

Improved application development: Part 2, Developing solutions with Rational Application Developer

Skill Level: Intermediate

[Nathaniel T. Schutta](#)

Software developer

Studio B

28 Jun 2005

Move from modeling your use cases to building the components of your application. Learn how Rational Application Developer enables you to create class diagrams while generating much of the template code for your components. With this outline in hand, add application-specific implementation code that extends the original model, which you will test in the final part of this tutorial by deploying your code to WebSphere, working through any bugs before moving to more formal testing.

Section 1. Before you start

About this tutorial

This tutorial is the second in a five-part tutorial series. We strongly recommend that you take all the tutorials in this series sequentially, because each tutorial builds upon something that was done in the previous tutorial.

In this tutorial, you build a class diagram, generating Enterprise JavaBean (EJB) code as you work. While this tutorial doesn't have the space to create a full-fledged application, you will end up with two of the main business components of a sample application in place, mostly functional, and tested. In this tutorial, you'll be working with the Auction application, an example that is used throughout the IBM® product family. This is a Web-based application that leverages Java™ 2 Platform, Enterprise

Edition (J2EE) technology to create an auction system similar to eBay, Yahoo, Amazon, and other auction sites and systems. As you would expect, users can put items up for bid, and other users can bid those items.

As you proceed through the tutorial, you will work with the newest version of a tool that greatly aids the development process: IBM Rational® Application Developer (formerly IBM WebSphere® Studio Application Developer), part of the IBM Software Development Platform. This new product helps unify a development organized on the Eclipse™ project from design through deployment and represents a small part of the complete solution for software development that is available in the IBM Software Development Platform.

This tutorial includes the following:

- **Getting started:** Start where you left off at the end of Part 1 of this series by setting up a project workspace. This section gets you ready for the rest of the tutorial.
- **Building the UML class diagram:** Next, visually create your classes, taking advantage of the newer, more powerful Unified Modeling Language (UML) modeling features of the Rational Suite®.
- **Adding business methods:** Modify the generated EJB code slightly while adding additional business methods. You do some manual coding and start to see the power of annotations.
- **Deploying and testing your beans:** With your EJBs mostly developed, deploy and run them in WebSphere, where you test them using the IBM Universal Test Client.

With the code fully built, it's time to start testing. In Part 3 of this tutorial series, you'll track these defects with IBM Rational ClearQuest® and trace the defects and other changes requests back into IBM Rational RequisitePro®.

Before you begin the tutorial, you should be familiar with the Java language and core J2EE concepts. Experience with UML and EJBs, though helpful, is not required.

Prerequisites

To complete the steps in this tutorial, you need Rational Application Developer Version 6.0. Download a [trial version](#). Downloads are available for Microsoft® Windows® 2000, Windows Server® 2003, and Windows XP along with x86-based Linux machines.

Section 2. Getting started

Create the project

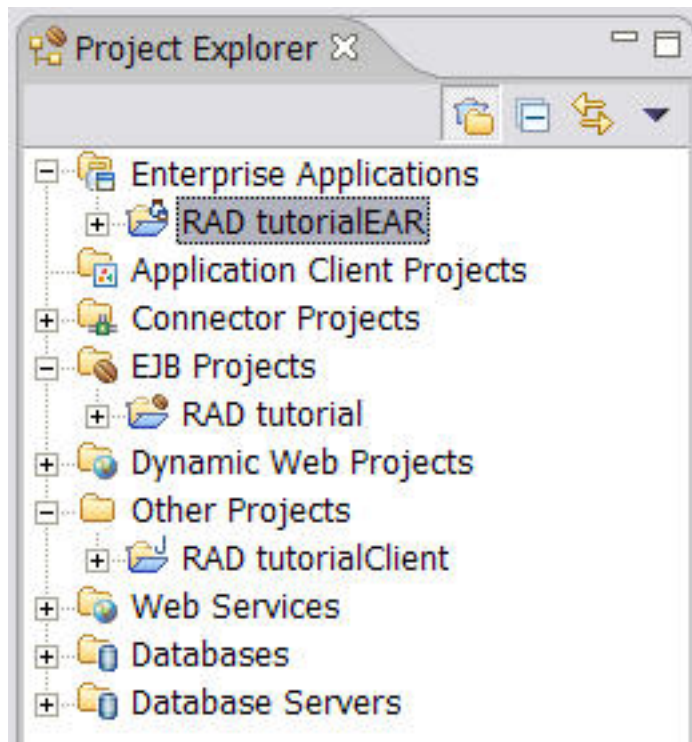
In this section, create the project spaces that are required to house your code. Because you are creating EJBs, you need to create an EJB project. Rational Application Developer automatically creates the supporting projects required for EJBs. Note that EJB projects require that a target server be defined. If you did not install the IBM WebSphere Application Server 6.0 Integrated Test Environment during the installation of Rational Application Developer, you should do so now. To create an EJB project:

1. Select **File > New > EJB Project**. Because the menus are context sensitive, you may not see EJB Project when you select **File > New**. If EJB Project is not visible, select **File > Other > EJB Project**.
2. The New EJB Project wizard opens. Enter a name for your project. Use `RAD tutorial` for this example.
3. Leave the project location as is.
4. If you want to develop to a different version of the EJB standard version or on a different target server, click **Show Advanced** and make the appropriate changes.
5. Click **Finish**.
6. If you are prompted to switch perspectives, click **Yes**.

At this point, Rational Application Developer has created the EJB project along with several supporting projects, as shown in Figure 1. The three projects are:

- The Enterprise Archive (EAR) project `RAD tutorialEAR` under the Enterprise Applications folder
- The EJB project `RAD tutorial` under the EJB Projects folder
- The EJB client project `RAD tutorialClient` under the Other Projects folder

Figure 1. The new projects



Create the UML class diagram

Now that you have your project defined, you can create the class diagram used to generate your EJBs. Start by creating a folder for the class diagram. From the Project Explorer view:

1. Select the EJB project **RAD tutorial**.
2. Right-click **RAD tutorial**, then select **New > Other**.
3. Scroll down and expand Simple.
4. Select **Folder**, then click **Next**.
5. Confirm that **RAD tutorial** is the parent folder.
6. Enter `diagrams` as the name of the folder.
7. Click **Finish**.

