Exercise Guide
Version 4.0

Introduction to Object-Oriented Programming

Introduction to Object-Oriented Programming

© 2017 Progress Software Corporation and/or one of its subsidiaries or affiliates. All rights reserved.

These materials and all Progress® software products are copyrighted and all rights are reserved by Progress Software Corporation. The information in these materials is subject to change without notice, and Progress Software Corporation assumes no responsibility for any errors that may appear therein. The references in these materials to specific platforms supported are subject to change.

Corticon, DataDirect (and design), DataDirect Cloud, DataDirect Connect, DataDirect Connect64, DataDirect XML Converters, DataDirect XQuery, Deliver More Than Expected, Icenium, Kendo UI, Making Software Work Together, NativeScript, OpenEdge, Powered by Progress, Progress, Progress Software Developers Network, Rollbase, SequeLink, Sitefinity (and Design), SpeedScript, Stylus Studio, TeamPulse, Telerik, Telerik (and Design), Test Studio, and WebSpeed are registered trademarks of Progress Software Corporation or one of its affiliates or subsidiaries in the U.S. and/or other countries. Analytics360, AppServer, BusinessEdge, DataDirect Spy, SupportLink, DevCraft, Fiddler, JustCode, JustDecompile, JustMock, JustTrace, OpenAccess, ProDataSet, Progress Results, Progress Software, ProVision, PSE Pro, Sitefinity, SmartBrowser, SmartComponent, SmartDataBrowser, SmartDataObjects, SmartDataView, SmartDialog, SmartFolder, SmartFrame, SmartObjects, SmartPanel, SmartQuery, SmartViewer, SmartWindow, and WebClient are trademarks or service marks of Progress Software Corporation and/or its subsidiaries or affiliates in the U.S. and other countries. Java is a registered trademark of Oracle and/or its affiliates. Any other marks contained herein may be trademarks of their respective owners.

Please refer to the Release Notes applicable to the particular Progress product release for any third-party acknowledgements required to be provided in the documentation associated with the Progress product.

Printed in the USA

Version 4.0

Progress Software Corporation (PSC) is concerned about the environment. To reduce waste and complete the recycling circle, we printed this manual and cover on stock that is recyclable. PSC has made every effort to look at environmental implications when deciding on this package.

Author: Annie Khan
Contributors: Elaine Rosenberg

Contents

GUIDED EXERCISE 1.1: SETTING UP YOUR APPLICATION DEVELOPMENT ENVIRONMENT 1
Part 1—Starting the Database Server *2*
Part 2—Setting up a workspace in Developer Studio *4*
Part 3—Setting up the Server project *7*
Guided Exercise 1.1: Setting up your application development environment, Wrap-up *13*
TRY IT 2.1: DEFINING CLASSES 15
Part 1—Creating the Emp class *16*
Part 2—Defining data members for the Emp class *17*
Part 3—Defining methods for the Emp class *18*
Part 4—Adding the include file for the ttEmployee temp-table *19*
Part 5—Creating the Dept class *20*
Part 6—Defining data members for the Dept class *21*
Part 7—Defining a constructor and methods for the Dept class *22*
Solution, Part 1—Creating the Emp class *23*
Solution, Part 2—Defining data members for the Emp class *26*
Solution, Part 3—Defining methods for the Emp class *30*
Solution, Part 4—Adding the include file for the ttEmployee temp-table *35*
Solution, Part 5—Creating the Dept class *37*
Solution, Part 6—Defining data members for the Dept class *39*
Solution, Part 7—Defining a constructor and methods for the Dept class *41*
Try It 2.1: Defining classes, Wrap-up *45*
TRY IT 2.2: WORKING WITH CLASSES 47
Part 1—Implementing the methods for the Emp class *48*
Part 2—Implementing the methods for the Dept class *49*
Solution, Part 1—Implementing the methods of the Emp class *50*
Solution, Part 2—Implementing the methods for the Dept class *52*
Try It 2.2: Working with classes, Wrap-up *54*
TRY IT 2.3: TESTING CLASSES 55
Part 1—Setting up a Test Project *56*
Part 2—Writing the test procedure for the Emp class *57*
Part 3—Testing the Emp class *58*
Part 4—Writing the test procedure for the Dept class *59*
Part 5—Testing the Dept class *61*
Solution, Part 1—Setting up a Test project *62*
Solution, Part 2—Writing the test procedure for the Emp class *68*
Solution, Part 3—Testing the Emp class *72*
Solution, Part 4—Writing the test procedure for the Dept class *74*
Solution, Part 5—Testing the Dept class *80*
Try It 2.3: Testing classes, Wrap-up *81*
TRY IT 3.1: USING INHERITANCE 83
Part 1—Modify the Emp class to support its derived classes *84*
Part 2—Creating the Manager class *85*
Part 3—Defining a constructor and a data member for the Manager class *86*
Part 4—Defining methods for the Manager class *87*
Part 5—Implementing the methods for the Manager class *88*
Part 6—Importing the TeamMember and Dept classes *89*
Part 7—Testing the inheritance hierarchy *90*
Solution, Part 1—Modify the Emp class to support its derived classes *85*
Solution, Part 2—Creating the Manager class *91*
Solution, Part 3—Defining a constructor and a data member for the Manager class *92*

© 2017 Progress Software Corporation. All rights reserved.

Solution, Part 4—Defining methods for the Manager class *95*
Solution, Part 5—Implementing the methods of the Manager class *99*
Solution, Part 6—Importing the TeamMember and Dept classes *101*
Solution, Part 7—Testing the inheritance hierarchy *103*
Try It 3.1: Using inheritance, Wrap-up *108*
TRY IT 3.2: USING AN INTERFACE CLASS 109
Part 1—Creating the IBusiness Unit interface class *110*
Part 2—Defining data members for the IBusiness Unit interface class *111*
Part 3—Defining methods for the IBusiness Unit interface class *112*
Part 4—Creating the Company class *113*
Part 5—Implementing a constructor, a destructor, and methods for the Company class *114*
Part 6—Importing the Franchise class *115*
Part 7—Testing the classes *116*
Solution, Part 1—Creating the IBusiness Unit interface class *117*
Solution, Part 2—Defining data members for the IBusinessUnit interface class *119*
Solution, Part 3—Defining methods for the IBusinessUnit interface class *121*
Solution, Part 4—Creating the Company class *123*
Solution, Part 5—Implementing a constructor, a destructor, and methods for the Company class *126*
Solution, Part 6—Importing the Franchise class *128*
Solution, Part 7—Testing the classes *129*
Try It 3.2: Using an interface class, Wrap-up *132*
TRY IT 3.3: USING A SINGLETON AND CREATING CLASSES DYNAMICALLY 133
Part 1—Importing the Corporation class and the ttBusinessUnit include file *134*
Part 2—Defining static data members for the Corporation class *135*
Part 3—Defining a static constructor for the Corporation class *136*
Part 4—Adding code to the InitializeBusinessUnit() method to create instances dynamically *137*
Part 5—Testing the Corporation class *138*
Solution, Part 1—Importing the Corporation class and the ttBusinessUnit include file *139*
Solution, Part 2—Defining static data members for the Corporation class *141*
Solution, Part 3—Defining a static constructor for the Corporation class *143*
Solution, Part 4—Adding code to the InitializeBusinessUnit() method to create instances dynamically *144*
Solution, Part 5—Testing the Corporation class *145*
Try It 3.3: Using a singleton and creating classes dynamically, Wrap-up *146*
TRY IT 3.4: USING EVENTS 147
Part 1—Defining and publishing an event in the Manager class *148*
Part 2—Modify the Dept class to subscribe to the event *149*
Part 3—Testing the Dept class event *150*
Solution, Part 1—Defining and publishing an event in the Manager class *151*
Solution, Part 2—Modify the Dept class to subscribe to the event *154*
Solution, Part 3—Testing the Dept class event *157*
Try It 3.4: Using events, Wrap-up *158*

© 2017 Progress Software Corporation. All rights reserved.

Guided Exercise 1.1: Setting up your application development environment

Overview

In this Guided Exercise, you will prepare your development environment for writing, testing, and debugging an ABL application.

Important: You must complete this Guided Exercise to perform subsequent Try It Exercises in this course.

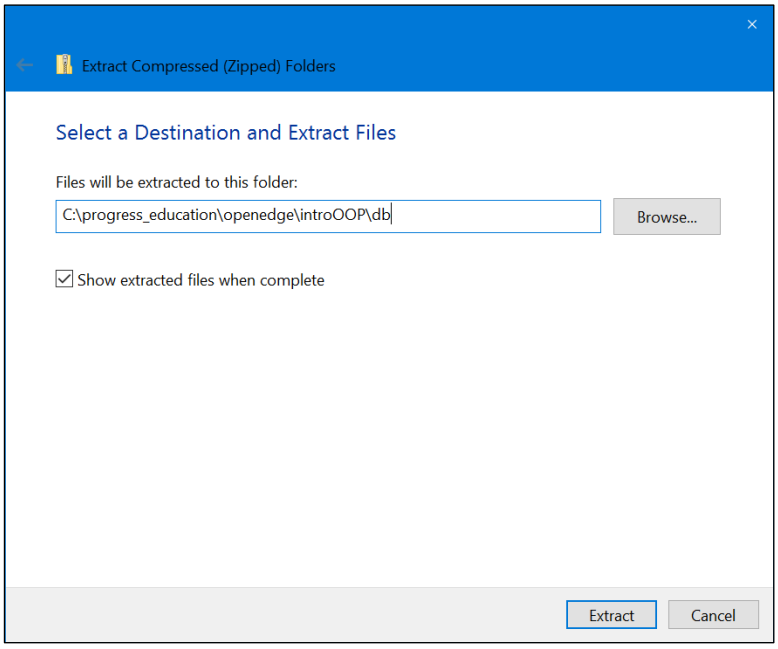
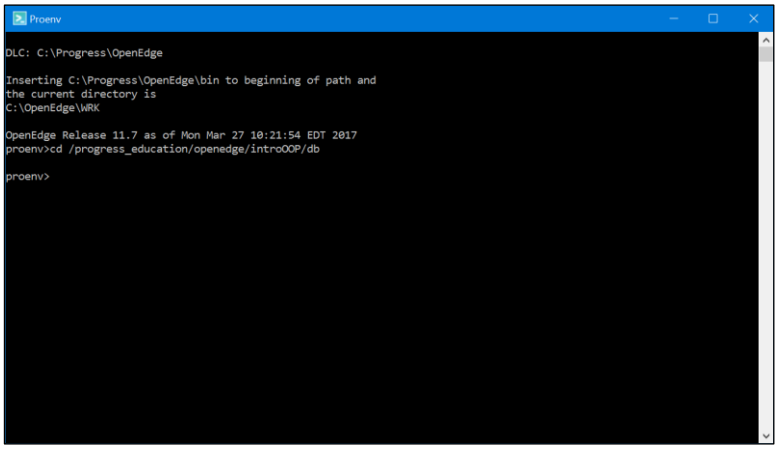
This exercise has 3 parts. The exercise steps take approximately 30 minutes to complete. You perform this exercise in your live version of Progress OpenEdge.

Before you begin, you must:

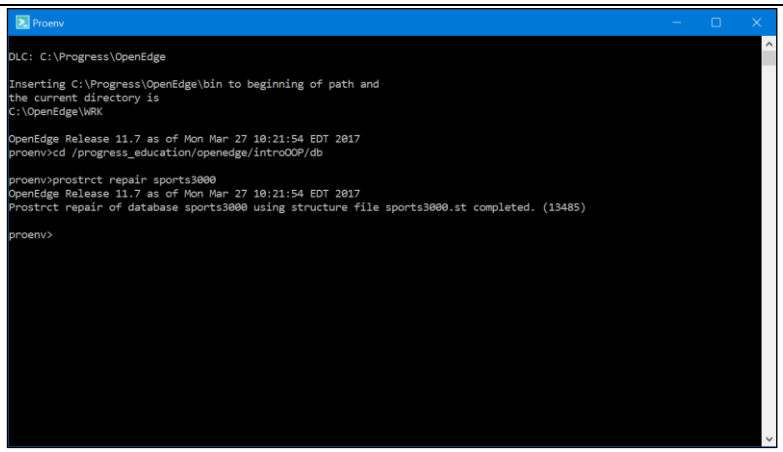
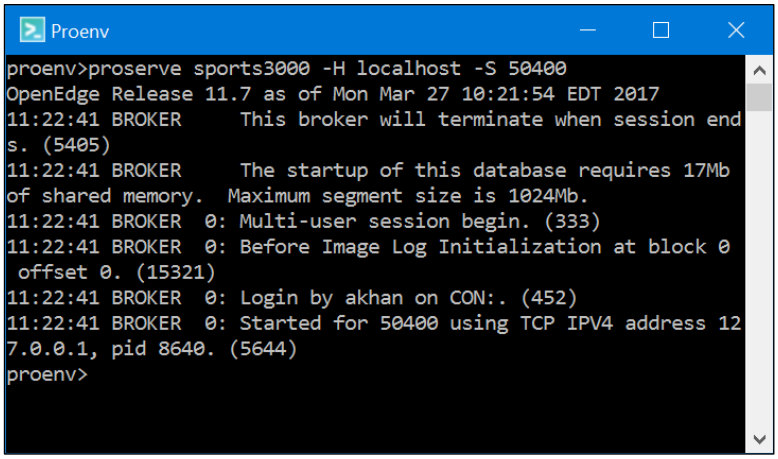
Step	Description
1.	Complete the exercise setup instructions, if you have not done so already.
2.	Install Progress Developer Studio for OpenEdge.

Part 1—Starting the Database Server

This course uses a database named sports3000. In this part of the Guided Exercise, you extract the database files and start the database server.

Step	Task
1.	Extract the files in Sports3000DatabaseFiles.zip to C:/progress_education/openedge/IntroOOP/db. The files extracted are the database files and a sub-folder of database triggers. 
2.	Open a Proenv window by selecting **Start** > **Progress** > **OpenEdge** > **Proenv** and navigate to **/progress_education/openedge/introOOP/db**. 
3.	Set the database location as follows: **prostrct repair sports3000**

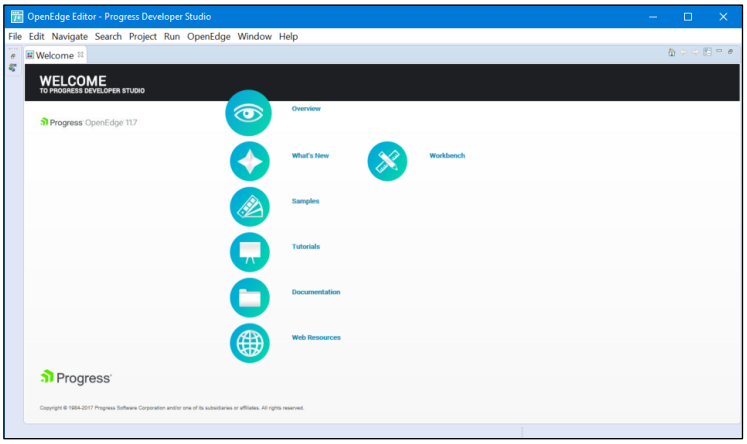
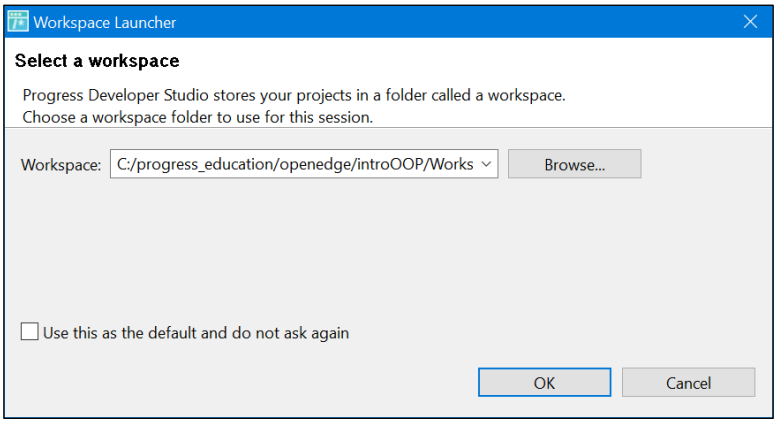
© 2017 Progress Software Corporation. All rights reserved.

	
4.	Start the database server for the sports3000 database as follows: **proserve sports3000 –H localhost –S 50400**  **Note**: If port 50400 is unavailable on your system, use a different port number.
5.	Close the Proenv window.

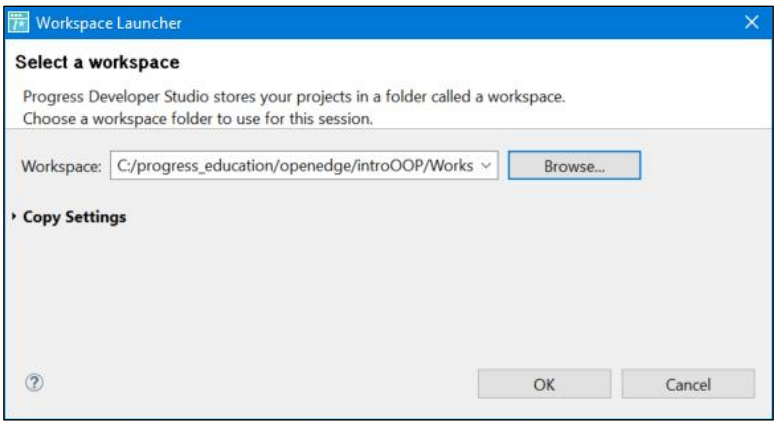
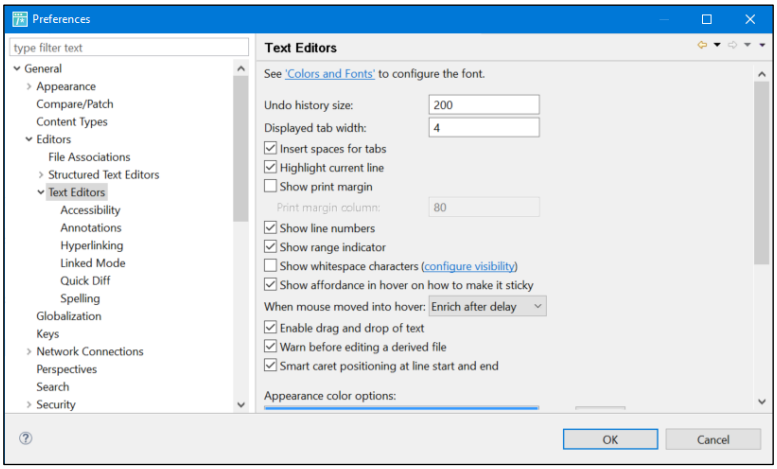
© 2017 Progress Software Corporation. All rights reserved.

Part 2—Setting up a workspace in Developer Studio

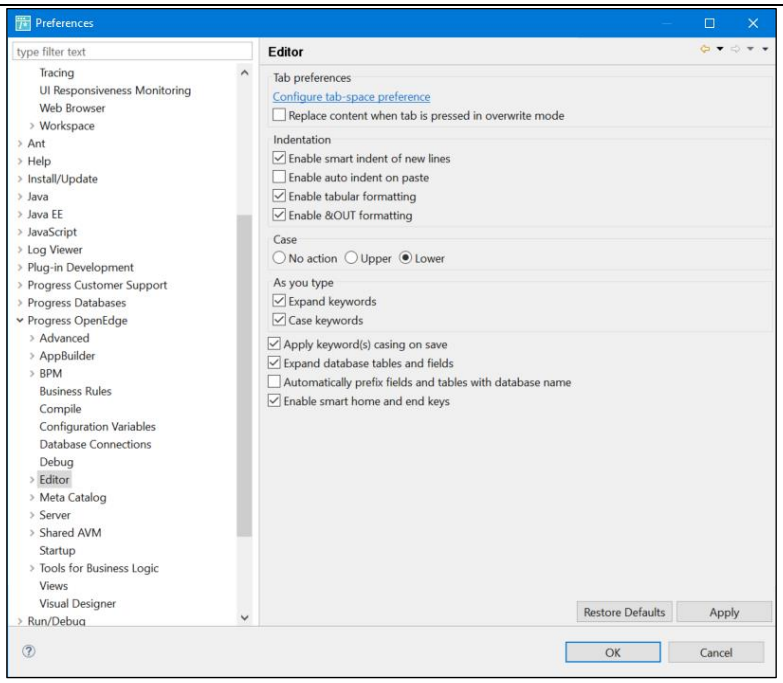
Next, you will create a new Developer Studio workspace area that will be used for your ABL development in this course.

Step	Task
1.	Start Developer Studio by selecting **Start** > **Progress** > **OpenEdge** > **Developer Studio**.
2.	You may first see the Welcome screen, as shown here. Click the Workbench icon to continue to the workspace for Developer Studio. 
3.	If this is your first time using Developer Studio, the first dialog is for you to specify where your workspace is located. If you do not see this Select a workspace window, go to step 4. a. Specify **/progress_education/openedge/IntroOOP/workspace** as the workspace location.  b. Click **OK**. c. Click the Workbench icon to start the workspace.
4.	If you have previously used Developer Studio for OpenEdge, you must switch to a new workspace.

© 2017 Progress Software Corporation. All rights reserved.

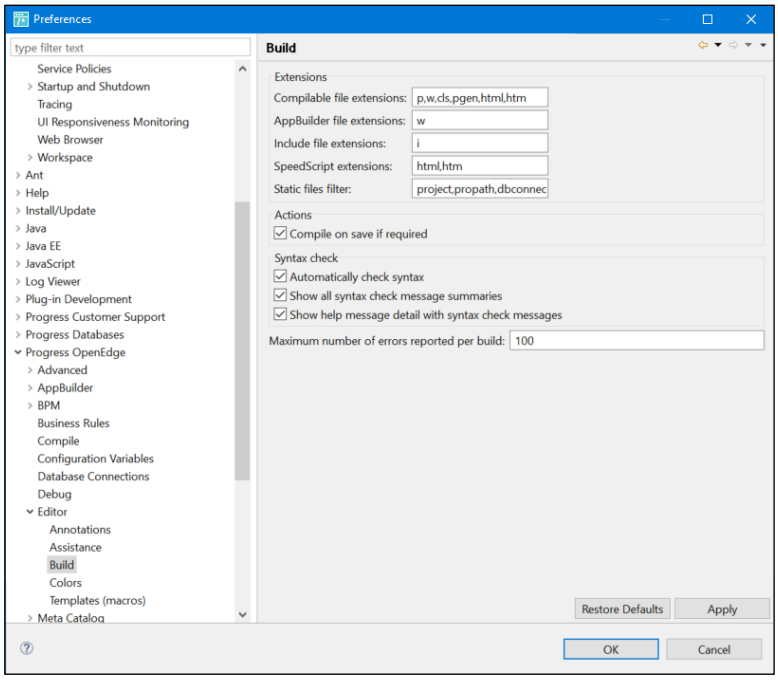
	a. Select **File** > **Switch Workspace** > **Other…**. b. Specify **/progress_education/openedge/IntroOOP/workspace** as the workspace location.  c. Click **OK**. Developer Studio restarts. d. Click the Workbench icon to start the workspace.
5.	Modify these workspace preferences as follows: a. Select **Window** > **Preferences**. b. In the Preferences window, navigate to **General** > **Editors** > **Text Editors**. c. Select **Show line numbers**. d. Click **Apply**.  e. Navigate to **Progress OpenEdge** > **Editor**. f. Select **Lower** in the **Case** area. g. Select **Expand keywords** and **Case keywords** in the **As you type** area. h. Select **Apply keyword(s) casing on save**. i. Select **Expand database tables and fields**. j. Click **Apply**.

© 2017 Progress Software Corporation. All rights reserved.



k. Navigate to **Progress OpenEdge** > **Editor** > **Build**.

l. Select all the boxes in the **Actions** and **Syntax check** areas.

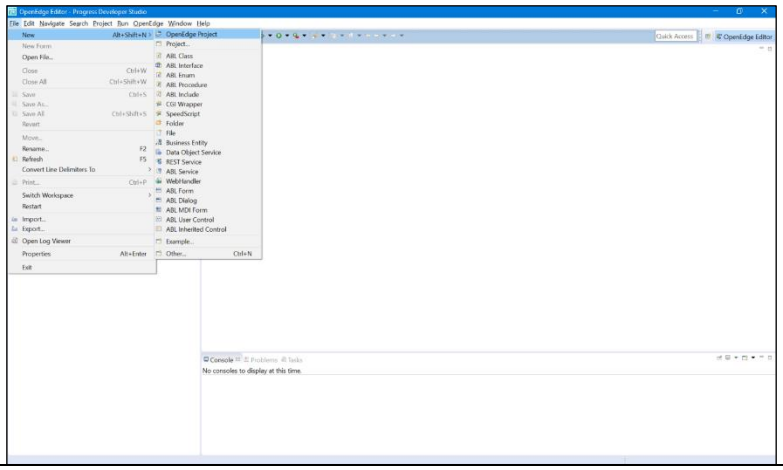
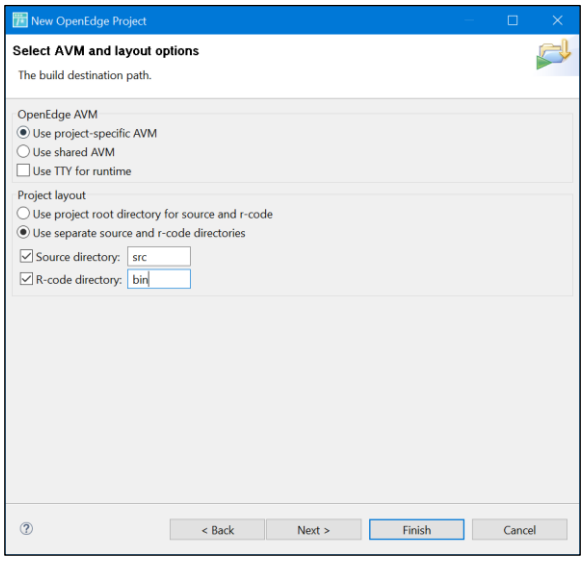


m. Click **OK**.

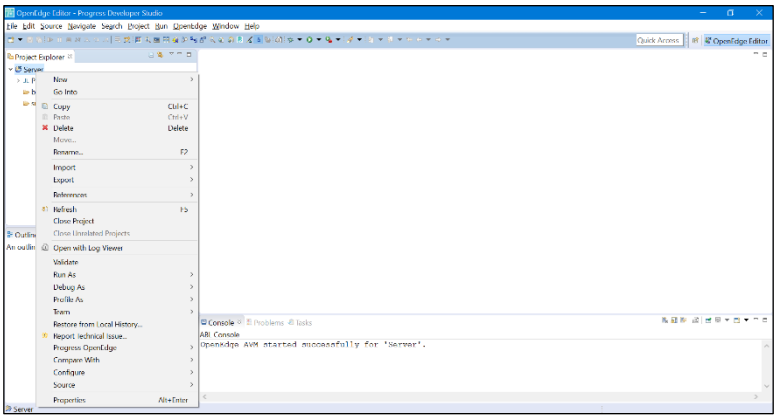
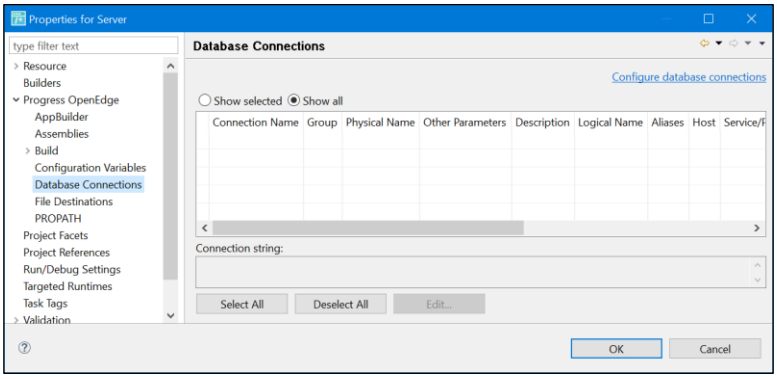
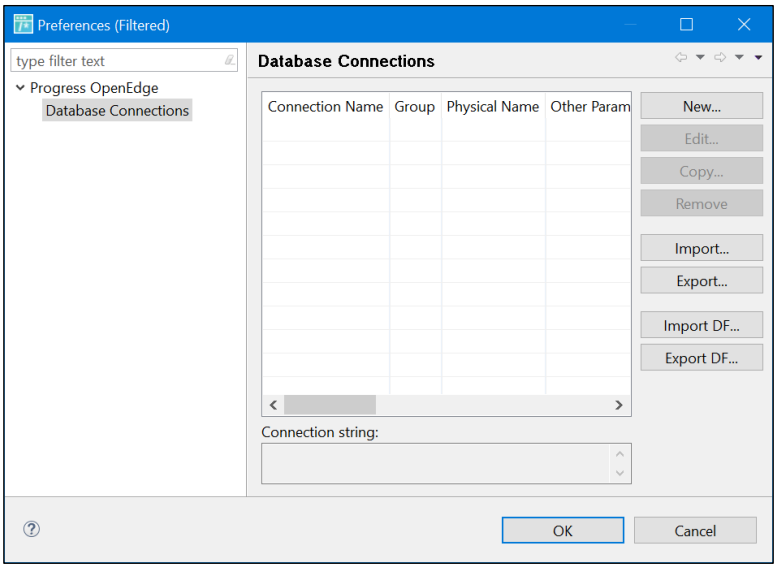
© 2017 Progress Software Corporation. All rights reserved.

Part 3—Setting up the Server project

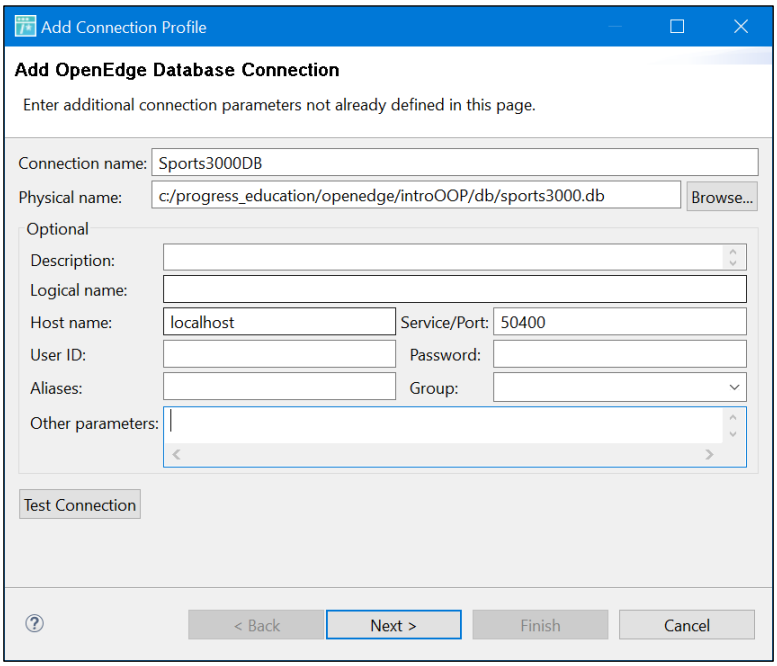
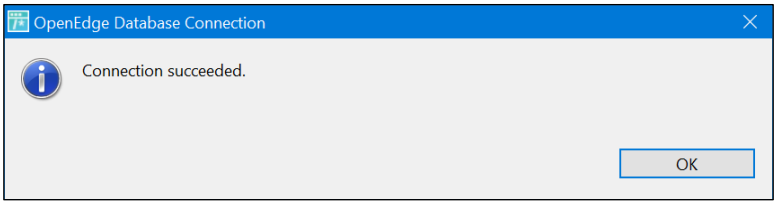
All server-side code resides in the Server project. It is from this project that the ABL code accesses the database. Follow these steps to create and configure this project to use the sports3000 database.

Step	Task
1.	Select **File** > **New** > **OpenEdge Project**. 
2.	In the New OpenEdge Project wizard: a. Enter **Server** for the project name. b. Click **Next**. c. In the Select AVM and layout options window, select **Use separate source and r-code directories**. d. Change **rcode** to **bin**.  e. Click **Finish**.

