Progress

Exercise Guide: Introduction to OpenEdge Object-Oriented Programming

OpenEdge

Table of Contents

Copyright

April 2021

Product version: Progress OpenEdge 12.2

1 Exercises

For details, see the following topics:

Guided Exercise 1.1: Set up your application development environment

Overview

In this guided exercise, you set up your development environment for this course using **Progress Developer Studio for OpenEdge (Developer Studio)**.

This exercise has several parts. The exercise steps take approximately 15 minutes to complete. You practice this task in your **Developer Studio**.

Important: You must complete this guided exercise to perform subsequent try it exercises in this course.

Before you begin

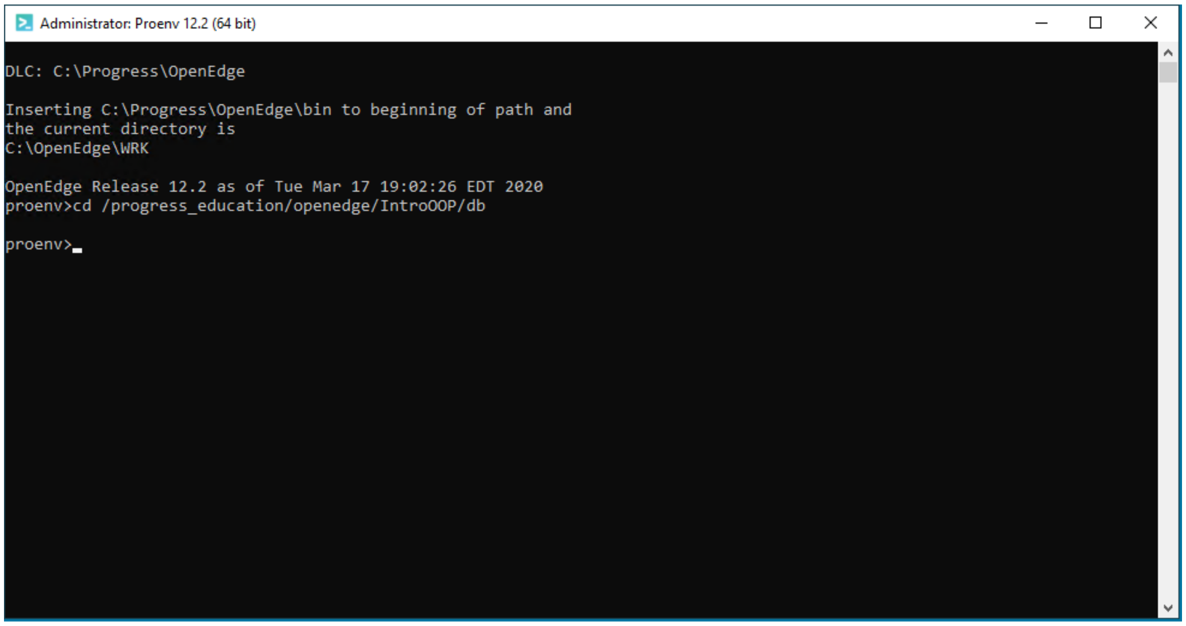
Before you begin this Guided Exercise, you must set up your exercise environment. If you have not done so already, then see the "Before you begin" section at the beginning of the course for details.

Part 1: Start the database server

This course uses a database named `sports2020`. In this part of the guided exercise, you create the database files and start the database server.

1. Open a Proenv window by selecting **Start** > **Progress** > **Proenv 12.2 (64 bit)**. Navigate to `/progress_education/OpenEdge/IntroOOP/db`, as follows:

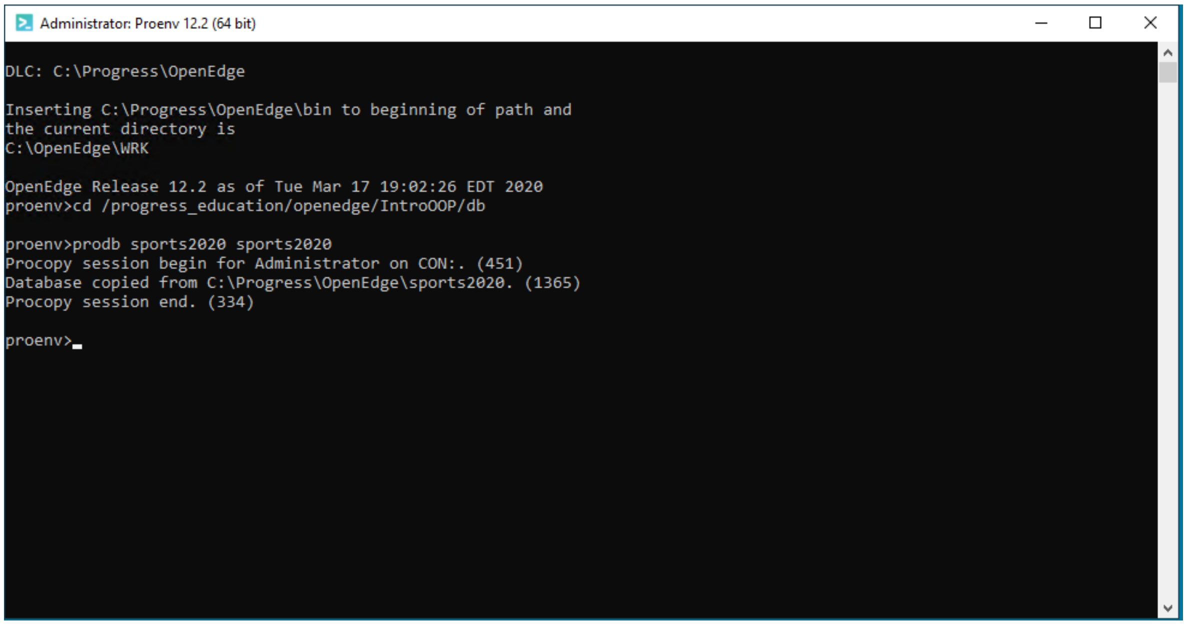
   ```
   cd /progress_education/openedge/IntroOOP/db
   ```


Administrator: Proenv 12.2 (64 bit)

```
DLC: C:\Progress\OpenEdge

Inserting C:\Progress\OpenEdge\bin to beginning of path and
the current directory is
C:\OpenEdge\WRK

OpenEdge Release 12.2 as of Tue Mar 17 19:02:26 EDT 2020
proenv>cd /progress_education/openedge/IntroOOP/db

proenv>
```

2. Create a copy of the `sports2020` database in the course db directory, as follows:

```
prodb sports2020 sports2020
```


Administrator: Proenv 12.2 (64 bit)

```
DLC: C:\Progress\OpenEdge

Inserting C:\Progress\OpenEdge\bin to beginning of path and
the current directory is
C:\OpenEdge\WRK

OpenEdge Release 12.2 as of Tue Mar 17 19:02:26 EDT 2020
proenv>cd /progress_education/openedge/IntroOOP/db

proenv>prodb sports2020 sports2020
Procopy session begin for Administrator on CON:. (451)
Database copied from C:\Progress\OpenEdge\sports2020. (1365)
Procopy session end. (334)

proenv>
```

3. Start the database server for the `sports2020` database, as follows:

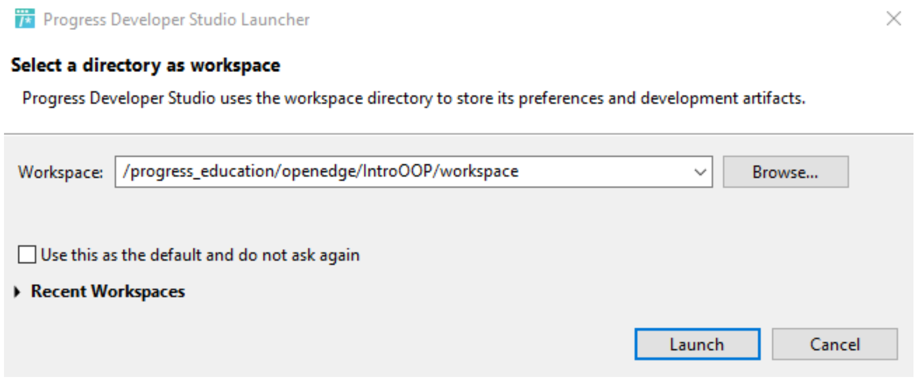
```
proserve sports2020 -H localhost -S 50400
```

4. Close the **Proenv** window.

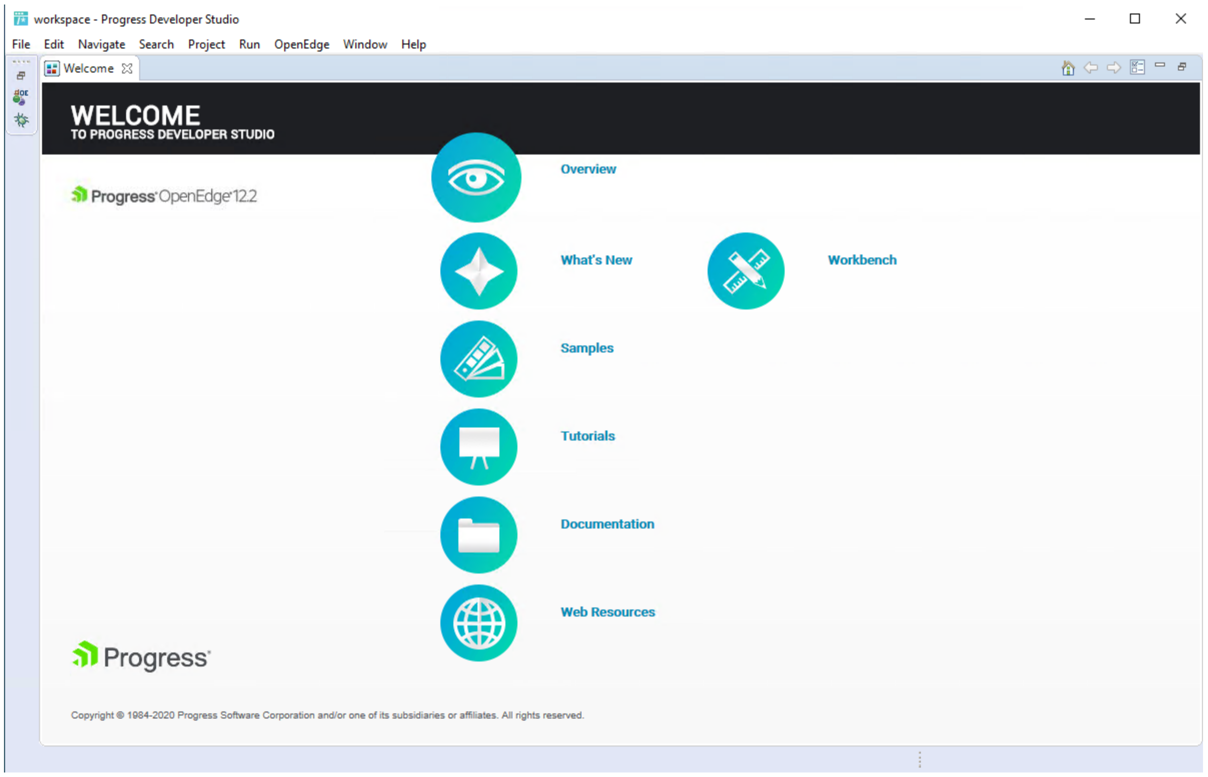
Part 2: Set up a workspace in Developer Studio

Create a new Developer Studio workspace that will be used for your ABL development. If this is the first time that you are using Developer Studio, then follow these instructions:

1. Start Developer Studio by selecting **Start** > **Progress** > **Developer Studio 12.2 (64 bit)**.
2. Specify `/progress_education/openedge/IntroOOP/workspace` as the workspace location.

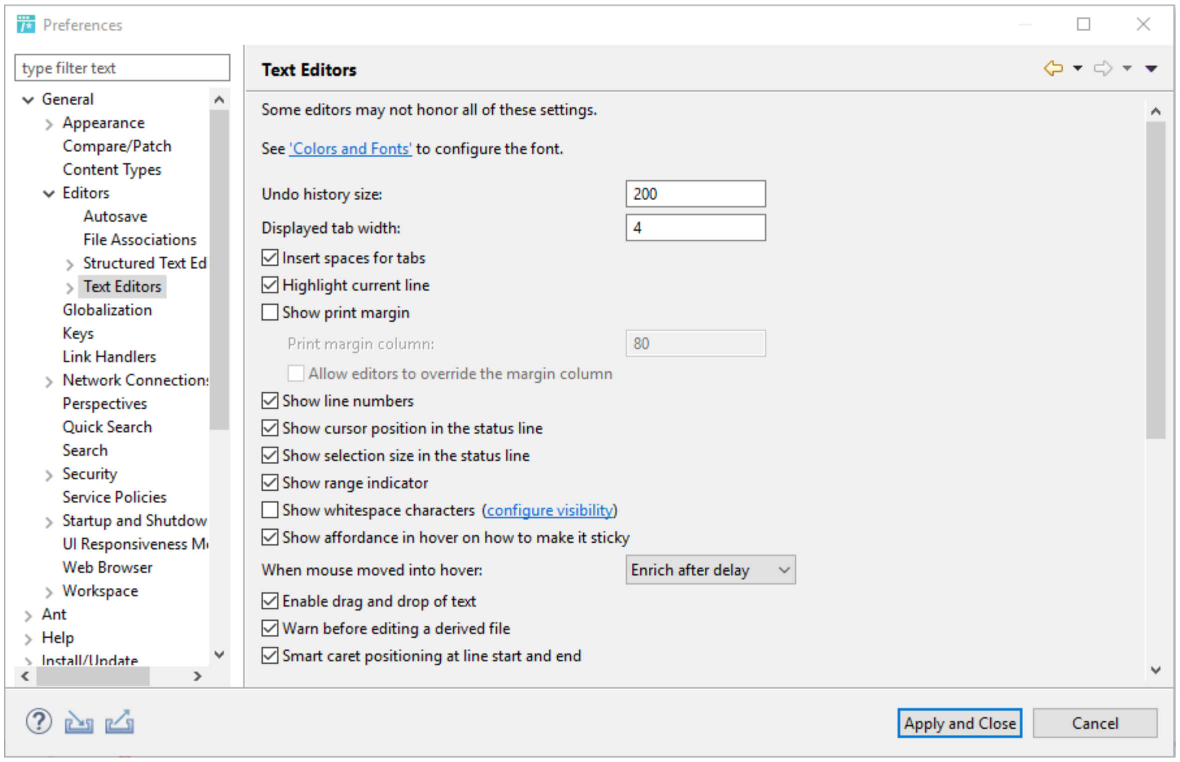

Progress Developer Studio Launcher
Select a directory as workspace
Progress Developer Studio uses the workspace directory to store its preferences and development artifacts.
Workspace: /progress_education/openedge/IntroOOP/workspace
Browse...
Use this as the default and do not ask again
Recent Workspaces
Launch
Cancel

3. Click **Launch**.
4. You may see the **Welcome screen**, as shown here. Click the **Workbench** icon to continue to the workspace for Developer Studio.


workspace - Progress Developer Studio
File Edit Navigate Search Project Run OpenEdge Window Help
Welcome
WELCOME
TO PROGRESS DEVELOPER STUDIO
Progress OpenEdge 12.2
Overview
What's New
Workbench
Samples
Tutorials
Documentation
Web Resources
Progress
Copyright © 1984-2020 Progress Software Corporation and/or one of its subsidiaries or affiliates. All rights reserved.