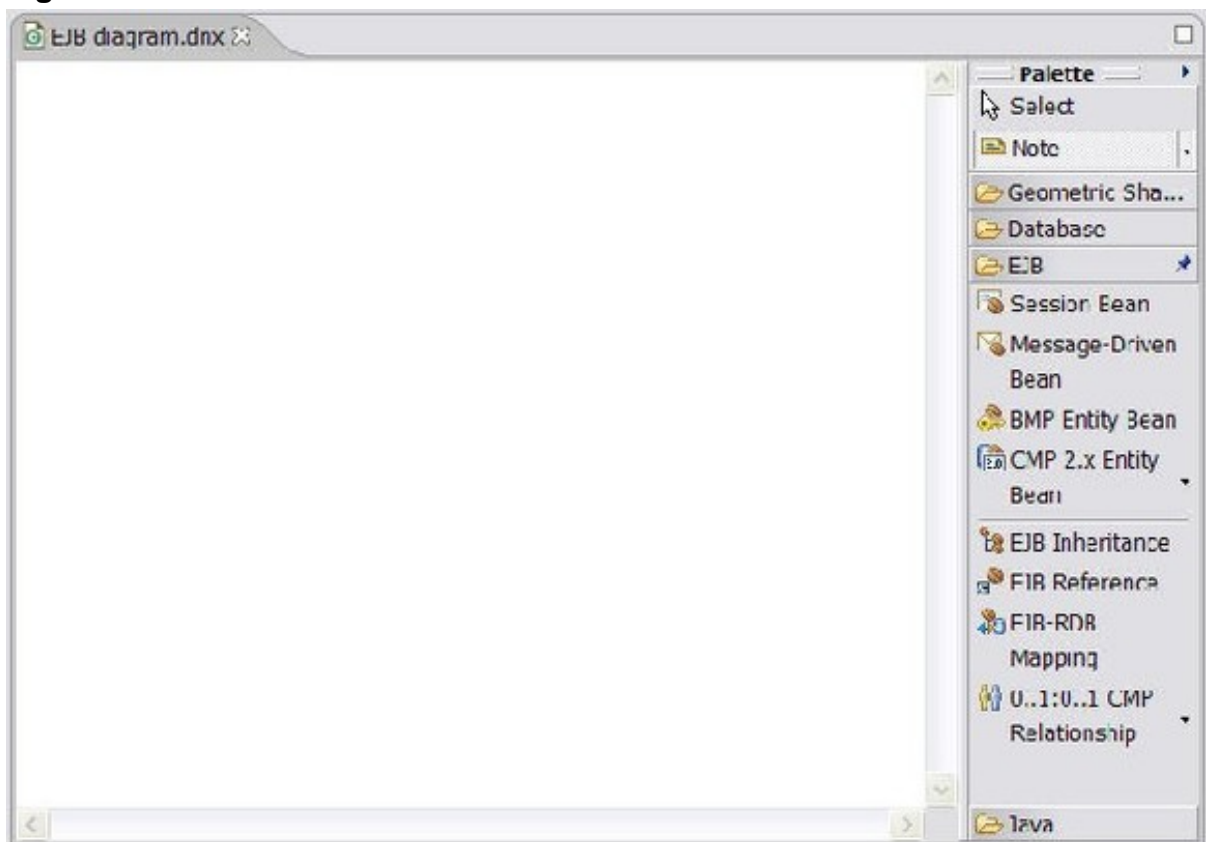
To create your UML class diagram:

1. Select the EJB project **RAD tutorial**.

2. Right-click **RAD tutorial**, then select **New > Class Diagram**.
3. Expand RAD tutorial.
4. Select the **diagrams** folder.
5. Change the name of the diagram to `EJB diagram`.
6. Click **Finish**.

You're now looking at an empty class diagram and the UML visual editor, which should look like Figure 2. The palette on the right side of the window is made up of several drawers that contain different items that can be created and edited within a class diagram. You can open and close drawers by simply clicking them. The stackable palette is new in this release of Rational Application Developer. Notice that the EJB drawer is opened. Because you created this diagram from within an EJB project, the EJB drawer is included on the palette.

Figure 2. UML visual editor



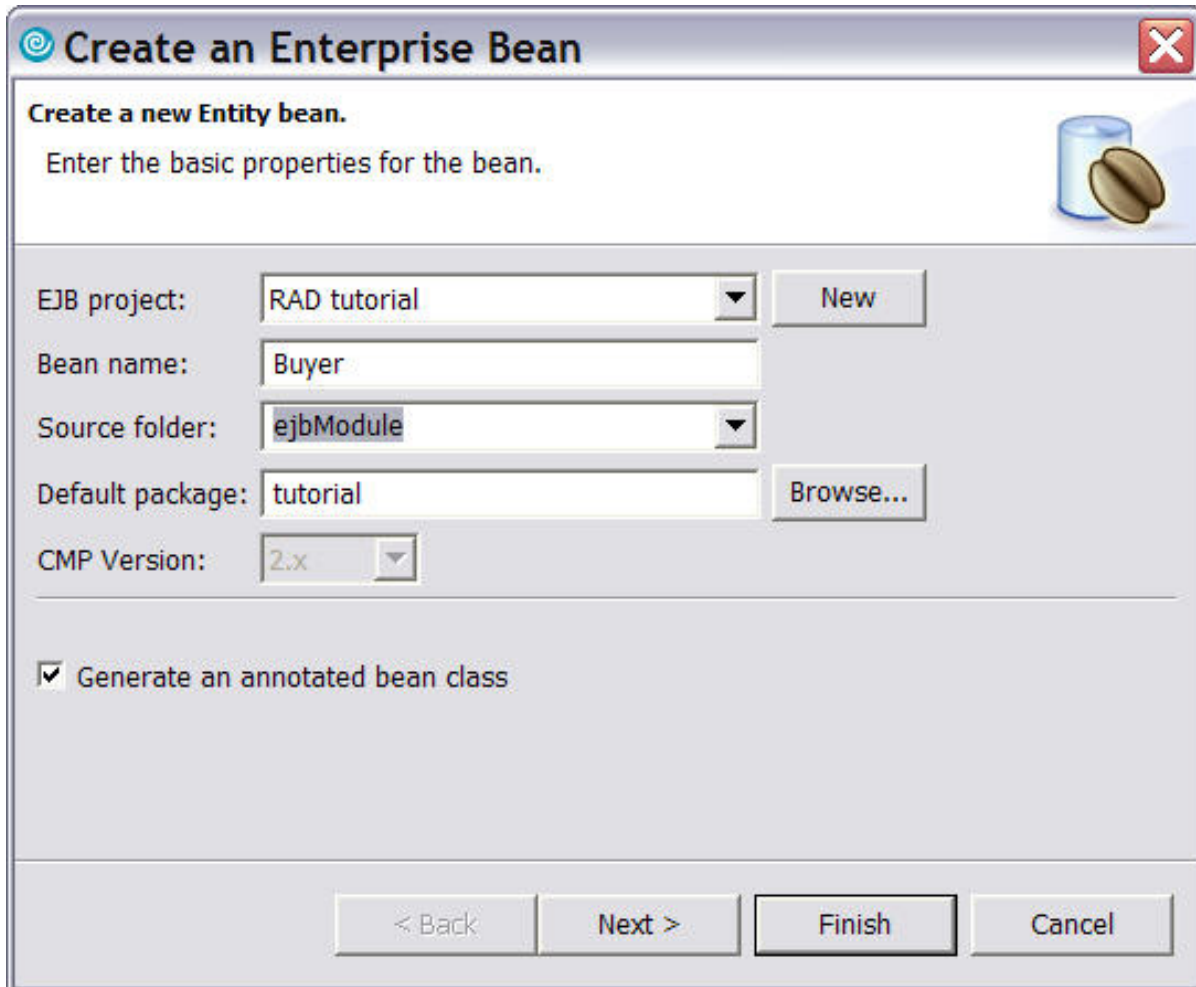
Section 3. Building the UML class diagram

Add entities to the UML class diagram

Now that all the prep work is done, you can get down to the business of actually creating your EJBs. Create four entity beans: *Buyer*, *Seller*, *Bid*, and *Item*. To create the Buyer bean:

1. Open the EJB drawer found in the palette of the Diagram view by clicking it.
2. Click the **CMP 2.x Entity Bean** found in the middle of the EJB drawer.
3. Move your mouse pointer onto the class diagram and click to place the bean. Doing so launches the Create an Enterprise Bean wizard.
4. Verify that the EJB project is **RAD tutorial**.
5. Enter `buyer` as the Bean name.
 - By convention, Java classes start with uppercase letters. The wizard knows this and warns you about your error.
 - Correct the name; notice that the alert goes away.
6. Verify that the Source folder is **ejbModule**.
7. Change the Default Package to `tutorial`.
8. Verify that the CMP Version is **2.x**.
9. Select the **Generate an annotated bean class** check box.
10. The wizard should now look like Figure 3. Click **Next** to proceed.