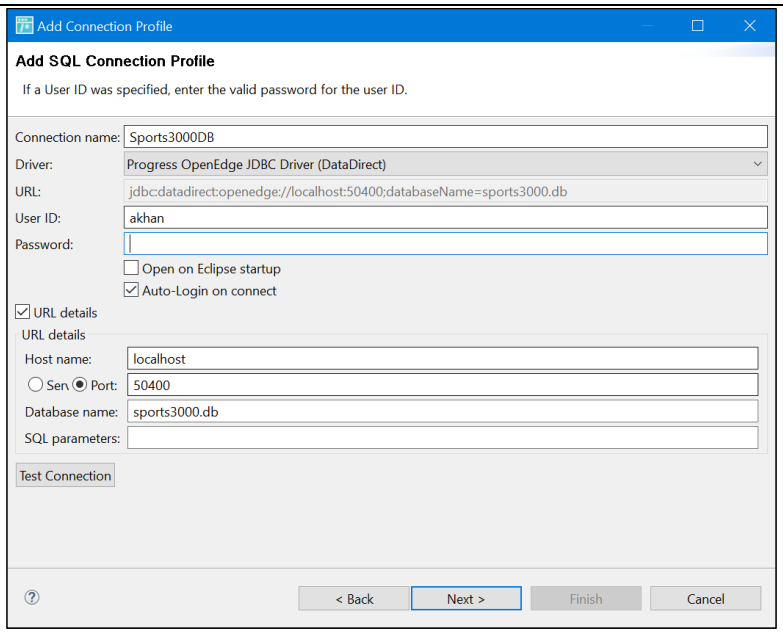
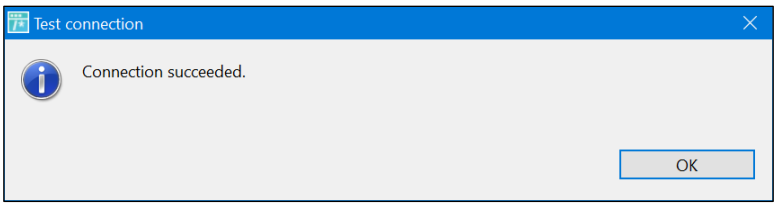
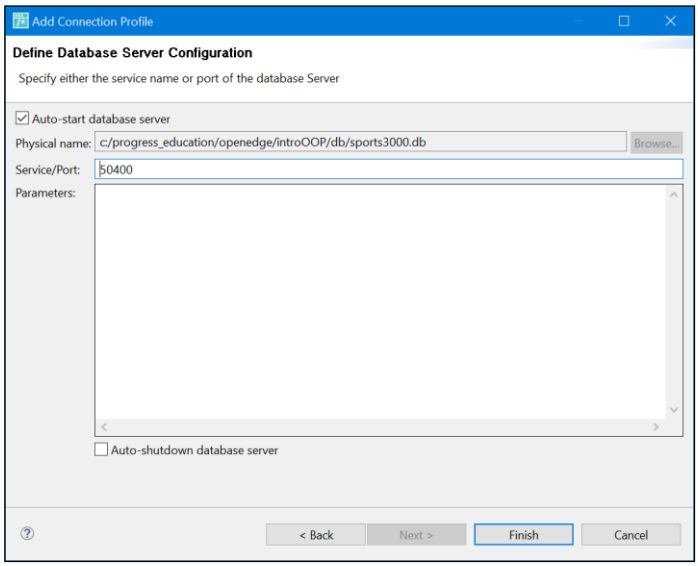
© 2017 Progress Software Corporation. All rights reserved.

3.	Next, you will add the database connection to the workspace and associate it with the **Server** project. Right-click the **Server** project and then select **Properties**. 
4.	Navigate to **Progress OpenEdge** > **Database Connections**. 
5.	Click the **Configure database connections** link. 
6.	Click **New** to open the **Add OpenEdge Database Connection** wizard.

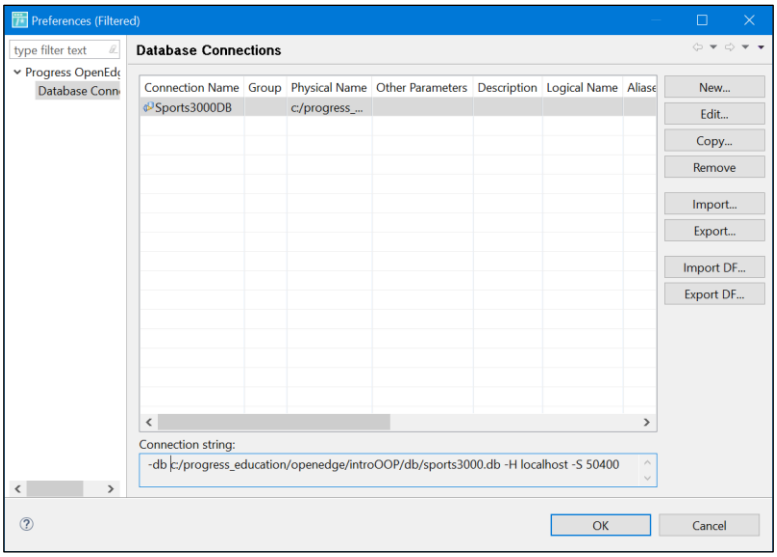
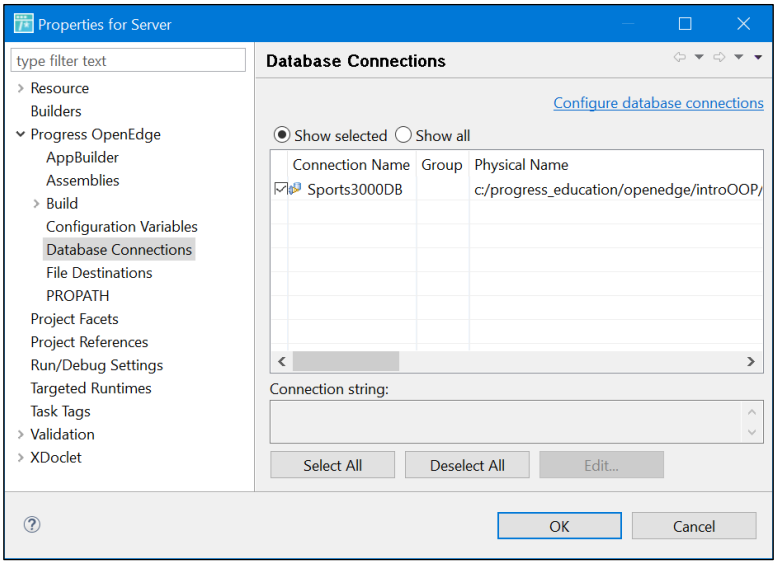
© 2017 Progress Software Corporation. All rights reserved.

7.	Enter the connection name as **Sports3000DB**.
8.	Click the **Browse** button and then navigate to and select **c:/progress_education/openedge/introOOP/db/sports3000.db**.
9.	Specify the host name as **localhost**.
10.	Specify **50400** as the port number. If port 50400 is unavailable on your system, use a different port number. It must be the same port number that you specified when you started the Database Server. 
11.	Click the **Test Connection** button.
12.	The AVM runs the code to test the connection. After the test completes successfully, click **OK**. If the connection does not succeed, you will need to investigate why it failed. Is the Database Server for the database running? 
13.	Click **Next**. The Define a SQL connection window opens.
14.	Click **Next**. The Add SQL Connection Profile window opens. Here, retain the values that have been automatically set for you by the wizard. **Note**: The Database name specified must be in lower case. If the name has upper-case letters, you must modify them to be lower case.

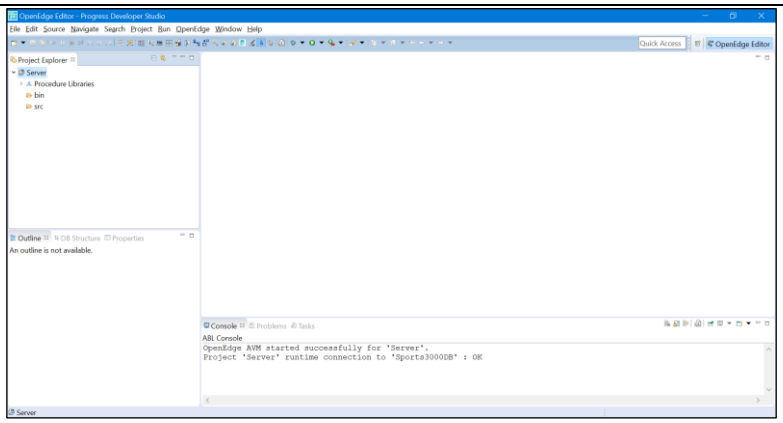
© 2017 Progress Software Corporation. All rights reserved.

	
15.	Click the **Test Connection** button. Then click **OK** when the connection succeeds. 
16.	Click Next.
17.	In the Define Database Server Configuration page, retain the values and click **Finish**. 

© 2017 Progress Software Corporation. All rights reserved.

18.	Back in the Database Connections window, click **OK**. 
19.	After the configuration is completed, the connection string is shown as: **-db c:/progress_education/openedge/introOOP/db/sports3000.db -H localhost -S 50400**
20.	Back in the Properties for Server project, Database Connections window, select the Sports3000DB connection that you just created. 
21.	Click **OK**.

© 2017 Progress Software Corporation. All rights reserved.



© 2017 Progress Software Corporation. All rights reserved.

Guided Exercise 1.1: Setting up your application development environment, Wrap-up

Exercise summary

You have set up your Developer Studio environment. You will use this environment to develop, debug, and test ABL code in the hands-on exercises of this course.

© 2017 Progress Software Corporation. All rights reserved.

Notes

© 2017 Progress Software Corporation. All rights reserved.

Try It 2.1: Defining classes

Overview

In this Try It, you will define data members, constructors, and methods for the *Emp* and *Dept* classes.

This exercise has 7 parts. The exercise steps take approximately 75 minutes to complete. You perform this exercise in your live version of Progress OpenEdge.

Before you begin

Before you begin, you must:

Step	Description
1.	Complete the Exercise Setup instructions, if you have not done so already.
2.	Complete the Guided Exercise 1.1.

Location of files:
Exercise files: **/progress_education/openedge/IntroOOP/Exercise/Lesson02**
Solution files: **/progress_education/openedge/IntroOOP/Solutions/Lesson02**

If at any time during this course, you need to restore your projects to match where you should have been prior to beginning a Try It, you can do the following:

1. Delete all projects in the workspace.
2. Import all projects in the archive (**.zip**) file for the previous Try It.
3. Ensure that Build Automatically is set for the workspace.

© 2017 Progress Software Corporation. All rights reserved.

Part 1—Creating the Emp class

In this part of the Try It, you will create the *Emp* class. This class will contain data and functionality for an employee.

If you need help, view the Solution.

Step	Task
1.	In Project Explorer, in the Server project, create a new folder named **Enterprise** under the **src** folder.
2.	Under the **Enterprise** folder, create a new folder named **HR**.
3.	In Project Explorer, navigate to the **Enterprise/HR** directory.
4.	In this directory, create a class named *Emp*. This class will not have a destructor, but it will have a default constructor. **Hint**: Use the *New ABL class* wizard.

© 2017 Progress Software Corporation. All rights reserved.

Part 2—Defining data members for the Emp class

Next, you will define the data members for the *Emp* class.

If you need help, view the Solution

Step	Task
1.	Define these properties that will be *public* with *private* setters: • *FirstName* as *character* • *LastName* as *character* • *JobTitle* as *character* • *EmpNum* as *integer* • *VacationHours* as *integer* **Hint**: Use the *Add Property* wizard.
2.	Define the *Address* property as *character*. It will be *public*.
3.	Define the *PostalCode* property as *character*. It will be *public* and will have an implementation for its setter.
4.	Define the *Phones* property as an array of size 3. It will be *public* and will have implementations for both of its accessors.
5.	Save your file. Ensure that it compiles without errors.

© 2017 Progress Software Corporation. All rights reserved.

Part 3—Defining methods for the Emp class

Next, you will define the methods for the *Emp* class.

If you need help, view the Solution.

Step	Task
1.	Define the *public Initialize*() method, which returns *void*. The input parameters for this method will be values for each of the data members of the class.
2.	Define the *public SetVacationHours*() method, which returns *void*. It has a single *input* parameter, *pHours* of type *integer*
3.	Define the *public SetJobTitle*() method, which returns *void*. It has a single *input* parameter, *pJobTitle* of type *character*.
4.	Define the *public* method *GetInfo*(), which returns *character* and takes no parameters.
5.	Define the *private* method *GetName*(), which returns *character* and takes no parameters.
6.	Save your file. Ensure that it compiles without errors.

© 2017 Progress Software Corporation. All rights reserved.

Part 4—Adding the include file for the ttEmployee temp-table

In this part of the Try It, you will create the *Dept* class. This class will contain data and functionality for a department.

If you need help, view the Solution.

Step	Task
1.	In Project Explorer, in the Server project, create a new folder named **Include** under the **src** folder.
2.	In this Include folder, import the ttEmployee.i file.

© 2017 Progress Software Corporation. All rights reserved.

Part 5—Creating the Dept class

In this part of the Try It, you will create the *Dept* class. This class will contain data and functionality for a department.

If you need help, view the Solution.

Step	Task
1.	In Project Explorer, navigate to the **Enterprise/HR** directory.
2.	In this directory, create a class named *Dept*. This class will have a destructor. Also, specify the default constructor that you will later modify to take parameters. **Hint**: Use the *New ABL Class* wizard.

© 2017 Progress Software Corporation. All rights reserved.

Part 6—Defining data members for the Dept class

Next, you will define the data members for the *Dept* class. The Dept class creates each of the Emp objects and populates the ttEmployee temp-table that represents the employees for the department.

If you need help, view the Solution.

Step	Task
1.	Define these properties that will be *public* with *private* setters: • *DeptName* as *character* • *DeptCode as character* • *ExpenseCode* as *character* • *NumEmployees as integer*
2.	Define the *private* temp-table by including the ttEmployee temp-table definition. **Hint**: Add this statement in the definitions part of the class: `{include/ttEmployee.i &ClassAccess = "private"}`
3.	Save your file. Ensure that it compiles without errors.

© 2017 Progress Software Corporation. All rights reserved.

Part 7—Defining a constructor and methods for the Dept class

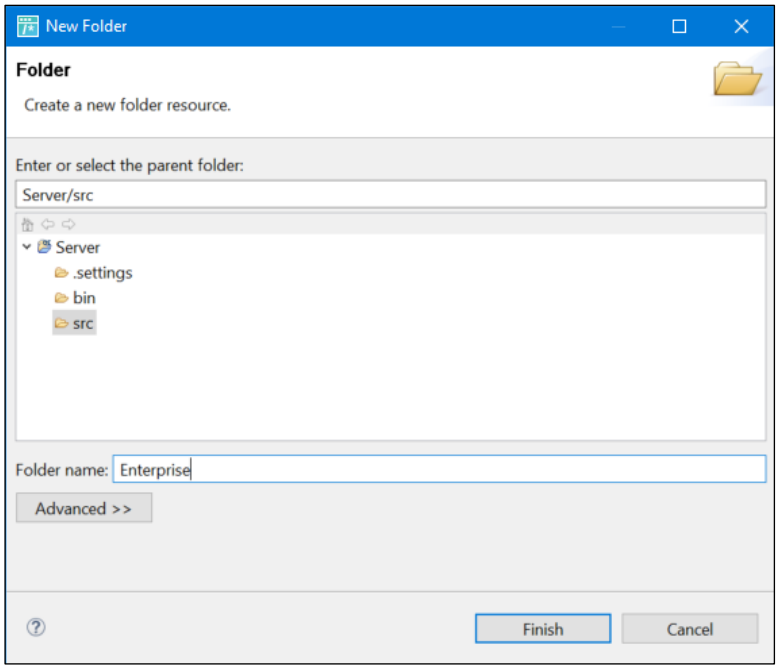
Next, you will modify the constructor and define the methods for the *Dept* class.

If you need help, view the Solution.

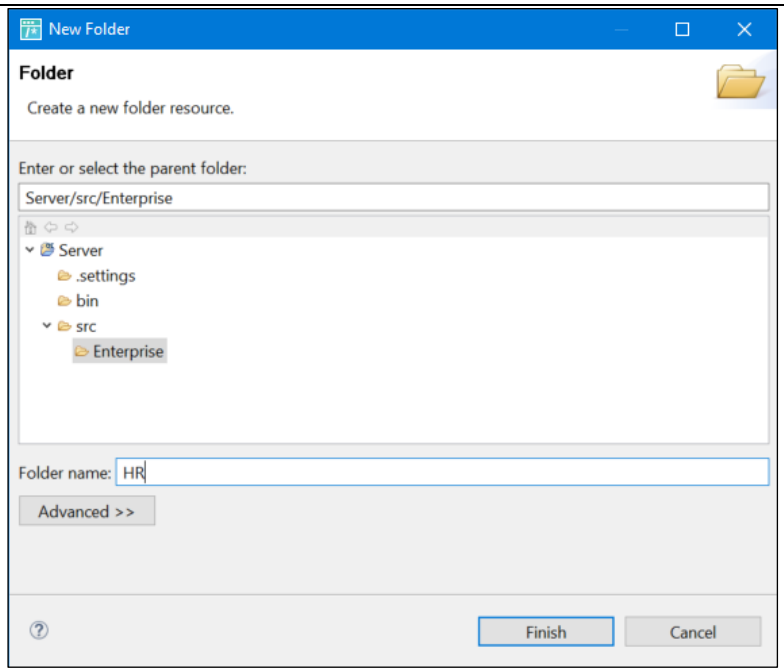
Step	Task
1.	Modify the default constructor to take as *input* parameters *pDeptName*, *pDeptCode*, and *pExpenseCode*.
2.	Since the Dept class uses instances of the Emp class, you must add a using statement for the Emp class at the beginning of the Dept class definition. Add this using statement as follows after the using Progress.Lang.* statement: `using Enterprise.Hr.Emp.` **Note**: You will learn more about using statements later in this lesson. Now, you will define the *public AddEmployee*() method that returns *void*. It takes a single *input* parameter, *pEmployee*, which is a reference to an *Emp* instance.
3.	Define the *public AddEmployee*() method that returns *void*. It has the same parameters you defined for the *Emp Initialize*() method. You will not be able to create this method with the Add Method wizard since there is already a method with the same name. Copy and paste the existing *AddEmployee*() method and then modify the parameters.
4.	Define the *public* method *GetEmployee*() that returns an *instance of Emp* and takes the first and last name parameters.
5.	Define the *public* method *GetEmployees*() that returns void, but has a ttEmployee table as an output parameter. **Hint**: The output table ttEmployee will be the parameter for this method. Having this as an output parameter, the entire ttEmployee temp-table is returned to the caller of this method.
6.	Save your file. Ensure that it compiles without errors.

© 2017 Progress Software Corporation. All rights reserved.

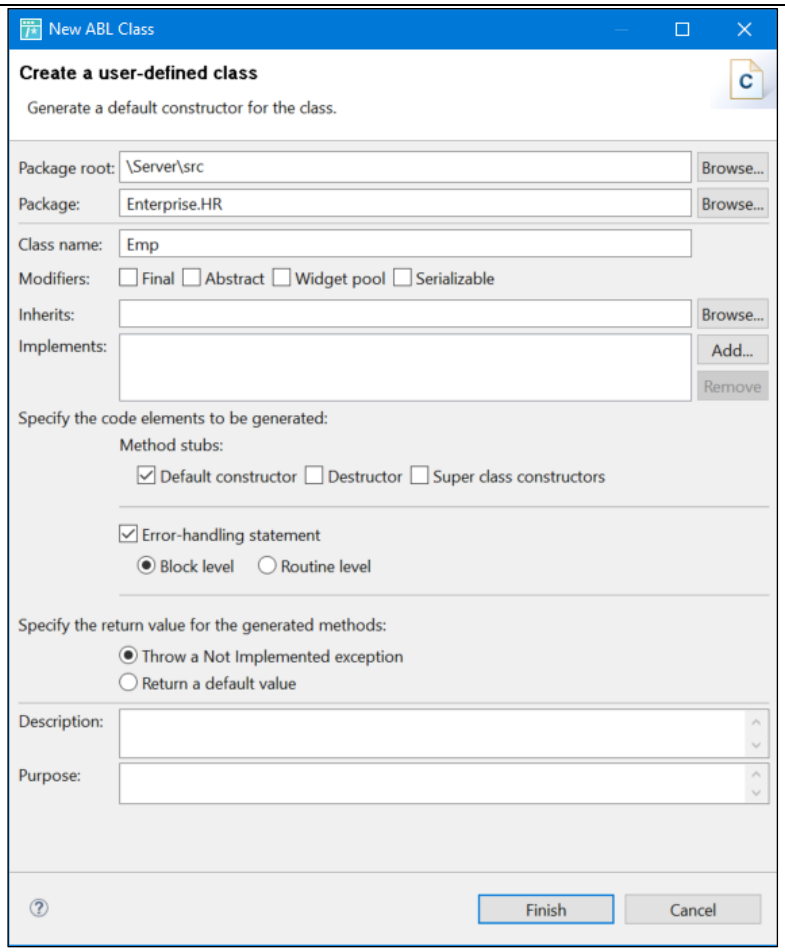
Solution, Part 1—Creating the Emp class

Step	Solution instructions
1.	In Project Explorer, in the **Server** project, create a new folder named **Enterprise** under the **src** folder. a. Right-click **src**. b. Select **New** > **Folder**. c. Enter **Enterprise** for the Folder name.  d. Click **Finish**.
2.	Under the **Enterprise** folder, create a new folder named **HR**. a. Right-click **Enterprise**. b. Select **New** > **Folder**. c. Enter **HR** for the Folder name.

© 2017 Progress Software Corporation. All rights reserved.

	 d. Click **Finish**.
3.	In Project Explorer, navigate to the **Enterprise/HR** directory.
4.	In this directory, create a class named *Emp*. This class will not have a destructor, but it will have a default constructor. **Hint**: Use the *New ABL class* wizard. a. Right-click **HR**. b. Select **New** > **ABL Class**. c. Enter **Emp** for the Class name. d. Select **Default constructor**. You can optionally add information to the description.

© 2017 Progress Software Corporation. All rights reserved.



e. Click **Finish**. The **Emp.cls** file opens in the editor.

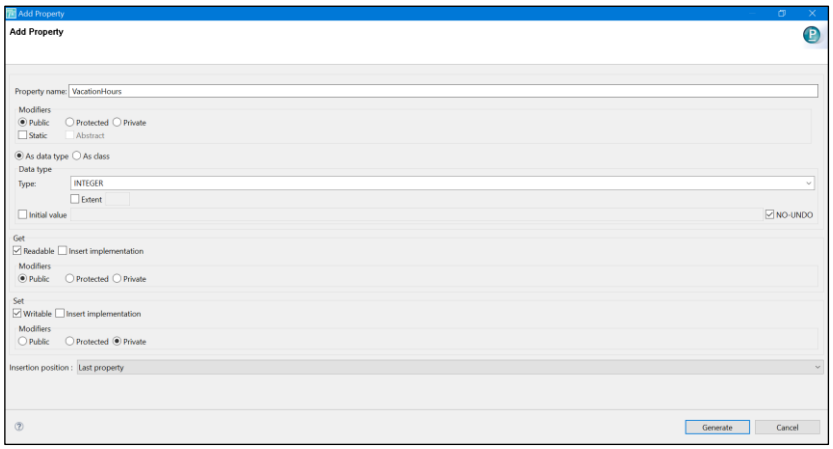
The generated file (without comments) should appear as follows:

```
using Progress.Lang.*.
block-level on error undo, throw.
class Enterprise.HR.Emp:
   constructor public Emp (  ):
      super ().
   end constructor.
end class.
```


© 2017 Progress Software Corporation. All rights reserved.

Solution, Part 2—Defining data members for the Emp class

Step	Solution instructions

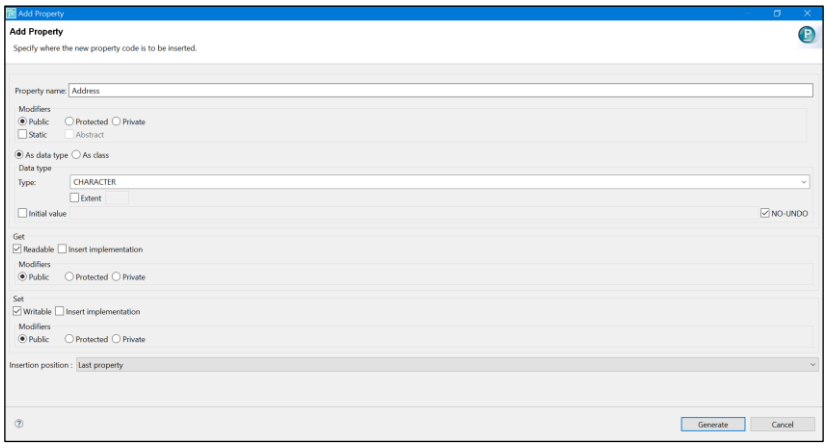
1.	Define these properties that will be *public* with *private* setters:
	• *FirstName* as *character*
	• *LastName* as *character*
	• *JobTitle* as *character*
	• *EmpNum* as *integer*
	• *VacationHours* as *integer*
	Hint: Use the *Add Property* wizard.
	a. In the editor, place the cursor anywhere in the source file.
	b. **Right-click** and then select **Source** > **Add Property**. The Add Property wizard opens.
	c. Enter **FirstName** for the Property name.
	d. Select the **CHARACTER** data type for the property from the drop-down list.
	e. Select **Private** for the Set accessor.
	f. Select **Last property** for the insertion position.
	 
	
	g. Click **Generate**.
	Repeat these steps for the definition of the remaining properties, making sure you select the correct type for each property.
	The generated code (without comments) should appear as follows:
	`define public property FirstName as character no-undo`
	`  get.`
	`  private set.`
	`define public property LastName as character no-undo`
	`  get.`
	`  private set.`
	`define public property JobTitle as character no-undo`

© 2017 Progress Software Corporation. All rights reserved.

```
        get.
        private set.
define public property EmpNum as integer no-undo
    get.
   private set.

define public property VacationHours as integer no-undo
    get.
    private set.
```

2. Define the *Address* property as *character*. It will be *public*.

a. In the editor, place the cursor anywhere in the source file.
b. Right-click and then select **Source** > **Add Property**. The Add Property wizard opens.
c. Enter **Address** for the Property name.
d. Select the **CHARACTER** data type for the property from the drop-down list.
e. Select **Last property** for the insertion position.



f. Click **Generate**.

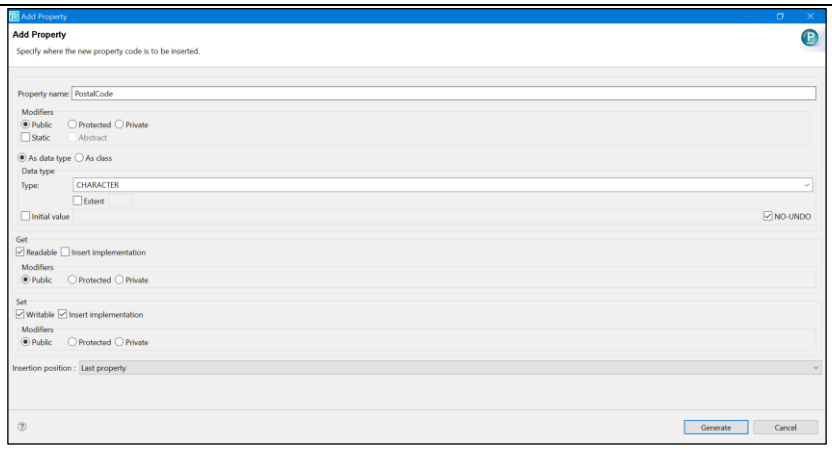
The generated code should appear as follows:

```
define public property Address as character no-undo
  get.
  set.
```

3. Define the *PostalCode* property as *character*. It will be *public* and will have an implementation for its setter.

a. In the editor, place the cursor anywhere in the source file.
b. Right-click and then select **Source** > **Add Property**. The Add Property wizard opens.
c. Enter **PostalCode** for the Property name.
d. Select the **CHARACTER** data type for the property from the drop-down list.
e. Select **Insert implementation** for the Set accessor.
f. Select **Last property** for the insertion position.

© 2017 Progress Software Corporation. All rights reserved.



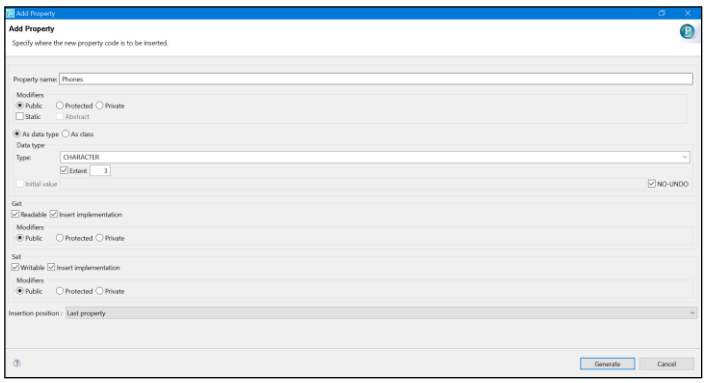
g. Click **Generate**.

The generated code should be as follows:

```
define public property PostalCode as character no-undo
  get.
  set(input arg as character):
end set.
```

4. Define the *Phones* property as an array of size 3. It will be *public* and have implementations for both of its accessors.

a. In the editor, place the cursor anywhere in the source code.
b. Right-click and then select **Source** > **Add Property**. The Add Property wizard opens.
c. Enter **Phones** for the Property name.
d. Select the **CHARACTER** data type for the property from the drop-down list.
e. Select **Extent** and then enter **3 i**n the box for the size.
f. Select **Insert implementation** for the Get accessor.
g. Select **Insert implementation** for the Set accessor.
h. Select **Last property** for the insertion position.



i. Click **Generate**.

The generated code should be as follows:

© 2017 Progress Software Corporation. All rights reserved.

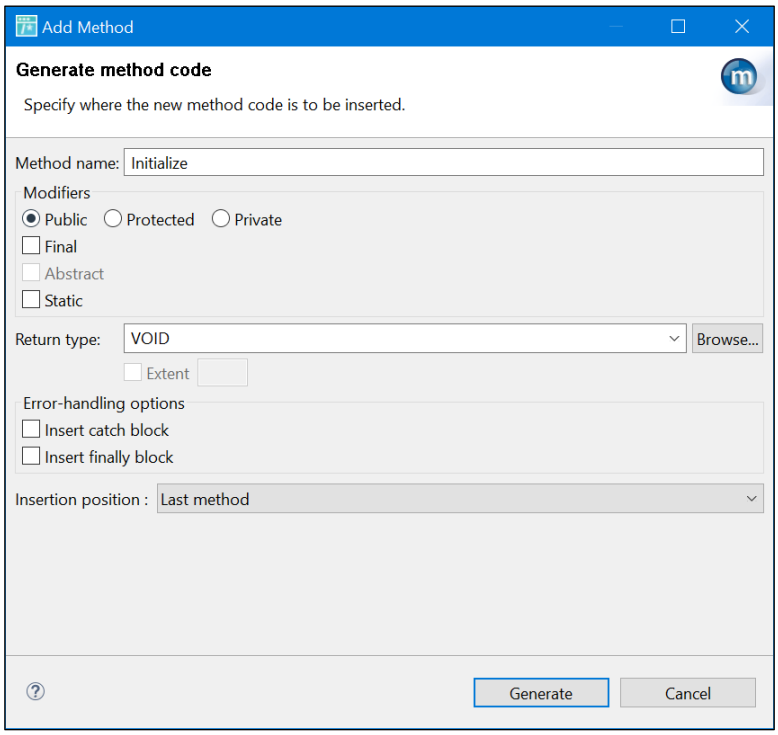
	``
	define public property Phones as character extent 3 no-
	undo
	get(input idx as integer):
	   return Phones[idx].
	end get.
	private set(input arg as character, input idx as
	integer):
	   Phones[idx] = arg.
	end set.
	``

5.	Save your file. Ensure that it compiles without errors.

© 2017 Progress Software Corporation. All rights reserved.