5. If you have previously used Developer Studio for OpenEdge, you must switch to the following workspace:
   a. Select **File** > **Switch Workspace** > **Other**.
   b. Specify `/progress_education/openedge/IntroOOP/workspace` as the workspace location.
6. 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**.



Text Editors

Some editors may not honor all of these settings.

See 'Colors and Fonts' to configure the font.

Undo history size: 200

Displayed tab width: 4

- [x] Insert spaces for tabs
- [x] Highlight current line
- [ ] Show print margin

Print margin column: 80

- [ ] Allow editors to override the margin column
- [x] Show line numbers
- [x] Show cursor position in the status line
- [x] Show selection size in the status line
- [x] Show range indicator
- [ ] Show whitespace characters (configure visibility)
- [x] Show affordance in hover on how to make it sticky

When mouse moved into hover: Enrich after delay

- [x] Enable drag and drop of text
- [x] Warn before editing a derived file
- [x] Smart caret positioning at line start and end

e. Navigate to **Progress OpenEdge** > **Editor**.

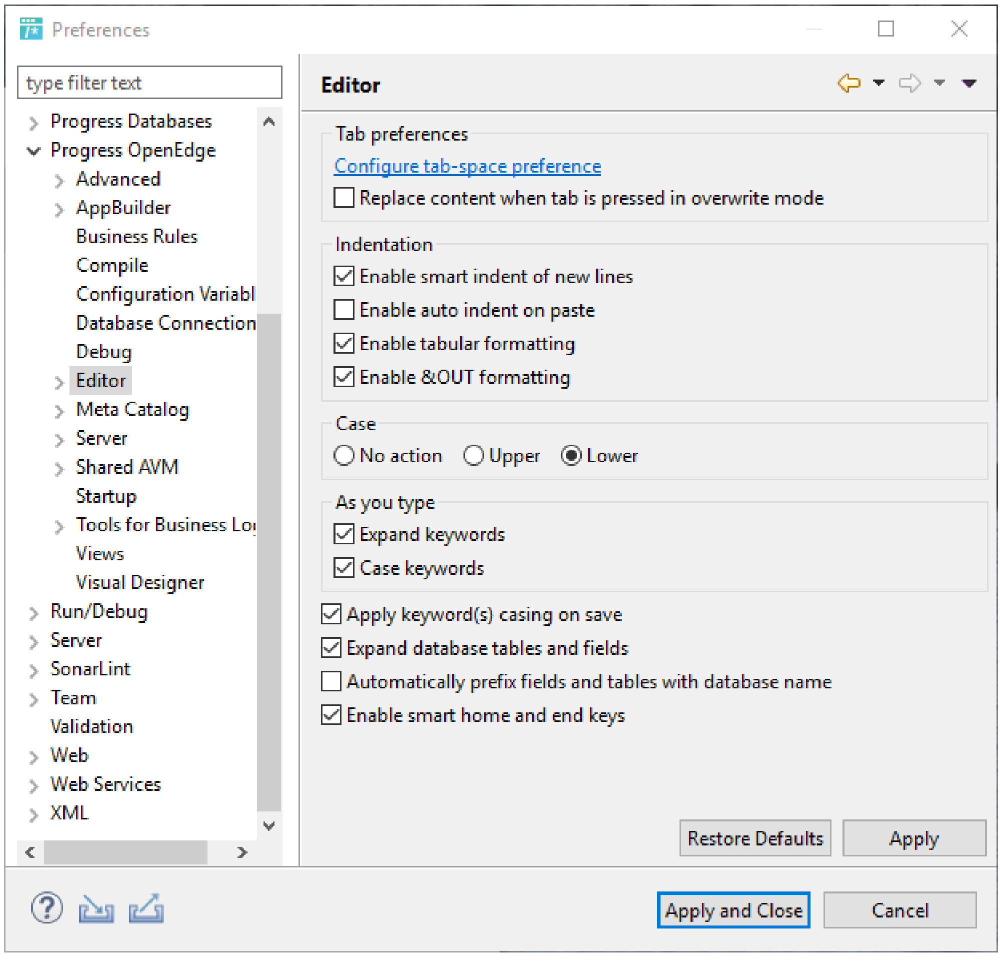
f. Select **Lower** in the **Case area**.

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**.



Preferences

type filter text

- Progress Databases
- Progress OpenEdge
  - Advanced
  - AppBuilder
  - Business Rules
  - Compile
  - Configuration Variabl
  - Database Connection
  - Debug
  - Editor
  - Meta Catalog
  - Server
  - Shared AVM
  - Startup
  - Tools for Business Lo
  - Views
  - Visual Designer
- Run/Debug
- Server
- SonarLint
- Team
- Validation
- Web
- Web Services
- XML

Editor

Tab preferences

Configure tab-space preference

☐ Replace content when tab is pressed in overwrite mode

Indentation

☑ Enable smart indent of new lines

☐ Enable auto indent on paste

☑ Enable tabular formatting

☑ Enable &OUT formatting

Case

○ No action ○ Upper ● Lower

As you type

☑ Expand keywords

☑ Case keywords

☑ Apply keyword(s) casing on save

☑ Expand database tables and fields

☐ Automatically prefix fields and tables with database name

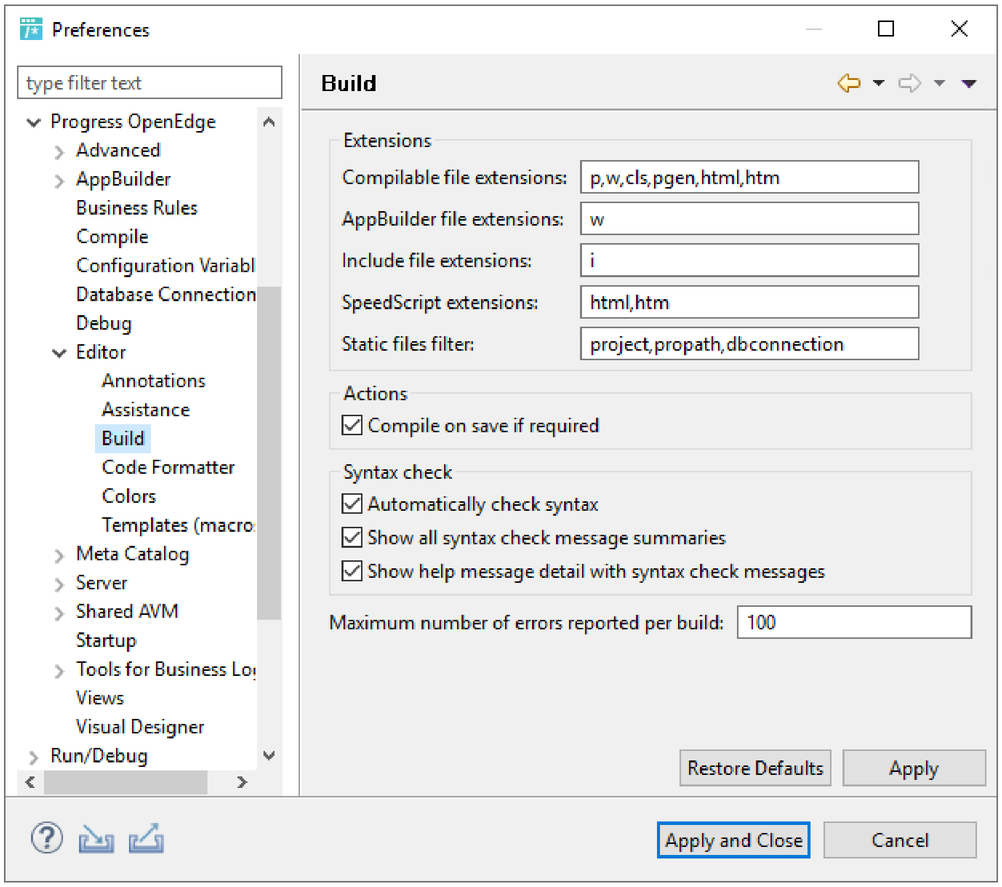
☑ Enable smart home and end keys

Restore Defaults Apply

Apply and Close Cancel

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

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



Build

Extensions

Compilable file extensions: p,w,cls,pgen,html,htm

AppBuilder file extensions: w

Include file extensions: i

SpeedScript extensions: html,htm

Static files filter: project,propath,dbconnection

Actions

☑ Compile on save if required

Syntax check

☑ Automatically check syntax

☑ Show all syntax check message summaries

☑ Show help message detail with syntax check messages

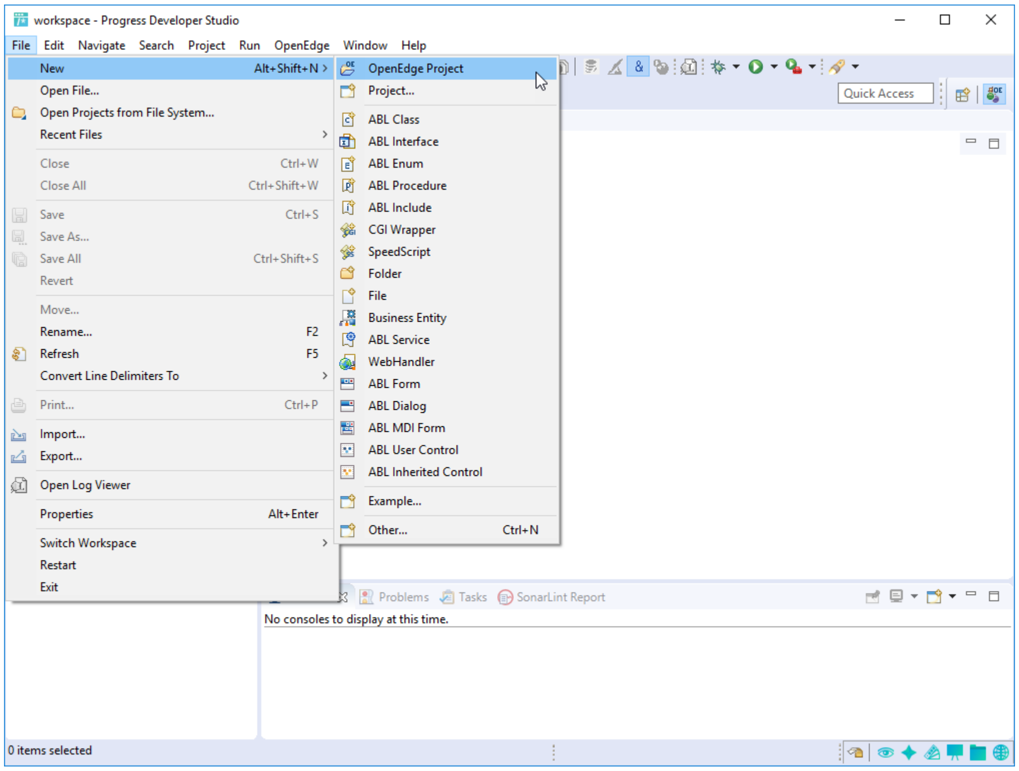
Maximum number of errors reported per build: 100

m. Click **Apply and Close**.

Part 3: Set 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 sports2020 database.

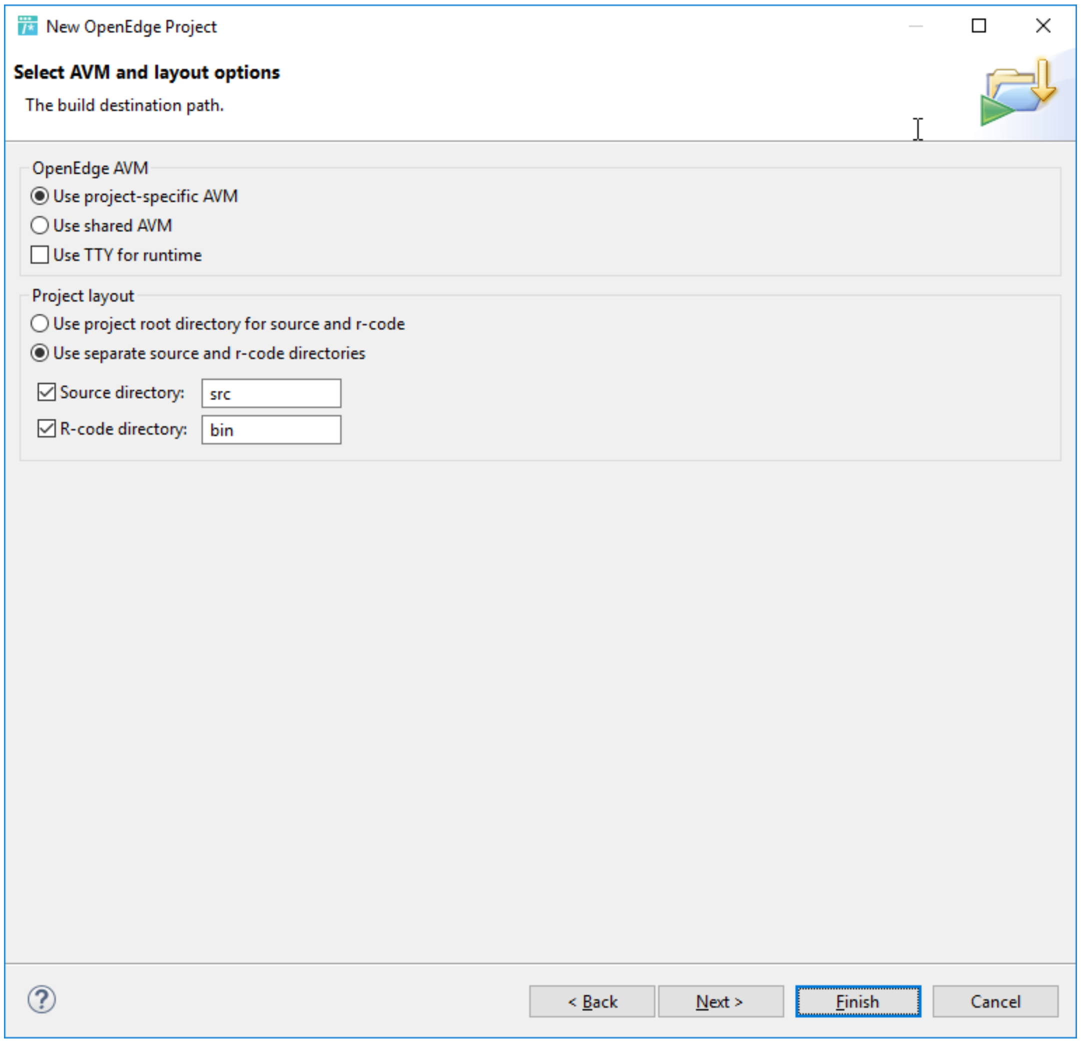
1. Select **File** > **New** > **OpenEdge Project**.



