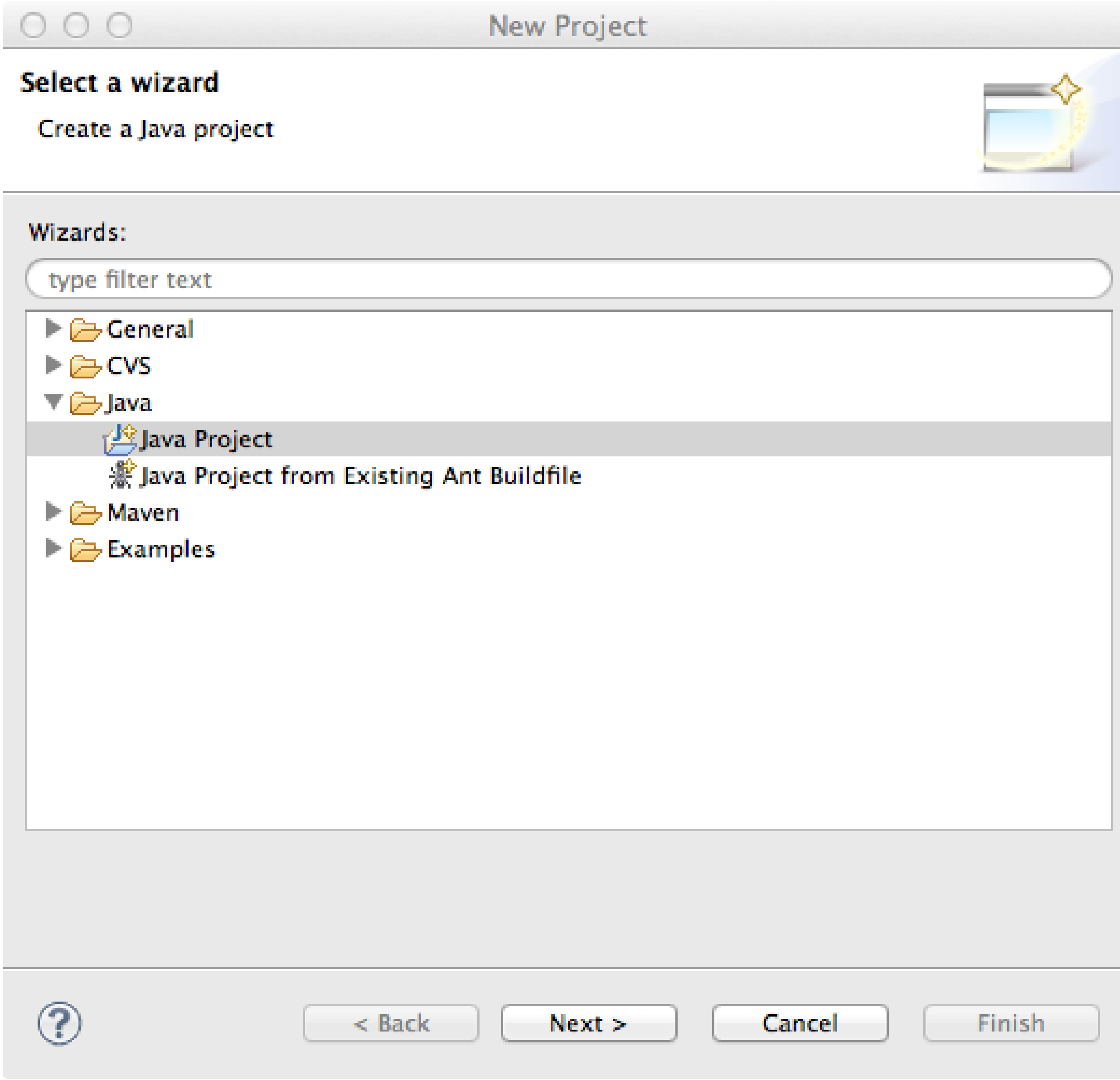