Solution, Part 3—Defining methods for the Emp class

Step	Solution instructions

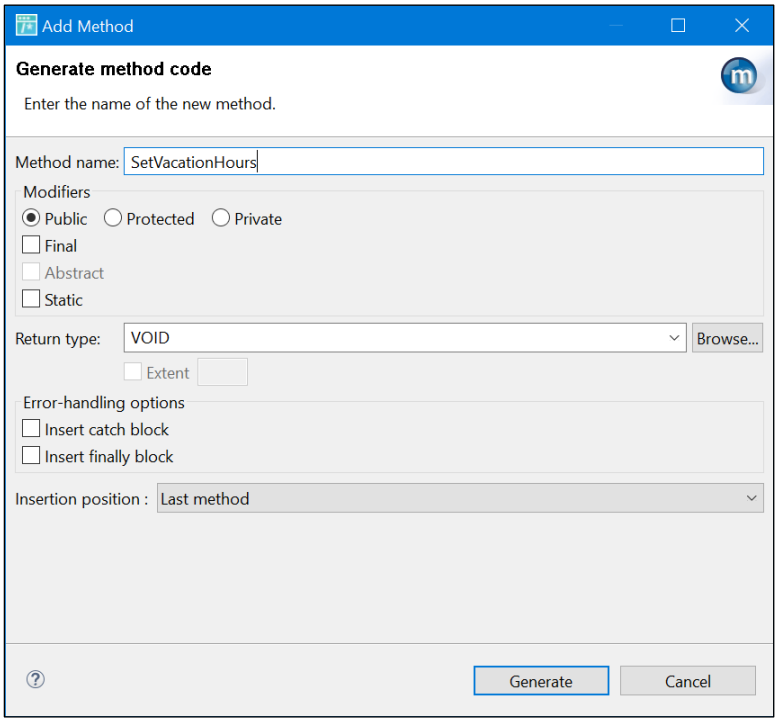
1.	Define the public *Initialize*() method, which returns *void*. The *input* parameters for this method will be values for each of the data members of the class.
	a. Place the cursor anywhere in the source file.
	b. Right-click and then select **Source** > **Add Method**.
	c. Enter **Initialize** for the method name.
	d. Select **Last method** for the insertion position.
	 
	
	e. Click **Generate**.
	You can enter information in the comment that precedes the method, or you can delete it.
	f. Modify the code in OpenEdge Editor to include the parameters for this method that correspond to the properties of this class.
	
	The definition of this method should be as follows:
	`method public void Initialize(input pEmpNum as integer,`
	`input pFirstName as character,`
	`input pLastName as character,`
	`input pAddress as character,`
	`input pPostalCode as character,`
	`input pPhones as character extent 3,`
	`input pVacationHours as integer,`
	`input pJobTitle as character ):`

© 2017 Progress Software Corporation. All rights reserved.

```
   return.
end method.
```

2. Define the *public SetVacationHours*() method, which returns *void*. It has a single *input* parameter, *pHours* of type *integer*.

a. Place the cursor anywhere in the source file.
b. Right-click and then select **Source** > **Add Method**.
c. Enter **SetVacationHours** for the method name.
d. Select **Last method** for the insertion position.



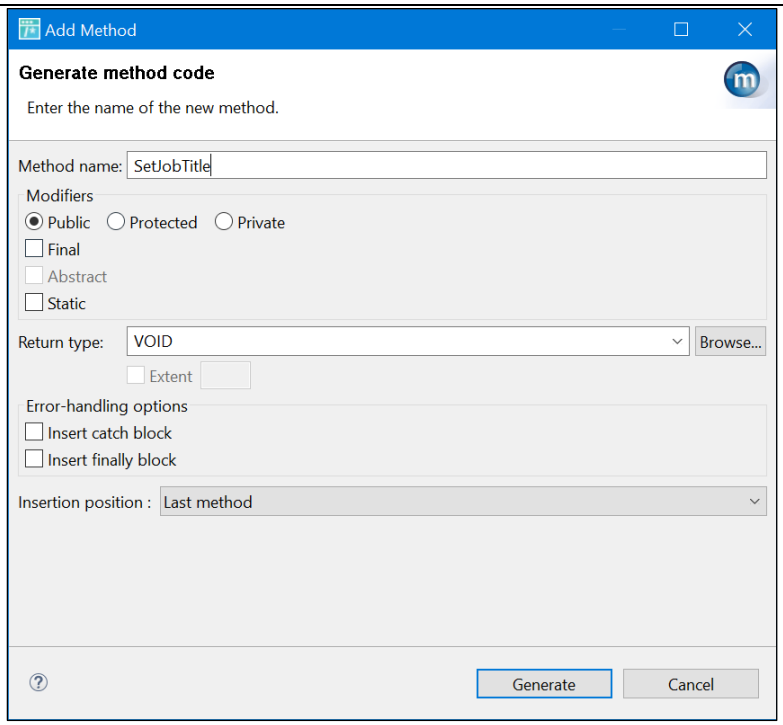
e. Click **Generate**.
You can enter information in the comment that precedes the method, or you can delete it.
f. Enter the parameter as:

```
method public void SetVacationHours(input pHours as
integer):
return.
end method.
```

3. Define the *public SetJobTitle*() method, which returns *void*. It has a single *input* parameter, *pJobTitle*, of type *character*.

a. Place the cursor cursor anywhere in the source file.
b. Right-click and then select **Source** > **Add Method**.
c. Enter **SetJobTitle** for the method name.
d. Select **Last method** for the insertion position.

© 2017 Progress Software Corporation. All rights reserved.



e. Click **Generate**.
You can enter information in the comment that precedes the method, or you can delete it.

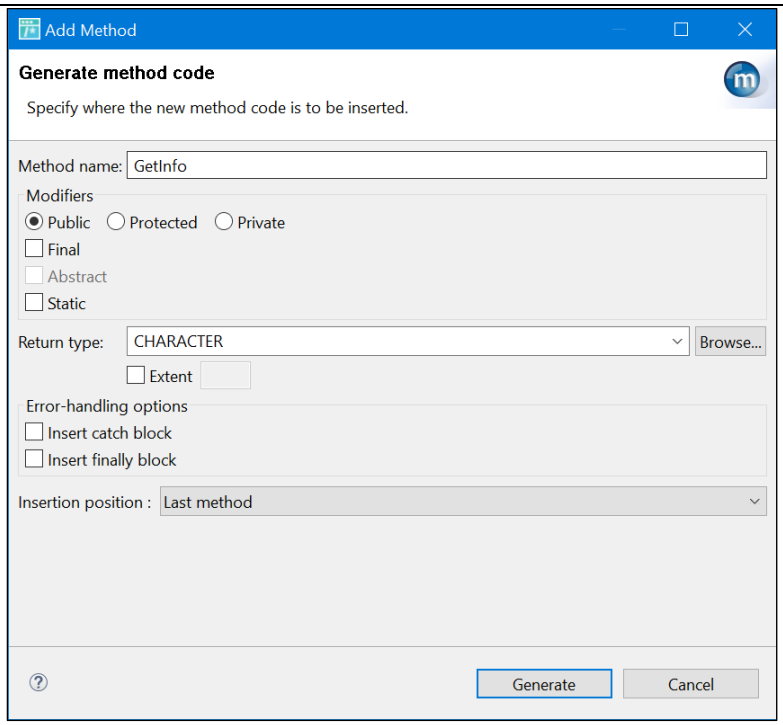
f. Enter the parameter as:

```
method public void JobTitle(input pJobTitle as
character):
return.
end method.
```

4. Define the *public* method *GetInfo*(), which returns *character* and takes no parameters.

a. Place the cursor anywhere in the source file.
b. Right-click and then select **Source** > **Add Method**.
c. Enter **GetInfo** for the method name.
d. Select the return type of **character**.
e. Select **Last method** for the insertion position.

© 2017 Progress Software Corporation. All rights reserved.



f. Click **Generate**.
You can enter information in the comment that precedes the method, or you can delete it.

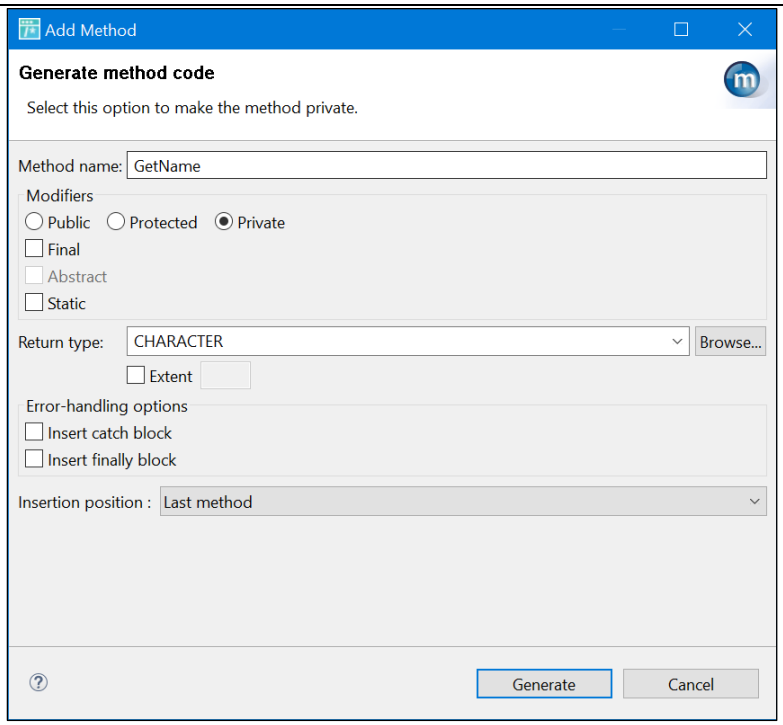
The code should appear as follows:

```
method public character GetInfo():
   define variable result as character no-undo.
   return result.
end method.
```

5. Define the *private* method *GetName*(), which returns *character* and takes no parameters.

a. Place the cursor anywhere in the source file.
b. Right-click and then select **Source** > **Add Method**.
c. Enter **GetName** for the method name.
d. Select **Private** for the modifier.
e. Select the return type as **character**.
f. Select **Last method** for the insertion position.

© 2017 Progress Software Corporation. All rights reserved.



g. Click **Generate**.
You can enter information in the comment that precedes the method, or you can delete it.

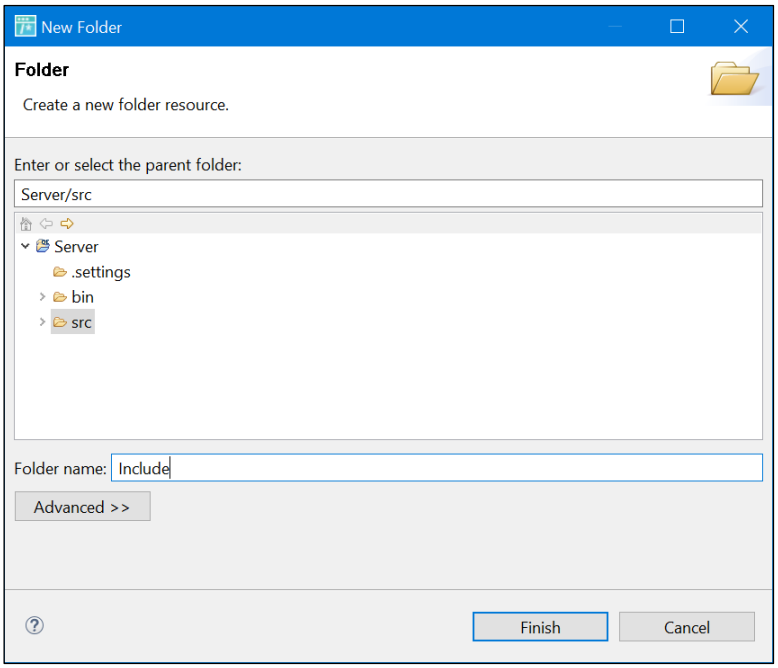
The code should appear as follows:

```
method private character GetName():
   define variable result as character no-undo.
   return result.
end method.
```

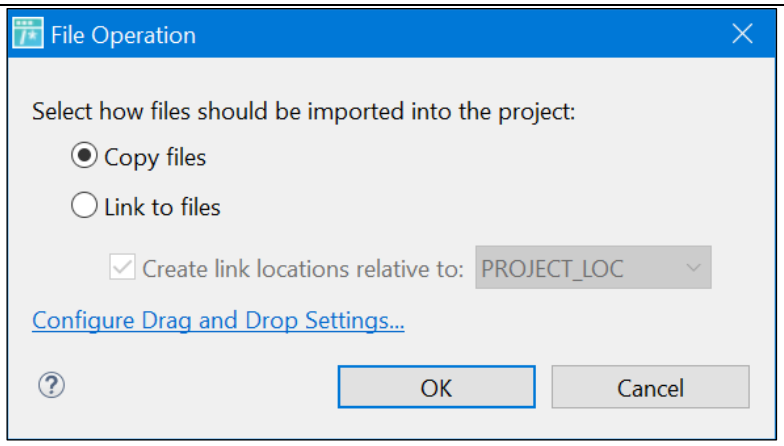
6. Save your file. Ensure that it compiles without errors.

© 2017 Progress Software Corporation. All rights reserved.

Solution, Part 4—Adding the include file for the ttEmployee temp-table

Step	Solution instructions
1.	In Project Explorer, in the **Server** project, create a new folder named **Include** under the **src** folder. a. Right-click **src**. b. Select **New** > **Folder**. c. Enter **Include** for the Folder name.  d. Click **Finish**.
2.	In this **Include** folder, import the **ttEmployee.i** file. a. In Windows Explorer, navigate to the **C:/progress_education/OpenEdge/introOOP /Exercise/Lesson02** directory and select the **ttEmployee.i** file. b. Drag and drop the **ttEmployee.i** file into the **Include** folder. c. Ensure that **Copy files** is selected.

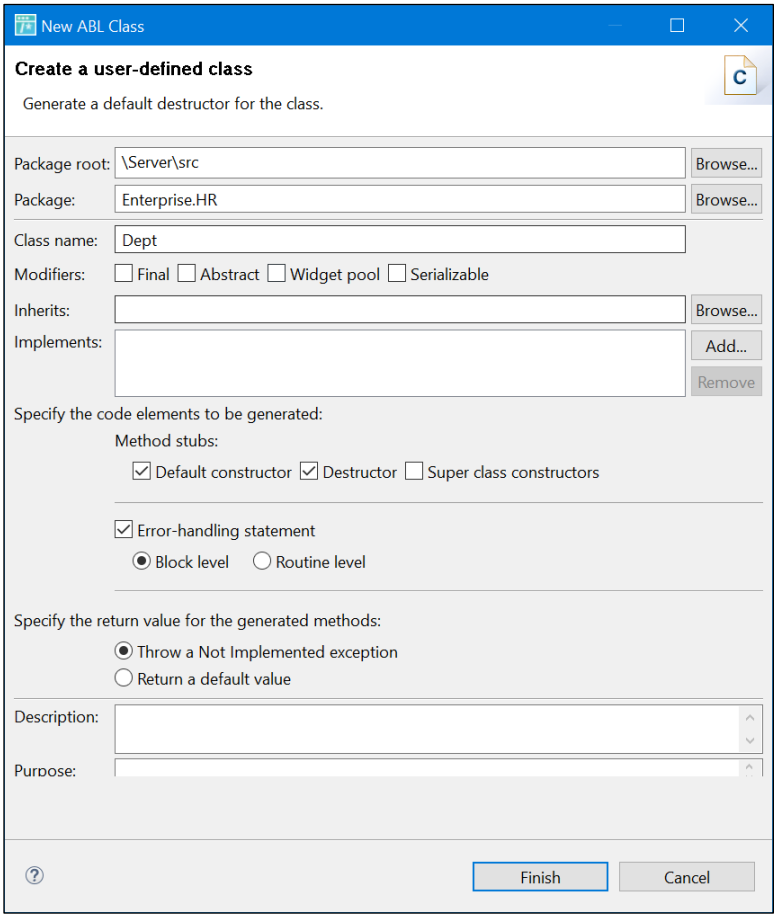
© 2017 Progress Software Corporation. All rights reserved.



d. Click **OK**.

© 2017 Progress Software Corporation. All rights reserved.

Solution, Part 5—Creating the Dept class

Step	Solution instruction
1.	In Project Explorer, navigate to the **Enterprise/HR** directory.
2.	In this directory, create a class named *Dept*. This class will have a destructor. Also, specify the default constructor that you will later modify to take parameters. **Hint**: Use the *New ABL Class* wizard. a. Select **Enterprise/HR**. b. Right-click **HR**. c. Select **New** > **ABL Class**. d. Enter **Dept** for the Class name. e. Select **Default constructor**. f. Select **Destructor**.  You can optionally add information to the description. g. Click **Finish**. The **Dept.cls** file opens in the editor. The generated code should be as follows:

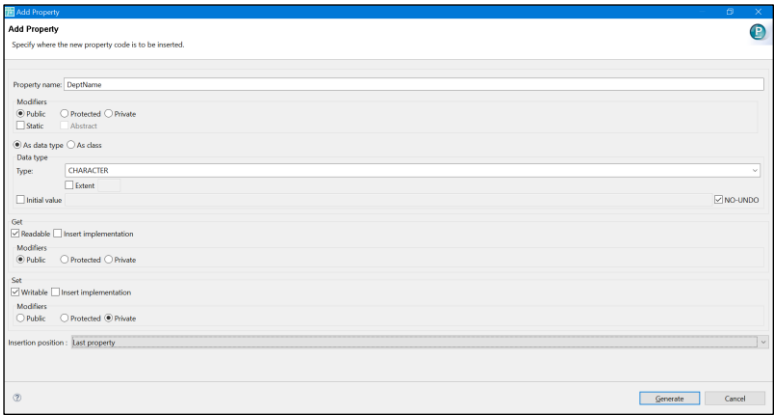
© 2017 Progress Software Corporation. All rights reserved.

```
using Progress.Lang.*.

block-level on error undo, throw.

class Enterprise.HR.Dept:

   constructor public Dept ():
      super ().
   end constructor.

   destructor public Dept ( ):

   end destructor.

end class.
```


© 2017 Progress Software Corporation. All rights reserved.

Solution, Part 6—Defining data members for the Dept class

Step	Solution instructions

1.	Define these properties which will be *public* with *private* setters:
	• *DeptName* as *character*
	• *DeptCode as character*
	• *ExpenseCode* as *character*
	• *NumEmployees as integer*
	a. In the editor, place the cursor anywhere in the source file.
	b. Right-click and then select **Source** > **Add Property**. The Add Property wizard opens.
	c. Enter **DeptName** for the Property name.
	d. Select the **CHARACTER** data type for the property from the drop-down list.
	e. Select **Private** for the Set accessor.
	f. Select **Last property** for the insertion position.
	 
	
	g. Click **Generate**.
	h. Repeat these steps for the other properties.
	
	The code should be as follows:
	`define public property DeptName as character no-undo`
	`  get.`
	`  private set.`
	`define public property DeptCode as character no-undo`
	`  get.`
	`  private set.`
	
	`define public property ExpenseCode as character no-undo`
	`  get.`
	`  private set.`
	
	`define public property NumEmployee as`
	`integer no-undo`
	`get.`

© 2017 Progress Software Corporation. All rights reserved.

	`private set.`
2.	Define the *private* temp-table by including the ttEmployee temp-table definition. a. In the editor, place the cursor at a blank line before the constructor. b. Add this code to the class file: `{include/ttEmployee.i &ClassAccess = "private"}`
3.	Save your file. Ensure that it compiles without errors.

© 2017 Progress Software Corporation. All rights reserved.

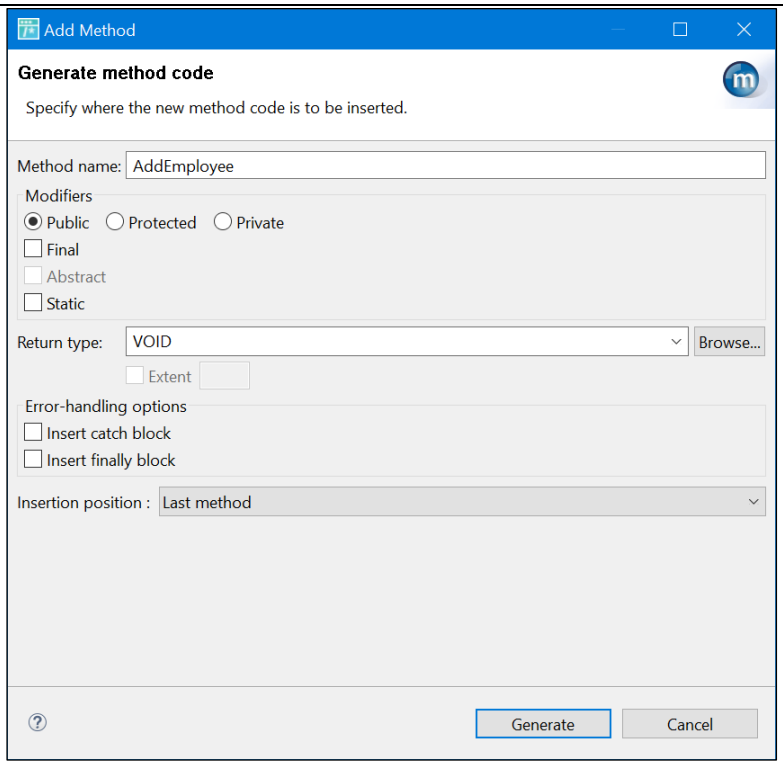
Solution, Part 7—Defining a constructor and methods for the Dept class

Step	Solution instruction

1.	Modify the default constructor to take as *input* parameters *pDeptName*, *pDeptCode*, and *pExpenseCode*.
	a. Place your cursor in the parameters area for the constructor.
	b. Add code for these parameters:
	`constructor public Dept(input pDeptName as character,`
	`input pDeptCode as character,`
	`input pExpenseCode as character):`
	`super ().`
	`end constructor.`

2.	Since the Dept class uses instances of the Emp class, you must add a using statement for the Emp class at the beginning of the Dept class definition. Add this using statement as follows after the using Progress.Lang.* statement: `using Enterprise.Hr.Emp.` **Note**: You will learn more about using statements later in this lesson. Now, you will define the *public AddEmployee*() method, that returns *void*. It takes a single *input* parameter, *pEmployee*, which is a reference to an *Emp* instance. a. Place the cursor anywhere in the source code. b. Right-click and then select **Source** > **Add Method**. c. Enter **AddEmployee** for the method name. d. Select **Last method** for the insertion position.

© 2017 Progress Software Corporation. All rights reserved.



e. Click **Generate**.
   You can enter information in the comment that precedes the method, or you can delete it.
f. Enter the parameter as:

```
method public void AddEmployee(input pEmployee as Emp):
return.
end method.
```

3. Define the *public AddEmployee*() method, which returns *void*. It has the same parameters you defined for the *Employee Initialize*() method. You will not be able to create this method with the Add Method wizard since there is already a method with the same name. Copy and paste the existing *AddEmployee*() method and then modify the parameters.

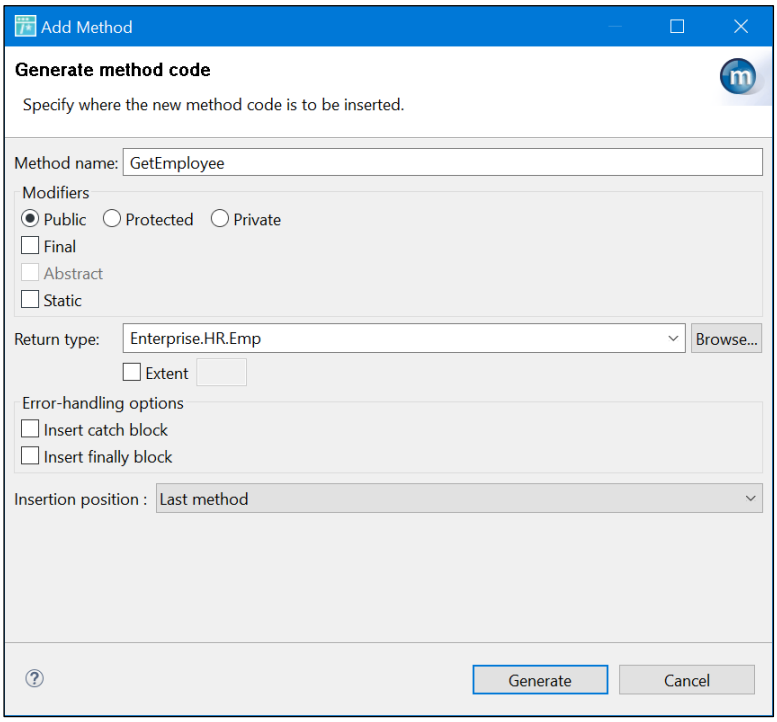
a. Select all the code in the *AddEmployee*() method and select copy.
b. Paste the code below the existing *AddEmployee*() method. You will see an error, because the methods are defined twice.
c. Delete what is currently in the parameter area and enter the parameters as:

```
method public void AddEmployee(input pEmpNum as integer,
input pFirstName as character,
input pLastName as character,
input pAddress as character,
input pPostalCode as character,
input pPhones as character extent 3,
input pVacationHours as integer,
input pJobTitle as character):
return.
end method.
```


© 2017 Progress Software Corporation. All rights reserved.

4. Define the *public* method *GetEmployee*() that returns an *instance of Emp* and takes the first and last name parameters.

a. Place the cursor anywhere in the source file.
b. Right-click and then select **Source** > **Add Method**.
c. Enter **GetEmployee** for the method name.
d. Select Browse and select the Enterprise.HR.Emp class.
e. Select **Last method** for the insertion position.



f. Click **Generate**.
You can enter information in the comment that precedes the method, or you can delete it.
g. Add code for the input parameters.

The code should appear as follows. You can change Enterprise.Hr.Emp to simply Emp since you defined the using statement earlier.

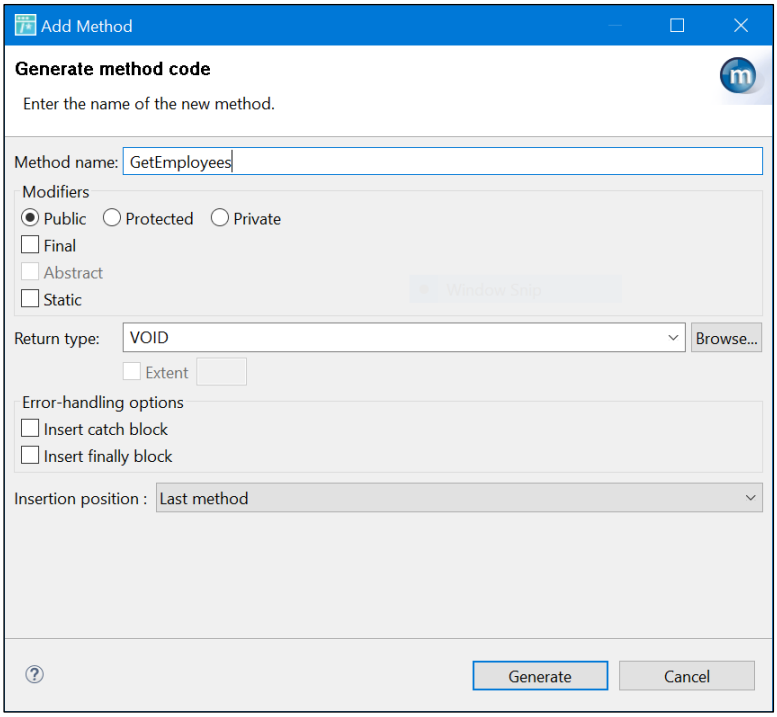
```
method public Emp GetEmployee
(input pFirstName as character,
input pLastName as character):
define variable result as Emp
return.
end method.
```

5. Define the *public* method *GetEmployees*() that returns void, but has a ttEmployee table as an output parameter.

Hint: The output table ttEmployee will be the parameter for this method. Having this as an output parameter, the entire ttEmployee temp-table is returned to the caller of this method.

a. Place the cursor anywhere in the source file.
b. Right-click and then select **Source** > **Add Method**.
c. Enter **GetEmployees** for the method name.

© 2017 Progress Software Corporation. All rights reserved.

	d. Select the return type **Void**.
	e. Select **Last method** for the insertion position.
	 
	
	f. Click **Generate**.
	g. Add the parameter information, which is output table ttEmployee.
	You can enter information in the comment that precedes the method, or you can delete it.
	The code should be as follows:
	`method public void GetEmployees(output table ttEmployee):`
	`  end method.`

6.	Save your file. Ensure that it compiles without errors.

© 2017 Progress Software Corporation. All rights reserved.

Try It 2.1: Defining classes, Wrap-up

Exercise summary

In this Try It, you defined data members, constructors, and methods for the *Emp* and *Dept* classes.

© 2017 Progress Software Corporation. All rights reserved.

Notes

© 2017 Progress Software Corporation. All rights reserved.

Try It 2.2: Working with Classes

Overview

Now that you have learned some basics about developing code for the constructors, methods, and destructor for a class, you will implement the *Emp* and *Dept* classes.

This exercise has 2 parts. The exercise steps take approximately 1 hour to complete. You perform this exercise in your live version of Progress OpenEdge.

Before you begin

Before you begin, you must complete the steps for Try It 2.1.

Location of files:
Exercise files: **/progress_education/openedge/IntroOOP/Exercise/Lesson02**
Solution files: **/progress_education/openedge/IntroOOP/Solutions/Lesson02**

If at any time during this course, you need to restore your projects to match where you should have been prior to beginning a Try It, you can do the following:

1. Delete all projects in the workspace.
2. Import all projects in the archive (**.zip**) file for the previous Try It.
3. Ensure that Build Automatically is set for the workspace.

© 2017 Progress Software Corporation. All rights reserved.

Part 1—Implementing the methods for the Emp class

In this part of the Try It, you add code to methods of the class that you have already defined.

If you need help, view the Solution.

Step	Task
1.	Implement the code for the *Initialize*() method. Assign the value of the *input* parameter to each of the data members. **Hint**: Use *Phones*[x] = *pPhones*[x] to initialize the three elements of the *Phoners* array.
2.	Implement the *SetVacationHours*() method to set *VacationHours* with the *input* parameter.
3.	Implement the *SetJobTitle*() method to set *JobTitle* with the *input* parameter.
4.	Implement the *GetInfo*() method as follows: • Set the value of *result* to be a concatenation of the value returned from *GetName*(), *Address*, *PostalCode*, *JobTitle*, and the *string* value for *VacationHours*
5.	Implement the *GetName*() method by returning the concatenation of *FirstName* and *LastName*.
6.	Save your file. Ensure that it compiles with no errors.

© 2017 Progress Software Corporation. All rights reserved.

Part 2—Implementing the methods for the Dept class

Unlike the *Emp* class that uses the default constructor and then uses the *Initialize*() method to initialize, for the *Dept* class, all instance initialization is performed in the constructor. In addition, the constructor retrieves employee data from the database to create all of the Emp instances and populate the ttEmployee temp-table. Next, you will add code to the constructor and the methods of the *Dept* class.

If you need help, view the Solution.

Step	Task
1.	Add code to the constructor to: • Set *DeptName* from the *input* parameter. • Set *ExpenseCode* from the *input* parameter. • Set *DeptCode* from the *input* parameter.
2.	Implement the *AddEmployee*() method that takes the parameter with type *Emp* to perform the following: • Create a ttEmployee record. • Set the values of the ttEmployee record from the input parameter. • Increment NumEmployees by 1.
3.	Implement the *AddEmployee*() method, which takes multiple parameters that can be used to create an *Emp* instance as follows: • Define a variable named *Empl* of type *Emp*. • Create an *Employee* instance assigning it to *Empl*. • Call the *Initialize*() method using *Empl*, providing the parameters from the *input* to this method • Create a ttEmployee record • Set the values of the ttEmployee record from the input parameters and the Emp instance. • Increment NumEmployees by 1.
4.	Implement the *GetEmployee*() method to return the reference to the *Emp* instance using the *input* to this method. Add code to the *GetEmployee*() method to find an employee based on the first and last name in the ttEmployee temp-table. If an employee is found, it should cast the EmpRef field and return the Emp instance. If an employee is not found, it should return unknown. **Hint**: You will need to cast the temp-table field ttEmployee.EmpRef to the type Emp.
5.	Add code to the destructor for the class to delete all ttEmployee records and delete their corresponding Emp objects. **Hint**: You will need to cast the temp-table field ttEmployee.EmpRef to the type Emp in order to call the Emp destructor.
6.	Note that you need not implement the GetEmployees() method. The ABL will automatically return the ttEmployee temp-table. Save your file. Ensure that it compiles without error.

© 2017 Progress Software Corporation. All rights reserved.

Solution, Part 1—Implementing the methods of the Emp class

Step	Solution instructions

1.	Implement the code for the *Initialize*() method. Assign the value of the *input* parameter to each of the data members.
	Hint: Use *Phones*[x] = *pPhones*[x] to initialize the three elements of the *Phones* array.
	