No consoles to display at this time.

0 items selected

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`.



Select AVM and layout options

The build destination path.

OpenEdge AVM

(●) Use project-specific AVM

( ) Use shared AVM

☐ Use TTY for runtime

Project layout

( ) Use project root directory for source and r-code

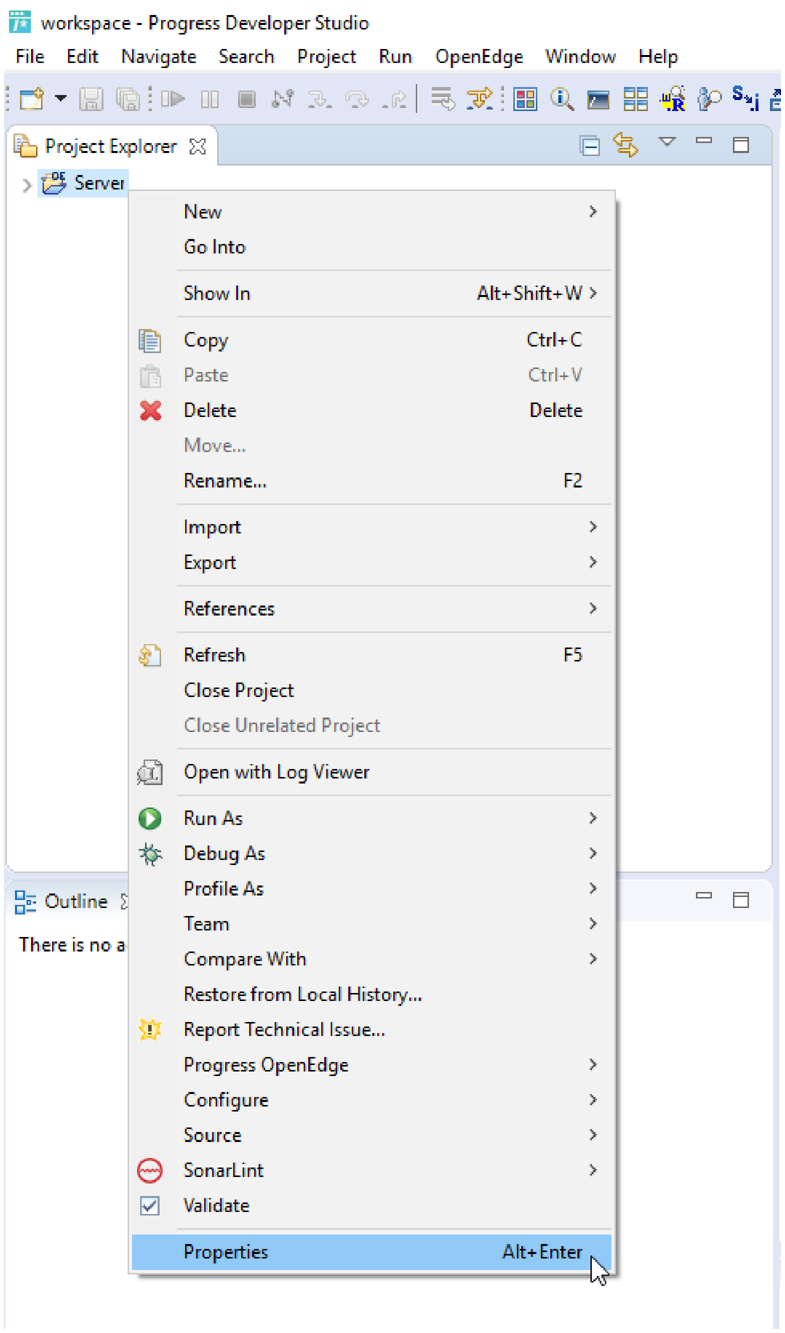
(●) Use separate source and r-code directories

☑ Source directory: src

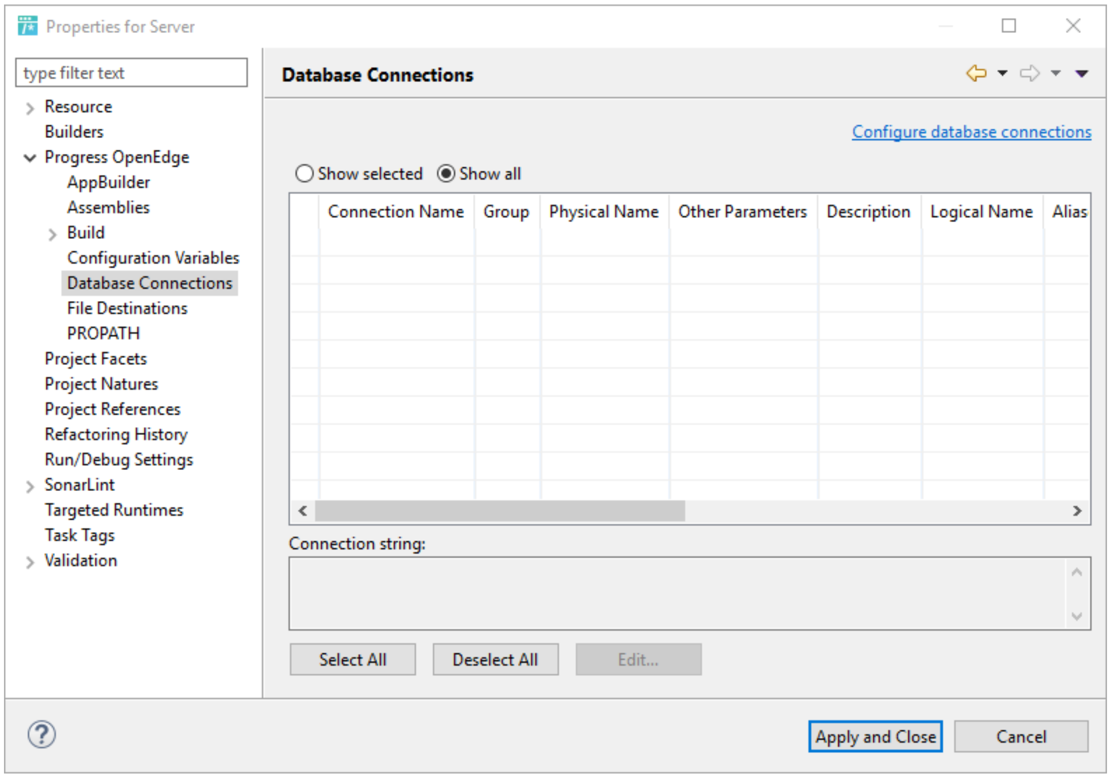
☑ R-code directory: bin

e. Click **Finish**.

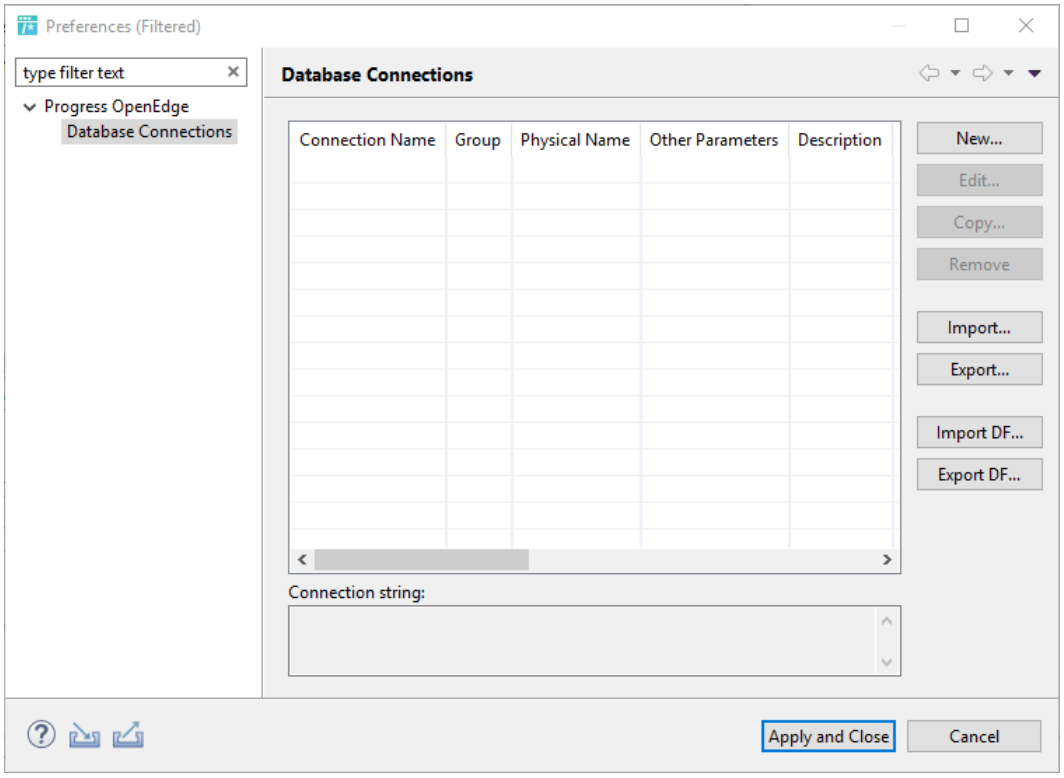
3. Add the database connection to the workspace and associate it with the **Server** project. Right-click the **Server** project, and then select **Properties**.



4. Navigate to **Progress OpenEdge** > **Database Connections**.



5. Click the **Configure database connections** link.

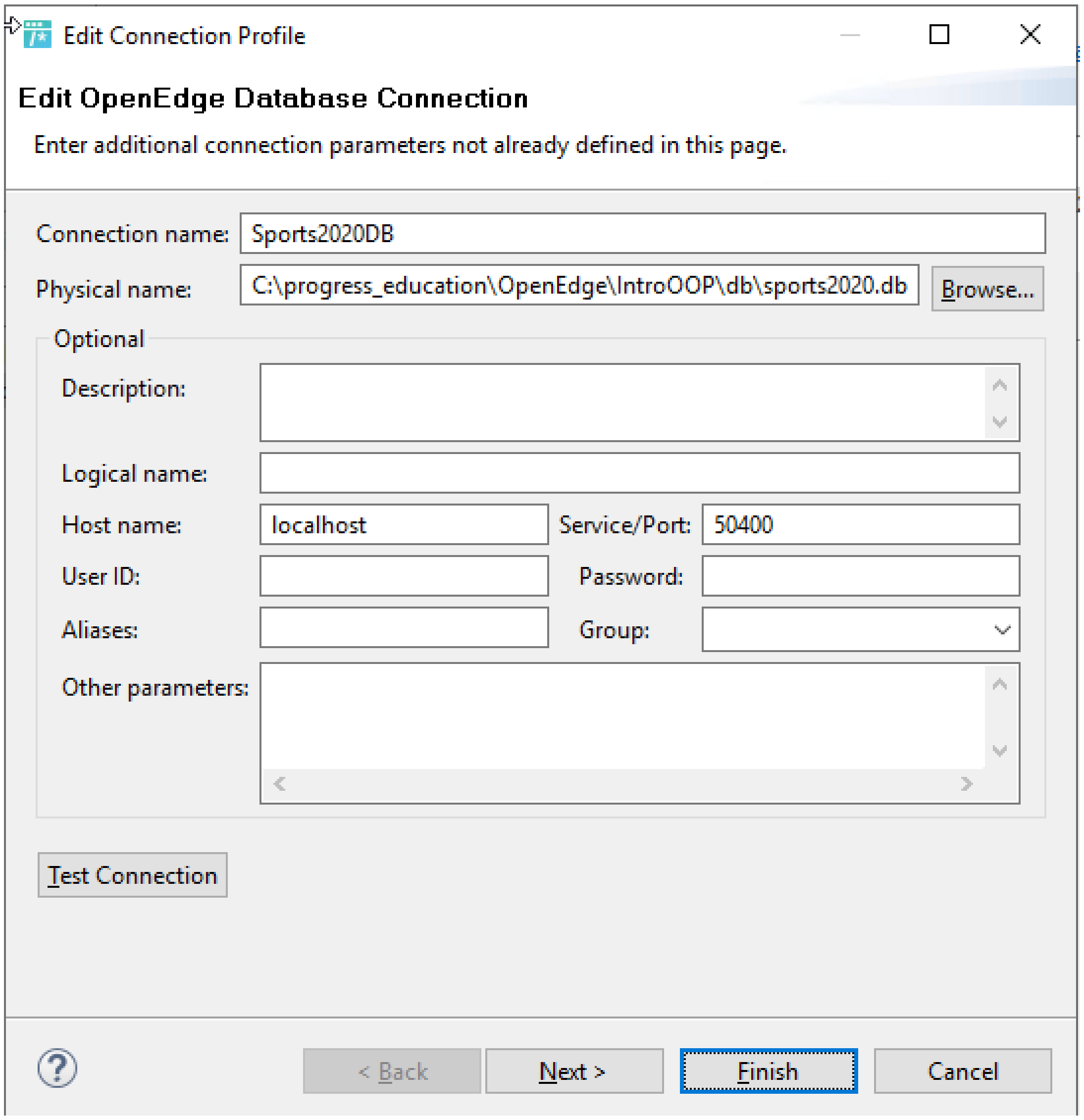


6. Click **New** to open the **Add OpenEdge Database Connection** wizard.

7. Enter the connection name as `Sports2020DB`.

8. Click the **Browse** button, and then navigate to and select `C:\progress_education\openedge\IntroOOP\db\sports2020.db`.
9. Specify the **host name** as `localhost`.
10. Specify `50400` as the **port number**.