Figure 3. Information in the Create a new Entity bean window



Create an Enterprise Bean

Create a new Entity bean.
Enter the basic properties for the bean.

EJB project: RAD tutorial

Bean name: Buyer

Source folder: ejbModule

Default package: tutorial

CMP Version: 2.x

Generate an annotated bean class

< Back Next > Finish Cancel

Note: Are you wondering what annotations are? If you've worked with EJBs before, you understand just how many components there are to keep in synch -- your home and component interfaces must match up with your actual bean class in very precise ways, and it is far too easy to make a mistake. While many integrated development environments (IDEs) have become "bean aware" and help keep everything together, you don't always have the luxury of using your favorite development tool. Because so many of the rules and relationships are mechanical in nature, a code generator can create most of the artifacts associated with EJBs for you. However, to do this, a certain amount of metadata is required, which is where annotations come in. The ultimate goal of annotation-based development is to limit the number of artifacts that developers are responsible for, thereby simplifying the development process. A new feature in Rational Application Developer version 6.0, annotation support is there right out of the box, making your life that much easier! To learn more, search the help files, check out the XDoclet code generation engine, the open source object-relational mapping project Hibernate, and the early draft version the EJB 3.0 specification.

Add details to the enterprise bean

The next page of the wizard adds the details to the enterprise bean. Buyer has three fields: a name, an address, and a rating. To add details:

1. Leave the Bean supertype text box blank.
2. Verify that the Bean class is `tutorial.BuyerBean`.
3. Leave the Remote client view check box clear.
4. Ensure that the Local client view check box is selected.
5. Verify that the Local home interface is `tutorial.BuyerLocalHome`.
6. Verify that the Local interface is `tutorial.BuyerLocal`.
7. Verify that the Key class is `java.lang.Integer` and that the Use the single key attribute type for the key class check box is selected.
8. Add the CMP attributes for this bean:
 1. Click **Add**.
 2. Enter `name` in the Name field and `java.lang.String` in the Type field. The Array and Key field check boxes should remain clear, and select the **Promote getter and setter methods to local interface** check box.
 3. Click **Apply**.
 4. Repeat for the remaining attributes: `address: java.lang.String` and `rating: int`.
 5. When you're finished adding attributes, click **Close**.
9. Your window should look like Figure 4. Click **Finish** on the Create an Enterprise Bean page.

Figure 4. Settings to create an EJB

Create an Enterprise Bean

Enterprise Bean Details

Select the supertype, Java classes, and CMP attributes for the container-managed bean.

Bean supertype:

Bean class:

Remote client view

Remote home interface:

Remote interface:

Local client view

Local home interface:

Local interface:

Key class:

Use the single key attribute type for the key class

CMP attributes:

Note that the entity in the class diagram has the methods ordered by the interfaces, a new feature in Rational Application Developer. Repeat these steps for the remaining three beans, adding each to the class model as CMP 2.x entity beans. For Seller:

- Name: `java.lang.String`

- Address: `java.lang.String`
- Rating: `int`

For Bid:

- Amount: `int`
- Timestamp: `java.sql.Timestamp`

For Item:

- reservePrice: `int`
- Title: `java.lang.String`
- Type: `java.lang.String`



When finished, save your diagram. Your class diagram should look something like Figure 5.

Figure 5. Completed class diagram



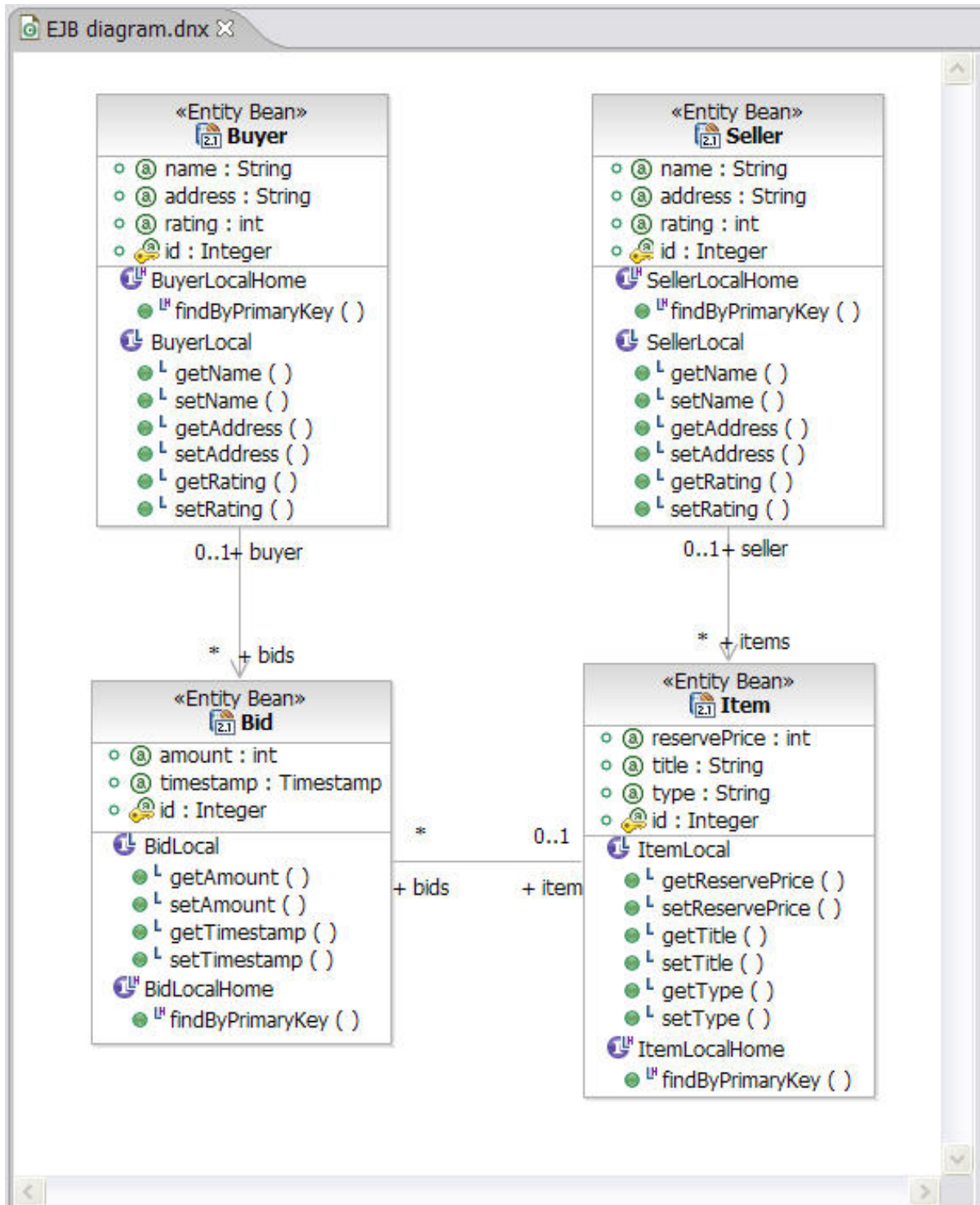
Add container-managed relationships to the UML class diagram