	Add this code to the method:
	
	`assign`
	`  EmpNum = pEmpNum`
	`  FirstName = pFirstName`
	`  LastName = pLastName`
	`  Address = pAddress`
	`  PostalCode = pPostalCode`
	`  Phones[1] = pPhones[1]`
	`  Phones[2] = pPhones[2]`
	`  Phones[3] = pPhones[3]`
	`  VacationHours = pVacationHours`
	`  JobTitle = pJobTitle`
	`.`

2.	Implement the *SetVacationHours*() method to set *VacationHours* with the *input* parameter. Add this code to the method: `VacationHours = pHours.`
3.	Implement the *SetJobTitle*() method to set *JobTitle* with the *input* parameter. Add this code to the method: `JobTitle = pJobTitle.`

4.	Implement the *GetInfo*() method as follows:
	• Set the value of *result* to be a concatenation of the value returned from *GetName*(), *Address*, *PostalCode*, *JobTitle*, and the *string* value for *VacationHours*
	
	Add this code to the method before *result* is returned:
	
	`result = GetName() + " " +`
	`  Address + " " + PostalCode + " " +`
	`  "Job Title: " + JobTitle + " " +`
	`  "Vacation Hours: " + string(VacationHours).`

© 2017 Progress Software Corporation. All rights reserved.

5.	Implement the *GetName*() method by returning the concatenation of *FirstName* and *LastName*. Add this code to the method before *result* is returned: `result = FirstName + " " + LastName.`
6.	Save your file. Ensure that it complies with no errors.

© 2017 Progress Software Corporation. All rights reserved.

Solution, Part 2—Implementing the methods for the Dept class

Step	Solution instructions

1.	Add code to the constructor to:
	• Set *DeptName* from the *input* parameter.
	• Set *ExpenseCode* from the *input* parameter.
	• Size the *Employees* array using the *input* parameter.
	
	Add this code to the constructor after the call to *super*():
	
	`assign`
	`  DeptName = pDeptName`
	`  DeptCode = pDeptCode`
	`  ExpenseCode = pExpenseCode`
	`  .`

2.	Implement the *AddEmployee*() method that takes the parameter with type *Emp* to perform the following:
	• Create a ttEmployee record.
	• Set the values of the ttEmployee record from the input parameter.
	• Increment NumEmployees by 1.
	
	Add this code to the method:
	
	`create ttEmployee.`
	`assign`
	`  ttEmployee.FirstName = pEmployee:FirstName`
	`  ttEmployee.LastName =`
	`  pEmployee:LastName`
	`  ttEmployee.EmpRef = pEmployee`
	`  NumEmployees = NumEmployees + 1  .`

3.	Implement the *AddEmployee*() method that takes multiple parameters that can be used to create an *Emp* instance as follows: • Define a variable named *Empl* of type *Emp*. • Create an *Emp* instance assigning it to *Empl*. • Call the *Initialize*() method using *Empl*, providing the parameters from the *input* to this method • Create a ttEmployee record. • Set the values of the ttEmployee record from the input parameters and the Emp instance. • Increment NumEmployees by 1. Add this code to the method: `define variable Empl as Emp no-undo.`

© 2017 Progress Software Corporation. All rights reserved.

```
Empl = new Emp().
Empl:Initialize(pEmpNum, pFirstName, pLastName,
                pAddress, pPostalCode, pPhones,
                pVacationHours, pJobTitle).
create ttEmployee.
assign
   ttEmployee.FirstName = pFirstName
ttEmployee.LastName = pLastName
ttEmployee.EmpRef = Empl
NumEmployees = NumEmployees + 1
.
```

4. Implement the *GetEmployee*() method to return the reference to the *Emp* instance using the *input* to this method. Add code to the *GetEmployee*() method to find an employee based on the first and last name in the ttEmployee temp-table. If an employee is found, it should cast the EmpRef field and return the Emp instance. If an employee is not found, it should return unknown.

   Hint: You will need to cast the temp-table field ttEmployee.EmpRef to the type Emp.

   Add this code to the method before it returns *result*:

```
find first ttEmployee where
ttEmployee.FirstName = pFirstName and
   ttEmployee.LastName = pLastName.
if available (ttEmployee)
  then
     result = cast(ttEmployee.EmpRef,Emp).
  else
     result = ?.
```

5. Add code to the destructor to delete all ttEmployee records and their corresponding Emp objects.

   Hint: You will need to cast the temp-table field ttEmployee.EmpRef to the type Emp in order to call the Emp destructor.

   Add this code to the destructor:

```
for each ttEmployee:
        delete object
cast(ttEmployee.EmpRef,Emp).
delete ttEmployee.
end.
```

6. Note that you need not implement the GetEmployees() method. The ABL will automatically return the ttEmployee temp-table. Save your file. Ensure that it compiles without error.

© 2017 Progress Software Corporation. All rights reserved.

Try It 2.2: Working with classes, Wrap-up

Exercise summary

In this Try It, you wrote ABL code to implement the constructors and methods for the *Emp* and *Dept* classes. In the next lesson, you will learn how to test these classes.

© 2017 Progress Software Corporation. All rights reserved.

Try It 2.3: Testing classes

Overview

In this Try It, you will develop simple test procedures to test the *Emp* class and the *Dept* class, and run them to ensure that the class code you have written executes correctly.

This exercise has 5 parts. The exercise steps take approximately 60 minutes to complete. You perform this exercise in your live version of Progress OpenEdge.

Before you begin

Before you begin, you must:

Step	Description
1.	Complete the Exercise Setup instructions, if you have not done so already.
2.	Complete Try It 2.2.

Location of files:
Exercise files: **/progress_education/openedge/IntroOOP/Exercise/Lesson02**
Solution files: **/progress_education/openedge/IntroOOP/Solutions/Lesson03**

If at any time during this course, you need to restore your projects to match where you should have been prior to beginning a Try It, you can do the following:

1. Delete all projects in the workspace.
2. Import all projects in the archive (.zip) file for the previous Try It.
3. Ensure that Build Automatically is set for the workspace.

© 2017 Progress Software Corporation. All rights reserved.

Part 1—Setting up a Test Project

In this part of the Try It, you will create a *Test* project. This project will contain procedures to test the ABL classes.

If you need help, view the Solution.

Step	Task
1.	Create a new OpenEdge Project named **Test** specifying Sports3000DB as the database and ensuring that its PROPATH is set to the Server/src and Server/bin folders.
2.	In Project Explorer, in the Test project, create a new folder named **Enterprise** under the **src** folder.
3.	Under the **Enterprise** folder, create a new folder named **HR**.

© 2017 Progress Software Corporation. All rights reserved.

Part 2—Writing the test procedure for the Emp class

You will write a procedure to create an Emp instance. You will then write statements to test each method of the *Emp* class. You will use *message* statements to write data to a file.

If you need help, view the Solution.

Step	Task
1.	In the **Enterprise/HR** directory of the Test project, create a *procedure* named **testEmp.p.**
2.	Add *using* statements after the error-handling statement. These *using* statements will ensure that the *Emp* class can be accessed.
3.	Define a variable named *EmpInstance* that will hold a reference to an instance of *Emp*.
4.	Define a variable named *Phones* as an extent with a type *character* and a fixed size of *3* with initial values.
5.	In the main block, add a statement to create an *Emp* instance, setting the reference of this instance to the *EmpInstance* variable.
6.	Add an *output to* statement in which you will write output from the test to a file named **testEmp.out**. It will be located in the **/progress_education/openedge/IntroOOP/log** directory. **Hint**: Use the full pathname of this file.
7.	Add a statement to initialize the *Emp* instance you just created with the constant values that have the same type as expected by the *Initialize*() method. For example, the employee number is 99, and the employee first name is "John". Use the *Phones* variable as input to this method.
8.	Add a *message* statement to write the employee data to the output file using the public data members and public methods of the class.
9.	Add a statement to call the *SetVacationHours*() method for the instance and set the vacation hours for this instance to *25*.
10.	Add a *message* statement to write the employee data to the output file.
11.	Add a statement to call the *SetJobTitle*() method for the instance and set the title for this instance as *Senior Architect*.
12.	Add a *message* statement to write the employee data to the output file.
13.	To test the GetInfo() method, add a *message* statement to write the employee data to the output file. You will call the *GetInfo*() method of the *Emp* class using the *EmpInstance* instance to do this.
14.	Add a statement to close the output file.
15.	Add a statement to delete the *Emp* instance.
16.	Save this file. Ensure that it compiles without errors.

© 2017 Progress Software Corporation. All rights reserved.

Part 3—Testing the Emp class

You will run the test procedure you wrote and confirm that the *Emp* class that you implemented in the second Try-it of this lesson executes correctly.

If you need help, view the Solution.

Step	Task
1.	Run **testEmp.p**. Does it run correctly and does the *Emp* class execute properly? View the output file.
2.	Examine the value for *PostalCode* in the output file. Was it set? Recall that in the first Try-it in this lesson, you created the PostalCode property using the New Property wizard. You specified that the setter for this property would be implemented later. The generated code has no implementation, so the property is not set. If you examine the code in **Emp.cls**, you will notice that the *set*() implementation for this accessor has no body.
3.	As the *PostalCode* data member of the *Emp* class was not set properly, you must correct the code. Add a statement to the *PostalCode set*() accessor so that the property will be set.
4.	Save your changes, and re-test until it runs correctly.

© 2017 Progress Software Corporation. All rights reserved.

Part 4—Writing the test procedure for the Dept class

Since the *Dept* class contains *Emp* instances, you will write code to retrieve data from the database and then initialize the *Emp* class instances, as you did earlier. You will write statements to test the constructor and each method of the *Dept* class. You will use message statements to write data to a file.

If you need help, view the Solution.

Step	Task
1.	In the **Test/Enterprise/HR** directory, create an ABL procedure file and name it **testDept.p**.
2.	In this procedure, you will be creating *Emp* instances and a *Dept* instance. Add *using* statements at the beginning of this file for the *Dept* and *Emp* classes.
3.	Define a variable named *DeptInstance*, which will be of type *Dept*. This variable will hold the reference to the *Dept* instance you create.
4.	Define the following other variables to the definition section. • EmpInstance as type Emp • retrievedEmp as type Emp • httEmployee as type handle
5.	Define a variable named *Phones* as an extent with a type *character* and a fixed size of *3*.
6.	Add a statement to include the ttEmployee temp-table.
7.	Add a statement to open a file for output. The name of the file you will be writing to is **testDept.out**. It will be located in the **/progress_education/openedge/IntroOOP/log** directory. **Hint**: Use the full pathname of this file.
8.	Write a statement to create an instance of a *Dept*, providing the three *input* values required by the constructor. You can specify them as these hard-coded values: • "Training" • 500 • "PROGRESS-3947" The name of the department is Training. It's expense code is PROGRESS-3947. The department code is 500. Assign the reference to this instance to the *DeptInstance* variable.
9.	Next, iterate through the Employee table in the database to create a set of Emp instances to add to the department. Since the department number is 500, write a FOR EACH statement to iterate through all Employee records that have a DeptNum equal to 500.
10.	Within the FOR EACH block, add a message statement to write the number of employees before adding a new employee.

© 2017 Progress Software Corporation. All rights reserved.

11.	Next, in the FOR EACH block, add the code for creating an Emp instance, assigning it to *EmpInstance*.
12.	Next, in the FOR EACH block, set the values the first and third elements of the Phones extent with the WorkPhone and HomePhone values from the current Employee record.
13.	Next, in the FOR EACH block, initialize the Emp instance by calling the Initialize() method, passing values from the current Employee buffer and the *Phones* extent.
14.	Next, in the FOR EACH block, add the created Emp instance to the DeptInstance. **Hint**: Use the AddEmployee() method that takes an Emp instance as a parameter.
15.	You are now done with the iteration through the Employee table and you will add code after the FOR EACH block. Next, you will add code to test the AddEmployee() method that creates and initializes the Emp instance using its parameters. First, add an *assign* statement to assign three phone numbers to each of the elements of the *Phones* extent. **Hints**: Use the index values 1,2,3 to access each element of the extent. Make sure the phone number values are in quotes as they are *character* types.
16.	Next, add a message statement to write the number of employees before adding a new employee.
17.	Add a statement to call AddEmployee() with the values for initializing the Emp instance. **Hint**: You can copy-paste the parameter values you used in testEmp.p for initializing the Emp instance.
18.	Add a message statement to write the number of employees (after adding an employee) to the output file.
19.	Next, you will test the *GetEmployee*() method. Add statements to get a particular employee by passing input values and assigning the value to retrievedEmp. Then display the retrieved employee name using the *GetInfo()* method of the Emp instance in a *message* statement. **Hint**: Get one employee that was created from the Employee table in the database and get another employee that you created manually.
20.	Next, you will test the *GetEmployees*() method. Add statements get all employees using the *GetEmployees()* method and then iterate through the *ttEmployee* temp-table to return the number of employees. **Hint**: You will need to use the cast function to call the DeptRef field of the temp-table.
21.	Add a statement to delete the *DeptInstance*. The destructor for this class also deletes the *Emp* instances.
22.	Add a statement to close the output file.
23.	Save this file. Ensure that it compiles without errors.

© 2017 Progress Software Corporation. All rights reserved.

Part 5—Testing the Dept class

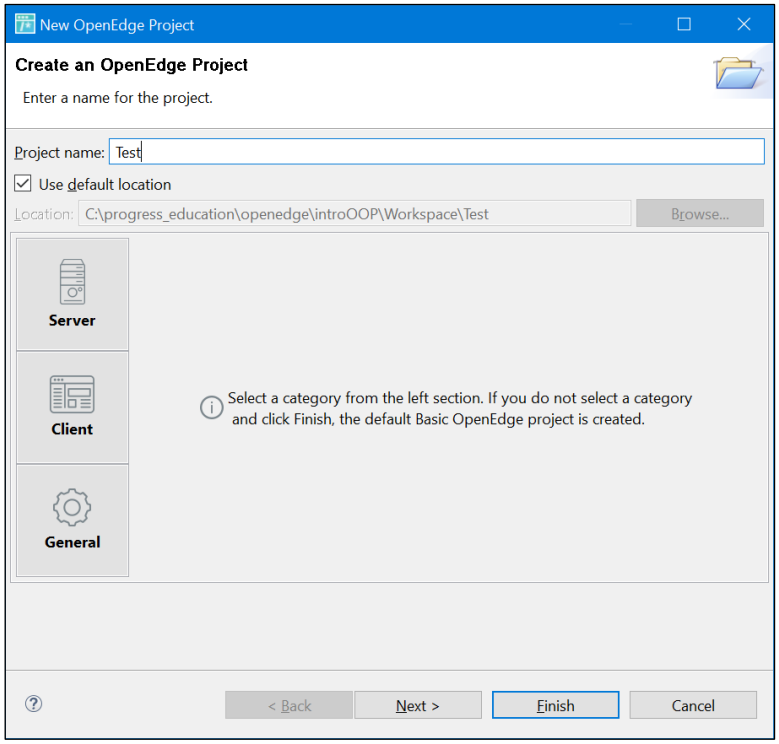
You will run the test procedure you wrote and confirm that the *Dept* class that you wrote executes correctly.

If you need help, view the Solution.

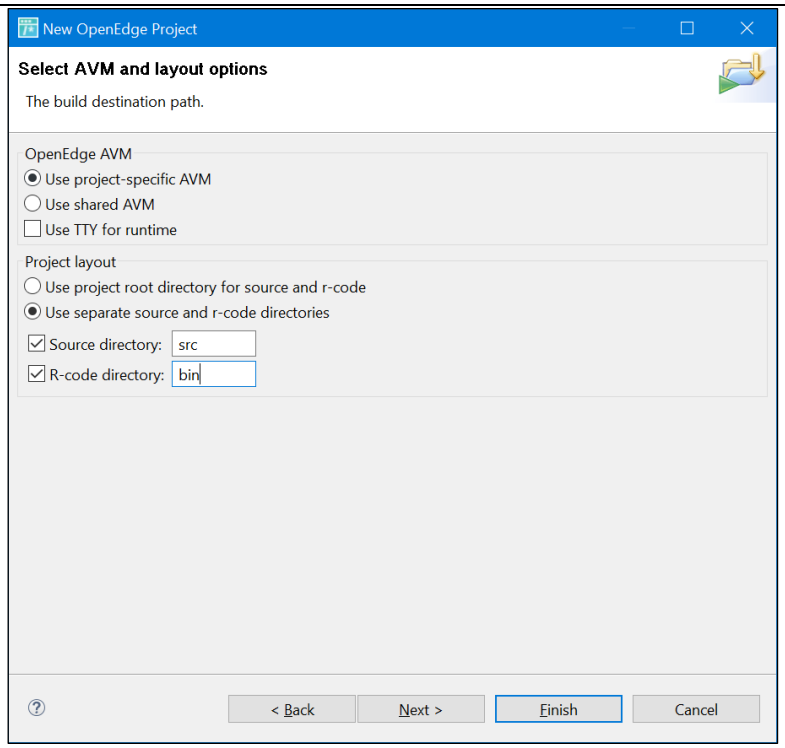
Step	Task
1.	Run **testDept.p**. Does it run correctly and does the *Dept* class execute properly? View the output file.
2.	If not, use the debugger to fix the problem, save your changes, and retest until it runs correctly.

© 2017 Progress Software Corporation. All rights reserved.

Solution, Part 1—Setting up a Test project

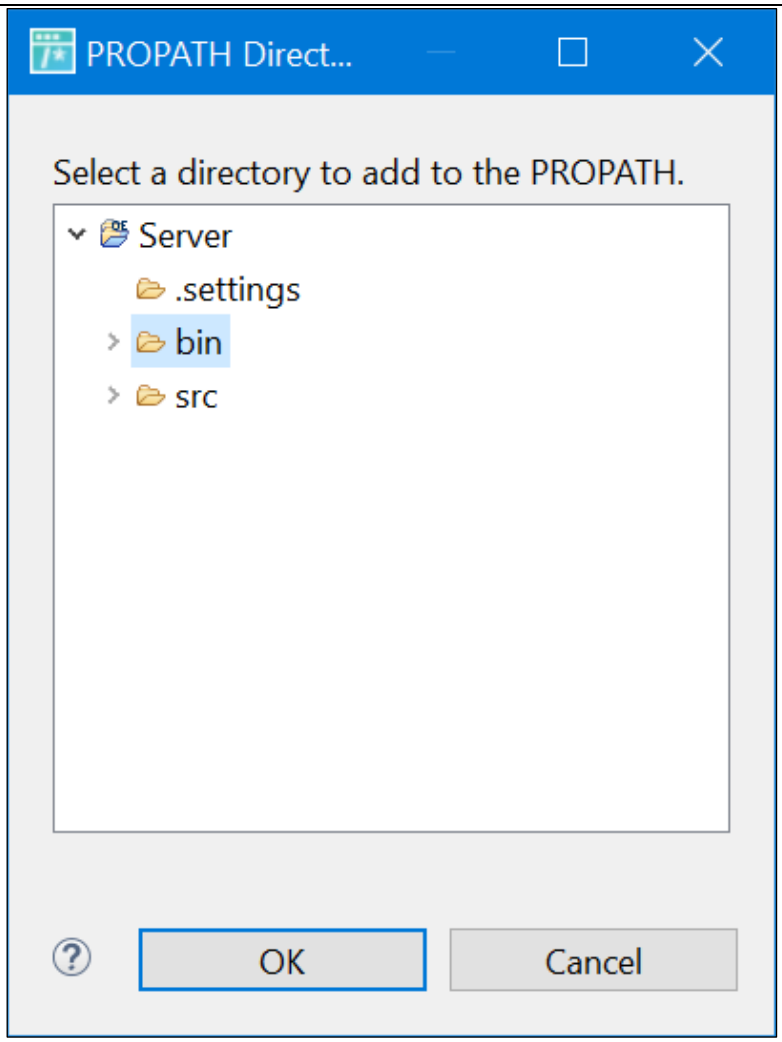
Step	Solution instructions
1.	Create a new OpenEdge Project named **Test** specifying Sports3000DB as the database and ensuring that its PROPATH is set to the Server/src and Server/bin folders. a. Select **File** > **New** > **OpenEdge Project**. b. Enter **Test** for the project name.  c. Click **Next**. d. Select **Use separate source and r-code directories**. e. Change **rcode** to **bin**.

© 2017 Progress Software Corporation. All rights reserved.



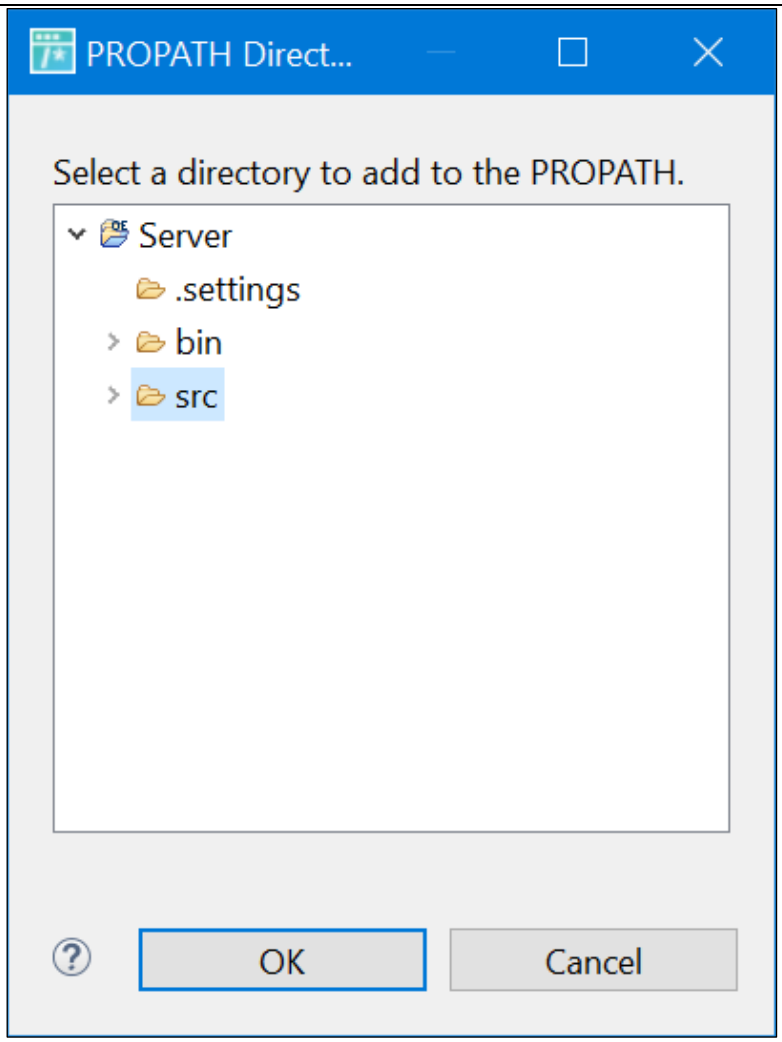
f. Click **Next**.
g. Click **Add Workspace Directory…**.
h. Select the **Server/bin** directory.

© 2017 Progress Software Corporation. All rights reserved.



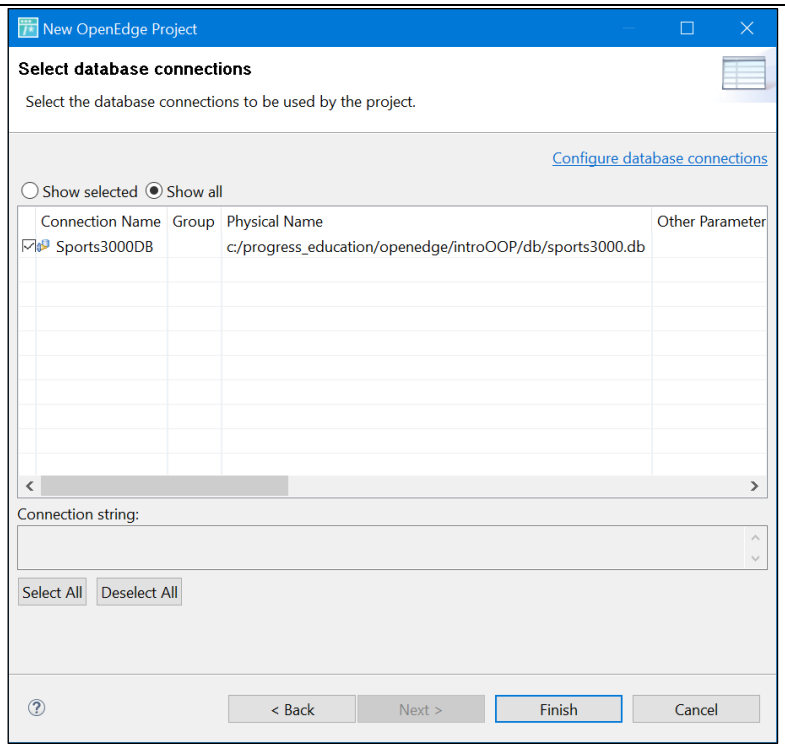
i. Click **OK**.
j. Click **Add Workspace Directory…**.
k. Select the **Server/src** directory.

© 2017 Progress Software Corporation. All rights reserved.



l. Click **OK**.
m. Click **Next**.
n. Select the **Sports3000DB** database connection.

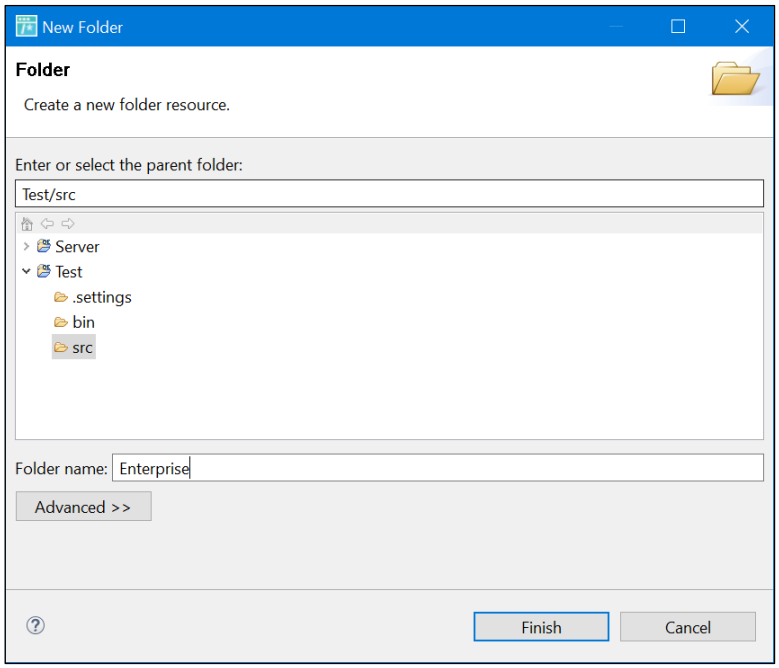
© 2017 Progress Software Corporation. All rights reserved.



o. Click **Finish**,

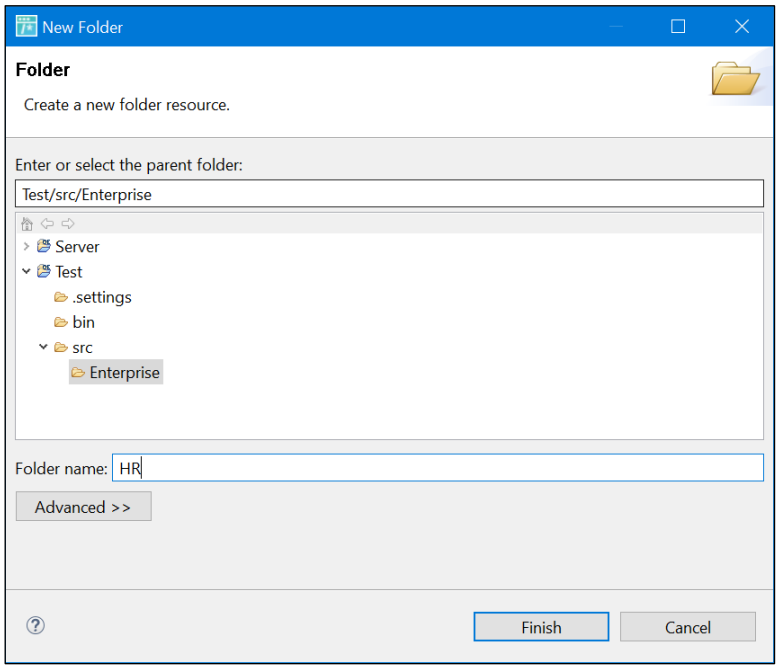
2. In Project Explorer, in the **Test** project, create a new folder named **Enterprise** under the **src** folder.

a. Right-click **src**.
b. Select **New** > **Folder**.
c. Enter **Enterprise** for the Folder name.



d. Click **Finish**.

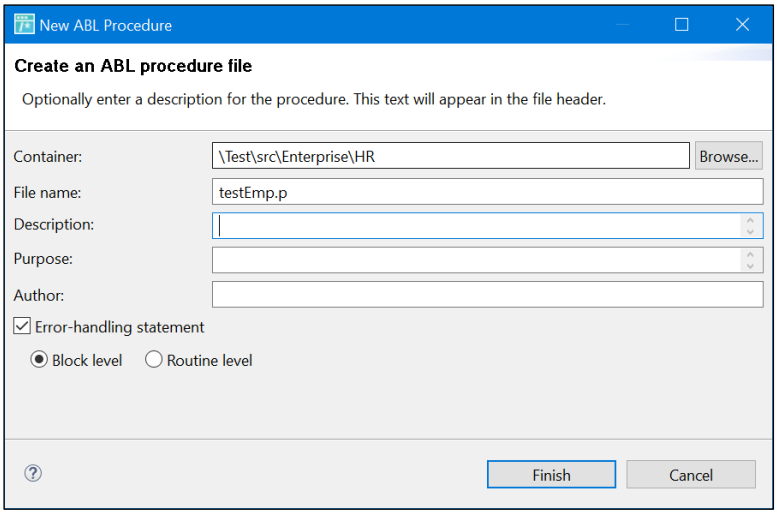
© 2017 Progress Software Corporation. All rights reserved.

3.	Under the **Enterprise** folder, create a new folder named **HR**. a. Right-click **Enterprise**. b. Select **New** > **Folder**. c. Enter **HR** for the Folder name.  d. Click **Finish**.

© 2017 Progress Software Corporation. All rights reserved.

Solution, Part 2—Writing the test procedure for the Emp class

Step	Solution instructions

1.	Create a new ABL procedure named **testEmp.p** in the **Enterprise/HR** directory of the Test project.
	a. Right-click the **HR** folder you created.
	b. Select **New** > **ABL Procedure**.
	c. Enter **testEmp.p** as the file name.
	d. Enter some descriptive information in the description field.
	 
	
	e. Click **Finish**.
	The generated code (without the heading comments) should be like this:
	`/* ******  Definitions  ****** */`
	`block-level on error undo, throw.`

2.	Add *using* statements after the error-handling statement. These *using* statements will ensure that the *Emp* class can be accessed.
	Add these statements at the beginning of the file after the error-handling statement:
	`using Progress.Lang.*.`
	`using Enterprise.HR.Emp.`

3.	Define a variable named *EmpInstance*, which will hold a reference to an instance of *Emp*. Then add this statement in the definitions section: `define variable EmpInstance as Emp no-undo.`

© 2017 Progress Software Corporation. All rights reserved.

4.	Define a variable named *Phones* as an extent with a type *character* and a fixed size of *3* with initial values. Add this statement: `define variable Phones as character extent 3 initial ["111-111-1111","222-222-2222","333-333-3333"] no-undo.`

5.	In the main block, add a statement to create an *Emp* instance, setting the reference of this instance to the *EmpInstance* variable.
	
	`/* create an Emp instance */`
	`EmpInstance = new Emp().`

6.	Add an *output to* statement in which you will write output from the test to a file named **test_Emp.out**. It will be located in the **/Progress_education/openedge/IntroOOP/log** directory.
	Hint: Use the full pathname to this file.
	
	In the main block area, add this statement:
	
	`/* set up the file for writing data*/`
	`output to`
	`/progress_education/openedge/IntroOOP/log/testEmp.out.`

7.	Add a statement to initialize the *Emp* instance you created with the constant values that have the same type as expected by the *Initialize*() method. For example, the employee number is 99, and the employee first name is "John". Use the *Phones* variable as input to this method.
	
	Add this code to the end of the procedure:
	
	`/* initialize the data members in the Emp instance */`
	`EmpInstance:Initialize(99,`
	`"John",`
	`"Doe",`
	`"123 Main Street",`
	`"01730",`
	`Phones,`
	`50,`
	`"Senior Developer").`

© 2017 Progress Software Corporation. All rights reserved.

8.	Add a *message* statement to write the employee data to the output file using the public data members and public methods of the class.
	
	Add this code to the end of the initialize method:
	
	`message "*********testInitialize()************" skip`
	`  EmpInstance:FirstName " "`
	`  EmpInstance:LastName ", "`
	`  EmpInstance:JobTitle skip`
	`  "Emp # " EmpInstance:EmpNum "-  Vacation hours: " EmpInstance:VacationHours skip`
	`  EmpInstance:Address " "`
	`EmpInstance:PostalCode skip`
	`  "Phones: " EmpInstance:Phones[1] " "`
	`EmpInstance:Phones[2] " " EmpInstance:Phones[3] " "`
	`  .`

9.	Add a statement to call the *SetVacationHours*() method for the instance and set the vacation hours for this instance to *25*.
	