    If port `50400` is unavailable on your system, then use a different port number. It must be the same port number that you specified when you started the Database Server.


Edit Connection Profile

Edit OpenEdge Database Connection

Enter additional connection parameters not already defined in this page.

Connection name: Sports2020DB

Physical name: C:\progress_education\OpenEdge\IntroOOP\db\sports2020.db

Optional

Description:

Logical name:

Host name: localhost

Service/Port: 50400

User ID:

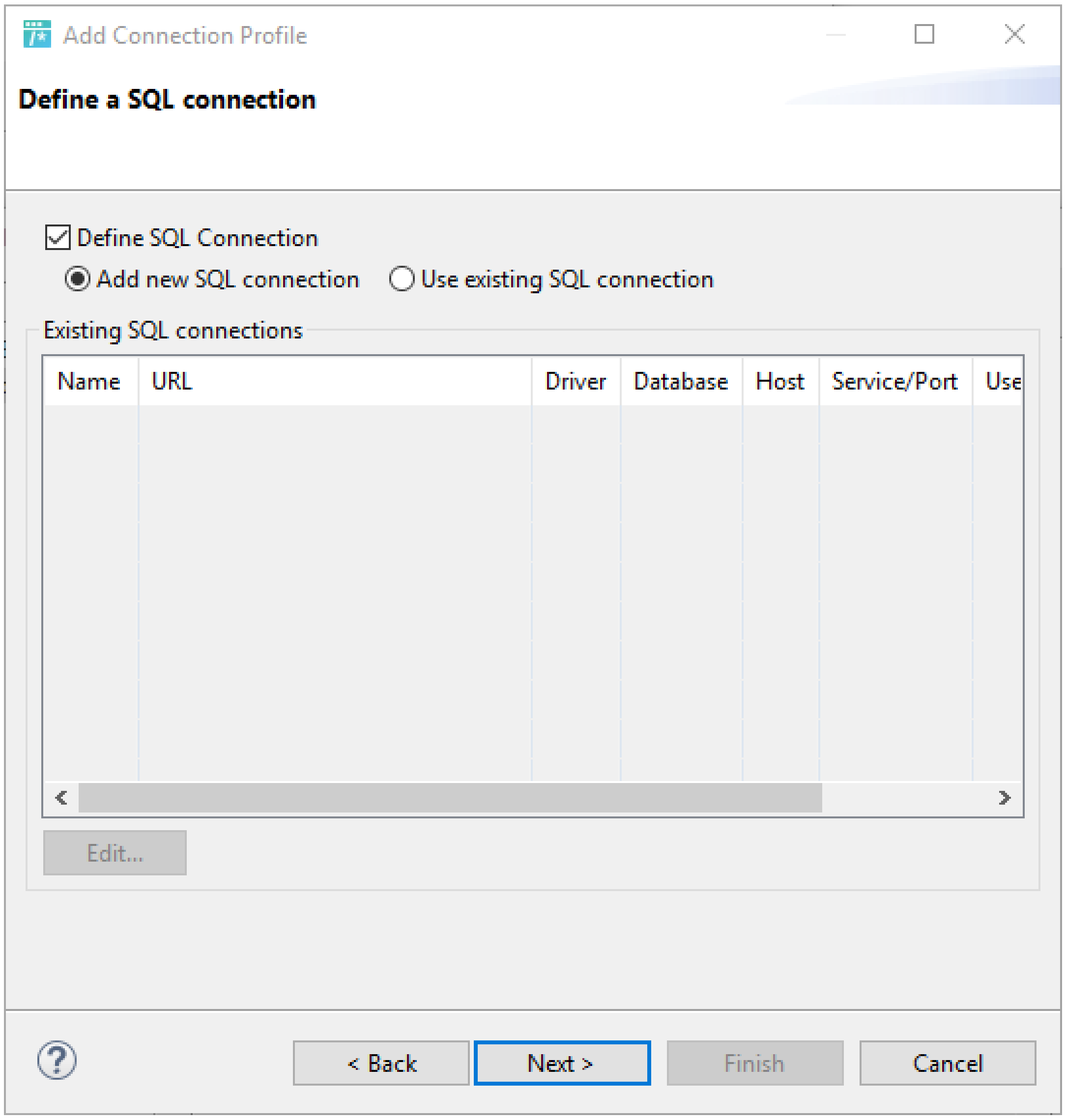
Password:

Aliases:

Group:

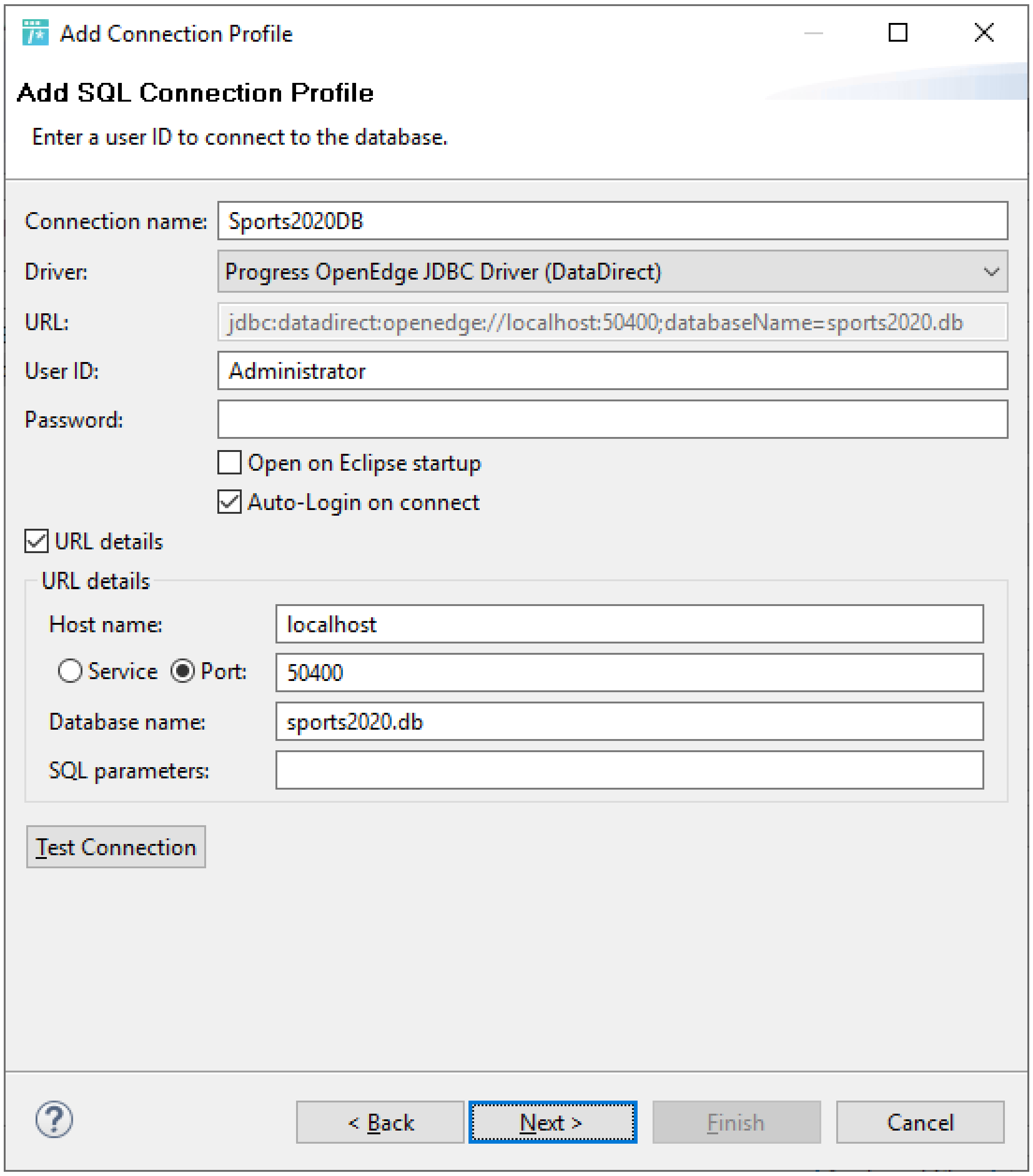
Other parameters:

11. Click **Next** in the **Add OpenEdge Database Connection** dialog box.
12. Click **Next** in the **Define a SQL connection** dialog box to keep the defaults.


Add Connection Profile
Define a SQL connection
[✓] Define SQL Connection
(●) Add new SQL connection
( ) Use existing SQL connection
Existing SQL connections

Name	URL	Driver	Database	Host	Service/Port	Use

13. Click **Next** in **Add SQL Connection Profile**.


Add Connection Profile

Add SQL Connection Profile

Enter a user ID to connect to the database.

Connection name:	Sports2020DB
Driver:	Progress OpenEdge JDBC Driver (DataDirect)
URL:	jdbc:datadirect:openedge://localhost:50400;databaseName=sports2020.db
User ID:	Administrator
Password:	

☐ Open on Eclipse startup

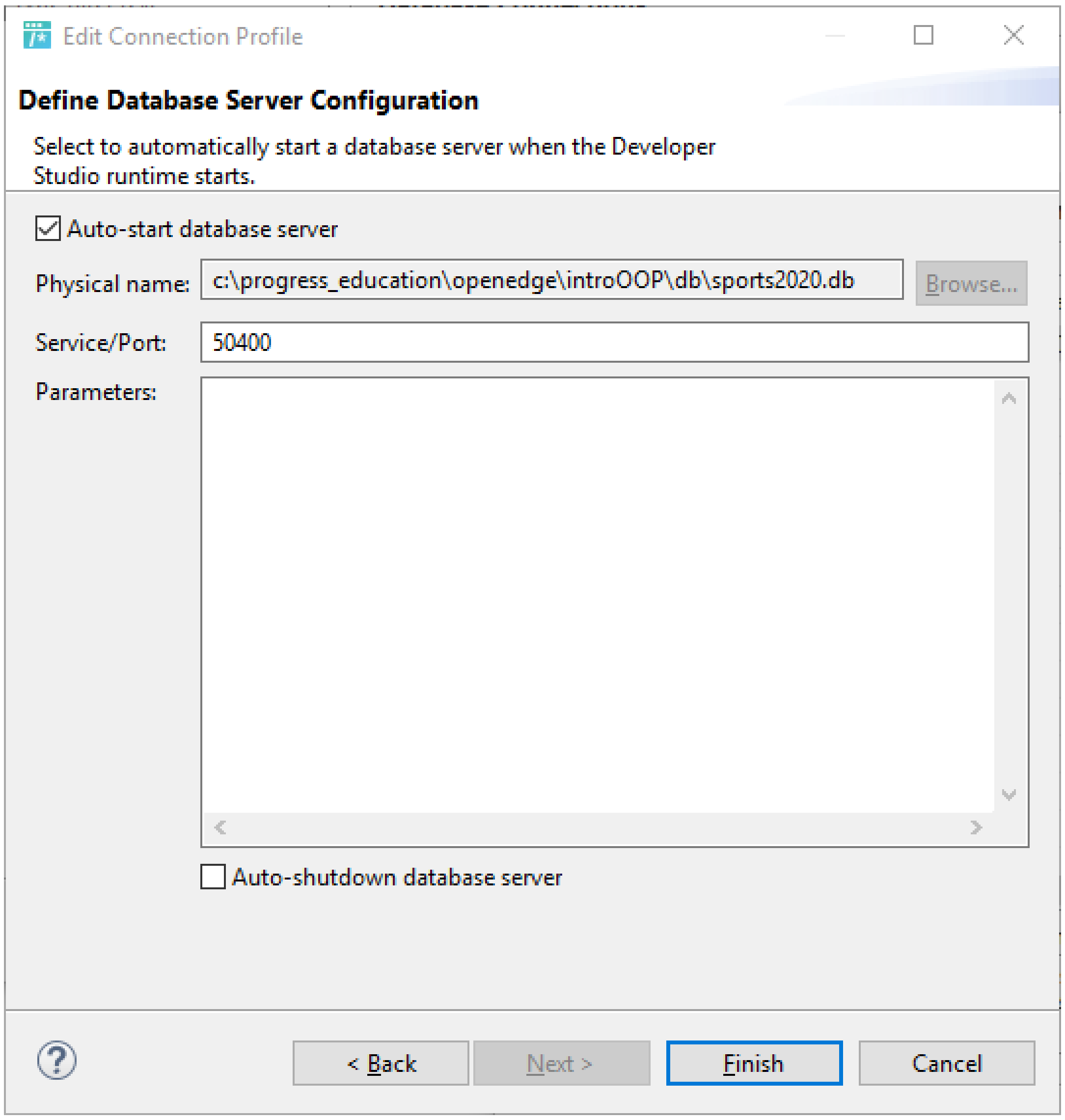
☑ Auto-Login on connect

☑ URL details

URL details

Host name:	localhost
○ Service ● Port:	50400
Database name:	sports2020.db
SQL parameters:	

14. Click **Next**.
15. In the **Defined Database Server Configuration** dialog box, select **Auto-start database server**.



Define Database Server Configuration

Select to automatically start a database server when the Developer Studio runtime starts.