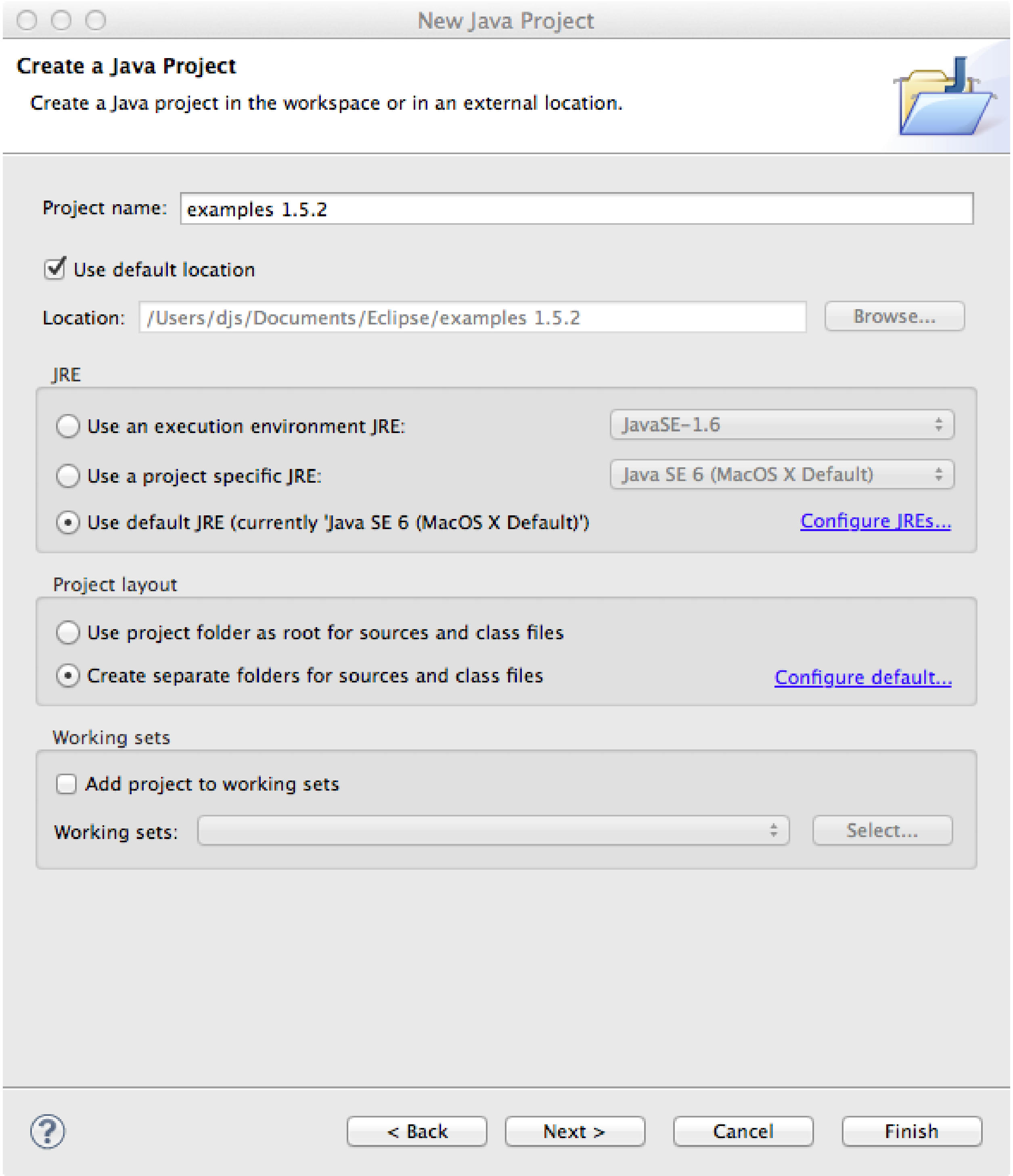