Now that you have your entities defined, it's time to add the associations between the beans. A given Buyer can have many Bids, and a given Seller can be selling many Items. Bids and Items are also related. Start by defining the relationship between a Buyer and their Bids and a Seller and their Items. To add an association:

1. If it is not already, open the EJB drawer found in the palette of the Diagram view by clicking it.
2. In the Palette view, click the down arrow on the CMP Relationship drawer to expose additional relationship types.
3. Select the **0..1:0..* Directed CMP Relationship**.
4. Move your mouse to the Buyer bean. An arrow icon () will appear if the relationship is valid. A relationship that is not valid results in the display of an arrow icon attached to the universal "no" sign () .
5. Click the Buyer bean; drag the cursor to the Bid bean, and release.
6. Double-click the `bid` label on the association, and rename the link to `bids`. Doing so generates a more meaningful `getBids()` method on the bean.
7. Press **Enter**, then save the diagram.

Repeat these steps to establish the same one-to-many relationship between Seller and Item (changing the item label to items). Add the relationship between Item and Bid in the same manner using a regular **0..1:0..* CMP Relationship**, and rename the `bid` label to `bids`. Your class diagram should look something like Figure 6.

Figure 6. Class diagram with container-managed relationships



Section 4. Adding business methods

Add business methods to the beans

At this point, you have modeled the EJB-related attributes, but two of your beans have additional business methods that you need to create. To do this, go into the actual bean code and add the methods that you need. Start by adding the `isHighest()` and `isSuccessful()` methods to the Bid bean, each of which return a Boolean value.

1. In the Project Explorer view, under EJB Projects, expand **RAD tutorial**.
2. Expand **ejbModule**.
3. Expand **tutorial**.
4. Double-click **BidBean** to open the BidBean within the Java editor.
5. Scroll down to the bottom and enter the code for the methods. The code should look something like this:

```
public boolean isHighest() {
    return false;
}

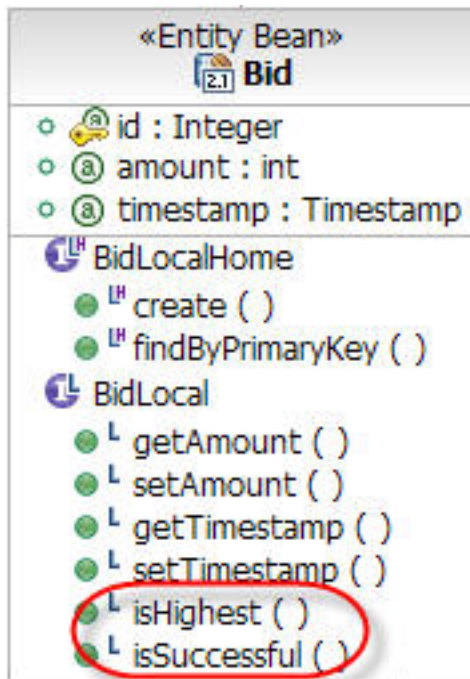
public boolean isSuccessful() {
    return false;
}
```

6. Save your work.

Now that you've created the business methods, make them accessible to clients by promoting them to the local interface. Because you created this class using annotations, you simply add the following code just prior to your methods:

```
/**
 * @ejb.interface-method
 */
```

This tag makes your methods visible on the local interface. Do this for the `isSuccessful()` method, as well. If you return to the class diagram, you should notice your methods under BidLocal within the BidBean class (see Figure 7).

Figure 7. BidBean with business methods

Now add the business methods for the ItemBean class. This class needs methods to return the current, maximum, and starting bids. You already created the collection of bids when you mapped the CMR between Items and Bids, so these three methods will iterate through that collection, looking for the appropriate Item. Open the ItemBean in a code editor and add the following code:

```

public BidLocal getStartingBid() {
    BidLocal startingBid = null;
    java.util.Collection c = getBids();
    java.util.Iterator it = c.iterator();
    while (it.hasNext()) {
        BidLocal thisBid = (BidLocal) it.next();
        java.sql.Timestamp thisBidTimestamp = thisBid.getTimestamp();
        if (startingBid == null) {
            startingBid = thisBid;
        }
        java.sql.Timestamp startingBidTimestamp = startingBid.getTimestamp();
        if (thisBidTimestamp.before(startingBidTimestamp)) {
            startingBid = thisBid;
        }
    }
    return startingBid;
}
  
```

Though not the most efficient code in the world, the Java commands are easy to understand (and will give IBM Rational PurifyPlus™ something to do later in this series). This code simply works through the collection of Bids; if the Bid under consideration has an earlier timestamp than the previous one, it is marked as the starting bid. The earliest Bid is returned by the method. The following code listing shows the implementation details of `getCurrentBid()` and `getMaximumBid()`,

both of which are similar to `getStartingBid()`:

```
public BidLocal getCurrentBid() {
    BidLocal currentBid = null;
    java.util.Collection c = getBids();
    java.util.Iterator it = c.iterator();
    while (it.hasNext()) {
        BidLocal thisBid = (BidLocal) it.next();
        java.sql.Timestamp thisBidTimestamp = thisBid.getTimestamp();
        if (currentBid == null) {
            currentBid = thisBid;
        }
        java.sql.Timestamp startingBidTimestamp = currentBid.getTimestamp();
        if (thisBidTimestamp.after(startingBidTimestamp)) {
            currentBid = thisBid;
        }
    }
    return currentBid;
}

public BidLocal getMaximumBid() {
    BidLocal maximumBid = null;
    java.util.Collection c = getBids();
    java.util.Iterator it = c.iterator();
    while (it.hasNext()) {
        BidLocal thisBid = (BidLocal) it.next();
        int thisBidAmount = thisBid.getAmount();
        if (maximumBid == null) {
            maximumBid = thisBid;
        }
        int maximumBidAmount = maximumBid.getAmount();
        if (thisBidAmount > maximumBidAmount) {
            maximumBid = thisBid;
        }
    }
    return maximumBid;
}
```

Finally, you need to add code with which you can add bids to the bid collection:

```
public void attachBid(BidLocal theBid) {
    //Get hold of the collection of bids, and throw the latest bid into it.
    getBids().add(theBid);
}
```

When finished, promote these new methods to the local interface just as you did with the `BidBean`: by adding the appropriate annotation. Verify that all four methods are visible on the class diagram under the `ItemLocal` interface.