☑ Auto-start database server

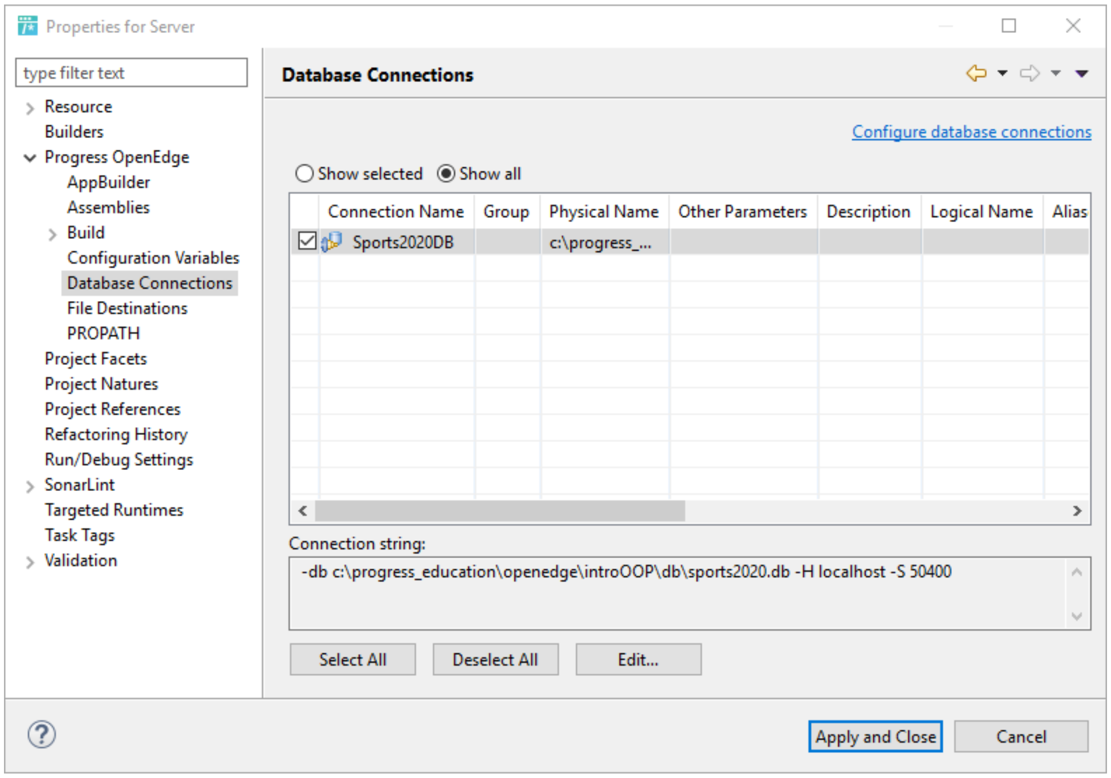
Physical name: c:\progress_education\openedge\introOOP\db\sports2020.db

Service/Port: 50400

Parameters:

☐ Auto-shutdown database server

16. Click **Finish** in the **Define Database Server Configuration** page.
17. Click **Apply and Close**.
18. Check the **Sports2020DB** checkbox in the **Database Connections** page.



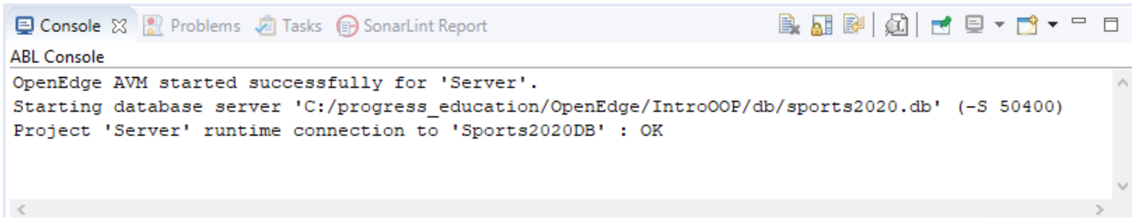
	Connection Name	Group	Physical Name	Other Parameters	Description	Logical Name	Alias
☑	Sports2020DB		c:\progress_...				

Connection string:

```
-db c:\progress_education\openedge\introOOP\db\sports2020.db -H localhost -S 50400
```

19. Click **Apply and Close**.

20. In the **Console** view, the following message appears:



```
OpenEdge AVM started successfully for 'Server'.
Starting database server 'C:/progress_education/OpenEdge/IntroOOP/db/sports2020.db' (-S 50400)
Project 'Server' runtime connection to 'Sports2020DB' : OK
```

Wrap-up

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.

Try It 2.1: Define 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:

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/Exercises/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. Add the database connection to the imported projects.
4. Ensure that **Build Automatically** is set for the workspace.

Part 1: Create 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.

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.

   Tip: Use the **New ABL class** wizard.

Part 2: Define data members for the Emp class

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

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*

   Tip: 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 have implementations for *both* of its accessors.
5. Save your file. Ensure that it compiles without errors.

Part 3: Define methods for the Emp class

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

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.

Part 4: Add 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.

1. In **Project Explorer**, in the **Server** project, create a new folder named *Include* under the **src** folder.
2. In the **Include** folder, import the `ttEmployee.i` file from the `C:/progress_education/OpenEdge/introOOP/Exercises/Lesson02` directory.

   Note: If you do not specify an access mode, the default access mode for a temp-table is `private`.

Part 5: Create 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.

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.

Tip: Use the **New ABL class** wizard.

Part 6: Define 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.

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.

Tip: Add this statement in the definitions part of the class:

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

Part 7: Define a constructor and methods for the Dept class

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

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.

Tip: 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.

Wrap-up

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

Try It 3.1: Work 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/Exercises/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. Add the database connection to the imported projects.
4. Ensure that Build Automatically is set for the workspace.

Part 1: Implement 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.

1. Implement the code for the *Initialize()* method for the *Emp* class. Assign the value of the input parameter to each of the data members.
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.

Part 2: Implement 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.

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.

Tip: 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.

Tip: 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.

Wrap-up

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.

Try It 3.2: Test 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:

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

Location of files:

Exercise files: `/progress_education/openedge/IntroOOP/Exercises/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. Add the database connection to the imported projects.
4. Ensure that *Build Automatically* is set for the workspace.

Part 1: Set 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.

1. Create a new OpenEdge Project named **Test** specifying *Sports2020DB* 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*.

Part 2: Write 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.

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.

   Tip: Use the full pathname of this file.

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.
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.

Part 3: Test 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.

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.

Part 4: Write 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.

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 variables in 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*. Add the definition of the Phones variable to the end of the definitions section.
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.

   Tip: 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.
11. 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. In the FOR EACH block, initialize the *Emp* instance by calling the *Initialize()* method, passing values from the current Employee buffer.
14. In the FOR EACH block, add the created *Emp* instance to the *DeptInstance*.

Tip: 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.

Tip: 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. 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.

Tip: 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. 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.

Tip: 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.

Tip: You will need to use the cast function to call the EmpRef 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.

Part 5: Test the Dept class

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

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.

Wrap-up

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.

Try It 4.1: Use 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:

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

Location of files:

Exercise files: **/progress_education/openedge/IntroOOP/Exercises/Lesson04**

Solution files: **/progress_education/openedge/IntroOOP/Solutions/Lesson04**

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. Add the database connection to the imported projects.
4. Do a clean build of the workspace.

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.

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.

Part 2: Create 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.

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.

   Tip: 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.

Part 3: Define a constructor and a data member for the Manager class

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

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.

   Tip: 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*).

Part 4: Define 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.

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 *GetManagersEmployees()* 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.

   Tip: 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)

Part 5: Implement 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.

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*.
   - Replace the return statement (return super:GetInfo().) with return result.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 *GetManagersEmployees()* 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.

Part 6: Import 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.

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.

Part 7: Test 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.

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.

Wrap-up

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.

Try it 5.1: Use 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:

1. Complete the Exercise Setup instructions, if you have not done so already.
2. Complete the Try It 4.1.

Location of files:

Exercise files: **/progress_education/openedge/IntroOOP/Exercises/Lesson05**

Solution files: **/progress_education/openedge/IntroOOP/Solutions/Lesson05**

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. Add the database connection to the imported projects.
4. Do a clean build of your workspace.

Part 1: Create the IBusiness Unit interface class

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

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.

Tip: 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.

Part 2: Define data members for the IBusiness Unit interface class

Define the data members for the *IBusiness Unit* interface class.

1. Define these properties as *public* with a `get` but no `set` method:
   - *Name* as *character*
   - *NumDepartments* as *Integer*
2. Save your file. Ensure that it compiles without errors.

Part 3: Define methods for the IBusiness Unit interface class

Define the methods for the *IBusiness Unit* interface class.

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.

Part 4: Create the Company class

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

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.

   Tip: 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.

Part 5: Implement a constructor, a destructor, and methods for the Company class

Modify the constructor and the destructor, and define methods for the *Company* class.

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*.

   Note: Define a `private` set method for the properties .

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.

   Tip: 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.

   Tip: 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*.

   Tip: 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.

Part 6: Import the Franchise class

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

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

Part 7: Test the classes

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

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.

Wrap-up

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.

Try It 5.2: Use 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:

1. Complete the Exercise Setup instructions, if you have not done so already.
2. Complete the Try It 5.1.

Location of files:

Exercise files: `/progress_education/openedge/IntroOOP/Exercises/Lesson05`

Solution files: `/progress_education/openedge/IntroOOP/Solutions/Lesson05`

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. Add the database connection to the imported projects.
4. Do a clean build of the workspace.

Part 1: Define and publish 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.

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.

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.

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.

   Tip: 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.

   Tip: 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.

   Tip: 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.

Part 3: Test 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.

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.

Wrap-up

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.

2

Solutions

For details, see the following topics:

Guided Exercise 1.1: Set up your application development environment

Overview

In this guided exercise, you set up your development environment for this course using **Progress Developer Studio for OpenEdge (Developer Studio)**.

This exercise has several parts. The exercise steps take approximately 15 minutes to complete. You practice this task in your **Developer Studio**.

Important: You must complete this guided exercise to perform subsequent try it exercises in this course.

Before you begin

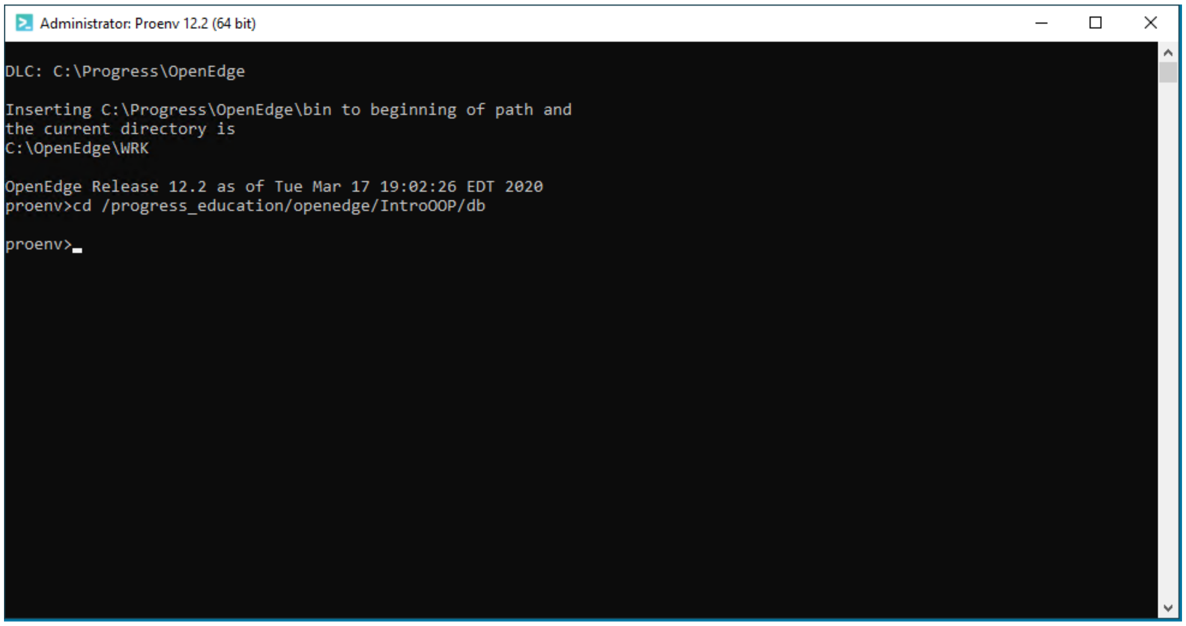
Before you begin this Guided Exercise, you must set up your exercise environment. If you have not done so already, then see the "Before you begin" section at the beginning of the course for details.

Part 1: Start the database server

This course uses a database named `sports2020`. In this part of the guided exercise, you create the database files and start the database server.