Figure 4.6. New Project

7. Provide a `Project name` and click on `Finish`.



Create a Java Project

Create a Java project in the workspace or in an external location.

Project name: examples 1.5.2

☑ Use default location

Location: /Users/djs/Documents/Eclipse/examples 1.5.2

JRE

○ Use an execution environment JRE: JavaSE-1.6

○ Use a project specific JRE: Java SE 6 (MacOS X Default)

● Use default JRE (currently 'Java SE 6 (MacOS X Default)')

Project layout

○ Use project folder as root for sources and class files

● Create separate folders for sources and class files

Working sets

☐ Add project to working sets

Working sets:

Figure 4.7. New Java Project

8. From the package explorer right click on the package name and select `Configure->Convert to Maven`.

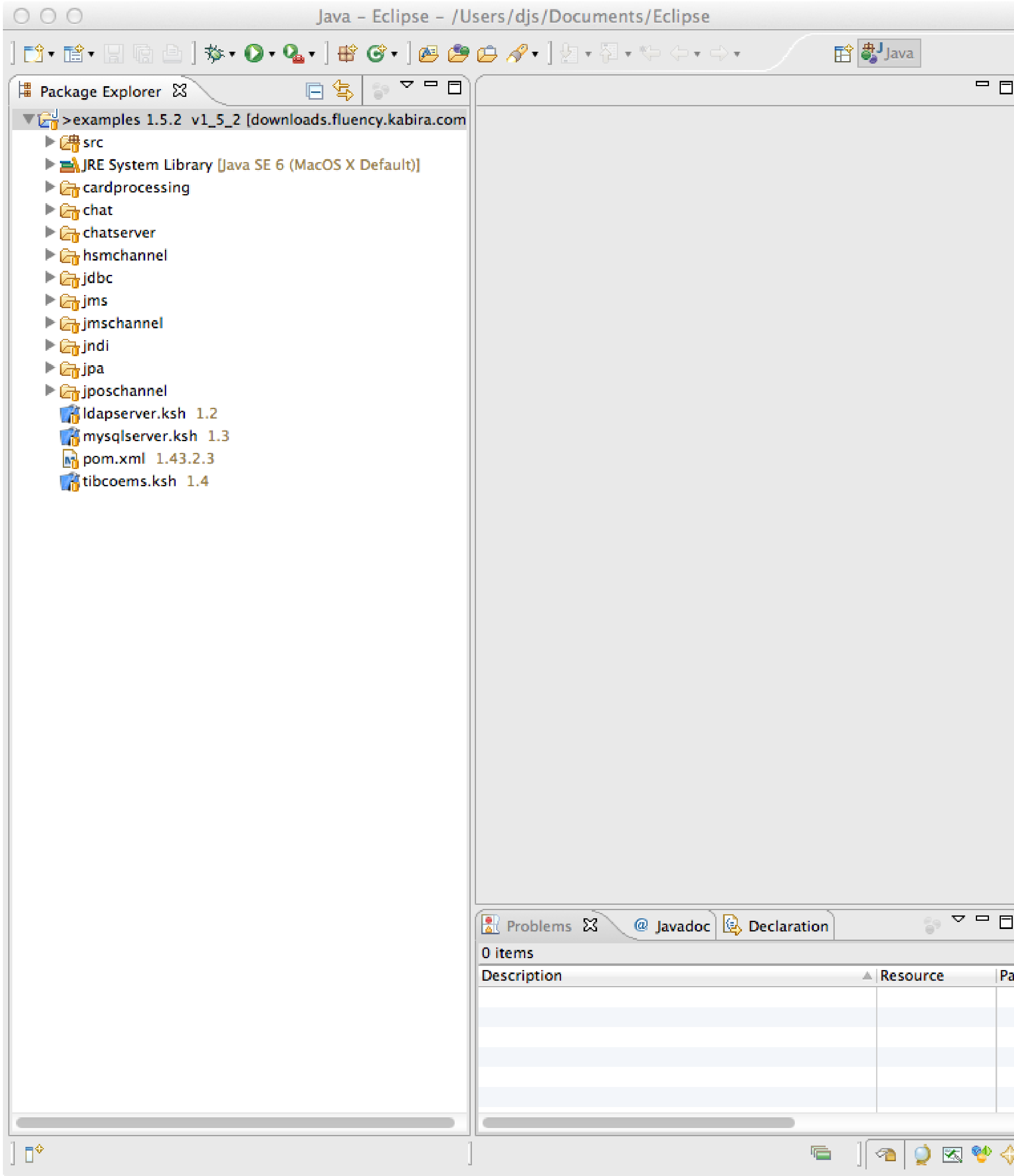