Another way to create an entity bean

So far, you've used the UML modeling power of Rational Application Developer to help you create your beans. This route isn't the only one you can go though. You can always directly create your beans within the project, as you'll do in this panel. Now that you have your core entities, you need to add a primary key generator so

that you can create your beans. To create the bean:

1. In the Project Explorer view, under EJB Projects, expand **RAD tutorial**.
2. Right-click **Deployment Descriptor: RAD tutorial**, then select **New > Enterprise Bean**.
3. Select **Entity bean with container-managed persistence (CMP) fields**.
4. Enter `UniqueKeyGenerator` for the Bean name.
5. Verify that the EJB project is **RAD tutorial**.
6. Verify that the Source folder is **ejbModule**.
7. Change the Default Package to `tutorial`.
8. Verify that the CMP Version is **2.x**.
9. Select the **Generate an annotated bean class** check box, then click **Next**.
10. Edit the default key:
 1. Select `id : java.lang.Integer` in the CMP attributes window.
 2. Click **Edit**.
 3. Change the Type to `java.lang.String`, then click **OK**.
11. Add the CMP attributes for this bean:
 1. Click **Add**.
 2. Enter `lastNumberUsed` in the Name field and `java.lang.Integer` in the Type field. The Array and Key field check boxes should both remain clear, and the Promote getter and setter methods to local interface check box should be clear.
 3. Click **Apply**, then click **Close**.
12. Click **Finish**. The Create an Enterprise Bean page should look like Figure 8.

Figure 8. Information for creating an enterprise bean



Create an Enterprise Bean

Create an Enterprise Bean

Select the EJB type and the basic properties of the bean.

Session bean

Message-driven bean

Entity bean with bean-managed persistence (BMP) fields

Entity bean with container-managed persistence (CMP) fields

EJB project: RAD tutorial

Bean name: UniqueKeyGenerator

Source folder: ejbModule

Default package: tutorial

CMP Version: 2.x

Generate an annotated bean class

This process launches a new class diagram (called *default* -- you can rename it if you choose) showing a UniqueKeyGenerator bean. Save the diagram. Rather than grant direct access to the `lastUsedNumber`, add a business method that increments and returns the `lastUsedNumber`. You're going to do this in a rather roundabout way, but the process demonstrates some of the power of Rational Application Developer. To add a business method:

1. In the Project Explorer view, under EJB Projects, expand **RAD tutorial**.
2. Expand **ejbModule**.

3. Expand **tutorial**.
4. Select **UniqueKeyGenerator.java**.
5. Drag the bean onto the new class diagram.
6. Right-click the Java class, then select **Add > Java Method** to launch the Create Java Method wizard.
7. Enter `getUniqueKey` for the Name.
8. Change the Type to `java.lang.Integer`.
9. Leave the defaults as they are, then click **Finish**.
10. Save the diagram.

Notice that you have an error in your code, because your new method body is empty in the `UniqueKeyGenerator` bean. Double-click the `UniqueKeyGenerator` bean in the Project Explorer view, then scroll down to your new method. Add the following code:

```
/**
 * @ejb.interface-method
 */
public java.lang.Integer getUniqueKey() {
    int intVal = getLastNumberUsed().intValue();
    intVal++;
    Integer inc = new Integer(intVal);
    setLastNumberUsed(inc);
    return inc;
}
```

When you save the class, the error symbol should disappear. Don't forget the annotation that makes this method available on the local interface!

While you're in the `UniqueKeyGenerator` bean, modify the `create` method so that it sets a starting last number for use in your other beans. This code isn't ideal, but it'll work for now:

```
/**
 * ejbCreate
 * @ejb.create-method
 * view-type="local"
 */
public java.lang.String ejbCreate(java.lang.String id)
    throws javax.ejb.CreateException {
    setId(id);
    setLastNumberUsed(new Integer(0));
    return null;
}
```

Save the class when you are finished.

Code for the creation of ItemBeans and BidBeans

Now that you have your unique key generator, its time to take advantage of it. In this panel, you'll edit the `ejbCreate()` methods in `ItemBean` and `BidBean` to do the minimum work required to create the bean by providing a primary key.

Edit `BidBean`'s and `ItemBean`'s `ejbCreate()` method so that it looks like the following code. Note that you are removing the argument that Rational Application Developer added by default for this method and are adding the annotation to say that this is a `create` method:

```
/**
 * @ejb.create-method
 */
public java.lang.Integer ejbCreate()
    throws javax.ejb.CreateException {
    Integer id = null;
    try {
        javax.naming.InitialContext ctx = new javax.naming.InitialContext();
        UniqueKeyGeneratorLocalHome ukgHome = (UniqueKeyGeneratorLocalHome)
            ctx.lookup("local:ejb/ ejb/tutorial/UniqueKeyGeneratorLocalHome");
        UniqueKeyGeneratorLocal ukg = ukgHome.findByPrimaryKey("KEY");
        id = ukg.getUniqueKey();
    } catch (Exception e) {
        e.printStackTrace();
        throw new RuntimeException();
    }
    setId(id);
    return id;
}
```

Of course, this code presupposes that a `UniqueKeyGenerator` already exists with an `id` of `KEY`, but you'll see to that in the testing phase. You should see an error indicating that the `ejbPostCreate()` must exist -- scroll down and remove the default `Integer` parameter on the `ejbPostCreate()` method, then save the bean.