1. Open a Proenv window by selecting **Start** > **Progress** > **Proenv 12.2 (64 bit)**. Navigate to `/progress_education/OpenEdge/IntroOOP/db`, as follows:

   ```
   cd /progress_education/openedge/IntroOOP/db
   ```

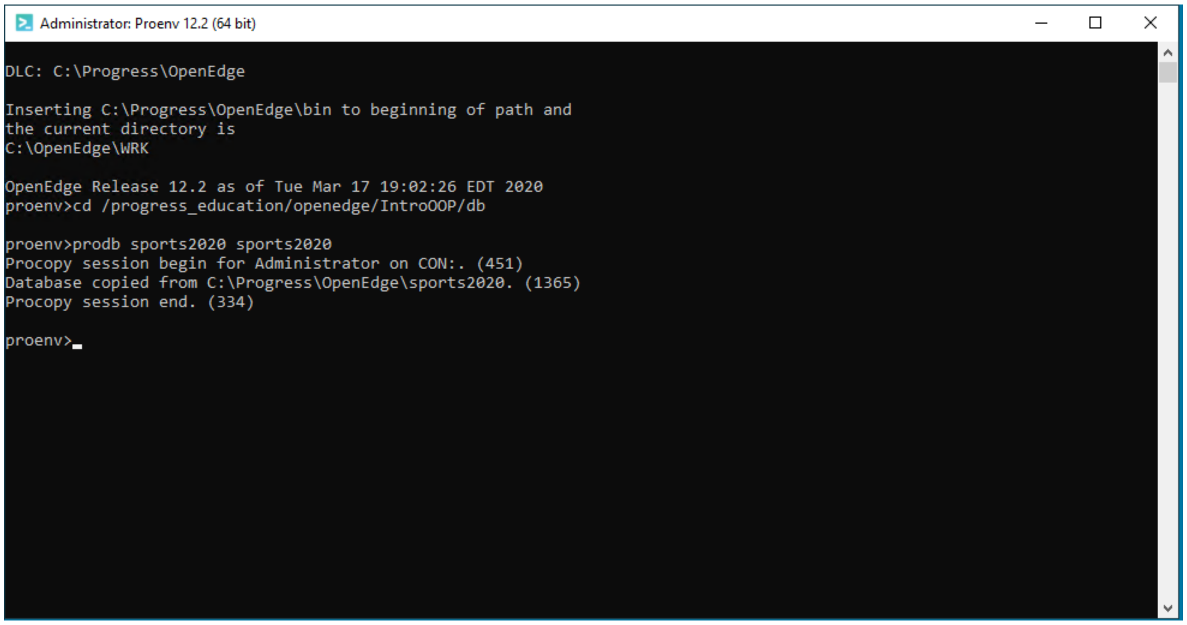


Administrator: Proenv 12.2 (64 bit)

```
DLC: C:\Progress\OpenEdge

Inserting C:\Progress\OpenEdge\bin to beginning of path and
the current directory is
C:\OpenEdge\WRK

OpenEdge Release 12.2 as of Tue Mar 17 19:02:26 EDT 2020
proenv>cd /progress_education/openedge/IntroOOP/db

proenv>_
```

2. Create a copy of the `sports2020` database in the course db directory, as follows:

```
prodb sports2020 sports2020
```



Administrator: Proenv 12.2 (64 bit)

```
DLC: C:\Progress\OpenEdge

Inserting C:\Progress\OpenEdge\bin to beginning of path and
the current directory is
C:\OpenEdge\WRK

OpenEdge Release 12.2 as of Tue Mar 17 19:02:26 EDT 2020
proenv>cd /progress_education/openedge/IntroOOP/db

proenv>prodb sports2020 sports2020
Procopy session begin for Administrator on CON:. (451)
Database copied from C:\Progress\OpenEdge\sports2020. (1365)
Procopy session end. (334)

proenv>_
```

3. Start the database server for the `sports2020` database, as follows:

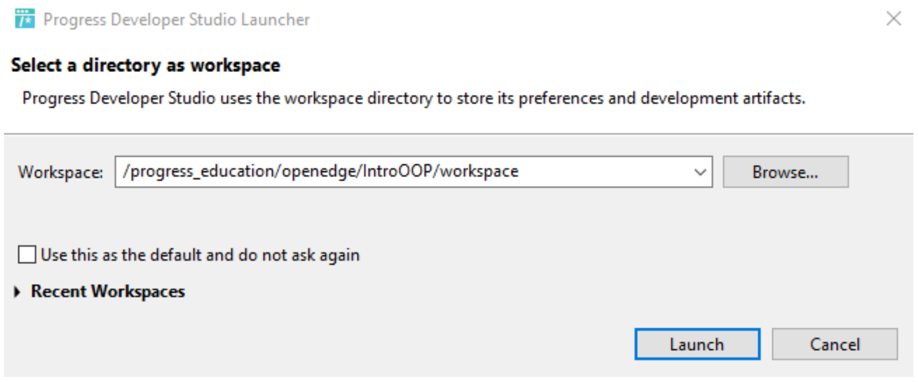
```
proserve sports2020 -H localhost -S 50400
```

4. Close the **Proenv** window.

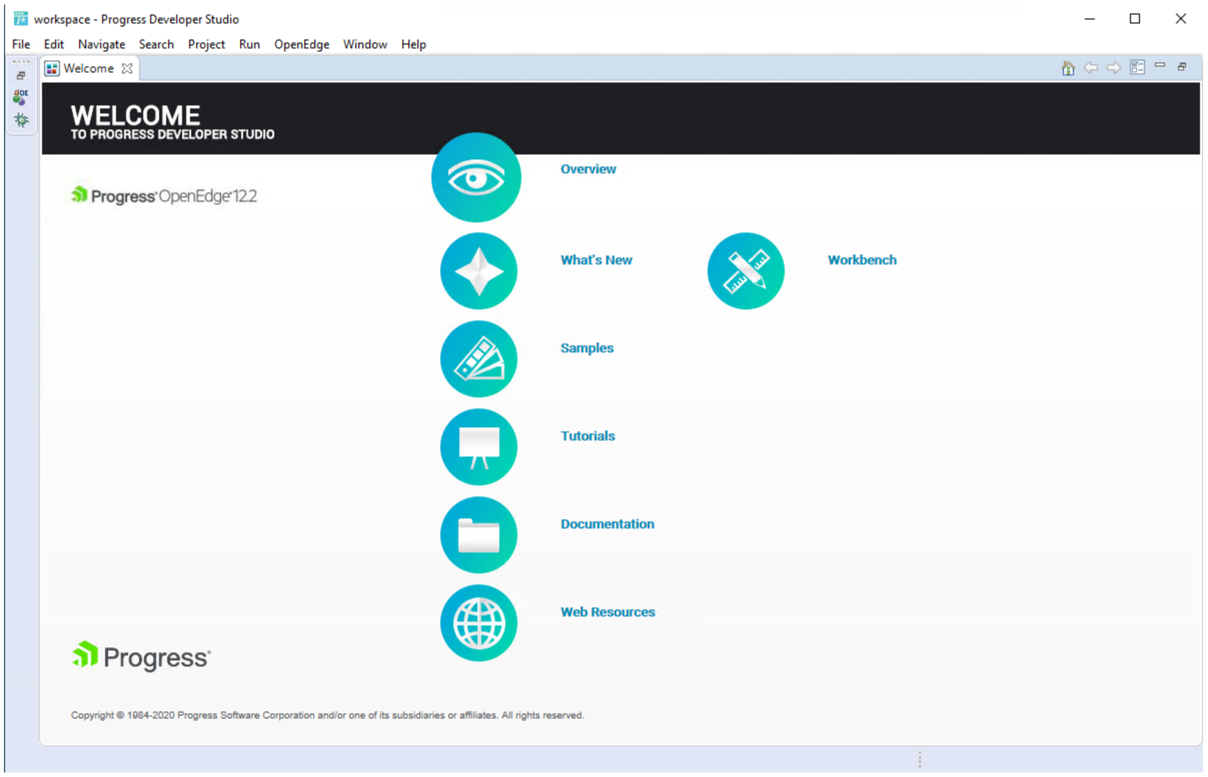
Part 2: Set up a workspace in Developer Studio

Create a new Developer Studio workspace that will be used for your ABL development. If this is the first time that you are using Developer Studio, then follow these instructions:

1. Start Developer Studio by selecting **Start** > **Progress** > **Developer Studio 12.2 (64 bit)**.
2. Specify `/progress_education/openedge/IntroOOP/workspace` as the workspace location.

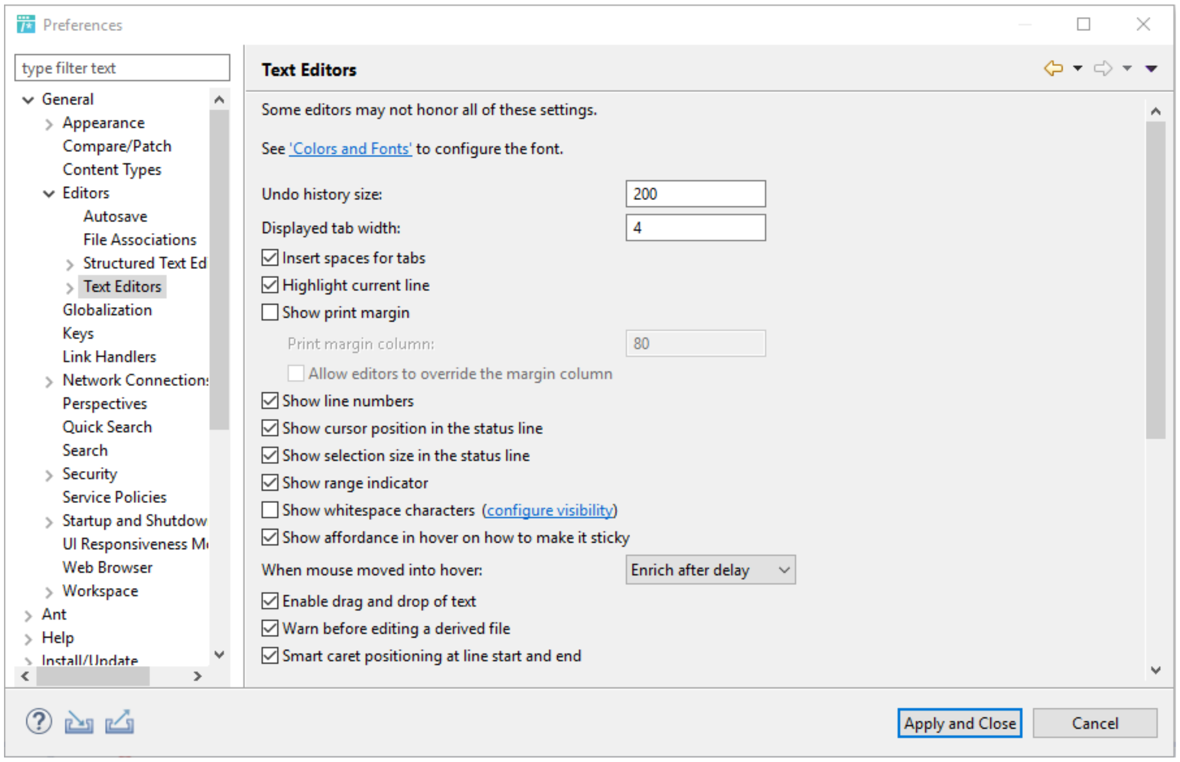

Progress Developer Studio Launcher
Select a directory as workspace
Progress Developer Studio uses the workspace directory to store its preferences and development artifacts.
Workspace: /progress_education/openedge/IntroOOP/workspace
Browse...
Use this as the default and do not ask again
Recent Workspaces
Launch
Cancel

3. Click **Launch**.
4. You may see the **Welcome screen**, as shown here. Click the **Workbench** icon to continue to the workspace for Developer Studio.


workspace - Progress Developer Studio
File Edit Navigate Search Project Run OpenEdge Window Help
Welcome
WELCOME
TO PROGRESS DEVELOPER STUDIO
Progress OpenEdge 12.2
Overview
What's New
Workbench
Samples
Tutorials
Documentation
Web Resources
Progress
Copyright © 1984-2020 Progress Software Corporation and/or one of its subsidiaries or affiliates. All rights reserved.

5. If you have previously used Developer Studio for OpenEdge, you must switch to the following workspace:
   a. Select **File** > **Switch Workspace** > **Other**.
   b. Specify `/progress_education/openedge/IntroOOP/workspace` as the workspace location.
6. 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**.


Preferences

Text Editors

Some editors may not honor all of these settings.

See 'Colors and Fonts' to configure the font.

Undo history size: 200

Displayed tab width: 4

☑ Insert spaces for tabs
☑ Highlight current line
☐ Show print margin
Print margin column: 80
☐ Allow editors to override the margin column
☑ Show line numbers
☑ Show cursor position in the status line
☑ Show selection size in the status line
☑ Show range indicator
☐ Show whitespace characters (configure visibility)
☑ Show affordance in hover on how to make it sticky

When mouse moved into hover: Enrich after delay

☑ Enable drag and drop of text
☑ Warn before editing a derived file
☑ Smart caret positioning at line start and end

e. Navigate to **Progress OpenEdge** > **Editor**.

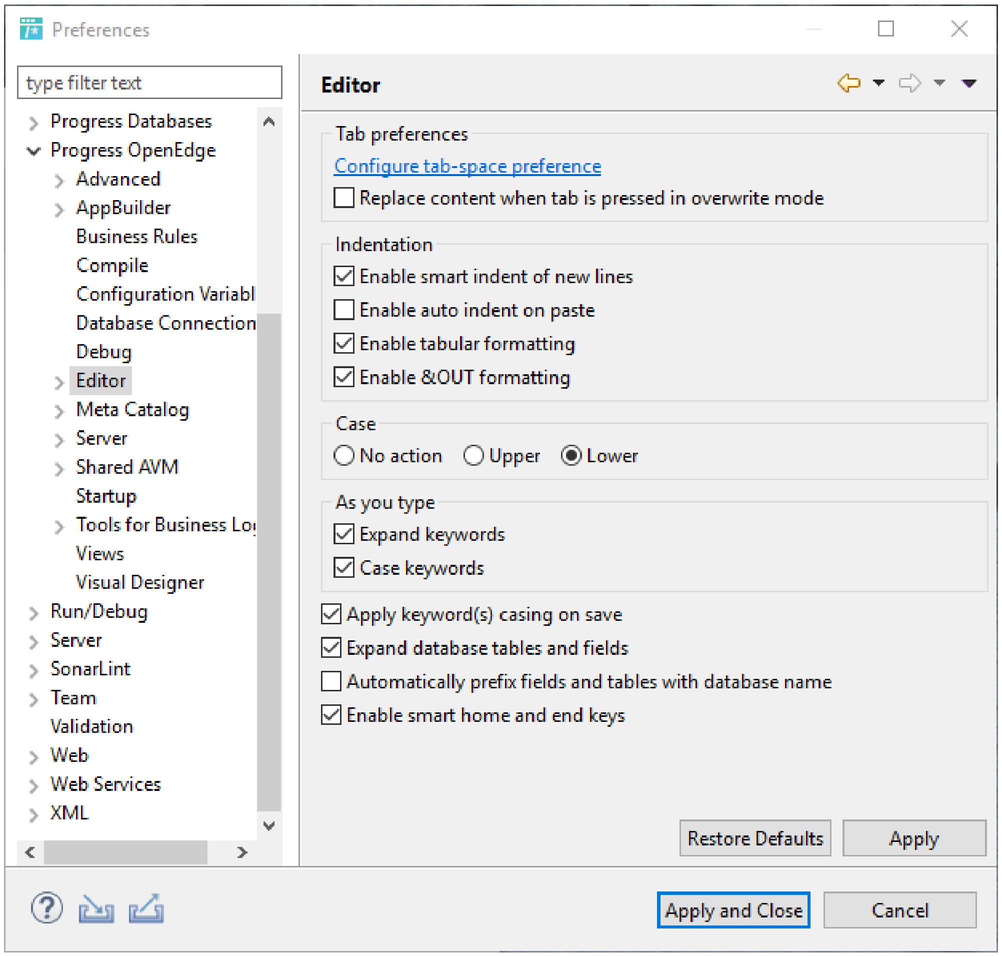
f. Select **Lower** in the **Case area**.