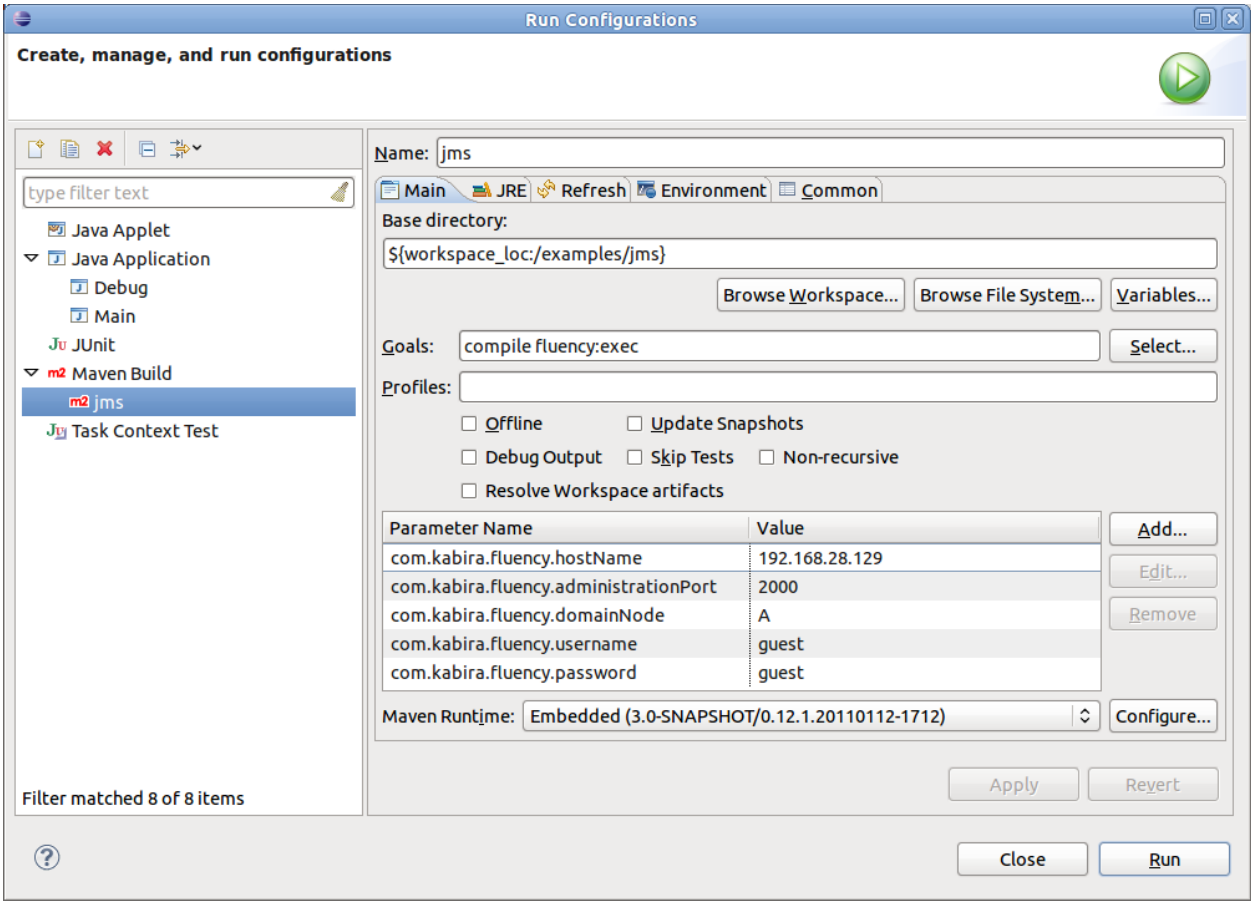
Figure 4.8. Package Explorer

Configure Eclipse to run the JMS example

Now we'll configure Eclipse to run the examples on the installed domain.

1. Under the "*Run*" menu, select "*Run Configurations*".
2. Right-click on on Maven Build, and select "*New*".
3. Rename "New_Configuration" to `jms`.
4. In the Main tab,
   - Set the Base directory: choose Browse Workspace, select `jms`, and click OK.
   - Set Goals to `compile fluency:exec`.
   - Set the following parameters:

com.kabira.fluency.hostName	(The IP address of the Domain Manager node)
com.kabira.fluency.administrationPort	2000
com.kabira.fluency.domainNode	A
com.kabira.fluency.username	guest
com.kabira.fluency.password	guest



Name: jms

Base directory: ${workspace_loc:/examples/jms}

Goals: compile fluency:exec

Profiles:

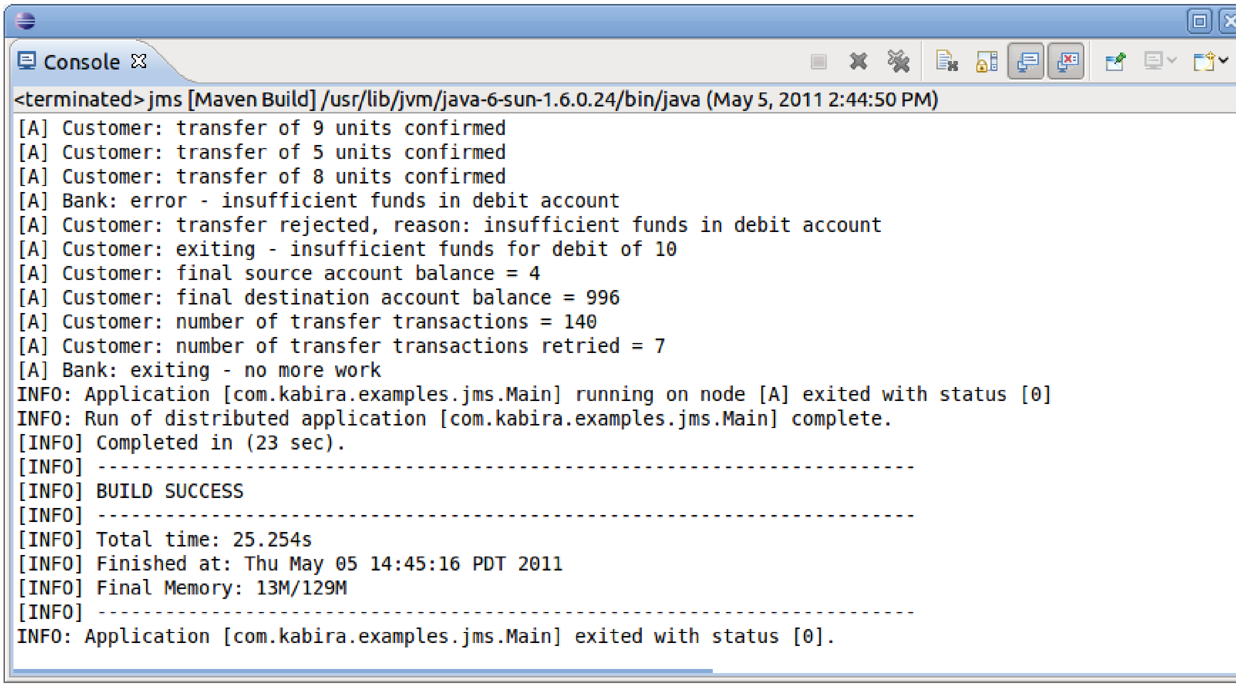
☐ Offline ☐ Update Snapshots
☐ Debug Output ☐ Skip Tests ☐ Non-recursive
☐ Resolve Workspace artifacts

Parameter Name	Value
com.kabira.fluency.hostName	192.168.28.129
com.kabira.fluency.administrationPort	2000
com.kabira.fluency.domainNode	A
com.kabira.fluency.username	guest
com.kabira.fluency.password	guest

Maven Runtime: Embedded (3.0-SNAPSHOT/0.12.1.20110112-1712)

Figure 4.9. Run Configurations - jms

5. Click the "*Apply*" button.
6. Click "*Run*" to build and run the JMS example. The console pane will open and you should see a bunch of output, ending with:



```
<terminated> jms [Maven Build] /usr/lib/jvm/java-6-sun-1.6.0.24/bin/java (May 5, 2011 2:44:50 PM)
[A] Customer: transfer of 9 units confirmed
[A] Customer: transfer of 5 units confirmed
[A] Customer: transfer of 8 units confirmed
[A] Bank: error - insufficient funds in debit account
[A] Customer: transfer rejected, reason: insufficient funds in debit account
[A] Customer: exiting - insufficient funds for debit of 10
[A] Customer: final source account balance = 4
[A] Customer: final destination account balance = 996
[A] Customer: number of transfer transactions = 140
[A] Customer: number of transfer transactions retried = 7
[A] Bank: exiting - no more work
INFO: Application [com.kabira.examples.jms.Main] running on node [A] exited with status [0]
INFO: Run of distributed application [com.kabira.examples.jms.Main] complete.
[INFO] Completed in (23 sec).
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 25.254s
[INFO] Finished at: Thu May 05 14:45:16 PDT 2011
[INFO] Final Memory: 13M/129M
[INFO] ------------------------------------------------------------------------
INFO: Application [com.kabira.examples.jms.Main] exited with status [0].
```

Figure 4.10. Console output for JMS example

Running other examples in Eclipse

The other examples in the examples project may also be run in Eclipse. See the site documentation on the Components and Examples [https://devzone.tibco.com/display/comp/Home] page of the DevZone [https://devzone.tibco.com].

Index