Section 5. Deploying and testing your beans

Map CMP beans to a relational database

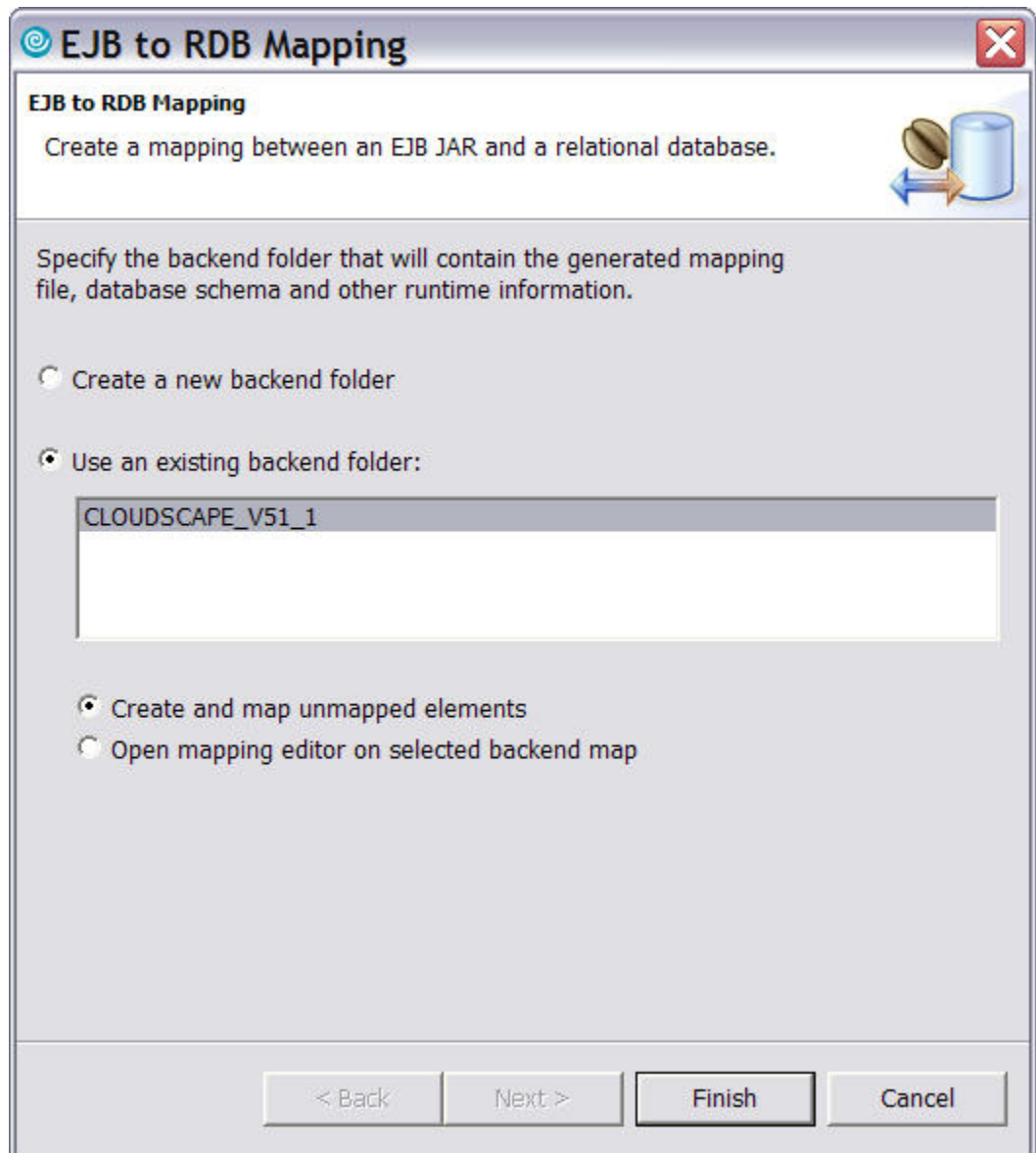
Of course, for your CMP beans to be of any use, you have to map them to a relational database. To do this, you must create database definition files for the

tables that your beans represent. The following steps show you how to map to the open source Cloudscape database, though you could map to virtually any relational database.

Because you've already defined your beans, you can engage in top-down mapping. It is possible to generate entity beans from existing tables (bottom-up mapping), and you could also map existing beans to existing tables (meet-in-the-middle mapping). To create the mapping files to Cloudscape:

1. Highlight the **BuyerBean** on the class diagram.
2. Right-click the bean, then select **EJB to RDB Mapping > Generate Map** to launch the EJB-to-RDB Mapping wizard (see Figure 9).

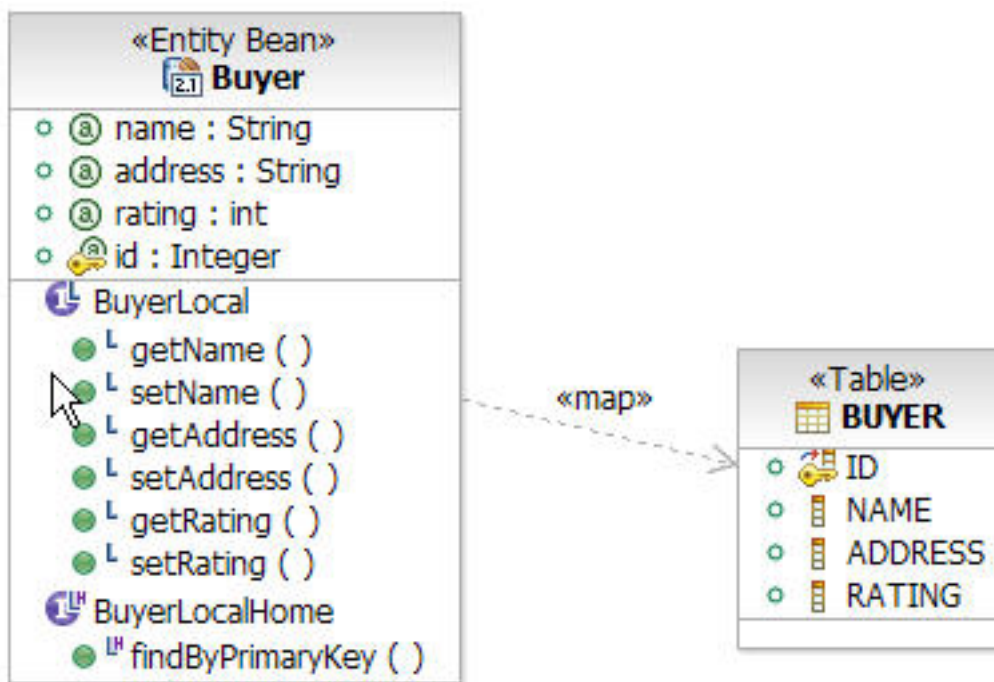
Figure 9. EJB-to-RDB mapping



3. Select the Use an existing backend folder option.
4. Verify that **CLOUDSCAPE_V51_1** is highlighted.
5. Select the Create and map unmapped elements option.
6. Click **Finish**. A mapping file is created, and the Buyer table is now displayed on the class diagram.
7. Save the diagram.

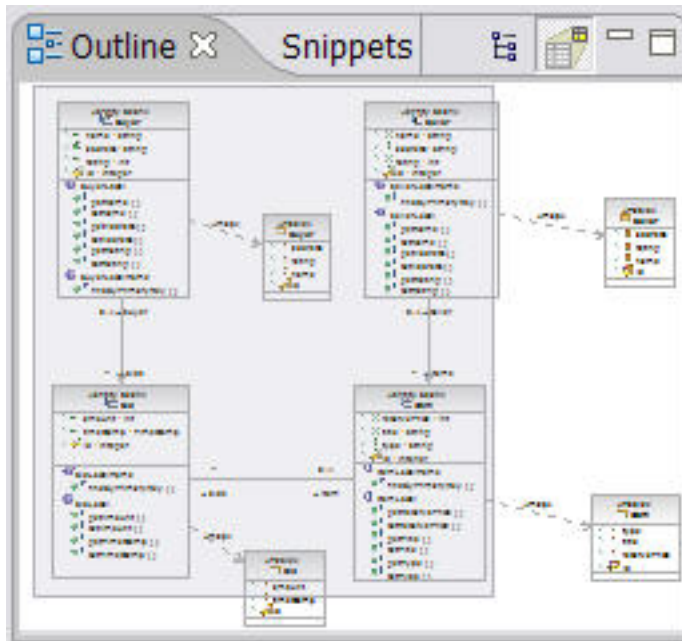
The Buyer in the class diagram should now look like Figure 10.

Figure 10. Class diagram with table visualized



Map the remaining beans, continuing to use the existing backend folder you used when mapping the BuyerBean. At this point, you might notice that you can't see your entire class diagram within the editor window (a common experience with most non-trivial class diagrams). To aid in viewing large diagrams, Rational Application Developer has an Outline view that you can use to scroll the diagram editor window to various portions of the diagram (see Figure 11). A highlighted rectangle outlines the area currently visible in the editor window.