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**.



Preferences

type filter text

- Progress Databases
- Progress OpenEdge
  - Advanced
  - AppBuilder
  - Business Rules
  - Compile
  - Configuration Variabl
  - Database Connection
  - Debug
  - Editor
  - Meta Catalog
  - Server
  - Shared AVM
  - Startup
  - Tools for Business Lo
  - Views
  - Visual Designer
- Run/Debug
- Server
- SonarLint
- Team
- Validation
- Web
- Web Services
- XML

Editor

Tab preferences

Configure tab-space preference

☐ Replace content when tab is pressed in overwrite mode

Indentation

☑ Enable smart indent of new lines

☐ Enable auto indent on paste

☑ Enable tabular formatting

☑ Enable &OUT formatting

Case

No action Upper [✓ Lower]

As you type

☑ Expand keywords

☑ Case keywords

☑ Apply keyword(s) casing on save

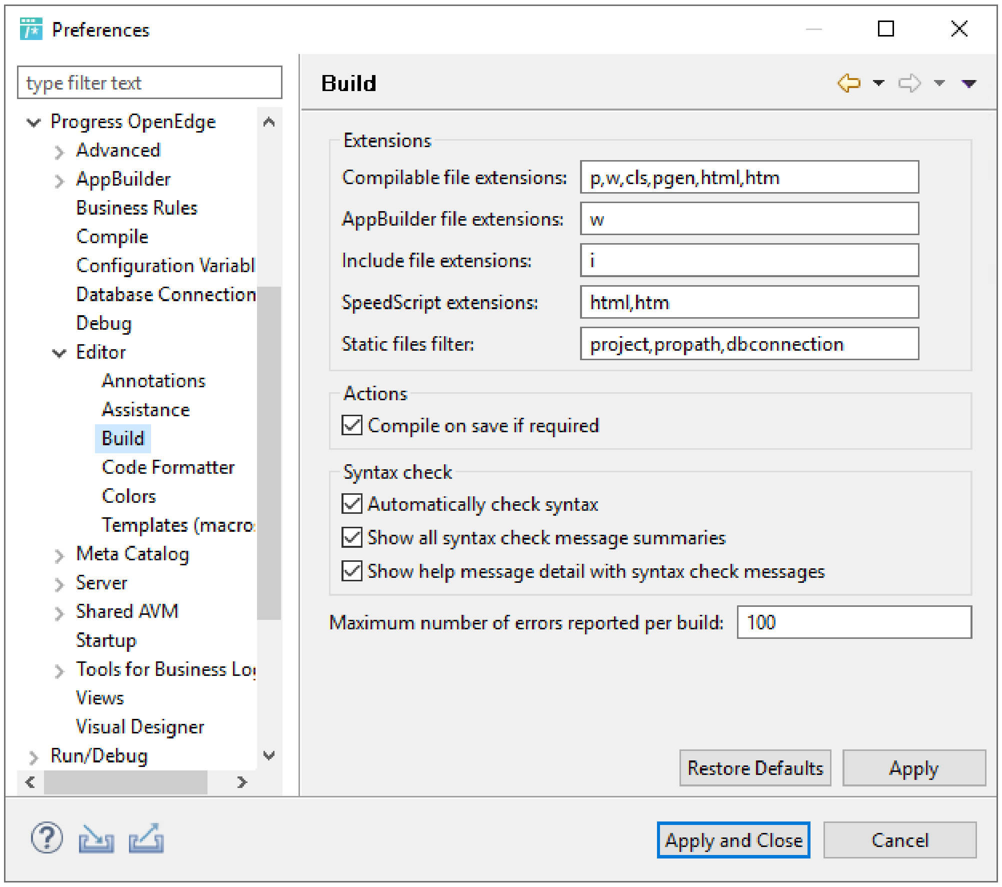
☑ Expand database tables and fields

☐ Automatically prefix fields and tables with database name

☑ Enable smart home and end keys

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

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



Build

Extensions

Compilable file extensions: p,w,cls,pgen,html,htm

AppBuilder file extensions: w

Include file extensions: i

SpeedScript extensions: html,htm

Static files filter: project,propath,dbconnection

Actions

☑ Compile on save if required

Syntax check

☑ Automatically check syntax

☑ Show all syntax check message summaries

☑ Show help message detail with syntax check messages

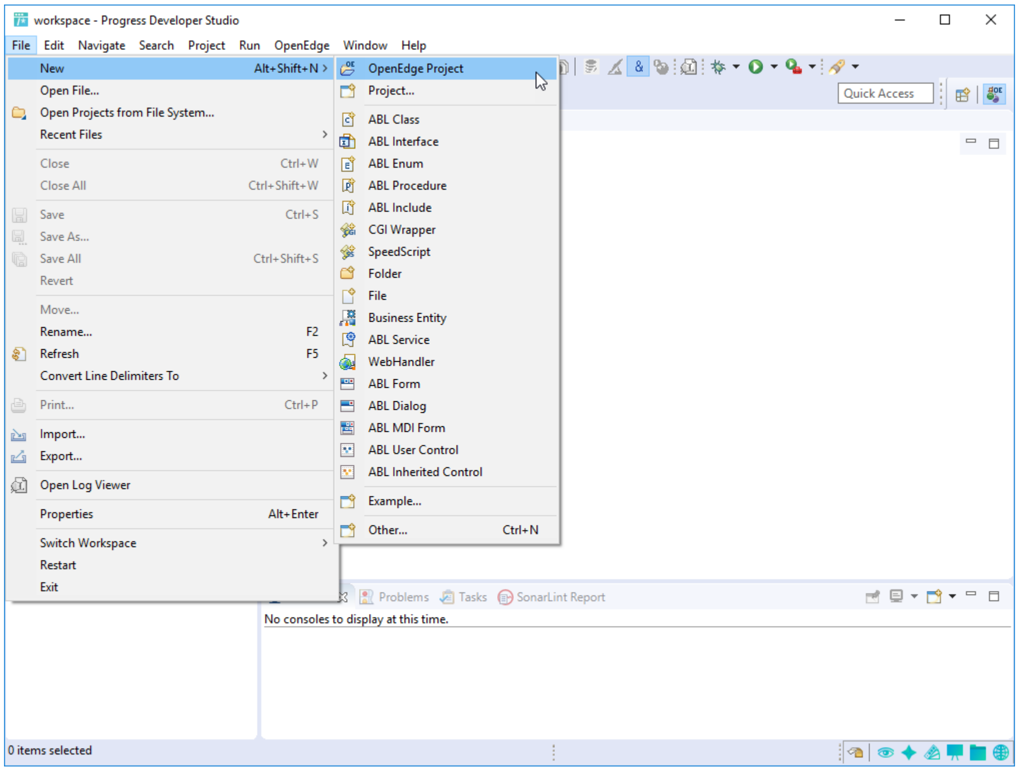
Maximum number of errors reported per build: 100

m. Click **Apply and Close**.

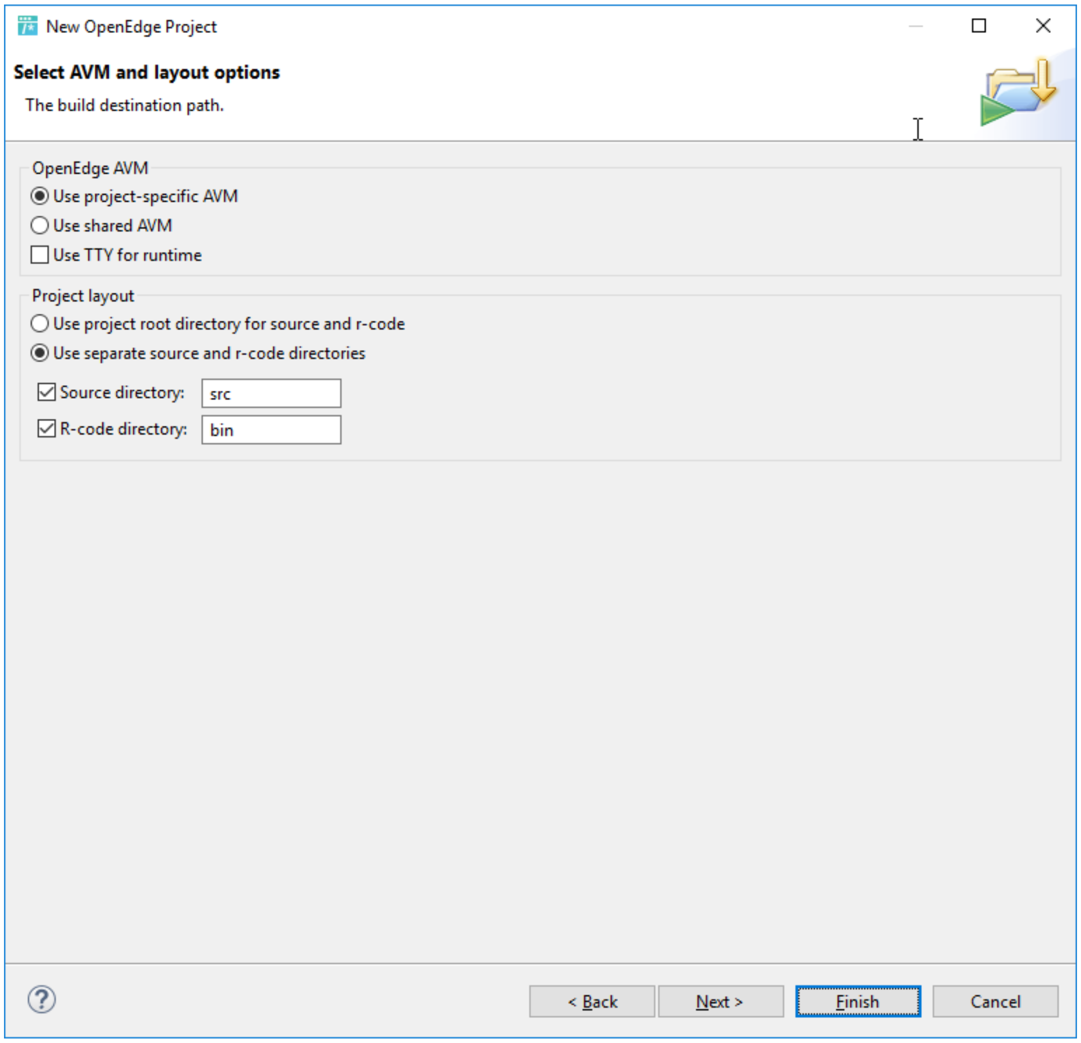
Part 3: Set up the Server project

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 sports2020 database.

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`.



Select AVM and layout options

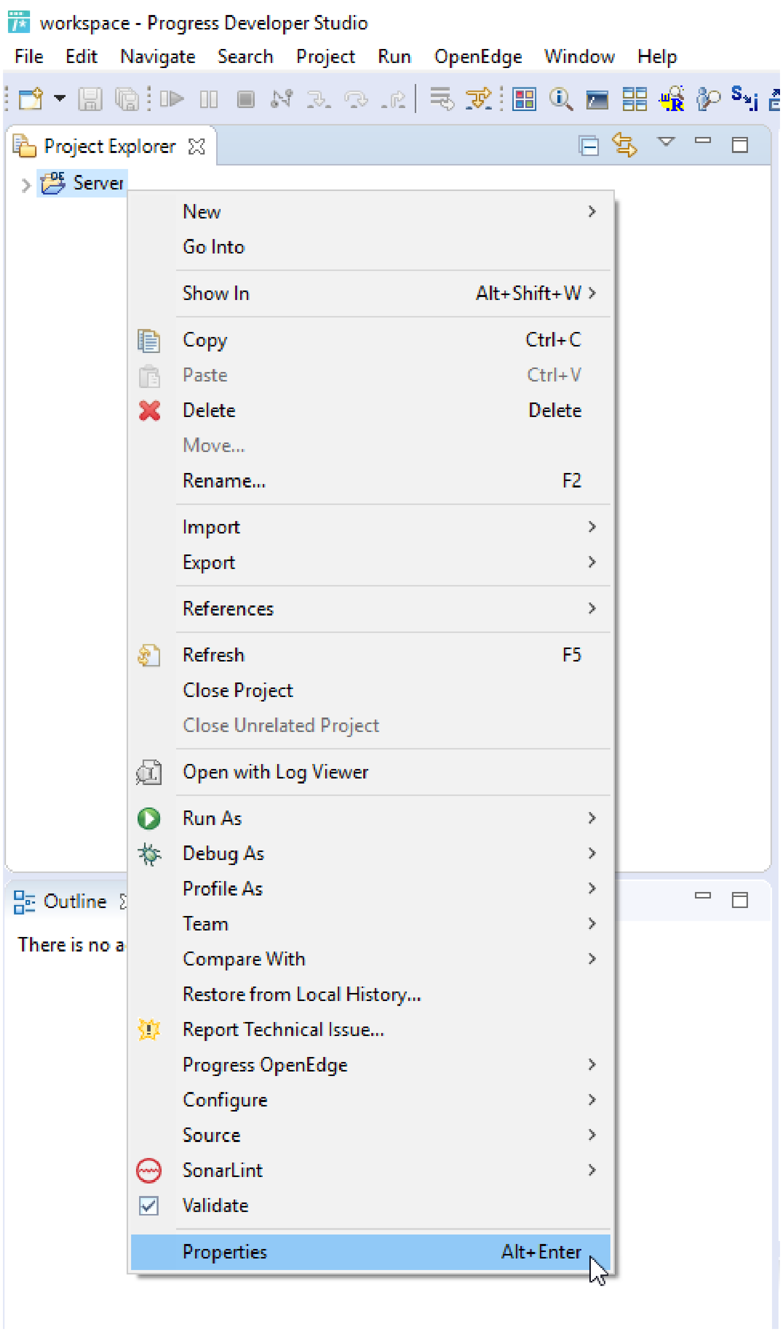
The build destination path.