	Add this code to the end of the procedure:
	
	`/* update the vacation hours for this employee */`
	`EmpInstance:SetVacationHours(25).`

10.	Add a *message* statement to write the employee data to the output file.
	
	Add this code to the end of the procedure:
	
	`message "*********testSetVacationHours()************" skip`
	`  "Vacation hours: "`
	`EmpInstance:VacationHours`
	`.`

11.	Add a statement to call the *SetJobTitle*() method for the instance and set the title for this instance as *Senior Architect*. Add this code to the end of the procedure: `EmpInstance:SetJobtitle("Senior Architect").`

12.	Add a *message* statement to write the employee data to the output file.
	
	Add this code to the end of the procedure:
	
	`message "*********testSetJobTitle()************" skip`
	`  "JobTitle: " EmpInstance:JobTitle.`

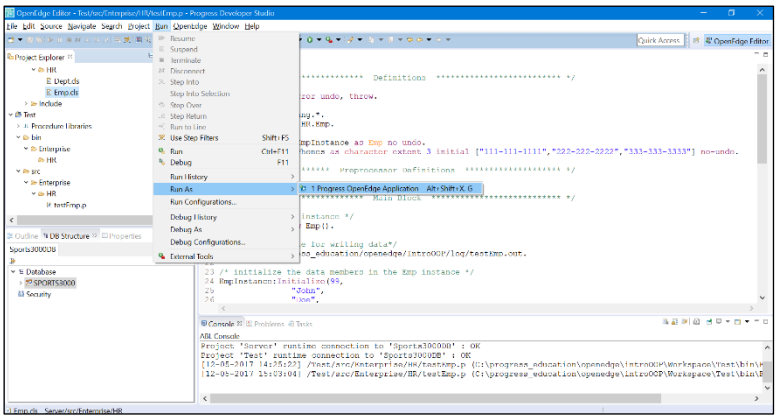
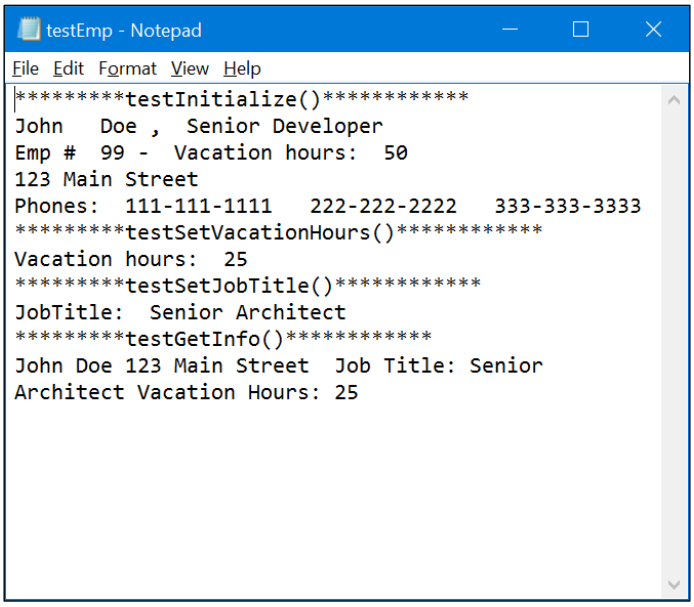
13.	To test the GetInfo() method, add a *message* statement to write the employee data to the output file. You will call the *GetInfo*() method of the *Emp* class using the *EmpInstance* instance to do this. Add this code to the end of the procedure:

© 2017 Progress Software Corporation. All rights reserved.

	``` /* write data to the output file */ message "*********testGetInfo()************" skip EmpInstance:GetInfo() . ```
14.	Add a statement to close the output file. Add this code at the end of the procedure. ``` output close. ```
15.	Add a statement to delete the *Emp* instance. Add this code to the end of the procedure: ``` /* delete the instance - freeing memory */ ```
16.	Save this file. Ensure that it compiles without errors.

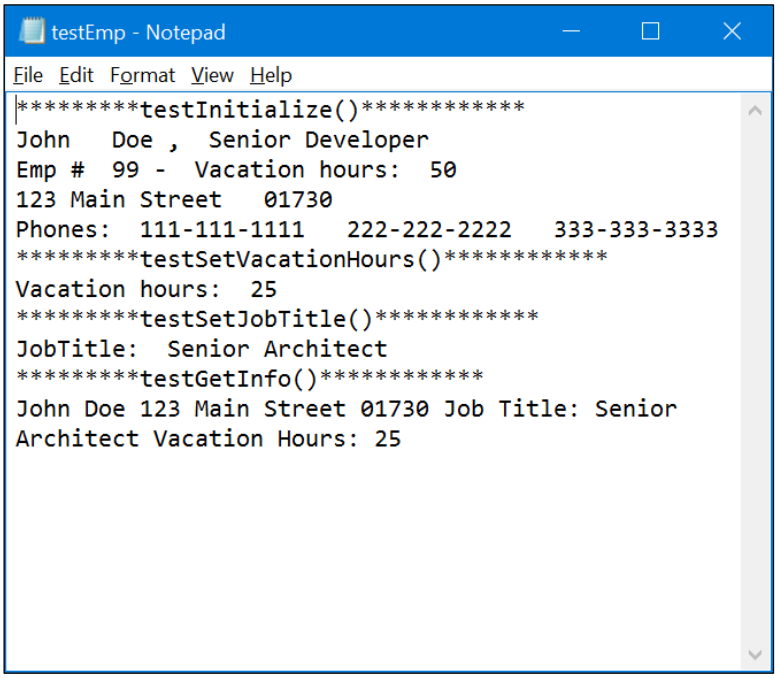
© 2017 Progress Software Corporation. All rights reserved.

Solution, Part 3—Testing the Emp class

Step	Solution instructions
1.	Run **testEmp.p**. Does it run correctly and does the *Emp* class execute properly? View the output file. a. With **testEmp.p** open in the editor, click the **Run** icon drop down. b. Select **Run As** > **Progress OpenEdge Application**.  c. Click **OK**.
2.	Examine the value for *PostalCode* in the output file. Was it set? Recall that in the first Try-it of this lesson, you created the PostalCode property using the New Property wizard. You specified that the setter for this property would be implemented later. The generated code has no implementation, so the property is not set. If you examine the code in **Emp.cls**, you will notice that the *set*() implementation for this accessor has no body.  testEmp - Notepad File Edit Format View Help *********testInitialize()************ John Doe , Senior Developer Emp # 99 - Vacation hours: 50 123 Main Street Phones: 111-111-1111 222-222-2222 333-333-3333 *********testSetVacationHours()************ Vacation hours: 25 *********testSetJobTitle()************ JobTitle: Senior Architect *********testGetInfo()************ John Doe 123 Main Street Job Title: Senior Architect Vacation Hours: 25

© 2017 Progress Software Corporation. All rights reserved.

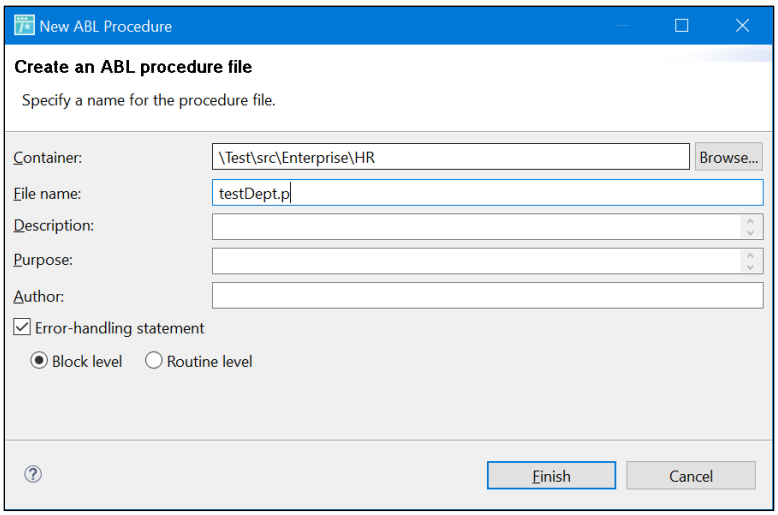
3.	As the *PostalCode* data member of the *Emp* class was not set properly, you must correct the code. Add a statement to the *PostalCode set*() accessor so that the property will be set. Your updated property should be as follows:
	
	`define public property PostalCode as character no-undo`
	`  get.`
	`  set(input arg as character):`
	`  PostalCode = arg.`
	`end set.`

4.	Save your changes, and re-test until it runs correctly. 

© 2017 Progress Software Corporation. All rights reserved.

Solution, Part 4—Writing the test procedure for the Dept class

Step	Solution instructions

1.	In the **Test/src/Enterprise/HR** directory, create an ABL procedure file and name it **testDept.p**.
	a. Right-click the **HR** folder you created.
	b. Select **New** > **ABL Procedure**.
	c. Enter **testDept.p** as the file name.
	d. Enter some descriptive information in the description field.
	 
	
	e. Click **Finish**.
	
	The generated code (without the header comments) should be like this:
	`/* ******  Definitions  ****** */`
	`block-level on error undo, throw.`

2.	In this procedure, you will be creating *Emp* instances and a *Dept* instance. Add *using* statements at the beginning of this file for the *Dept* and Emp classes.
	
	Add these statements at the beginning of the procedure, after the error handling statement:
	`using Progress.Lang.*.`
	`using Enterprise.HR.Emp.`
	`using Enterprise.HR.Dept.`

3.	Define a variable named *DeptInstance*, which will be of type *Dept*. This variable will hold the reference to the *Dept* instance you create. Add this statement after the using statements:

© 2017 Progress Software Corporation. All rights reserved.

```
define variable DeptInstance as Dept no-undo.
```

4. Define the following other variables to the definition section.

   - EmpInstance as type Emp
   - retrievedEmp as type Emp
   - httEmployee as type handle

   Place your cursor after the definition of the *DeptInstance* variable to add these definitions.

```
define variable EmpInstance as Emp no-undo.
define variable retrievedEmp as Emp no-undo.
define variable httEmployee as handle no-undo.
```

5. Define a variable named *Phones* as an extent with a type *character* and a fixed size of *3*. Add the definition of the Phones variable to the end of the definitions section.

```
define variable Phones as character extent 3 no-undo.
```

6. Add a statement to include the ttEmployee temp-table.

```
{include/ttEmployee.i}
```

7. Add a statement to open a file for output. The name of the file you will be writing to is **testDept.out**. It will be located in the **/progress_education/openedge/IntroOOP/log** directory.

   Hint: Use the full pathname of this file.

   In the main block area, add this statement:

```
output to
/progress_education/openedge/IntroOOP/log/testDept.out.
```

8. Write a statement to create an instance of *Dept*, providing the three *input* values required by the constructor. You can specify them as these hard-coded values:

   - "Training"
   - "500"
   - "PROGRESS-3947"

   The name of the department is Training. It's expense code is PROGRESS-3947. The department code is 500. Assign the reference of this instance to the *DeptInstance* variable.

   Add this statement after the *output to* statement:

```
/* create a Deptinstance using hard-coded values for
initializing it */
DeptInstance = new Dept("Training",
                        "500",
                        "PROGRESS-3947").
```


© 2017 Progress Software Corporation. All rights reserved.

9.	Next, iterate through the Employee table in the database to create a set of Emp instances to add to the department. Since the department number is 500, write a FOR EACH statement to iterate through all Employee records that have a DeptNum equal to 500.
	Add this code to the end of the procedure:
	`for each Employee where Employee.Dept = "500":`
	`end.`

10.	Within the FOR EACH block, add a message statement to write the number of employees before adding a new employee.
	Add this code to the FOR EACH block:
	`message`
	`"************testAddEmployeeWithEmpInstance()************`
	`" skip`
	`"Before adding employee, number of employees is: "`
	`DeptInstance:NumEmployees.`

11.	Next, in the FOR EACH block, add the code for creating an Emp instance, assigning it to *EmpInstance*. Add this code as the next statement in the block: `EmpInstance = new Emp().`

12.	Next, in the FOR EACH block, set the values the first and third elements of the Phones extent with the WorkPhone and HomePhone values from the current Employee record.
	Add this code next in the block:
	`assign`
	`Phones[1] = Employee.WorkPhone`
	`Phones[3] = Employee.HomePhone`
	`.`

13.	Next, in the FOR EACH block, initialize the Emp instance by calling the Initialize() method, passing values from the current Employee buffer and the Phones extent.
	Add this code next in the block:
	`EmpInstance:Initialize(Employee.EmpNum,`
	`Employee.FirstName,`
	`Employee.LastName,`
	`Employee.Address,`
	`Employee.PostalCode,`
	`Phones,`
	`Employee.VacationDaysLeft * 8,`
	`Employee.Position).`

© 2017 Progress Software Corporation. All rights reserved.

14.	Next, in the FOR EACH block, add the created Emp instance to the DeptInstance. **Hint**: Use the AddEmployee() method that takes an Emp instance as a parameter. Add this code next in the block: `DeptInstance:AddEmployee (EmpInstance).`

15.	You are now done with the iteration through the Employee table and you will add code after the FOR EACH block. Next, you will add code to test the AddEmployee() method that creates and initializes the Emp instance using its parameters. First, add an *assign* statement to assign three phone numbers to each of the elements of the *Phones* extent.
	Hints: Use the index values 1,2,3 to access each element of the extent. Make sure the phone number values are in quotes as they are *character* types.
	
	Add this code after the FOR EACH block's end statement:
	
	`/* Test add Employee with initialization values */`
	`assign`
	`  Phones[1] = "111-111-1111"`
	`  Phones[2] = "222-222-2222"`
	`  Phones[3] = "333-333-3333"`
	`  .`

16.	Next, add a message statement to write the number of employees before adding a new employee.
	
	Add this code to the end of the procedure:
	
	`message`
	`"************testAddEmployeeWithInitializationValues()**********" skip`
	`  "Before adding employee, number of employees is: "`
	`DeptInstance:NumEmployees.`

17.	Add a statement to call AddEmployee() with the values for initializing the Emp instance.
	Hint: You can copy-paste the parameter values you used in testEmp.p for initializing the Emp instance.
	
	Add this code to the end of the procedure:
	
	`DeptInstance:AddEmployee(999,`
	`  "Jane",`
	`  "Doe",`
	`  "321 Main Street",`
	`  "01730",`
	`  Phones,`
	`  40,`
	`  "Instructor").`

© 2017 Progress Software Corporation. All rights reserved.

18.	Add a message statement to write the number of employees (after adding an employee) to the output file. Add this statement to the end of the procedure: `message "After adding employee, number of employees is: " DeptInstance:NumEmployees.`

19.	Next, you will test the *GetEmployee*() method. Add statements to get a particular employee by passing input values and assigning the value to retrievedEmp. Then display the retrieved employee name using the *GetInfo()* method of the Emp instance in a *message* statement.
	Hint: Get one employee that was created from the Employee table in the database (Luke Sanders) and get another employee that you created manually (Jane Doe).
	
	Add this code to the end of the procedure:
	
	`retrievedEmp = DeptInstance:GetEmployee("Luke", "Sanders").`
	`if valid-object(retrievedEmp)`
	`  then`
	`    message`
	`      "************testGetEmployee( ) for Luke Sanders ************"`
	`      retrievedEmp:GetInfo().`
	`retrievedEmp = DeptInstance:GetEmployee("Jane", "Doe").`
	`if valid-object(retrievedEmp)`
	`  then`
	`    message`
	`      "************testGetEmployee() for Jane Doe ************"`
	`    retrievedEmp:GetInfo().`

20.	Next, you will test the *GetEmployees*() method. Add statements to get all employees using the *GetEmployees()* method and then iterate through the *ttEmployee* temp-table to write the employee names to the output file.
	Hint: You will need to use the cast function to call GetInfo() for the EmpRef field of the temp-table.
	
	Add this code to the end of the procedure:
	
	`message "************testGetEmployees()**************".`
	`httEmployee = temp-table ttEmployee:handle.`
	`empty temp-table ttEmployee no-error.`
	`DeptInstance:GetEmployees(output table ttEmployee).`
	`  message`
	`      "Number of employees in this dept: "`
	`DeptInstance:NumEmployees.`
	`for each ttEmployee:`
	`      message`
	`          cast(ttEmployee.EmpRef,Emp):GetInfo().`
	`end.`

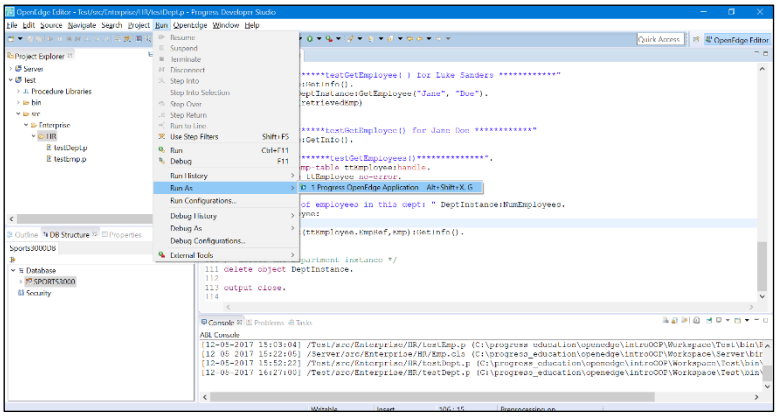
© 2017 Progress Software Corporation. All rights reserved.

21.	Add a statement to delete the *DeptInstance*. The destructor for this class also deletes the *Emp* instances.
	
	Add this code to the end of the procedure:
	
	`/* delete the Department instance */`
	`delete object DeptInstance.`

22.	Add a statement to close the output file. Add this code to the end of the procedure: `output close.`
23.	Save this file. Ensure that it compiles without errors.

© 2017 Progress Software Corporation. All rights reserved.

Solution, Part 5—Testing the Dept class

Step	Solution instructions
1.	Run **testDept.p**. Does it run correctly and does the *Dept* class execute properly? View the output file. a. With **testDept.p** open in the editor, click the **Run** icon drop-down. b. Select **Run As** >**Progress OpenEdge Application**.  c. Click **OK**.
2.	If not, use the debugger to fix the problem, save your changes, and re-test until it runs correctly.

© 2017 Progress Software Corporation. All rights reserved.

Try It 2.3: Testing classes, Wrap-up

Exercise summary

In this Try It, you wrote procedures that test the code you developed for the *Emp* and *Dept* classes. You must ensure that all code that you develop is thoroughly tested.

© 2017 Progress Software Corporation. All rights reserved.

Notes

© 2017 Progress Software Corporation. All rights reserved.

Try It 3.1: Using inheritance

Overview

In this Try It, you will use the *Emp* class as a super class for the *Manager* and *TeamMember* derived classes. First, you will modify a data member of the *Emp* super class so that it can be used by its derived classes. Then you will create and develop the code for the *Manager* class. Next, you will import existing code for the *TeamMember* and *Dept* classes, along with test procedures for testing your class hierarchy. Finally, you will test the inheritance hierarchy.

This exercise has 7 parts. The exercise steps take approximately 60 minutes to complete. You perform this exercise in your live version of Progress OpenEdge.

Before you begin

Before you begin, you must:

Step	Description
1.	Complete the Exercise Setup instructions, if you have not done so already.
2.	Complete the Guided Exercise 2.3.

Location of files:
Exercise files: **/progress_education/openedge/IntroOOP/Exercise/Lesson03**
Solution files: **/progress_education/openedge/IntroOOP/Solutions/Lesson03**

If at any time during this course, you need to restore your projects to match where you should have been prior to beginning a Try It, you can do the following:

1. Delete all projects in the workspace.
2. Import all projects in the archive (**.zip**) file for the previous Try It.
3. Do a clean build of the workspace.

© 2017 Progress Software Corporation. All rights reserved.

Part 1—Modify the Emp class to support its derived classes

In this part of the Try It, you modify the *Emp* class, so that it can be used by its derived classes – *Manager* and *TeamMember* classes. You will change the *GetName*() method from private to protected to make it available to the derived classes.

If you need help, view the Solution.

Step	Task
1.	At the end of the definition section of the *Emp* class, define the *EmpType* public property as a character with a protected setter.
2.	Change the *GetName*() method to protected.
3.	Save your file. Ensure that it compiles without errors.

© 2017 Progress Software Corporation. All rights reserved.

Part 2—Creating the Manager class

In this part of the Try It, you will create the *Manager* class. This class will inherit the data members, constructors, and methods from the *Emp* class.

If you need help, view the Solution.

Step	Task
1.	In Project Explorer, in the Server project, create a new folder named **Role** under the **src/Enterprise/HR** folder.
2.	In this folder, create a class named *Manager* that uses *Emp* as its super class. This class will have a default constructor and a destructor. **Hint**: Use the *New ABL class* wizard.
3.	Add a *using* statement after the error-handling statement. This *using* statement will ensure that the *TeamMember* class can be accessed. **Note**: You will see an error in this statement because you have not yet created the *TeamMember* class.

© 2017 Progress Software Corporation. All rights reserved.

Part 3—Defining a constructor and a data member for the Manager class

Next, you will define the data member for the *Manager* class.

If you need help, view the Solution.

Step	Task
1.	Modify the default constructor to set the *EmpType property* to be *Manager*.
2.	Define the *private* temp-table by including the *ttEmployee* temp-table definition. **Hint**: Add this statement in the definitions part of the class.
3.	Save your file. Ensure that it compiles without errors (except for the error related to the reference to *TeamMember*).

© 2017 Progress Software Corporation. All rights reserved.

Part 4—Defining methods for the Manager class

Next, you will define the methods for the *Manager* class. Recall that all the methods of the *Emp* super class are available for the *Manager* class. However, you will add two methods and override one inherited method.

If you need help, view the Solution.

Step	Task
1.	Define the public *AddTeamMember*() method that takes a *TeamMember* as input and returns *void*. **Note**: You will see an error in this statement because you have not yet created the *TeamMember* class.
2.	Define the public *GetManagerEmployees*() method that has an output parameter, which is the *ttEmployee* table, and returns *void*.
3.	Define the *public* method *GetInfo*(), which returns *character* and takes no parameters. It will override the method in the *Emp* class. **Hint**: Use the Override/Implement Members wizard.
4.	Save your file. Ensure that it compiles without errors. (except for the errors related to the *TeamMember* reference)

© 2017 Progress Software Corporation. All rights reserved.

Part 5—Implementing the methods for the Manager class

In this part of the Try It, you add code to methods of the class that you have already defined. Here, you will override the *GetInfo*() method to add additional vales to the result string.

If you need help, view the Solution.

Step	Task
1.	Implement the *AddTeamMember*() method that takes a parameter with type *TeamMember* to perform the following: • Create a *ttEmployee* record. • Set the values of the *ttEmployee* record from the input parameter. **Note**: You will see errors in your code because you have not yet created the *TeamMember* class.
2.	Implement the *GetInfo*() override method as follows: • Define a *character* variable named *result*. • Set the value of *result* to be a concatenation of the value returned from *GetName*(), *"Manager"*, *Address*, *PostalCode*, *JobTitle*, and the *string* value for *VacationHours.* • Return the result.
3.	Add code to the destructor for the class to delete all *ttEmployee* records.
4.	Note that you need not implement the *GetManagerEmployees*() method. ABL will automatically return the *ttEmployee* temp-table. Save your file. It will have compilation errors because the *TeamMember* class has not yet been created. If you see other syntax errors not related to *TeamMember*, correct them.

© 2017 Progress Software Corporation. All rights reserved.

Part 6—Importing the TeamMember and Dept classes

You have practiced creating one derived class. In this part of the Try It, to complete the inheritance hierarchy you will first import the *TeamMember.cls* file. This class will also inherit the data members, constructors, and methods from the *Emp* class.

Then, you will import a new version of the **Dept.cls** file to utilize *Manager* and *TeamMember* instances, rather than *Emp* instances. The new version of the **Dept.cls** has been updated to support the *Manager* and *TeamMember* classes.

If you need help, view the Solution.

Step	Task
1.	In the **Role** folder of the Server project, import the **TeamMember.cls** file.
2.	Clean and rebuild the Server project. Note that the compilation errors for **Manager.cls** should now no longer exist.
3.	In the **HR** folder of the Server project, import the **Dept.cls** file. **Note**: Click **OK** to replace the existing **Dept.cls** file.
4.	Clean and rebuild the workspace. You should have no compilation errors.

© 2017 Progress Software Corporation. All rights reserved.

Part 7—Testing the inheritance hierarchy

In this part of the Try It, you will import the **testManager.p** and **testTeamMember.p** procedure files for testing the *Manager* and *TeamMember* classes. Then, you will import a new version of **testDept.p** to test the *Dept* class.

Next, you will run the test procedures to test the inheritance hierarchy and how the *Dept* class uses the derived classes.

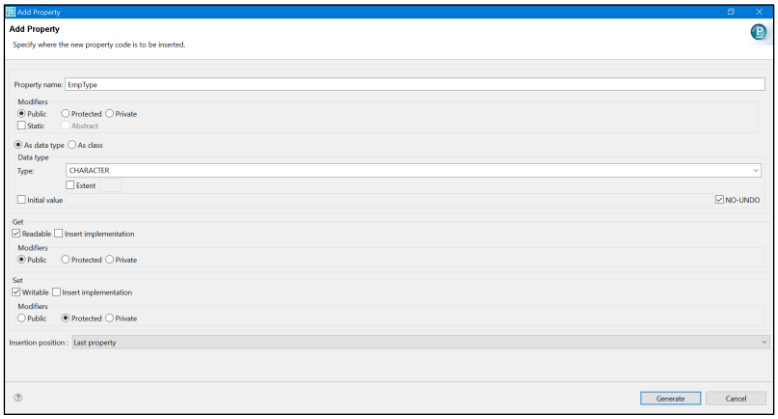
If you need help, view the Solution.

Step	Task
1.	In the Test project, create a folder named **Role** in the Enterprise/HR directory.
2.	In the **Enterprise/HR/Role** directory of the Test project, import the procedure files named **testManager.p** and **testTeamMember.p.**
3.	Run **testManager.p**. Does it run correctly and was the *Manager* information added to the output file? View the output file.
4.	Run **testTeamMember.p**. Does it run correctly and was the *TeamMember* added with *Manager* information? View the output file.
5.	If the files did not run as expected, use the debugger to fix the problem, save your changes, and retest until it runs correctly.
6.	In the **Test/Enterprise/HR** directory, import an ABL procedure file **testDept.p**. View the file to see how each method is defined for testing. **Note**: You will replace the existing **testDept.p** file.
7.	Run **testDept.p**. Does it run correctly and does the *Dept* class execute properly? View the output file named **TestDept2.out**.
8.	If it does not run correctly, use the debugger to fix the problem, save your changes, and retest until it runs correctly.

© 2017 Progress Software Corporation. All rights reserved.

Solution, Part 1—Modify the Emp class to support its derived classes

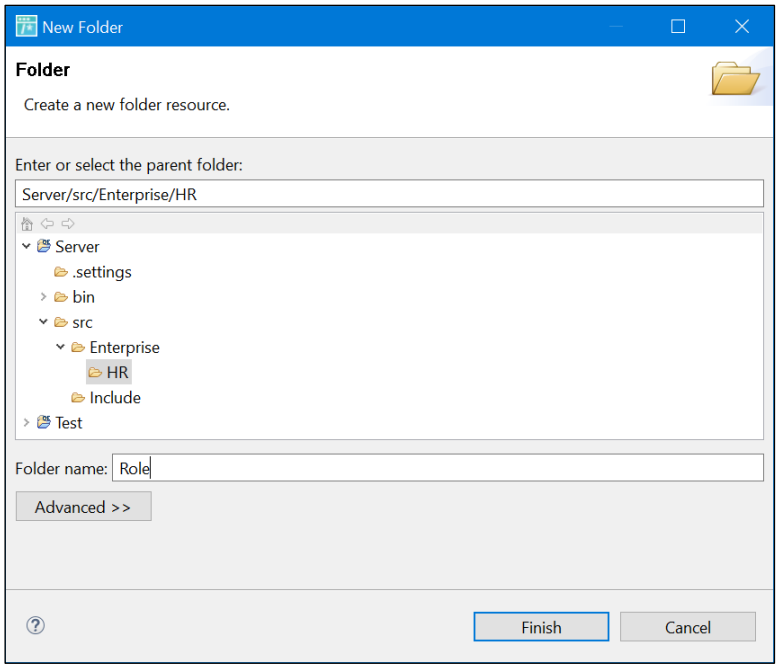
Step	Solution Instructions

1.	At the end of the definition section of the *Emp* class, define the *EmpType* public property as a character *EmpType* as *character* with a protected setter.
	Hint: Use the *Add Property* wizard.
	a. In the editor, place the cursor at a blank line before the constructor.
	b. **Right-click** and then select **Source** > **Add Property**. The Add Property wizard opens.
	c. Enter **EmpType** for the Property name.
	d. Select **Character**.
	e. Select **Public** for Get accessor.
	f. Select **Protected** for the Set accessor.
	g. Select **Last Property** for the insertion position.
	 
	
	h. Click **Generate**.
	The generated code (without comments) should appear as follows:
	`define public property EmpType as character no-undo`
	`  get.`
	`  protected set.`

2.	Change the *GetName*() method from *private* to *protected*. `method protected character GetName( ):`
3.	Save your file. Ensure that it compiles without errors.

© 2017 Progress Software Corporation. All rights reserved.

Solution, Part 2—Creating the Manager class

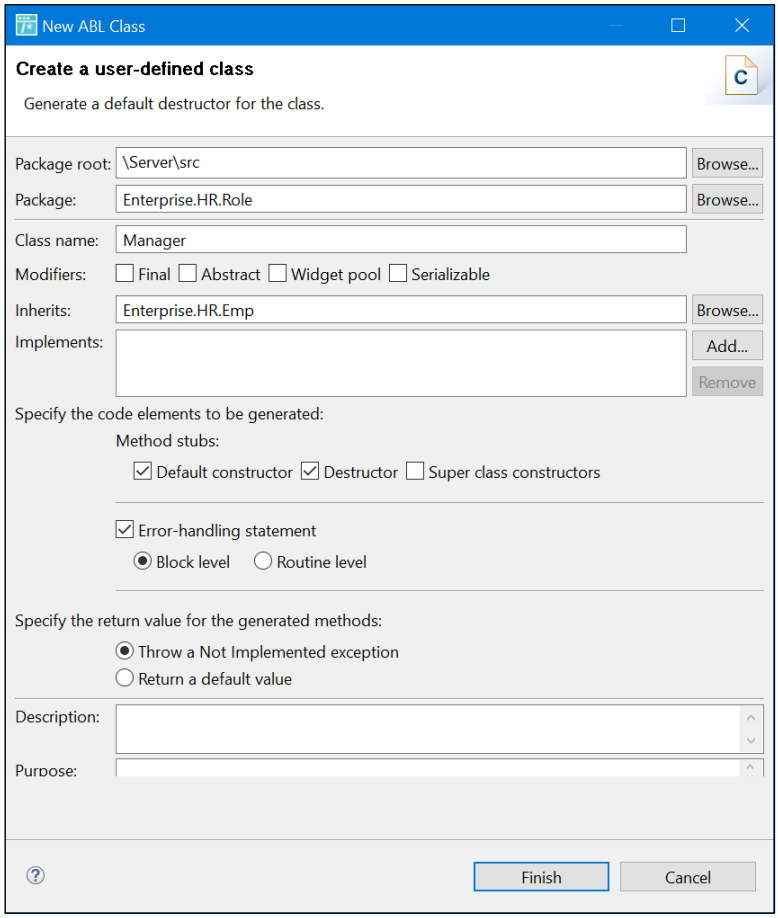
Step	Solution instructions
1.	In Project Explorer, in the **Server** project, create a new folder named **Role** under the **src/Enterprise/HR** folder. a. Right-click **HR**. b. Select **New** > **Folder**. c. Enter **Role** for the Folder name.  d. Click **Finish**.

© 2017 Progress Software Corporation. All rights reserved.

2. In this directory, create a class named *Manager* that uses *Emp* as its super class. This class will have a default constructor and a destructor.

Hint: Use the *New ABL class* wizard.

a. Right-click **Role**.
b. Select **New** > **ABL Class**.
c. Enter **Manager** for the Class name.
d. Browse and select **Emp.cls (Enterprise.HR.Emp)** for the Inherits field.
e. Select **Default Constructor**.
f. Select **Destructor**.
g. You can optionally add information to the description.



h. Click **Finish**.
The **Manager.cls** file opens in the editor.

The generated file should appear as follows:

```
using Progress.Lang.*.
using Enterprise.HR.Emp.
block-level on error undo, throw.
class Enterprise.HR.Role.Manager inherits Emp:
constructor public Manager():
super ().
end constructor.
destructor public Manager ():
end destructor.
  end class.
```


© 2017 Progress Software Corporation. All rights reserved.

3.	Add a *using* statement after the error-handling statement. These *using* statements will ensure that the *TeamMember* class can be accessed. **Note**: You will see an error in this statement because you have not yet created class *TeamMember* class. Add this statements at the beginning of the file before the error-handling statement: `using Enterprise.HR.Role.TeamMember.`

© 2017 Progress Software Corporation. All rights reserved.

Solution, Part 3—Defining a constructor and a data member for the Manager class

Step	Solution instructions

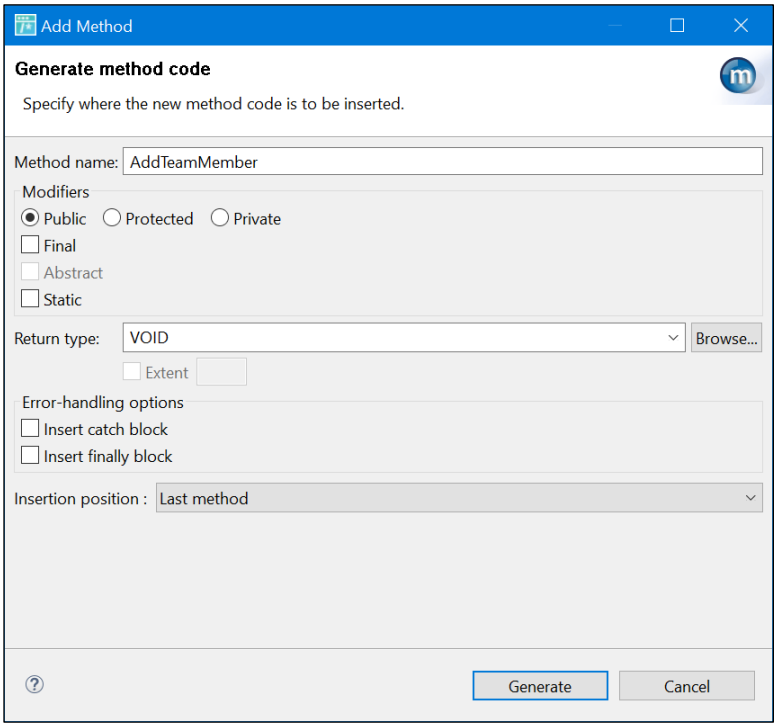
1.	Modify the default constructor to set the *EmpType* property to be *Manager*.
	
	Place your cursor in the parameters area for the constructor. Add code for these parameters:
	
	`constructor public Manager( ):`
	`super ().`
	`EmpType = "Manager".`
	`end constructor.`

2.	Define the *private* temp-table by including the ttEmployee temp-table definition. a. In the editor, place the cursor at a blank line before the constructor. b. Add this code to the class file: `{include/ttEmployee.i &ClassAccess = "private"}`
3.	Save your file. Ensure that the code added compiles without errors (except for the using statement for *TeamMember* class).

© 2017 Progress Software Corporation. All rights reserved.

Solution, Part 4—Defining methods for the Manager class

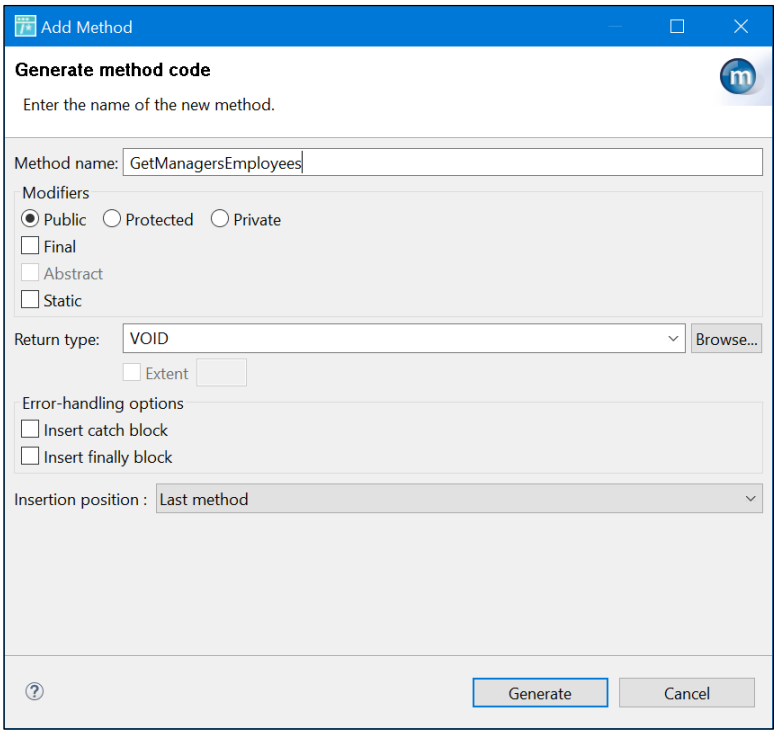
Step	Solution instructions

1.	Define the public *AddTeamMember*() method that takes a *TeamMember* as input and returns *void*.
	a. Place the cursor just before the anywhere in the class.
	b. Right-click and then select **Source** > **Add Method**.
	c. Enter **AddTeamMember** for the method name.
	d. Select **void** as the return type.
	e. Select **Last method** for the insertion position.
	 
	
	f. Click **Generate**.
	You can enter information in the comment that precedes the method, or you can delete it.
	g. Enter the input parameter as *ptm* with a type *TeamMember*.
	
	Your code should appear as follows:
	`method public AddTeamMember(input ptm as TeamMember)`
	`return.`
	`end method.`

© 2017 Progress Software Corporation. All rights reserved.

2. Define the public *GetManagersEmployees*() method that has an output parameter, which is the *ttEmployee* table, and returns a *void*. Place the cursor anywhere in the class.

a. Right-click and then select **Source** > **Add Method**.
b. Enter **GetManagersEmployees** for the method name.
c. Select **void** as the return type.
d. Select **Last method** for the insertion position.

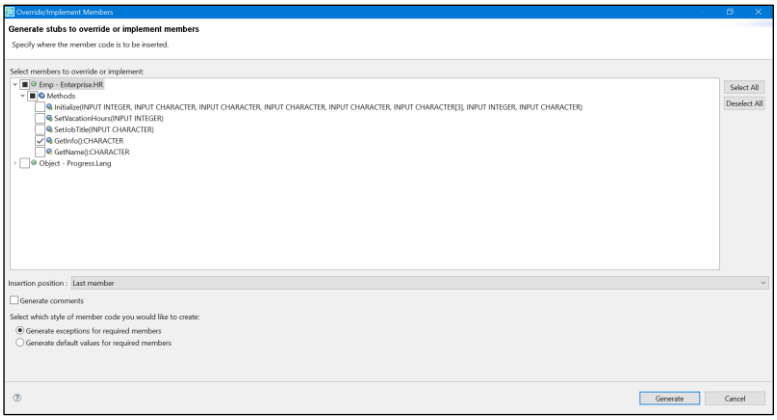


e. Click **Generate**.
You can enter information in the comment that precedes the method, or you can delete it.
f. Enter the output parameter as table *ttEmployee*:

Your code should appear as follows:

```
method public void GetManagersEmployees(output table
ttEmployee)
return.
end method.
```


© 2017 Progress Software Corporation. All rights reserved.

3.	Define the *public* method *GetInfo*(), which returns *character* and takes no parameters. It will override the method in the *Emp* class.
	
	a. Place the cursor anywhere in the class.
	b. Right-click and then select **Source** > **Override/Implement Members…**.
	c. Navigate to and select the **GetInfo** method in the *Enterprise.HR.Emp* class.
	d. Select **Last member** for the insertion position.
	
	 
	
	e. Click **Generate**.
	You can enter information in the comment that precedes the method, or you can delete it.
	
	The code should appear as follows:
	
	`method override public character GetInfo():`
	`  return super:GetInfo().`
	`  end method.`

4.	Save your file. Ensure that the code added compiles without errors (except for the using statement for *TeamMember* class).

© 2017 Progress Software Corporation. All rights reserved.

Solution, Part 5—Implementing the methods of the Manager class

Step	Solution instructions

1.	Implement the *AddTeamMember*() method that takes the parameter with type *TeamMember* to perform the following:
	• Create a *ttEmployee* record.
	• Set the values of the *ttEmployee* record from the input parameter.
	
	This method should now appear as follows:
	
	`method public void AddTeamMember`
	`(input ptm as TeamMember):`
	`create ttEmployee.`
	`assign`
	`  ttEmployee.FirstName = ptm:FirstName`
	`ttEmployee.LastName = ptm:LastName`
	`ttEmployee.EmpRef = ptm.`
	`return.`
	` end method.`

2.	Implement the *GetInfo*() method as follows:
	• Define a *character* variable named *result*.
	• Set the value of *result* to be a concatenation of the value returned from *GetName*(), *" Manager"*, *Address*, *PostalCode*, *JobTitle*, and the *string* value for *VacationHours*.
	• Replace the return statement (return super:GetInfo().) with return result.
	The code for this method should now appear as follows:
	
	`method override public character GetInfo( ):`
	`define variable result as character no-undo.`
	`result = GetName() + "Manager" +`
	`  Address + " " + PostalCode + " " +`
	`  "Job Title: " + JobTitle + " " +`
	`  "Vacation Hours: "+ string(VacationHours).`
	
	`return result.`
	`end method.`

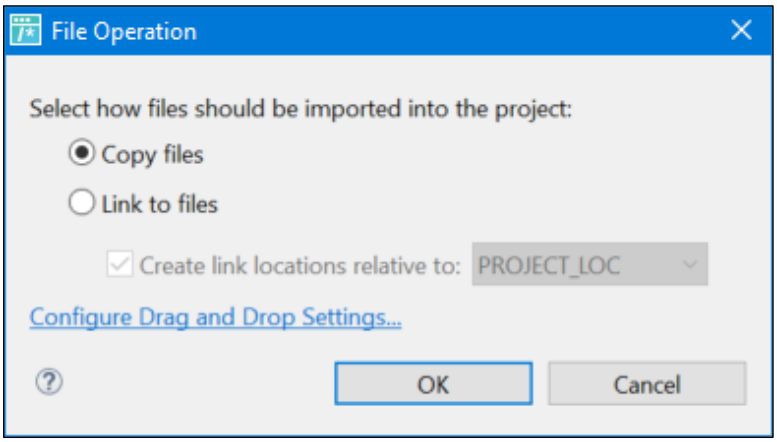
3.	Add code to the destructor to delete all *ttEmployee* records.
	
	Add this code to the destructor:
	
	`for each ttEmployee:`
	`  delete ttEmployee.`
	`end.`

© 2017 Progress Software Corporation. All rights reserved.

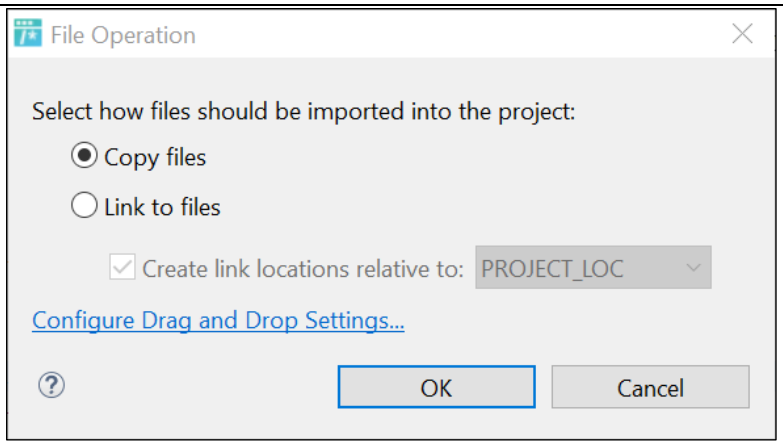
4.	Note that you need not implement the *GetManagersEmployees*() method. The ABL will automatically return the *ttEmployee* temp-table. Save your file. Ensure that it compiles without error (except for the using statement for *TeamMember* class).

© 2017 Progress Software Corporation. All rights reserved.

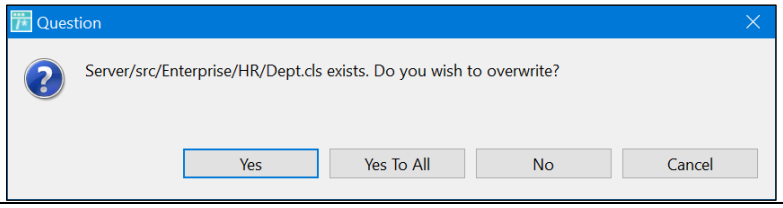
Solution, Part 6—Importing the TeamMember and Dept classes

Step	Solution instructions
1.	In the **Role** folder of the Server project, import the **TeamMember.cls** file. a. In Windows Explorer, navigate to the / **/progress_education/OpenEdge/introOOP/Exercise/Lesson03** directory and select the **TeamMember.cls** file. b. Drag and drop the **TeamMember.cls** file into the **Role** folder. c. Ensure that Copy files is selected.  d. Click **OK**.
2.	Clean and rebuild the Server project. Note that the compilation errors for the **Manager.cls** should now no longer exist.
3.	In the **HR** folder of the Server project, import the **Dept.cls** file. a. In Windows Explorer, navigate to the **/progress_education/OpenEdge/introOOP/Exercise/Lesson03** directory and select the **Dept.cls** file. b. Drag and drop the **Dept.cls** file into the **HR** folder. c. Ensure that Copy files is selected.

© 2017 Progress Software Corporation. All rights reserved.



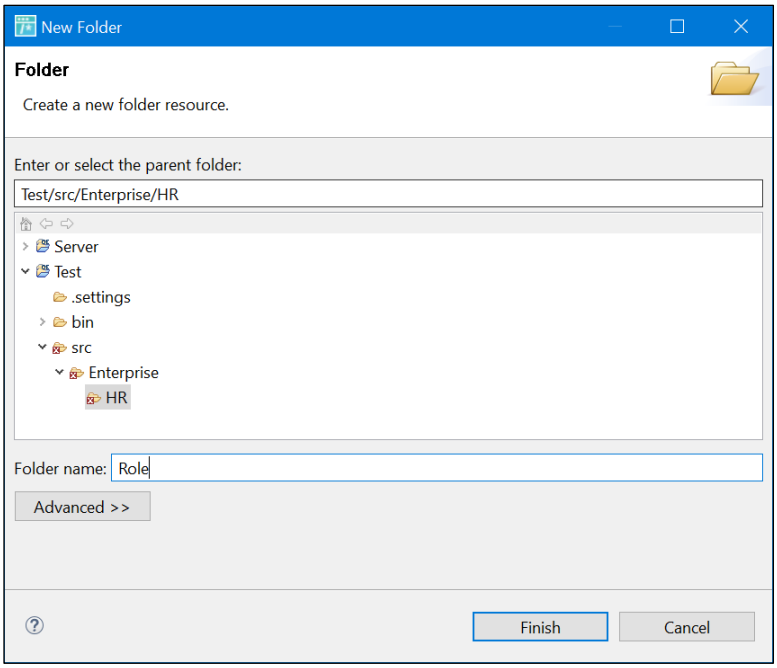
d. When asked if you want to replace the **Dept.cls** file, respond **Yes**.



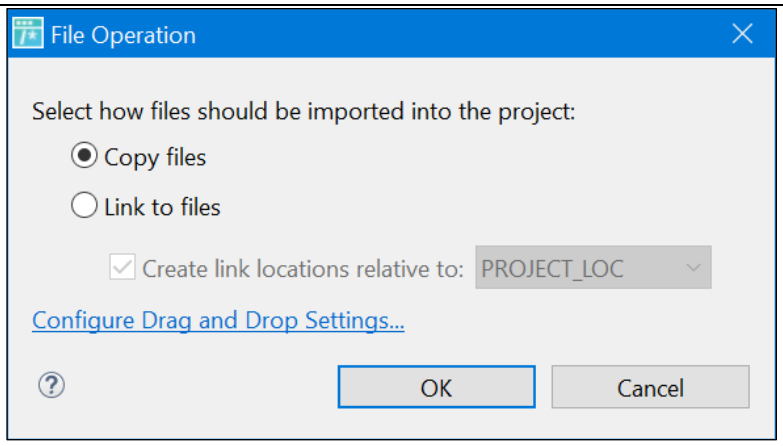
4. Clean and rebuild the workspace. You should have no compilation errors in the Server project.
Note: You will see an error in the Test project due to replacement of the **Dept.cls** file which can be ignored for now.

© 2017 Progress Software Corporation. All rights reserved.

Solution, Part 7—Testing the inheritance hierarchy

Step	Solution instructions
1.	In the Test project, create a folder named **Role** in the Enterprise/HR directory. a. Right-click the **HR** folder of the Test project. b. Select **New>Folder**. c. Enter **Role** as the folder name.  d. Click **Finish**.
2.	In the **Enterprise/HR/Role** directory of the Test project, import the procedure files named **testManager.p** and **testTeamMember.p.** a. In Windows Explorer, navigate to the **/progress_education/OpenEdge/introOOP/Exercise/Lesson03** directory and select the **testManager.p** and **testTeamMember.p** files. b. Drag and drop the **testManager.p** and **testTeamMember.p** files into the **Role** folder. c. Ensure that Copy files is selected.

© 2017 Progress Software Corporation. All rights reserved.

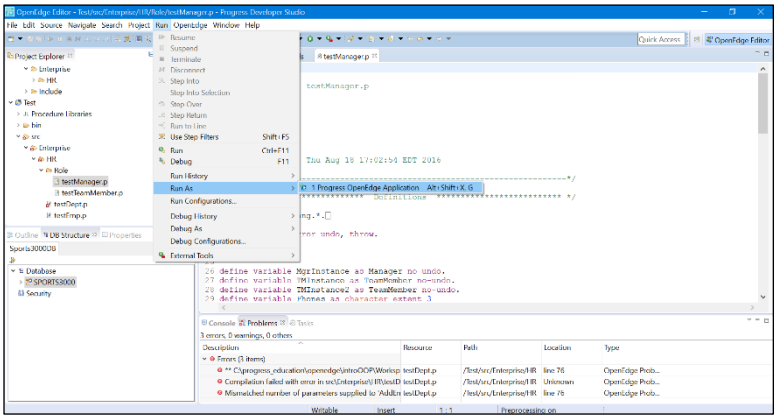


d. Click OK.

3. Run **testManager.p**. Does it run correctly and was the *TeamMember* information added to the output file for each *TeamMember* under this *Manager*? View the output file.

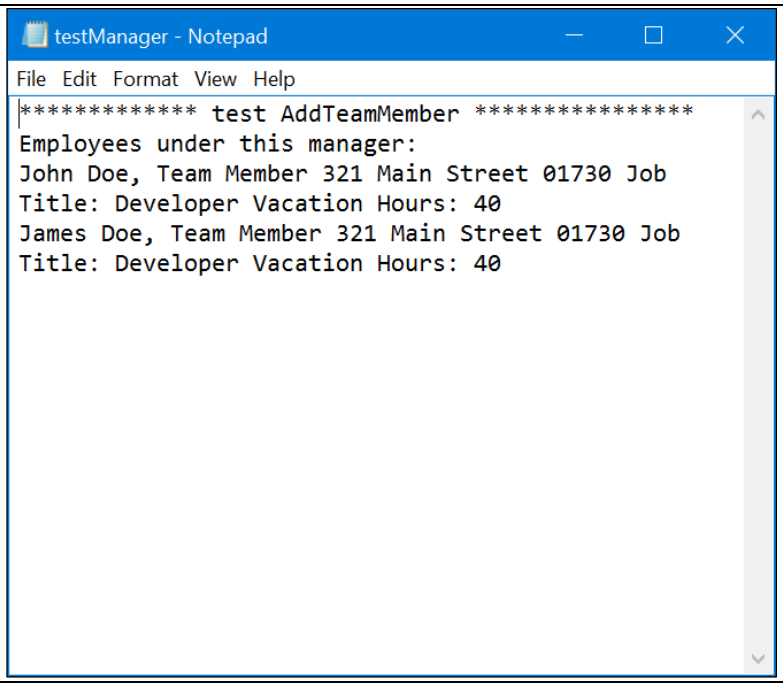
a. With **testManager.p** open in the editor, click the **Run** icon.

b. Select **Progress OpenEdge Application**.



c. Click **OK**.
Note: You will receive a message that the project has errors. This is because the **testdept.p** file needs to be updated (later in this try it). It does not impact the current test.

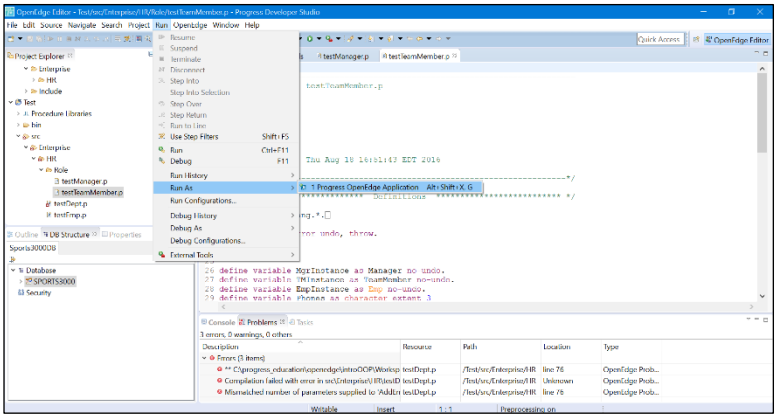
© 2017 Progress Software Corporation. All rights reserved.



4. Run **testTeamMember.p**. Does it run correctly and was the *TeamMember* added with *Manager* information? View the output file.

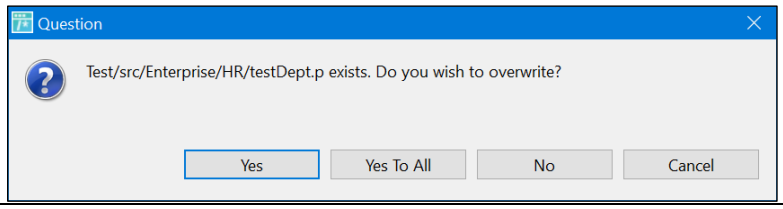
a. With **testTeamMember.p** open in the editor, click the **Run** icon.

b. Select **Progress OpenEdge Application**.

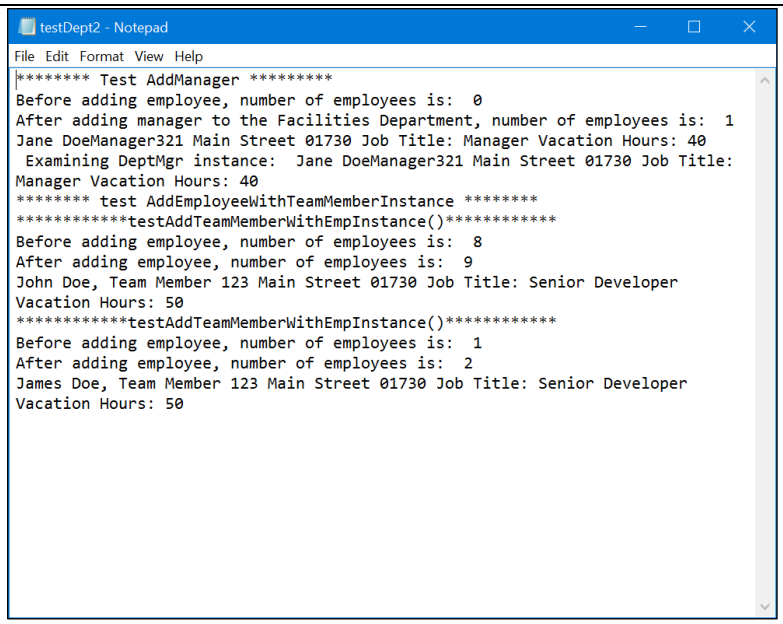


c. Click **OK**.
Note: You will receive a message that the project has errors. This is because the **testdept.p** file needs to be updated (later in this try it). It does not impact the current test.

© 2017 Progress Software Corporation. All rights reserved.

5.	If the files did not run as expected, use the debugger to fix the problem, save your changes, and retest until it runs correctly.
6.	In the **Test/Enterprise/HR** directory, import an ABL procedure file **testDept.p**. View the file to see how each method is defined for testing. **Note**: You will replace the existing **testDept**.p file. a. In Windows Explorer, navigate to the **/progress_education/OpenEdge/introOOP/Exercise/Lesson03** directory and select the **testDept.p** file. b. Drag and drop the **testDept.p** file into the **HR** folder. c. Ensure that Copy files is selected. d. Reply Yes when asked if you want to replace the existing file. 
7.	Run **testDept.p**. Does it run correctly and does the *Dept* class execute properly? View the output file named **TestDept2.out**. a. With **testDept.p** open in the editor, click the **Run** icon. b. Select **Progress OpenEdge Application**. c. Click **OK**.
8.	If it did not run correctly, use the debugger to fix the problem, save your changes, and retest until it runs correctly.

© 2017 Progress Software Corporation. All rights reserved.



© 2017 Progress Software Corporation. All rights reserved.

Try It 3.1: Using inheritance, Wrap-up

Exercise summary

In this Try It, you used the *Emp* class as a super class for the *Manager* and *TeamMember* derived classes. You then modified a data member of the *Emp* super class so that it can be used by its derived classes. Then, you created and developed the code for the *Manager* class. Next, you imported existing code for the *TeamMember* and *Dept* classes, along with test procedures for testing your class hierarchy. Finally, you tested the class hierarchy.

© 2017 Progress Software Corporation. All rights reserved.

Try It 3.2: Using an interface class

Overview

The sample application you are working with in this course supports a Corporation that consists of a number of Business Units. Our application supports two types of Business Units – Company and Franchise.

In this Try It, you will create the *IBusinessUnit* interface class that defines the required methods of the *Company* and *Franchise* classes. You will then create the *Company* class and import the *Franchise* class. Next, you will test the classes.

This exercise has 7 parts. The exercise steps take approximately 45 minutes to complete. You perform this exercise in your live version of Progress OpenEdge.

Before you begin

Before you begin, you must:

Step	Description
1.	Complete the Exercise Setup instructions, if you have not done so already.
2.	Complete the Try It 3.1.

Location of files:
Exercise files: **/progress_education/openedge/IntroOOP/Exercise/Lesson03**
Solution files: **/progress_education/openedge/IntroOOP/Solutions/Lesson03**

If at any time during this course, you need to restore your projects to match where you should have been prior to beginning a Try It, you can do the following:

1. Delete all projects in the workspace.
2. Import all projects in the archive (**.zip**) file for the previous Try It.
3. Do a clean build of your workspace.

© 2017 Progress Software Corporation. All rights reserved.

Part 1—Creating the IBusiness Unit interface class

In this part of the Try It, you will create the *IBusiness Unit* interface class.

If you need help, view the Solution.

Step	Task
1.	In Project Explorer, in the Server project, create a new folder named **BusinessUnit** under the **src/Enterprise** folder.
2.	In Project Explorer, navigate to the **Enterprise/BusinessUnit** directory.
3.	In this directory, create an interface class named *IBusinessUnit*. This class will define a set of properties and methods for the *Company* and *Franchise* classes. **Hint**: Use the *New ABL Interface* wizard.
4.	Add a *using* statement after the error-handling statement. This *using* statement will ensure that the *Dept* class can be accessed.

© 2017 Progress Software Corporation. All rights reserved.

Part 2—Defining data members for the IBusiness Unit interface class

Next, you will define the data members for the *IBusiness Unit* interface class.

If you need help, view the Solution.

Step	Task
1.	Define these properties as *public* with *private* setters: • *Name* as *character* • *NumDepartments* as *Integer*
2.	Save your file. Ensure that it compiles without errors.

© 2017 Progress Software Corporation. All rights reserved.

Part 3—Defining methods for the IBusiness Unit interface class

Next, you will define the methods for the *IBusiness Unit* interface class.

If you need help, view the Solution.

Step	Task
1.	Define the public *AddDepartment*() method with return type *void*. The input parameter for this method will be *pDept* of type *Dept*.
2.	Define the public *GetDepartment* method with return type *Dept*. It has a single *input* parameter, *pDeptCode*, of type *character*.
3.	Define the public *NumberEmployees*() method, which returns *integer* and takes no parameters.
4.	Save your file. Ensure that it compiles without errors.

© 2017 Progress Software Corporation. All rights reserved.

Part 4—Creating the Company class

In this part of the Try It, you will create the *Company* class. This class will implement the *IBusinessUnit* class. Before you add the *Company* class, you will import the include file for the *ttDepartment* temp-table, which will be a data member in the *Company* class.

If you need help, view the Solution.

Step	Task
1.	In Project Explorer, navigate to the **Include** sub-folder under the **src** folder.
2.	In the **Include** folder, import the **ttDepartment.i** file.
3.	In Project Explorer, navigate to the **Enterprise/BusinessUnit** directory.
4.	In this directory, create a class named *Company*. This class will implement the *IBusiness* unit interface class and have a default constructor and destructor that you will later modify to take parameters. **Hint**: Use the *New ABL Class* wizard.
5.	Add a *using* statement before the error-handling statement that will ensure that the *Dept* class can be accessed.
6.	Define the *private* temp-table by including the *ttDepartment* temp-table definition.

© 2017 Progress Software Corporation. All rights reserved.

Part 5—Implementing a constructor, a destructor, and methods for the Company class

Next, you will modify the constructor and the destructor, and define methods for the *Company* class.

If you need help, view the Solution.

Step	Task
1.	Modify the default constructor to take as *input* parameters *pCompanyName* which is of type *character*. In the constructor, assign the *Name* property from the value of *pCompanyName*. Assign *NumDepartments* a value of 0.
2.	Modify the *public AddDepartment*() method that returns *void*. Create a ttDepartment record in the temp-table and assign values to it using the input parameter. Increment the value of NumDepartments by 1. **Hint**: Replace the undo, throw statement with your own code.
3.	Modify the *public* method *GetDepartment*() that returns an instance of *Dept* and takes the department code input parameter. Use a find statement to find the department record in the *ttDepartment* temp-table based upon the input parameter. If the record is found, return a *Dept* instance. **Hints**: You will need to cast the *Object* in the temp-table to the *Dept* class. Replace the undo, throw statement with your own code.
4.	Modify the *public* method *NumberEmployees*() that returns *integer*. Add statements to: • Define an *integer* variable named *iNumEmployees*. • Iterate through each *ttDepartment* record to retrieve the number of employees for each department. You will need to cast the *DeptRef* field to the *Dept* class to call *NumEmployees*() for the instance. Add this number to *iNumEmployees*. **Hint**: Replace the undo, throw statement with your own code.
5.	Add code to the destructor to delete all the created *Department* objects and records in the *ttDepartment* temp-table.
6.	Save your file. Ensure that it compiles without errors.

© 2017 Progress Software Corporation. All rights reserved.

Part 6—Importing the Franchise class

In this part of the Try It, you will import the **Franchise.cls** file. This class also implements the *IBusinessUnit* class.

If you need help, view the Solution.

Step	Task
1.	In the **Role** folder of the Server project, import the **Franchise.cls** file.
2.	Clean and rebuild the Server project.

© 2017 Progress Software Corporation. All rights reserved.

Part 7—Testing the classes

In this part of the Try It, you will import the **testCompany.p** and **testFranchise.p** procedure files for testing the *Company* and *Franchise* classes. Then, you will test them one by one.

The **testCompany.p** file has a *Company* instance, *CompanyInstance*. This file tests the methods of the *Company* class and uses *message* statements to write data to a file. The **testFranchise.p** file has a *Franchise* instance, *FranchiseInstance*. This file tests the methods of the *Franchise* class and uses *message* statements to write data to a file.

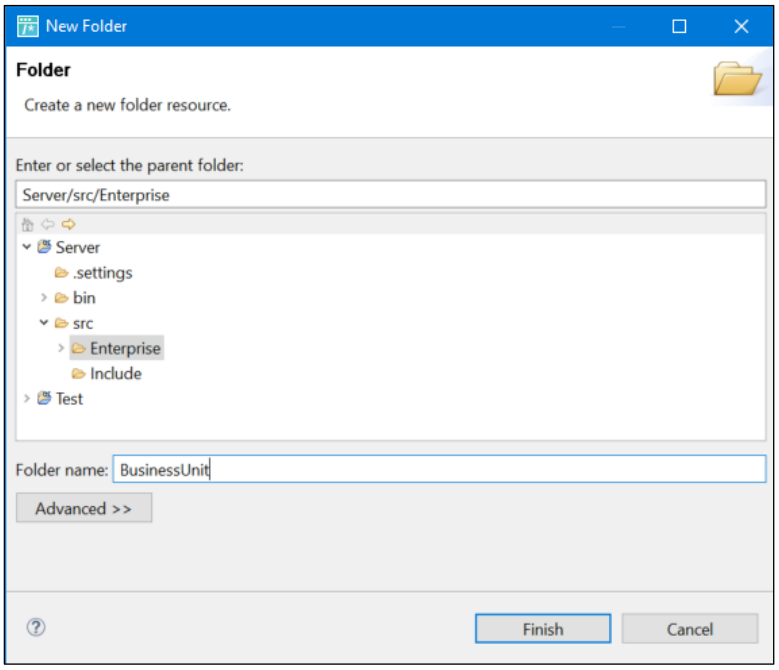
You can view the test procedure files to see how they are written.

If you need help, view the Solution.

Step	Task
1.	In the Test project, create a folder named **BusinessUnit** in the Enterprise directory.
2.	In the **Enterprise/BusinessUnit** directory of the Test project, import the procedure files named **testCompany.p** and **testFranchise.p.**
3.	Run **testCompany.p**. Does it run correctly and was the *Company* information added to the output file? View the output file.
4.	Run **testFranchise.p**. Does it run correctly and was the *Franchise* information added to the output file? View the output file.
5.	If the files did not run as expected, use the debugger to fix the problem, save your changes, and retest until they run correctly.

© 2017 Progress Software Corporation. All rights reserved.

Solution, Part 1—Creating the IBusiness Unit interface class

Step	Solution instructions
1.	In Project Explorer, in the **Server** project, create a new folder named **BusinessUnit** under the **src/Enterprise** folder. a. Right-click **src>Enterprise**. b. Select **New** > **Folder**. c. Enter **BusinessUnit** for the Folder name.  d. Click **Finish**.
2.	In Project Explorer, navigate to the **Enterprise/BusinessUnit** directory.
3.	In this directory, create an interface class named *IBusinessUnit*. This class will be a template with a set of properties and methods for the *Company* and *Franchise* classes that you will create later in this Try It. **Hint**: Use the *New ABL Interface* wizard. a. Right-click **BusinessUnit** b. Select **New** > **ABL Interface**. c. Enter **IBusinessUnit** for the Interface name. d. You can optionally add information to the description. e. Click **Finish**. The **IBusinessUnit.cls** file opens in the editor.

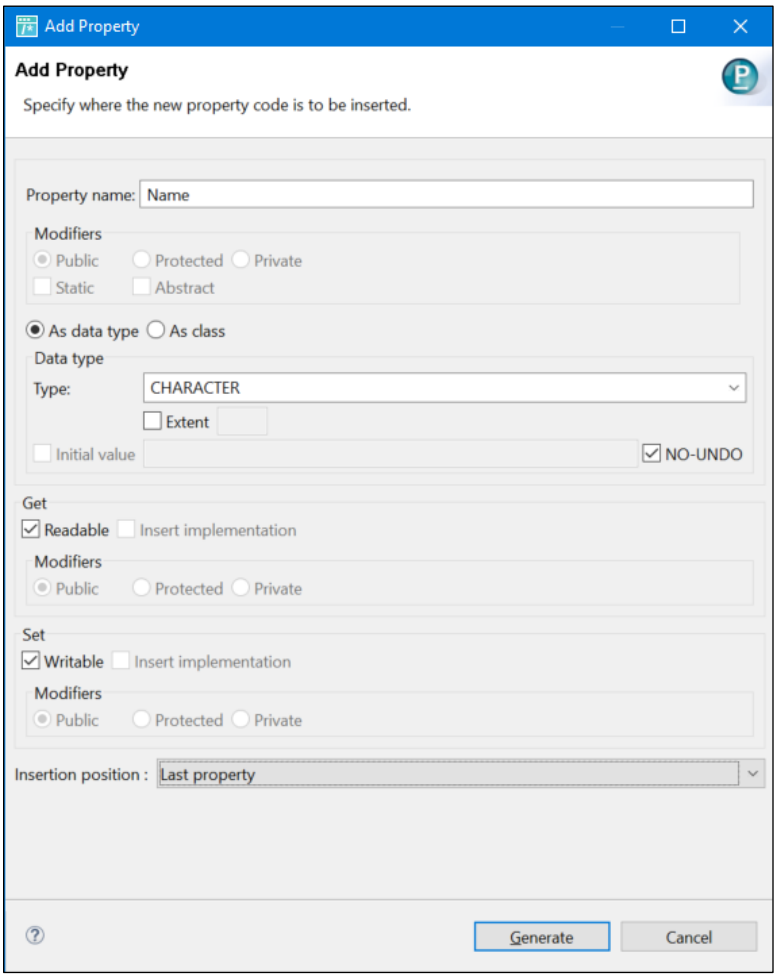
© 2017 Progress Software Corporation. All rights reserved.

4.	The generated file should appear as follows:
	
	`/*----------------------------------------`
	`    File        : IBusinessUnit`
	`    Purpose     :`
	`    Syntax      :`
	`    Description :`
	`    Author(s)   : haribabu`
	`    Created     : Fri Aug 26 11:07:43 IST 2016 --------`
	`----------------------*/`
	`using Progress.Lang.*.`
	`interface Enterprise.BusinessUnit.IBusinessUnit:`
	`end interface.`

5.	Add a *using* statement after the error-handling statement. This *using* statement will ensure that the *Dept* class can be accessed.
	
	`using Progress.Lang.*.`
	`using Enterprise.HR.Dept.`

© 2017 Progress Software Corporation. All rights reserved.

Solution, Part 2—Defining data members for the IBusinessUnit interface class

Step	Solution instructions
1.	Define these properties that will be *public.* • *Name* as *character* • *NumDepartments* as *integer* **Hint**: Use the *Add Property* wizard. a. **Right-click** anywhere in the **IBusinessUnit.cls** file and then select **Source** > **Add Property**. The Add Property wizard opens. b. Enter **Name** for the Property name. c. Select the **CHARACTER** data type for the property from the drop-down list. d. Select **Last Property** for the insertion position.  e. Click **Generate**.

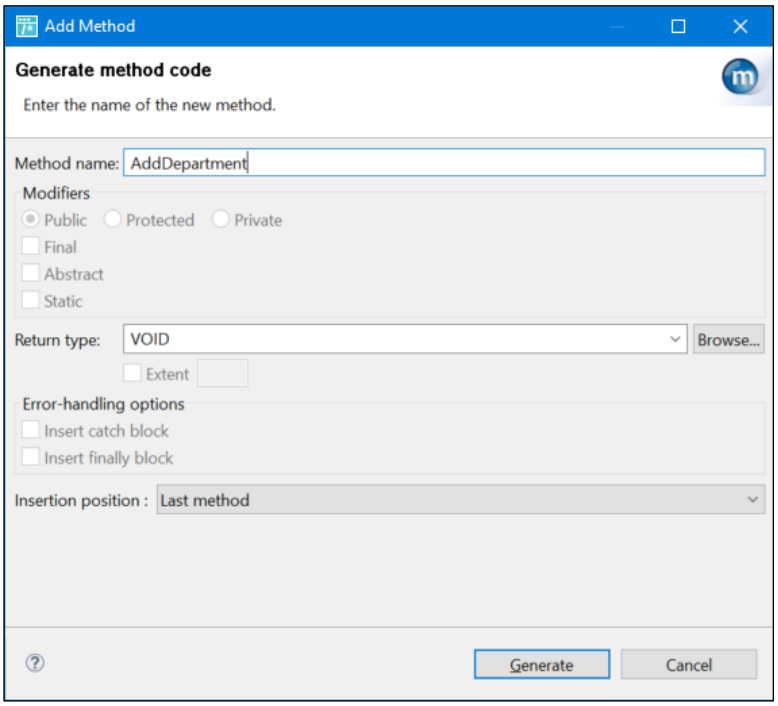
© 2017 Progress Software Corporation. All rights reserved.

	Repeat these steps for the definition of the *NumDepartments* property, making sure you select the *integer* type.
	
	The generated code should appear as follows:
	
	`define public property Name as character no-undo`
	`  get.`
	`  set.`
	
	`define public property NumDepartments as integer no-undo`
	`  get.`
	`  set.`

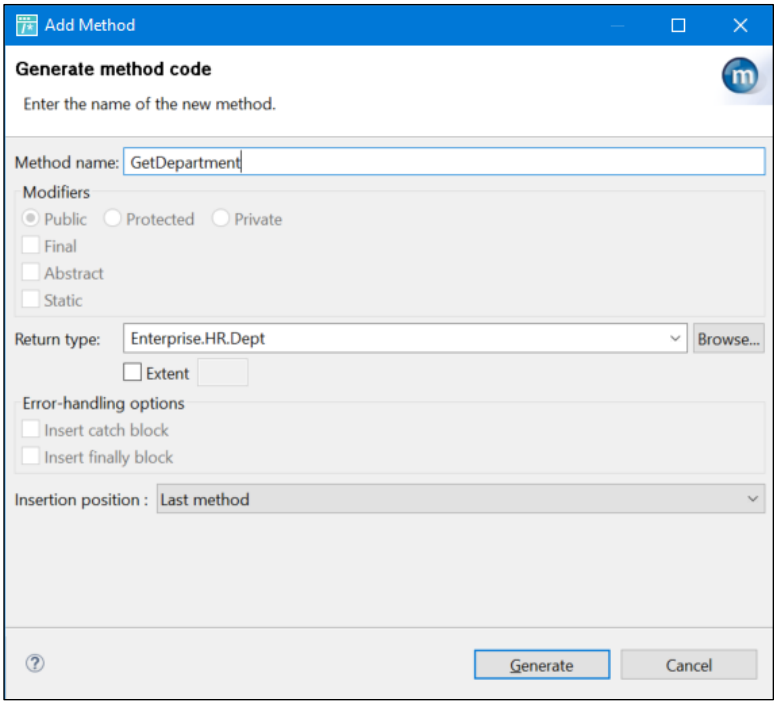
2.	Save your file. Ensure that it compiles without errors.

© 2017 Progress Software Corporation. All rights reserved.

Solution, Part 3—Defining methods for the IBusinessUnit interface class

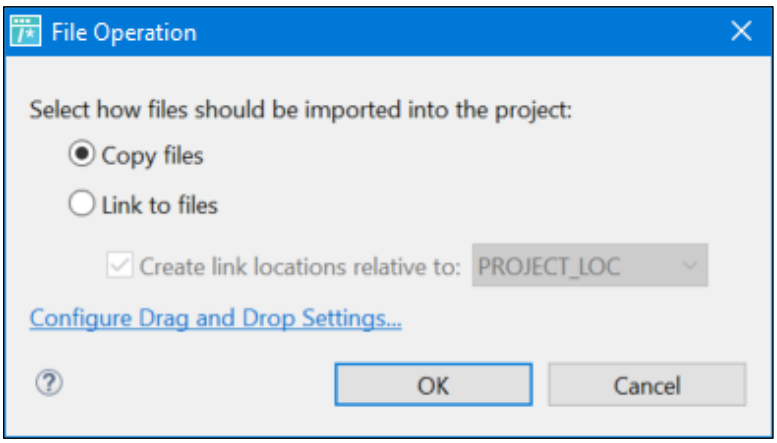
Step	Solution instructions
1.	Define the public *AddDepartment*() method with return type *void*. The input parameters for this method will be *pDept* of type *Dept*. a. Place the cursor anywhere in the file. b. Right-click and then select **Source** > **Add Method**. c. Enter the method name as **AddDepartment**.  d. Click **Generate**. You can enter information in the comment that precedes the method, or you can delete it. e. Specify **input pDept** as **Dept** for the parameter. The definition of this method should be as follows: `method public void AddDepartment(input pDept as Dept).`

© 2017 Progress Software Corporation. All rights reserved.

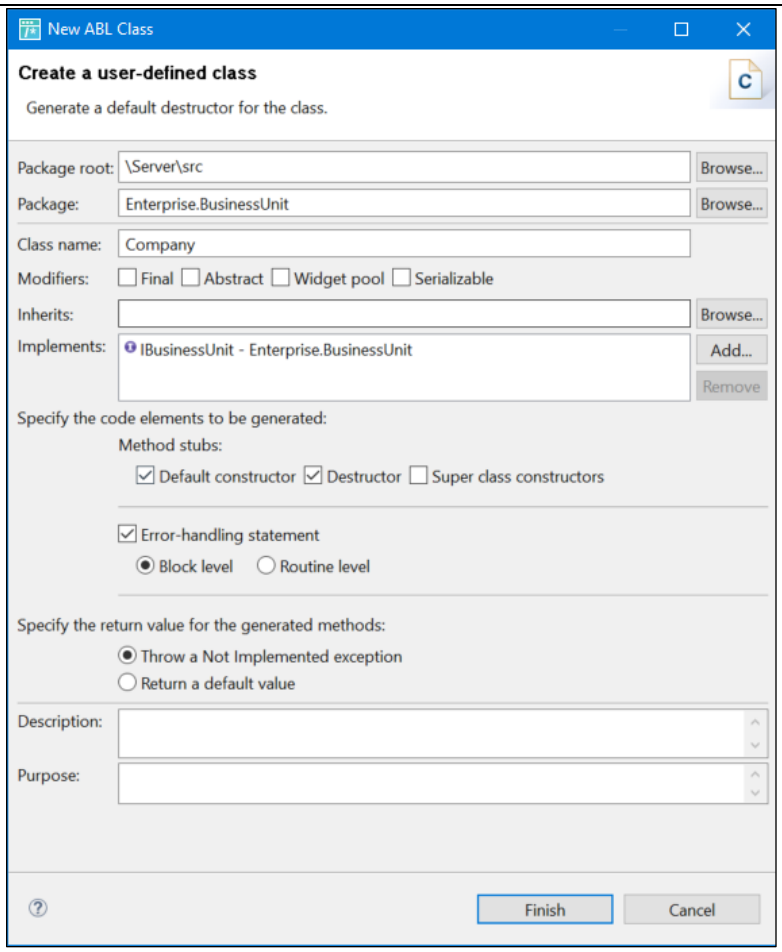
2.	Define the public *GetDepartment* method with return type *Dept*. It has a single *input* parameter, *pDeptCode*, of type *character*. a. Place the cursor anywhere in the source file. b. Right-click and then select **Source** > **Add Method**. c. Enter the method name as **GetDepartment**. d. Browse and select **Dept (Enterprise.HR.Dept)** as return type.  e. Click **Generate**. You can enter information in the comment that precedes the method, or you can delete it. f. Specify **input pDeptCode as character** for the parameter. The definition of this method should be as follows: `method public Enterprise.HR.Dept GetDepartment(input pDeptCode as character).`
3.	Define the public *NumberEmployees*() method, which returns *integer* and takes no parameters. `method public integer NumberEmployees().`
4.	Save your file. Ensure that it compiles without errors.

© 2017 Progress Software Corporation. All rights reserved.

Solution, Part 4—Creating the Company class

Step	Solution instructions
1.	In Project Explorer, navigate to the **Include** sub-folder under the **src** folder.
2.	In this Include folder, import the **ttDepartment.i** file. a. In Windows Explorer, navigate to the **/progress_education/OpenEdge/introOOP /Exercise/Lesson03** directory and select the **ttDepartment.i** file. b. Drag and drop the **ttDepartment.i** file into the **Include** folder. c. Ensure that Copy files is selected.  d. Click **OK**.
3.	In Project Explorer, navigate to the **Enterprise/BusinessUnit** directory.
4.	In this directory, create a class named *Company*. This class will implement the *IBusiness* unit interface class and have a default constructor and destructor that you will later modify to take parameters. **Hint**: Use the *New ABL Class* wizard. a. Select **Enterprise/BusinessUnit**. b. Right-click **BusinessUnit**. c. Select **New** > **ABL Class**. d. Enter **Company** for the Class name. e. Add **IBusinessUnit** interface class in the **Implements** field. f. Select **Default constructor**. g. Select **Destructor**. You can optionally add information to the description.

© 2017 Progress Software Corporation. All rights reserved.



h. Click **Finish**. The **Company.cls** file opens in the editor.

4. The generated code should be as follows:
Note: The methods are implemented from the IBusinessUnit interface class. However, the methods need to be implemented.