Figure 11. Outline view in Rational Application Developer



Deploy and test your beans

At this point, your coding work is done -- now it's time to test your beans! First things first: You need to generate all the ancillary code required to make EJBs dance. In Rational Application Developer, this process is actually pretty easy. To generate the code you need, in the Project Explorer view, right-click **RAD tutorial**, then select **Deploy**.

This process might take a while -- be patient. When it finishes, if you expand **ejbModule** and then expand **tutorial**, you'll notice quite a few more classes. After you've generated the code, run your beans on the test server:

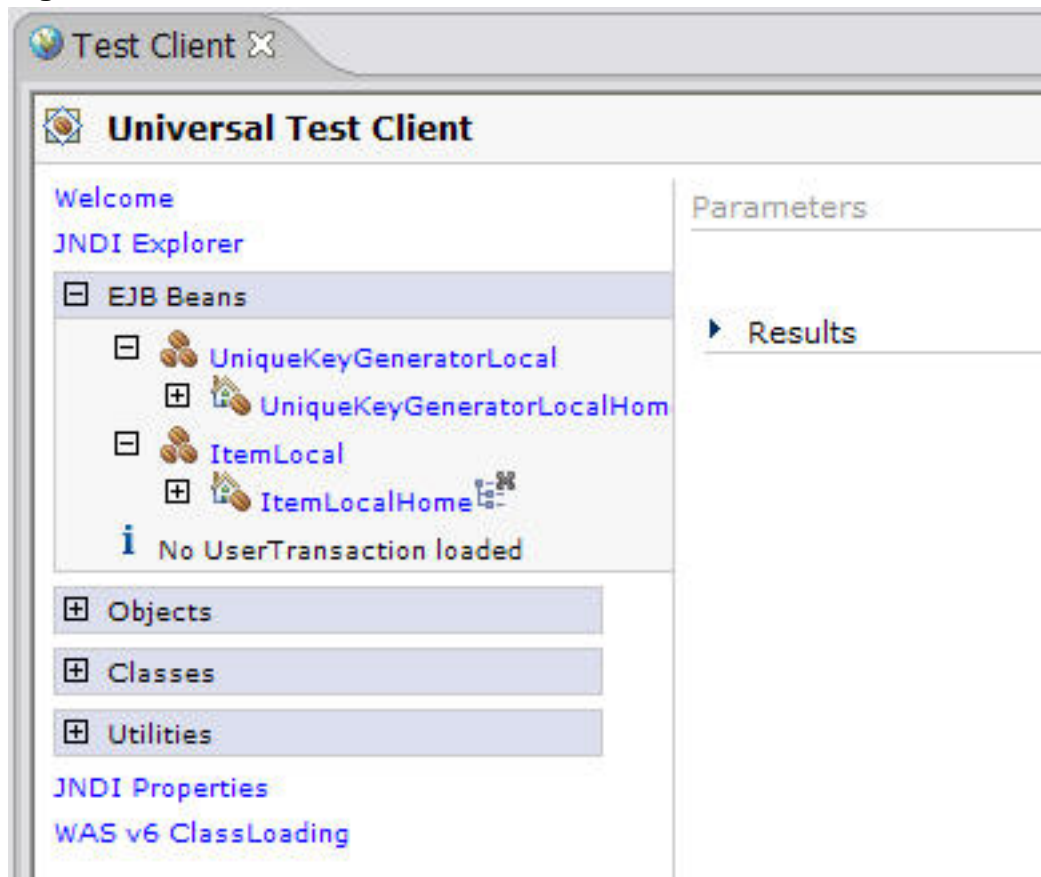
1. In the Project Explorer view, right-click **RAD tutorial** and select **Run > Run on Server**.
2. Select the Choose an existing server option.
3. Verify that **WebSphere Application Server V6.0** is selected as the server you want to use, then click **Next**.
4. Verify that **RAD tutorialEAR** is in the Configured Projects window, then click **Next**.
5. Select the Create tables with data sources option.
6. Click **Finish**.

Again, this process might take a while. When the Table and Data Source Creator window appears, click **OK**. When the Universal Test Client opens, you are ready to test your beans. The Universal Test Client gives you an explorer-like interface to work with your beans. However, you first need to "run" your beans on the server:

1. In the Project Explorer view, expand **EJB Projects**.
2. Expand **Deployment Descriptor : RAD tutorial**.
3. Expand **Entity Beans**.
4. Right-click any bean, then select **Run > Run on Server**.
5. Click **Finish** in the Server Selection window.

Your bean should now appear in the Universal Test Client under the EJB Beans section (see Figure 12).

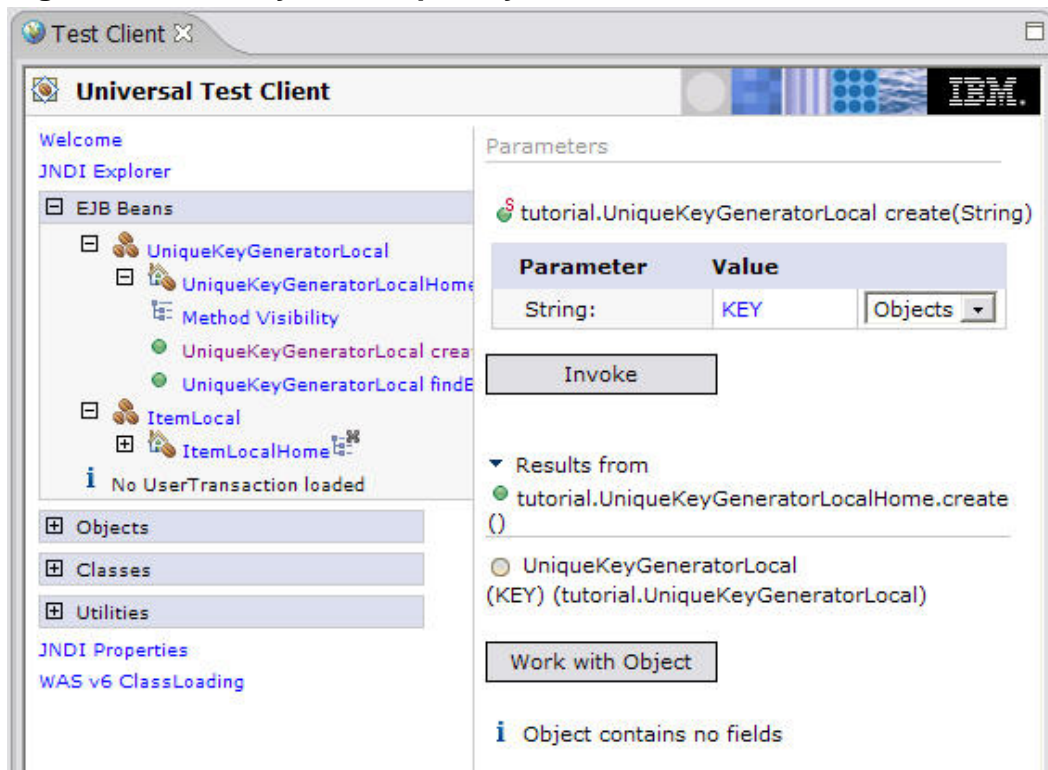
Figure 12. New bean in the Universal Test Client



You can now drill down into the methods on your beans and actually execute them. Let's start by creating your default UniqueKeyGenerator:

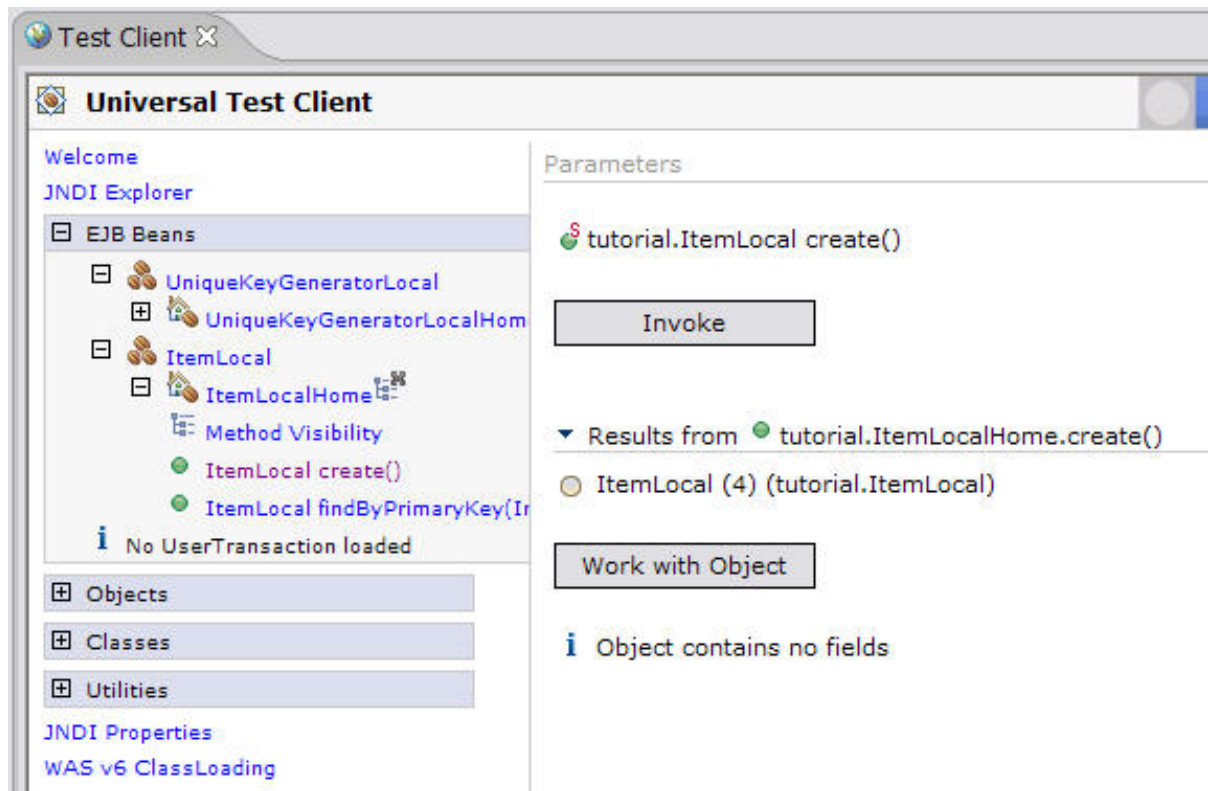
1. Expand **UniqueKeyGeneratorLocalHome**.
2. Select the `create(String)` method.
3. Because the `create` method takes parameters, the main window allows you to enter a value -- enter `KEY`.
4. Click **Invoke**. If the method works without errors, you will get local reference back (see Figure 13).

Figure 13. Create your UniqueKeyGenerator



At this point, you could choose to work with this object by clicking **Work with Object**. For now, though, you should try creating an `Item` and seeing what happens. If everything goes right, your panels should look something like Figure 14.

Figure 14. Create your item



You have created an Item bean. If you click **Work with Object**, the actual bean will be available in the Explorer window, and its public methods will be visible.

Test other methods

By now, you should have some familiarity with the test client; it's really quite simple to use. If you want to check out the other methods you've written, such as `attachBid()` and `getStartingBid()`, here is a sample process you can use to test out the Item bean:

1. Click the **JNDI Explorer** for the test client.
2. Expand **[Local EJB Beans]**, then click **BidLocalHome**. The bean now appears in the Explorer under EJB Beans.
3. Create a bid by using the same technique you used to create your item:
 1. Expand **BidLocalHome**.
 2. Select **create()**.
 3. Click **Invoke**.

4. Click **Work with Object**.
 5. Set an amount and a timestamp. Fortunately, the complex format defaults for you.
-
4. Add the newly created bid to your existing item. When you select the `attachBid()` method, the Value drop-down menu will contain the existing bids.
 5. Create a few more bids as above and add them to the same item.
 6. Return to your Item and try out the business methods: `getStartingBid()`, `getCurrentBid()`, and `getMaximumBid()`. See if they pull out the expected bids.

Obviously, you are just scratching the surface of development and testing. However, what you've done today exploits many of the features of Rational Application Developer and should help you get started with your own projects.

Section 6. Summary

In this tutorial, you've taken your requirements and started building your application. The IBM Software Development Platform lets you easily and quickly model your use cases while at the same time generating the majority of the "plumbing" code, freeing you to focus on the business methods that your customers need. Working with both the model and the code, you saw how changing one was reflected in the other.

When you finished the code, you deployed your beans into the WebSphere environment, where you can polish your code and iron out bugs before your code is released for formal testing. The tight integration of WebSphere and the development environment greatly simplifies the move to testing.

As you move into the next phase -- testing and change requests, which is the topic of Part 3 of this series -- you'll begin to see the model taking shape. At the same time, the original requirements and functionality of the system will head closer to their final format.

Resources

Learn

- [Collating requirements for an application, Part 1](#) is the first part of this tutorial series.
- [Hibernate](#) is an open source object-relational service.
- Check out the [EJB 3.0 Early Draft Specification](#).
- "EJB 3.0 in a nutshell," Anil Sharma (*JavaWorld*, August 2004) provides a summary of what's to come in the future EJB spec.
- [IBM Software Development Platform](#) product page. Find more resources and overview information.
- "Reduce complexity with model driven development," Milena Litoiu (developerWorks, September 2004) for more on using the IBM Software Development Platform.
- "Using the new EJB Visual Editor in Rational Application Developer for WebSphere software," Manish Bhargava (developerWorks, December 2004) to learn more about visual development.
- Tackle EJB development in "Easiest breeziest EJB components: Using Rational Application Developer for WebSphere Software," Jeff K. Wilson (developerWorks, January 2005).
- Learn more about [recent enhancements](#) to IBM Rational software.
- [developerWorks Rational zone](#) provides many more resources for you.

Get products and technologies

- [Rational Application Developer Version 6.0](#)

Discuss

- [Participate in the discussion forum for this content.](#)

About the author

Nathaniel T. Schutta

Nathaniel T. Schutta, a [Studio B](#) author, is a software developer with extensive experience in the financial services industry, primarily developing Java-based Web applications. A self-proclaimed JUnit evangelist, he has contributed to two corporate Java frameworks, led several study groups, served as a mentor, and developed training

materials. A Sun Certified Web Component Developer for the Java 2 Platform Enterprise Edition 1.4, Nathaniel's primary focus is on developing usable, intuitive user interfaces.