```
using Progress.Lang.*.
using Enterprise.BusinessUnit.IBusinessUnit.
block-level on error undo, throw.
class Enterprise.BusinessUnit.Company implements
IBusinessUnit:
       define public property Name as character no-undo
       get.
       set.
define public property NumDepartments as integer no-undo
       get.
       set.
       constructor public Company (  ):
              super ().
       end constructor.
       method public void AddDepartment( input pDept as
Enterprise.HR.Dept )
              undo, throw new
Progress.Lang.AppError("METHOD NOT IMPLEMENTED").
       end method.
       method public Enterprise.HR.Dept
GetDepartment(input pDeptCode as character ):
```


© 2017 Progress Software Corporation. All rights reserved.

```
        undo, throw new Progress.Lang.AppError("METHOD NOT
IMPLEMENTED").
        end method.
        method public integer NumberEmployees(  ):

                undo, throw new
Progress.Lang.AppError("METHOD NOT IMPLEMENTED").
        end method.
        destructor public Company ( ):
        end destructor.
end class.
```

5. Add a *using* statement before the error-handling statement. This *using* statement will ensure that the *Dept* class can be accessed.

```
using Enterprise.HR.Dept.
```

6. Define the *private* temp-table by including the ttDepartment temp-table definition.

   a. In the editor, place the cursor at a blank line before the constructor.
   b. Add this code to the class file:

```
{include/ttDepartment.i &ClassAccess = "private"}
```


© 2017 Progress Software Corporation. All rights reserved.

Solution, Part 5—Implementing a constructor, a destructor, and methods for the Company class

Step	Solution instructions

1.	Modify the default constructor to take as *input* parameter *pCompanyName*, which is of type *character*. In the constructor, assign the *Name* property from the value of *PCompanyName*. Assign *NumDepartments* a value of 0.
	a. Place your cursor in the parameters area for the constructor.
	b. Add code for this parameter:
	c. Add an assign statement to set the *Name* and *NumDepartments* properties.
	
	The code for the constructor should appear as follows:
	
	`constructor public Company (input pCompanyName as character):`
	`  super ().`
	`  assign`
	`  Name = pCompanyName`
	`  NumDepartments = 0`
	`  .`
	`end constructor.`

2.	Modify the public *AddDepartment*() method that returns *void*. Create a *ttDepartment* record in the temp-table and assign values to it using the input parameter. Increment the value of *NumDepartments* by 1.
	Hint: Replace the undo, throw statement with your own code.
	
	The code for this method should appear as follows:
	
	`method public void AddDepartment( input pDept as Enterprise.HR.Dept ):`
	`  create ttDepartment.`
	`  assign`
	`  ttDepartment.Name = pDept:DeptName`
	`  ttDepartment.DeptCode = pDept:DeptCode`
	`  ttDepartment.DeptRef = pDept`
	`  NumDepartments = NumDepartments + 1.`
	`  .`
	`  return.`
	`end method.`

3.	Modify the *public* method *GetDepartment*() that returns an instance of *Dept* and takes the department code input parameter. Use a *find* statement to find the department record in the *ttDepartment* temp-table based upon the input parameter. If the record is found, return a *Dept* instance. **Hints**: You will need to cast the *Object* in the temp-table to the *Dept* class. Replace the undo, throw statement with your own code. The code for this method should appear as follows:

© 2017 Progress Software Corporation. All rights reserved.

```
method public Enterprise.HR.Dept GetDepartment( input
pDeptCode as character ):
find ttDepartment where ttDepartment.DeptCode =
pDeptCode no-error.
if available(ttDepartment)
              then return.
cast(ttDepartment.DeptRef,Dept).
              else return ?.
end method.
```

4. Modify the *public* method *NumberEmployees*() that returns *integer*. Add statements to:

- Define an *integer* variable named *iNumEmployees*.
- Iterate through each *ttDepartment* record to retrieve the number of employees for each department. You will need to cast the *DeptRef* field to the *Dept* class to call *NumEmployees*() for the instance. Add this number to *iNumEmployees*.

Hint: Replace the undo, throw statement with your own code.

The code for this method should appear as follows:

```
method public integer NumberEmployees(  ):

       define variable iNumEmployees as integer no-undo
initial 0.
       for each ttDepartment:
                  iNumEmployees =
cast(ttDepartment.DeptRef,Dept):NumEmployees +
iNumEmployees.
        end.
        return iNumEmployees.
end method.
```

5. Add code to the destructor to delete all the created Department objects and records in the *ttDepartment* temp-table.

The code for the destructor should appear as follows:

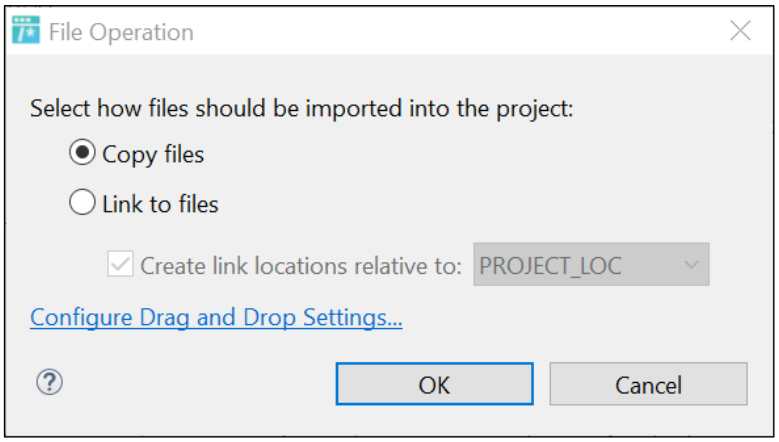
```
destructor public Company ( ):
       /* delete all the Department objects */
       for each ttDepartment:
          delete object
cast(ttDepartment.DeptRef,Dept).
          delete ttDepartment.
       end.
end destructor.
```

6. Save your file. Ensure that it compiles without errors.

© 2017 Progress Software Corporation. All rights reserved.

Solution, Part 6—Importing the Franchise class

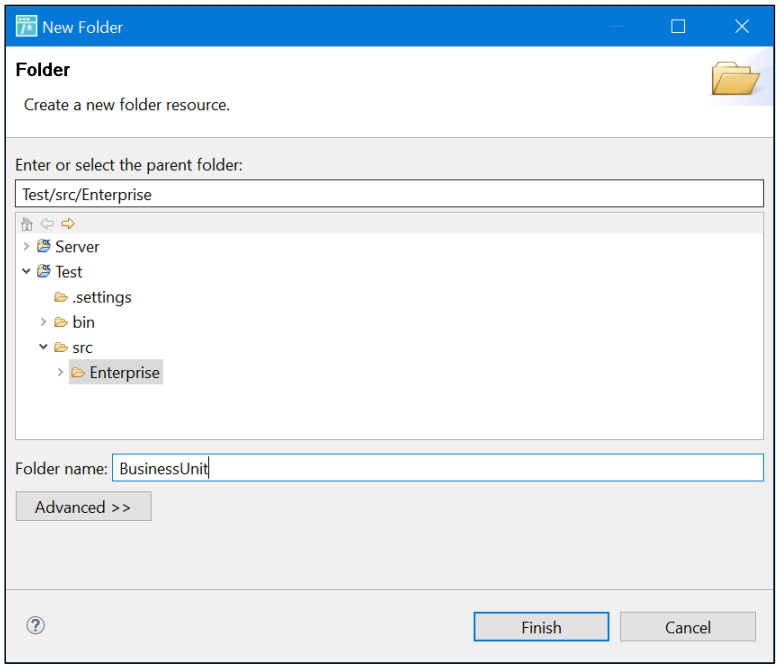
Step	Solution instructions

1.	In the BusinessUnit folder of the Server project, import the Franchise.cls file.
	a. In Windows Explorer, navigate to the **/progress_education/OpenEdge/introOOP/Exercise/Lesson03** directory and select the **Franchise.cls** file.
	b. Drag and drop the **Franchise.cls** file into the **BusinessUnit** folder.
	c. Ensure that Copy files is selected.
	 
	
	d. Click OK.
	Note: This is what the beginning of the file should look like.
	`/*----------------------------------------`
	`File : Franchise`
	`Purpose :`
	`Syntax :`
	`Description :`
	`Author(s) :`
	`Created : Fri Aug 19 14:45:07 EDT 2016`
	`----------------------------------------*`
	`block-level on error undo, throw.`
	`using Enterprise.BusinessUnit.Franchise from propath.`
	`using Enterprise.BusinessUnit.IBusinessUnit from propath.`
	`using Enterprise.HR.Dept from propath.`
	`using Enterprise.HR.Emp from propath.`
	`class Enterprise.BusinessUnit.Franchise implements IBusinessUnit:`

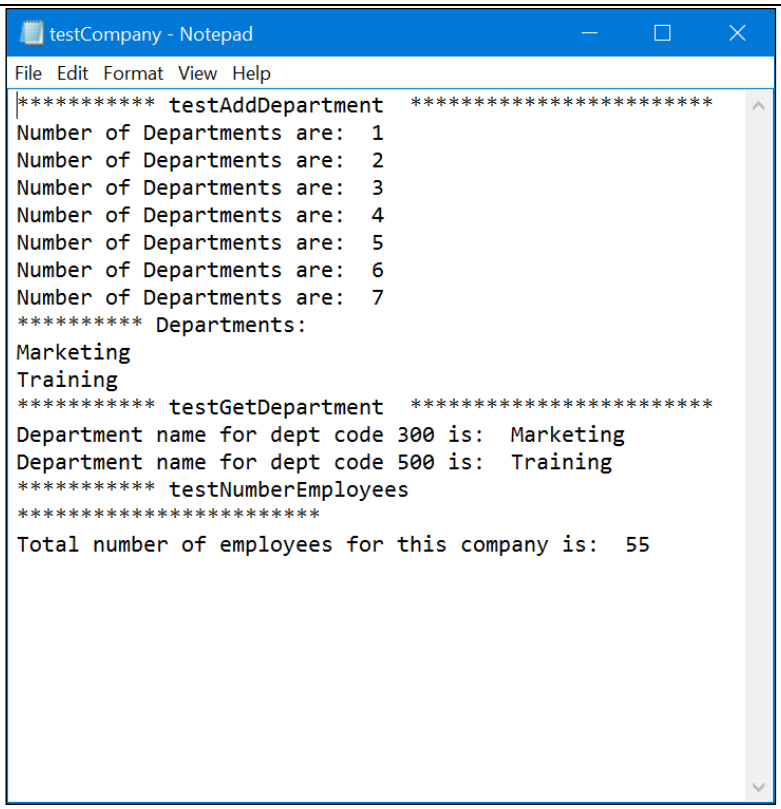
2.	Clean and rebuild the Server project.

© 2017 Progress Software Corporation. All rights reserved.

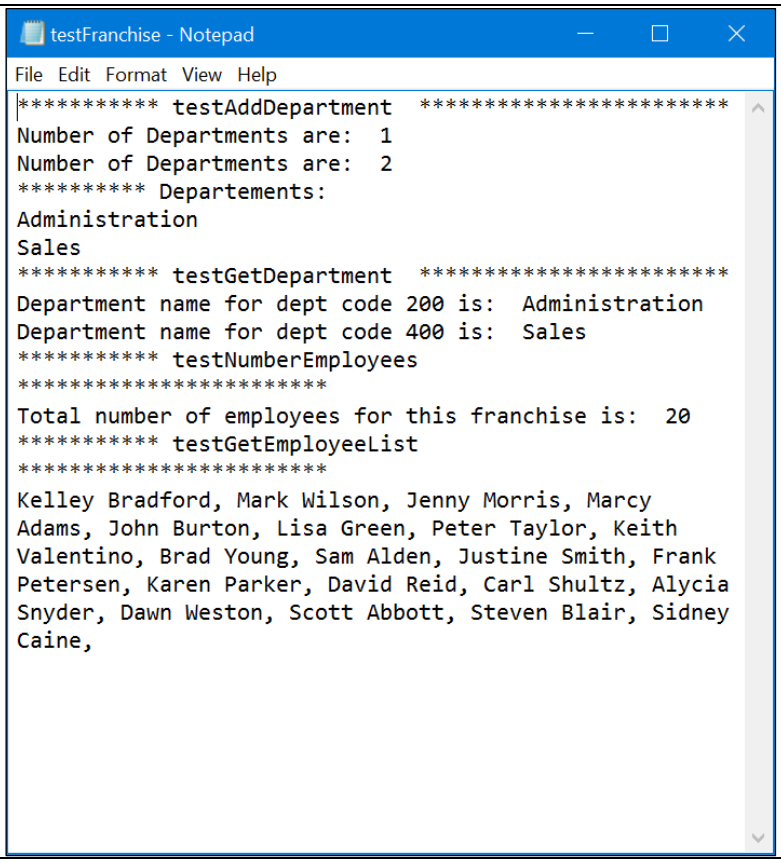
Solution, Part 7—Testing the classes

Step	Solution instructions
1.	In the Test project, create a folder named **BusinessUnit** in the Enterprise directory. a. Right-click the **Enterprise** folder of the Test project. b. Select **New>Folder**. c. Enter **BusinessUnit** as the folder name.  d. Click **Finish**.
2.	In the **Enterprise/BusinessUnit** directory of the Test project, import the procedure files named **testCompany.p** and **testFranchise.p.** a. In Windows Explorer, navigate to the **/progress_education/OpenEdge/introOOP/Exercise/Lesson03** directory and select the **testCompany.p** and **testFranchise.p** files. b. Drag and drop the **testCompany.p** and **testFranchise.p** files into the **BusinessUnit** folder. c. Ensure that Copy files is selected. d. Click **OK**.
3.	Run **testCompany.p**. Does it run correctly and was the *Company* information added to the output file? View the output file. a. With **testCompany.p** open in the editor, click the **Run** icon. b. Select **Progress OpenEdge Application**. c. Click **OK**.

© 2017 Progress Software Corporation. All rights reserved.

	
4.	Run **testFranchise.p**. Does it run correctly and was the *Franchise* information added to the output file? View the output file. a. With **testFranchise.p** open in the editor, click the **Run** icon. b. Select **Progress OpenEdge Application**. c. Click **OK**.

© 2017 Progress Software Corporation. All rights reserved.

	
5.	If the files did not run as expected, use the debugger to fix the problem, save your changes, and retest until they run correctly.

© 2017 Progress Software Corporation. All rights reserved.

Try It 3.2: Using an interface class, Wrap-up

Exercise summary

In this Try It, you created the *IBusinessUnit* class as an interface class that defines the required methods of the *Company* and *Franchise* classes. You then created the *Company* class and imported the *Franchise* class which use the interface class and tested them.

© 2017 Progress Software Corporation. All rights reserved.

Try It 3.3: Using a singleton and creating classes dynamically

Overview

In this Try It, you will import the *Corporation* class and add a static data member and a static constructor to it so that it can be used as a singleton. Then, you will add code to the *Corporation* class to create the business unit instances (of types *Company* and *Franchise*) of the corporation dynamically. The *Corporation* instance will be created automatically when the AVM detects the first use of *Corporation*.

This exercise has 5 parts. The exercise steps take approximately 30 minutes to complete. You perform this exercise in your live version of Progress OpenEdge.

Before you begin

Before you begin, you must:

Step	Description
1.	Complete the Exercise Setup instructions, if you have not done so already.
2.	Complete the Try It 3.2.

Location of files:
Exercise files: **/progress_education/openedge/IntroOOP/Exercise/Lesson03**
Solution files: **/progress_education/openedge/IntroOOP/Solutions/Lesson03**

If at any time during this course, you need to restore your projects to match where you should have been prior to beginning a Try It, you can do the following:

1. Delete all projects in the workspace.
2. Import all projects in the archive (**.zip**) file for the previous Try It.
3. Do a clean build of the workspace.

© 2017 Progress Software Corporation. All rights reserved.

Part 1—Importing the Corporation class and the ttBusinessUnit include file

In this part of the Try It, you will import the *Corporation* class and the *ttBusinessUnit* include file that you will use in this exercise.

If you need help, view the Solution.

Step	Task
1.	In Project Explorer, navigate to the **Enterprise** directory and import the **Corporation.cls** file. **Note**: You will see compilation errors that will be resolved as you add code to the file.
2.	Navigate to the **Include** directory and import the **ttBusinessUnit.i** file. This file contains the temp-table definition for a set of business units. Each record in the temp-table will hold a reference to an instance of a business unit.

© 2017 Progress Software Corporation. All rights reserved.

Part 2—Defining static data members for the Corporation class

Next, you will define the static data members for the *Corporation* class.

If you need help, view the Solution.

Step	Task
1.	Define static properties that will be *public* with *private* setters: • *CorpID* as *character* • *Instance* as *Enterprise.Corporation* **Hints**: Use the *Add Property* wizard. Each setter will have an implementation that enables the property to be set.
2.	Add code in the implementation for the set accessor for the *CorpID* property to set its value from the input *arg* parameter.
3.	Add code in the implementation for the set accessor for the *Instance* property to set its value from the input *arg* parameter.

© 2017 Progress Software Corporation. All rights reserved.

Part 3—Defining a static constructor for the Corporation class

Next, you will modify the default constructor and define a static constructor for the *Corporation* class.

If you need help, view the Solution.

Step	Task
1.	Modify the default constructor to set the *CorpID* property to "*Progress Software Corporation"*.
2.	Define a static constructor for the *Corporation* class to create a new instance of *Corporation* and assign it to the *Instance* property.

© 2017 Progress Software Corporation. All rights reserved.

Part 4—Adding code to the InitializeBusinessUnit() method to create instances dynamically

Next, you will add code to the *for each* block of the *InitializeBusinessUnits()* method to create *Company* or *Franchise* instances dynamically.

If you need help, view the Solution.

Step	Task
1.	Locate the public *InitializeBusinessUnits*() method and view the code that is there already. This code iterates through all business unit records in the *ttBusinessUnit* temp-table. It reads a department JSON file to initialize the departments for each business unit.
2.	In the *InitializeBusinessUnits*() method, add code at the beginning of the *for each* block to test if a business unit is of type *Company*. To do this, you will test the value of the type field in the *ttBusinessUnit* temp-table. If the business unit is of type *Company*, add a statement to create a new *Company* object dynamically. Otherwise, create a new *Franchise* object dynamically.
3.	Save your file. Ensure that it compiles without errors.

© 2017 Progress Software Corporation. All rights reserved.

Part 5—Testing the Corporation class

The **testCorporation.p** file will test if the static *Corporation* instance is created automatically and if instances of *Company* and *Franchise* are accessible. The procedure file also uses message statements to write data to a file.

You can view the test procedure file to see how it is written.

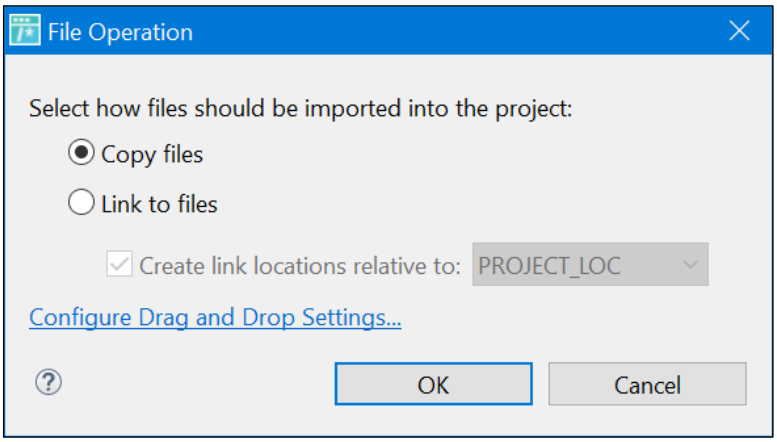
If you need help, view the Solution.

Step	Task
1.	In the **Enterprise** folder of the Test project, import the procedure file named **testCorporation.p.**
2.	Run **testCorporation.p**. Does it run correctly and was the *Corporation* information added to the output file? View the output file.
3.	If the file did not run as expected, use the debugger to fix the problem, save your changes, and retest until it runs correctly.

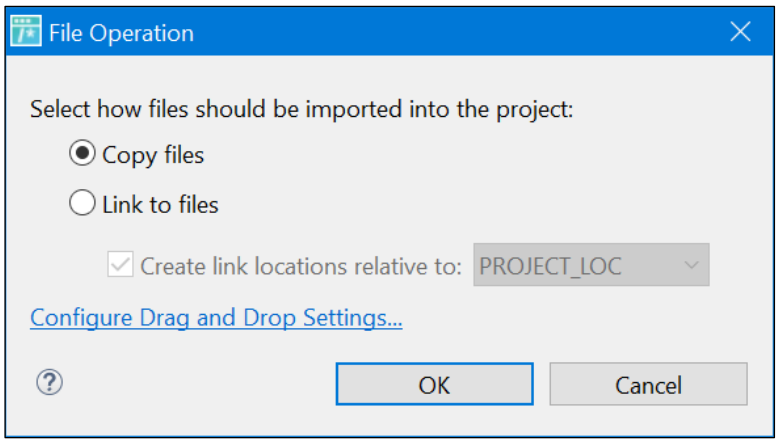
© 2017 Progress Software Corporation. All rights reserved.

Solution, Part 1—Importing the Corporation class and the ttBusinessUnit include file

Step	Solution instructions

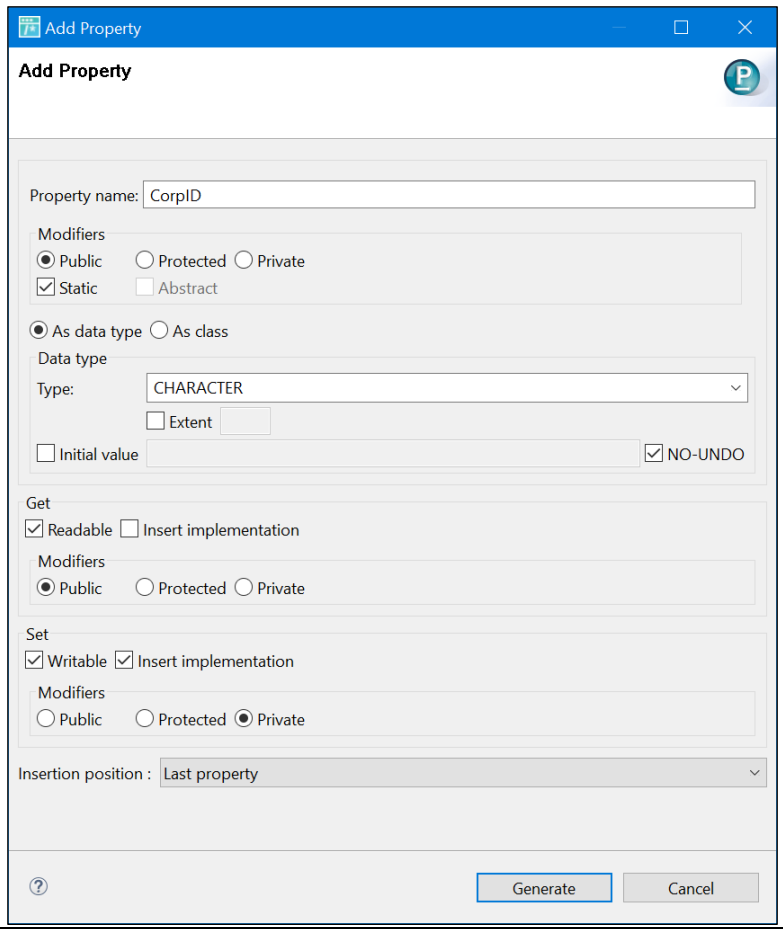
1.	In Project Explorer, navigate to the **Enterprise** directory and import the **Corporation.cls** file.
	a. In Windows Explorer, navigate to the **/progress_education/OpenEdge/introOOP/Exercise/Lesson03** directory and select the **Corporation.cls** file.
	b. Drag and drop the **Corporation.cls** file into the **Enterprise** folder.
	c. Ensure that Copy files is selected.
	 
	
	d. Click **OK**.
	Note: You will see compilation errors that will be resolved as you add code to the file.
	Note: This is what the beginning of the file should look like.
	`/*----------------------------------------`
	`File : Corporation`
	`Purpose :`
	`Syntax :`
	`Description :`
	`Author(s) : haribabu`
	`----------------------------------*/`
	`block-level on error undo, throw.`
	`using Enterprise.BusinessUnit.IBusinessUnit from propath.`
	`using Enterprise.Corporation from propath.`
	`using Enterprise.HR.Dept from propath.`
	`using Enterprise.HR.Role.TeamMember from propath.`

© 2017 Progress Software Corporation. All rights reserved.

2.	Navigate to the **Include** directory and import the **ttBusinessUnit.i** file. This file contains the temp-table definition for a set of business units. Each record in the temp-table will hold a reference to an instance of a business unit. a. In Windows Explorer, navigate to the **/progress_education/OpenEdge/introOOP/Exercise/Lesson03** directory and select the **ttBusinessUnit.i** file. b. Drag and drop the **ttBusinessUnit.i** file into the **Include** folder. c. Ensure that Copy files is selected.  d. Click **OK**.

© 2017 Progress Software Corporation. All rights reserved.

Solution, Part 2—Defining static data members for the Corporation class

Step	Solution instructions
1.	Define static properties that will be *public* with *private* setters: • *CorpID* as *character* • *Instance* as *Enterprise.Corporation* **Hints**: Use the *Add Property* wizard. Each setter will have an implementation that enables the property to be set. a. In the editor, place the cursor anywhere in the source file. b. **Right-click** and then select **Source** > **Add Property**. The Add Property wizard opens. c. Enter **CorpID** for the **Property** name. d. Select **Static** for the type of the property. e. Select **Character** for the property type. f. Set **Insert implementation** for the Set accessor. g. Select **Private** for the Set accessor. h. Select **Last property** for the insertion position. 

© 2017 Progress Software Corporation. All rights reserved.

	i. Click **Generate**.
	
	Repeat these steps for the definition of the *Instance* property, making sure you type *Enterprise.Corporation* for the class for the property type.
	
	The generated code should appear as follows:
	
	`define public static property CorpID as character no-undo`
	`  get.`
	`  private set (input arg as character):`
	`  end set.`
	`define public static property Instance as Enterprise.Corporation no-undo`
	`  get.`
	`  private set(input arg as Enterprise.Corporation):`
	`  end set.`

2.	Add code in the implementation for the set accessor for the *CorpID* property to set its value from the input *arg* parameter.
	
	The *CorpID* property should now appear as follows:
	
	`define public static property CorpID as character no-undo`
	`  get.`
	`  private set(input arg as character):`
	`  CorpId = arg.`
	`  end set.`

3.	Add code in the implementation for the set accessor for the *Instance* property to set its value from the input *arg* parameter.
	
	The Instance property should now appear as follows:
	
	`define public static property Instance as Enterprise.Corporation no-undo`
	`  get.`
	`  private set(input arg as Enterprise.Corporation):`
	`  Instance = arg.`
	`  end set.`

© 2017 Progress Software Corporation. All rights reserved.

Solution, Part 3—Defining a static constructor for the Corporation class

Step	Solution instructions

1.	Modify the default constructor to set the value for the *CorpID* property to "*Progress Software Corporation"*.
	
	The definition of the default constructor should be as follows:
	
	`constructor  Corporation():`
	`    super().`
	`    CorpID = "Progress Software Corporation".`
	`end constructor.`

2.	Define a static constructor for the *Corporation* class to create a new instance of Corporation and assign it to the Instance property.
	
	The definition of this constructor should be as follows:
	
	`constructor static Corporation():`
	`    Instance = new Enterprise.Corporation().`
	`end constructor.`

© 2017 Progress Software Corporation. All rights reserved.

Solution, Part 4—Adding code to the InitializeBusinessUnit() method to create instances dynamically

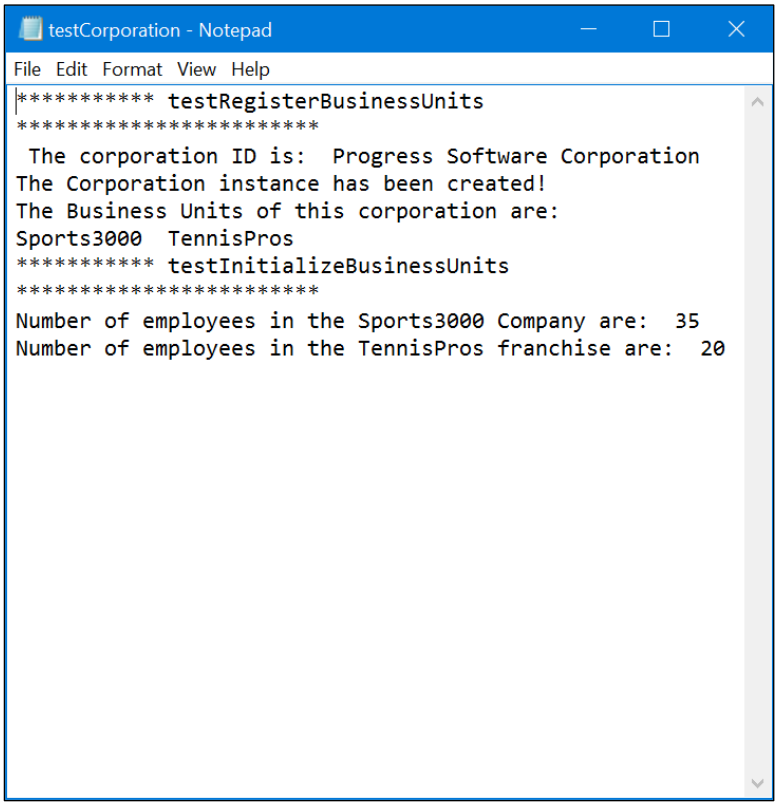
Step	Solution instructions
1.	Locate the public *InitializeBusinessUnits*() method and view the code that is there already. This code iterates through all business unit records in the *ttBusinessUnit* temp-table. It reads a department JSON file to initialize the departments for each business unit.

2.	In the *InitializeBusinessUnits*() method, add code at the beginning of the *for each* block to test if a business unit is of type *Company*. To do this, you will test the value of the type field in the *ttBusinessUnit* temp-table.
	If the business unit is of type *Company*, add a statement to create a new *Company* object dynamically. Otherwise, create a new *Franchise* object dynamically.
	
	The beginning of this *for each* block should appear as follows:
	
	`for each ttBusinessUnit:`
	`  /* add code here to dynamically create the business unit depending on the Type value  */`
	`if ttBusinessUnit.Type = "Company"`
	`then`
	`ttBusinessUnit.BusinessUnitRef = dynamic-new (ttBusinessUnit.Classname) (ttBusinessUnit.Name).`
	`else`
	`ttBusinessUnit.BusinessUnitRef = dynamic-new (ttBusinessUnit.Classname) (ttBusinessUnit.Name,ttBusinessUnit.MaxNumDepartments).`
	`empty temp-table ttDepartment.`

3.	Save your file. Ensure that it compiles without errors.

© 2017 Progress Software Corporation. All rights reserved.

Solution, Part 5—Testing the Corporation class

Step	Solution instruction
1.	In the **Enterprise** folder of the Test project, import the procedure file named **testCorporation.p.** a. In Windows Explorer, navigate to the **/progress_education/OpenEdge/introOOP/Exercise/Lesson03** directory and select the **testCorporation.p** file. b. Drag and drop the **testCorporation.p** file into the **Enterprise** folder. c. Ensure that Copy files is selected. d. Click **OK**.
2.	Run **testCorporation.p**. Does it run correctly and was the *Corporation* information added to the output file? View the output file. a. With **testCorporation.p** open in the editor, click the **Run** icon. b. Select **Progress OpenEdge Application**. c. Click **OK**.
3.	If the file did not run as expected, use the debugger to fix the problem, save your changes, and retest until it runs correctly. 

© 2017 Progress Software Corporation. All rights reserved.

Try It 3.3: Using a singleton and creating classes dynamically, Wrap-up

Exercise summary

In this Try It, you imported the *Corporation* class and added a static data member and a static constructor to it so that it can be used as a singleton. Then, you added code to the *Corporation* class to create the business unit instances (of types *Company* and *Franchise*) of the corporation dynamically.

© 2017 Progress Software Corporation. All rights reserved.

Try It 3.4: Using events

Overview

In this Try It, you will define and publish an event in the *Manager* class and then subscribe to it in the *Dept* class. You will also write an event handler to handle the event. Finally, you will test your event handling code to ensure it executes correctly.

This exercise has 3 parts. The exercise steps take approximately 30 minutes to complete. You perform this exercise in your live version of Progress OpenEdge.

Before you begin

Before you begin, you must:

Step	Description
1.	Complete the Exercise Setup instructions, if you have not done so already.
2.	Complete the Try It 3.3.

Location of files:
Exercise files: **/progress_education/openedge/IntroOOP/Exercise/Lesson03**
Solution files: **/progress_education/openedge/IntroOOP/Solutions/Lesson03**

If at any time during this course, you need to restore your projects to match where you should have been prior to beginning a Try It, you can do the following:

1. Delete all projects in the workspace.
2. Import all projects in the archive (**.zip**) file for the previous Try It.
3. Do a clean build of the workspace.

© 2017 Progress Software Corporation. All rights reserved.

Part 1—Defining and publishing an event in the Manager class

In this part of the Try It, you will define and publish an event in the *Manager* class. This event will be subscribed to by the *Dept* class later in this Try It.

If you need help, view the Solution.

Step	Task
1.	In Project Explorer, open the *Manager* class and define a new public event *DischargeEmployee*.
2.	In the *DischargeEmployee* event, define the following input parameters. • *pFirstname* as *character* • *pLastname* as *character*
3.	Define the *public DischargeTeamMember*() method with return type *void*.
4.	To the public *DischargeTeamMember*() method pass the following parameters. • *pFirstname* as *character* • *pLastname* as *character*
5.	Modify the public *DischargeTeamMember*() method to publish the *DischargeEmployee* event.
6.	Save this file. Ensure that it compiles without errors.

© 2017 Progress Software Corporation. All rights reserved.

Part 2—Modify the Dept class to subscribe to the event

In this part of the Try It, you will modify the *Dept* class to subscribe to the *DischargeEmployee* event defined and published in the *Manager* class. You also write the event handler method that handles the event.

If you need help, view the Solution.

Step	Task
1.	In Project Explorer, open the *Dept* class; at the end of the *AddManager*() method, add a statement to subscribe to the *DischargeEmployee* event specifying the *DischargeEmployee_Handler* method. **Hint**: You will see an error in this statement because you have not yet created the *DischargeEmployee_Handler*() method.
2.	Define the public *DischargeEmployee_Handler*() method with return type *void*.
3.	To the public *DischargeEmployee_Handler*() method add the following input parameters. • *pFirstname* as *character* • *pLastname* as *character*
4.	Modify the public *DischargeEmployee_Handler*() method to find the discharged employee in the *ttEmployee* temp-table based on the first and last name.
5.	If an employee is found, you should add code in a *do* block to: • Add a statement to open a file for output. The name of the file you will be writing to is **Discharges.out**. It will be located in the **/progress_education/openedge/IntroOOP/log** directory. **Hint**: Use the full pathname of this file. • Add a *message* statement to write the number of employees terminated. • Delete the employee record by casting the *EmpRef* field and return the *Emp* instance. **Hint**: You will need to cast the temp-table field *ttEmployee.EmpRef* to the type *Emp*.
6.	In the *do* block, add a statement to delete the *ttEmployee* temp-table record that was found.
7.	In the *do* block, add a statement to decrease the employee count by 1.
8.	In the *do* block, add a statement to close the output file.
9.	Save this file. Ensure that it compiles without errors.

© 2017 Progress Software Corporation. All rights reserved.

Part 3—Testing the Dept class event

You will run a test procedure to confirm that the *Dept* class event executes correctly.

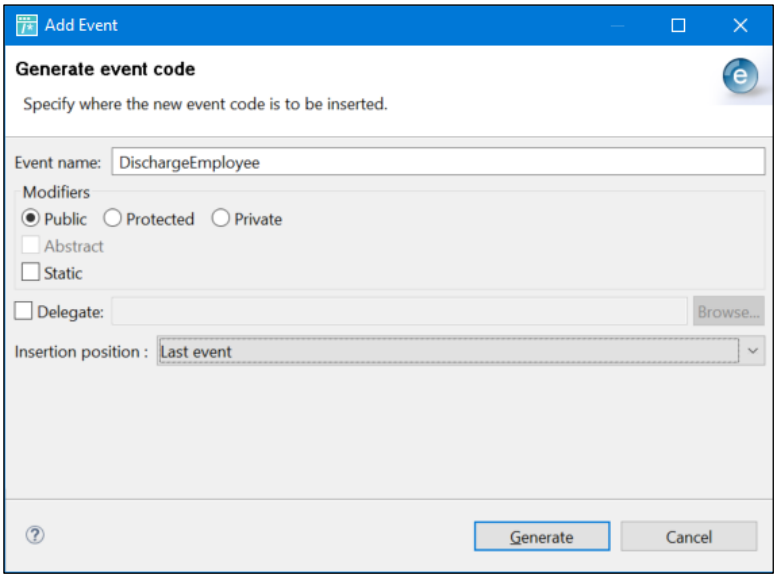
You can view the test procedure file to see how it is written.

If you need help, view the Solution.

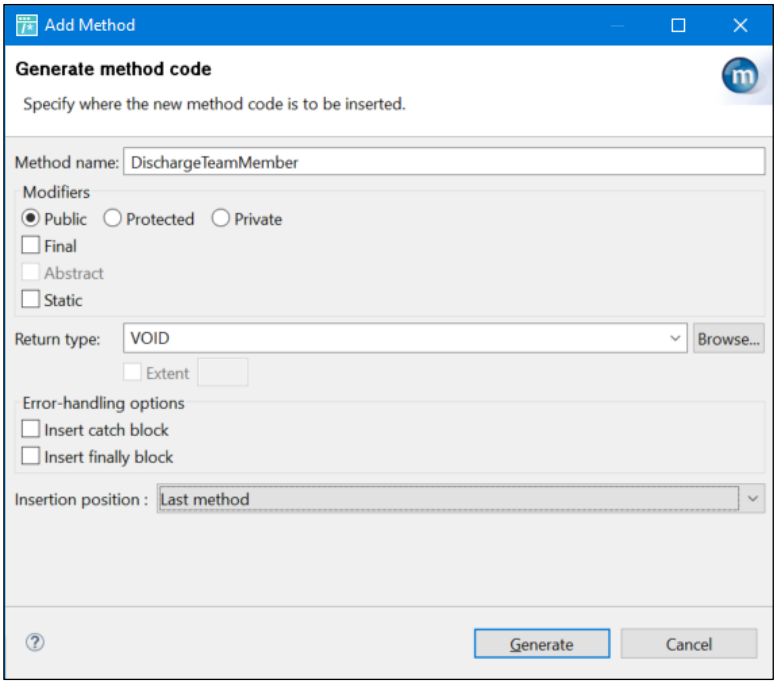
Step	Task
1.	In the **Enterprise/HR** folder of the Test project, import the procedure file named **testEvents.p.**
2.	Run **testEvents.p**. Does it run correctly and was the discharged employee information added to the output file? View the output file.
3.	If the file did not run as expected, use the debugger to fix the problem, save your changes, and retest until it runs correctly.

© 2017 Progress Software Corporation. All rights reserved.

Solution, Part 1—Defining and publishing an event in the Manager class

Step	Solution instructions
1.	In Project Explorer, open the Manager class and define a new public event *DischargeEmployee*. **Hint**: Use the *Add Event* wizard. a. In the editor, place the cursor anywhere in the file. b. **Right-click** and then select **Source** > **Add Event**. The Add Event wizard opens. c. Enter *DischargeEmployee* for the event name. d. Select **Last event** for the insertion position.  e. Click **Generate**. The generated code should appear as follows: `define public event DischargeEmployee signature void ().`
2.	In the *DischargeEmployee* event, pass the following input parameter. • *pFirstname* as *character* • *pLastname* as *character* The modified event should appear as follows: `define public event DischargeEmployee signature void (input pFirstname as character, input pLastName as character).`

© 2017 Progress Software Corporation. All rights reserved.

3.	Define the public *DischargeTeamMember*() method with return type *void*.
	a. Place the cursor anywhere in the file.
	b. Right-click and then select **Source** > **Add Method**.
	c. Enter the method name as **DischargeTeamMember**.
	d. Select **void** as return type.
	 
	
	e. Click **Generate**.
	
	You can enter information in the comment that precedes the method, or you can delete it.
	
	The definition of this method should be as follows:
	
	`method public void DischargeTeamMember ():`
	`  return.`
	`end method.`

4.	To the public *DischargeTeamMember*() method pass the following parameters.
	• *pFirstname* as *character*
	• *pLastname* as *character*
	
	`method public void DischargeTeamMember (input pFirstname as character, pLastname as character):`
	`  return.`
	`end method.`

© 2017 Progress Software Corporation. All rights reserved.

5.	Modify the public *DischargeTeamMember*() method to publish the *DischargeEmployee* event.
	
	The modified method should appear as follows:
	
	`method public void DischargeTeamMember (input pFirstname as character, pLastname as character): DischargeEmployee:Publish(pFirstname, pLastname) no-error.`
	`end method.`

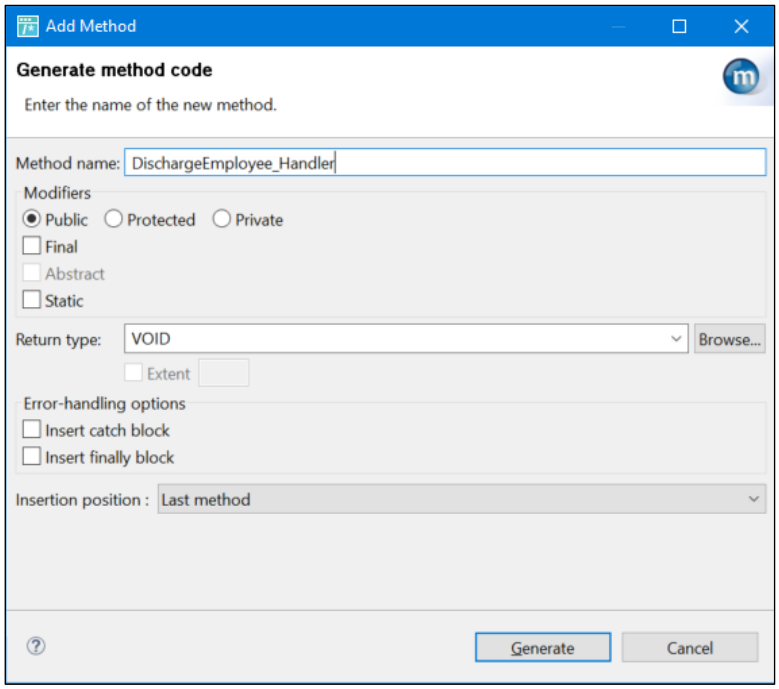
6.	Save this file. Ensure that it compiles without errors.

© 2017 Progress Software Corporation. All rights reserved.

Solution, Part 2—Modify the Dept class to subscribe to the event

Step	Solution instructions

1.	In Project Explorer, open the *Dept* class. At the end of the *AddManager*() method, add a statement to subscribe to the *DischargeEmployee* event specifying the *DischargeEmployee_Handler* method.
	Hint: You will see an error on this statement because you have not yet created the *DischargeEmployee_Handler* method.
	`DeptMgr:DischargeEmployee:`
	`Subscribe(DischargeEmployee_Handler).`

2.	Define the public *DischargeEmployee_Handler*() method with return type *void*.
	Hint: Use the *Add Method* wizard.
	a. In the editor, place the cursor anywhere in the *Dept* class.
	b. **Right-click** and then select **Source** > **Add Method**. The Add Method wizard opens.
	c. Enter *DischargeEmployee_Handler* as the event name.
	d. Select **Last method** for the insertion position.
	 
	
	e. Click **Generate**.
	The generated code (without comments) should appear as follows:
	`method public void DischargeTeamMember_Handler ():`
	`  return.`
	`end method.`

© 2017 Progress Software Corporation. All rights reserved.

3.	To the public *DischargeEmployee_Handler*() method add the following input parameters.
	• *pFirstname* as *character*
	• *pLastname* as *character*
	
	`method public void DischargeTeamMember_Handler (input pFirstName as character,`
	`input pLastName as character):`
	`  return.`
	`end method.`

4.	Modify the public *DischargeEmployee_Handler*() method to find the discharged employee in the *ttEmployee* temp-table based on the first and last name. `find ttEmployee where ttEmployee.FirstName = pFirstName and ttEmployee.LastName = pLastName no-error.`

5.	If an employee is found, you should add code in a do block to:
	• Add a statement to open a file for output. The name of the file you will be writing to is **Discharges.out**. It will be located in the **/progress_education/openedge/IntroOOP/log** directory.
	Hint: Use the full pathname of this file.
	• Add a *message* statement to write the number of employees discharged.
	• Delete the employee record by casting the *EmpRef* field and return the *Emp* instance.
	Hint: You will need to cast the temp-table field *ttEmployee.EmpRef* to the type *Emp*.
	
	The code you add should appear as follows:
	
	`if available(ttEmployee)`
	`  then do:`
	`  output to "/progress_education/openedge/introOOP/log/Discharges.out" append.`
	`  message "Employee: " + pFirstName + " " + pLastName +`
	`  " ("`
	`cast(ttEmployee.EmpRef,Emp):EmpNum  ")"  +`
	`  "has terminated effective: "`
	`today.`
	`  delete object cast(ttEmployee.EmpRef,Emp).`
	
	`end.`

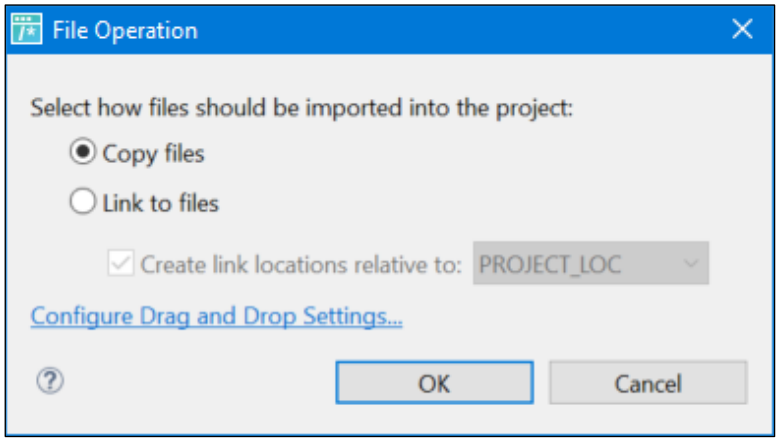
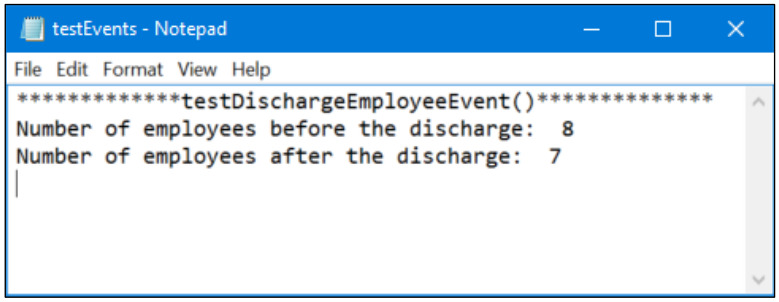
6.	In the *do* block, add a statement to delete the *ttEmployee* temp-table record that was found. `delete ttEmployee.`
7.	In the *do* block, add a statement to decrease the employee count by 1.

© 2017 Progress Software Corporation. All rights reserved.

	`NumEmployees = NumEmployees - 1.`
8.	In the *do* block, add a statement to close the output file. `output close.`
9.	Save this file. Ensure that it compiles without errors.

© 2017 Progress Software Corporation. All rights reserved.

Solution, Part 3—Testing the Dept class event

Step	Solution instructions
1.	In the **Enterprise/HR** folder of the Test project, import the procedure file named **testEvents.p.** a. In Windows Explorer, navigate to the **/progress_education/OpenEdge/introOOP/Exercise/Lesson03** directory and select the **testEvents.p** file. b. Drag and drop the **testEvents.p** file into the **Enterprise/HR** folder. c. Ensure that **Copy files** is selected.  d. Click **OK**.
2.	Run **testEvents.p**. Does it run correctly and was the discharged employee information added to the output file? View the output file. a. With **testEvents.p** open in the editor, click the **Run** icon. b. Select **Progress OpenEdge Application**. c. Click **OK**.
3.	If the file did not run as expected, use the debugger to fix the problem, save your changes, and retest until it runs correctly. 

© 2017 Progress Software Corporation. All rights reserved.

Try It 3.4: Using events, Wrap-up

Exercise summary

In this Try It, you defined and published an event in the *Manager* class and then subscribed to it in the *Dept* class. You also wrote an event handler to handle the event. Finally, you tested your event-handling code to ensure it executes correctly.

© 2017 Progress Software Corporation. All rights reserved.