OpenEdge AVM
- (●) Use project-specific AVM
- ( ) Use shared AVM
- [ ] Use TTY for runtime

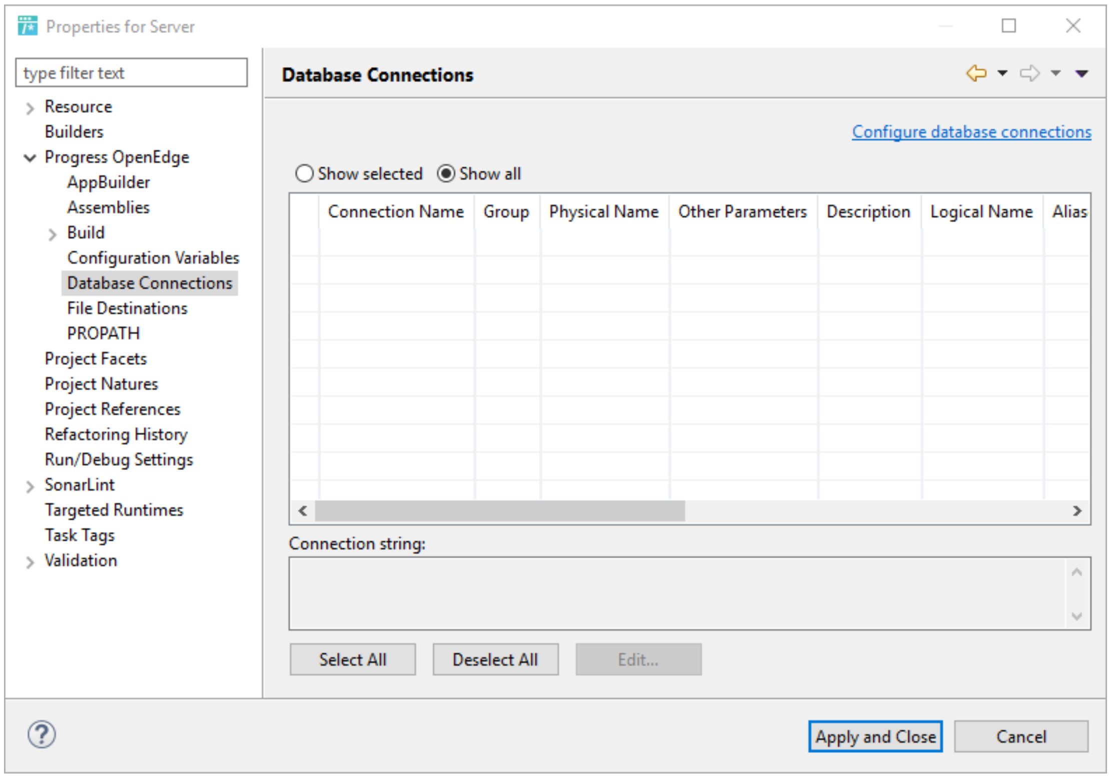
Project layout
- ( ) Use project root directory for source and r-code
- (●) Use separate source and r-code directories
- [✓] Source directory: src
- [✓] R-code directory: bin

e. Click **Finish**.

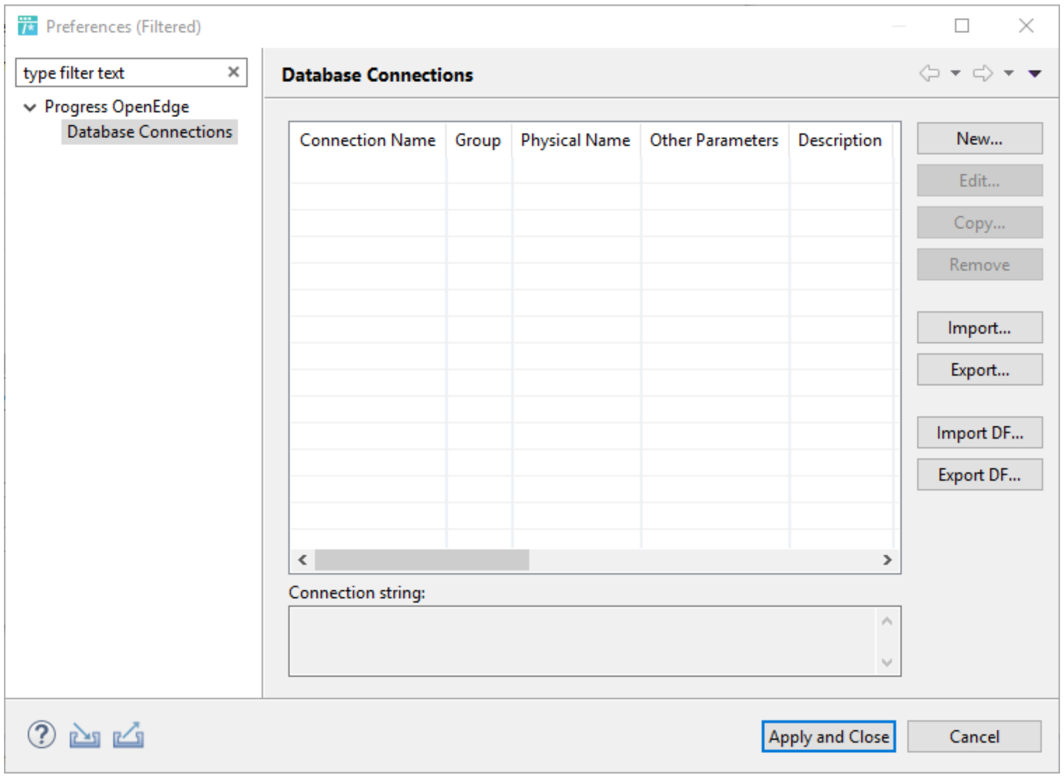
3. Add the database connection to the workspace and associate it with the **Server** project. Right-click the **Server** project, and then select **Properties**.



4. Navigate to **Progress OpenEdge** > **Database Connections**.



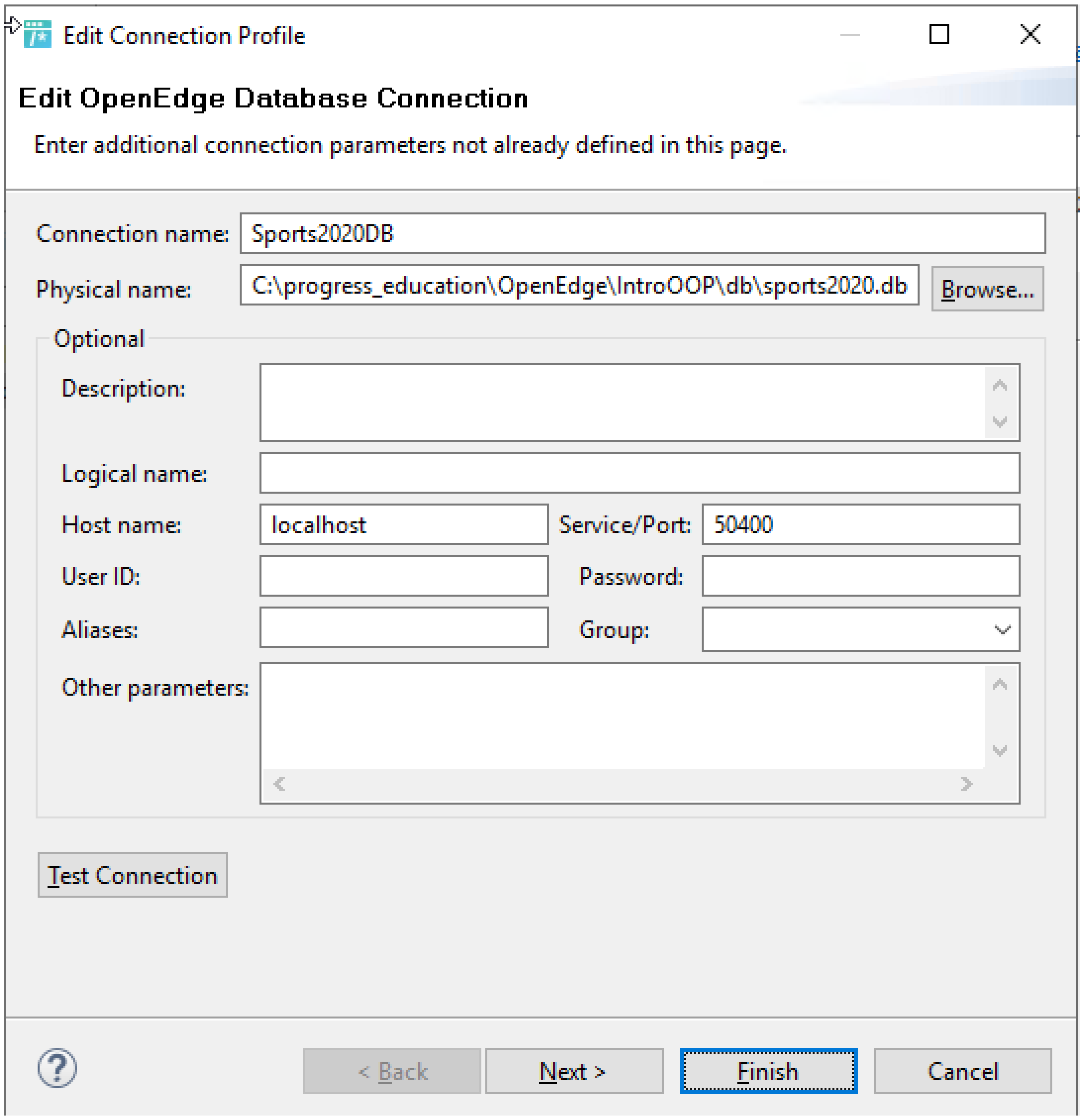
5. Click the **Configure database connections** link.



6. Click **New** to open the **Add OpenEdge Database Connection** wizard.
7. Enter the connection name as `Sports2020DB`.

8. Click the **Browse** button, and then navigate to and select `C:\progress_education\openedge\IntroOOP\db\sports2020.db`.
9. Specify the **host name** as `localhost`.
10. Specify `50400` as the **port number**.

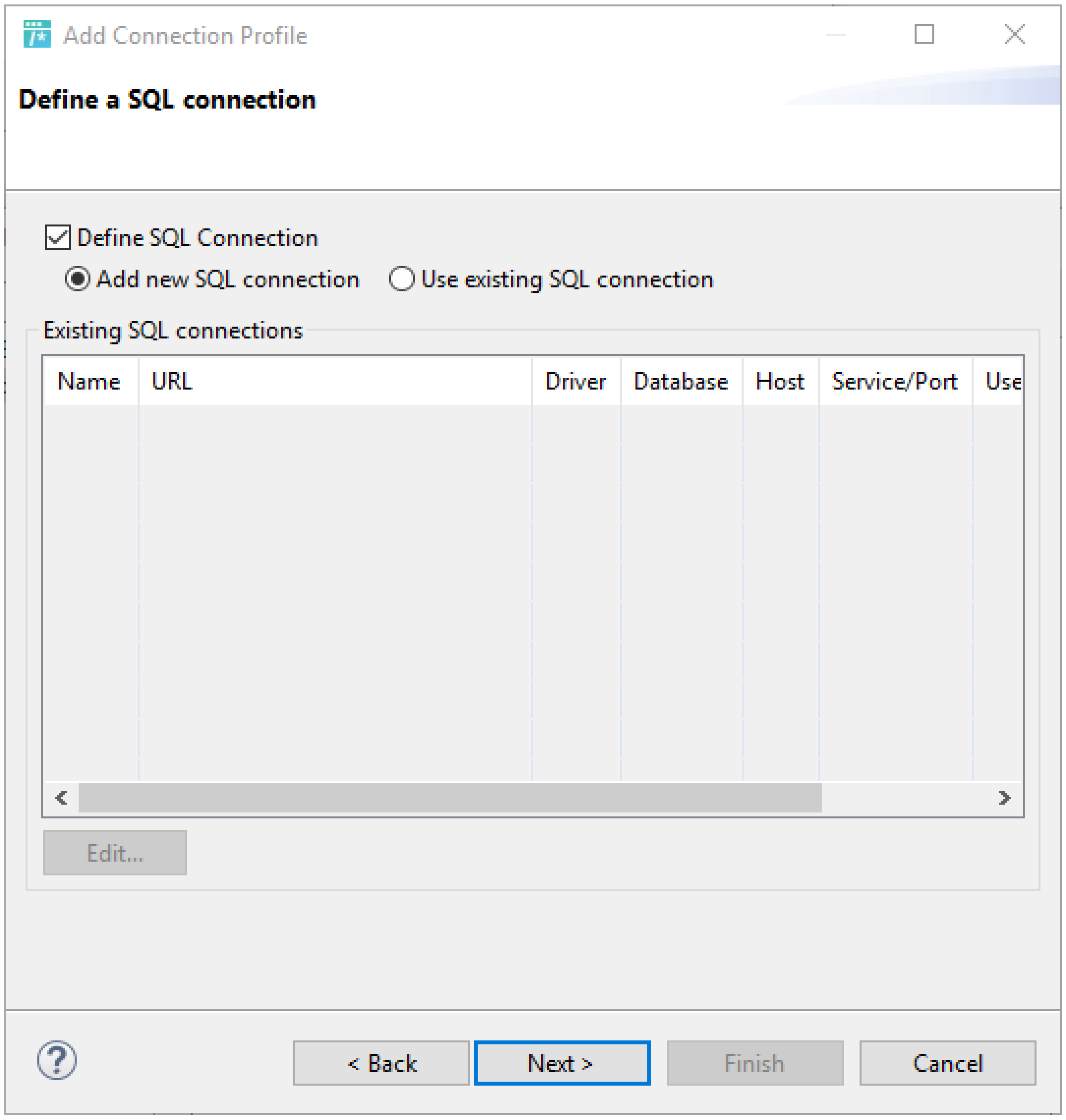
    If port `50400` is unavailable on your system, then use a different port number. It must be the same port number that you specified when you started the Database Server.



Enter additional connection parameters not already defined in this page.

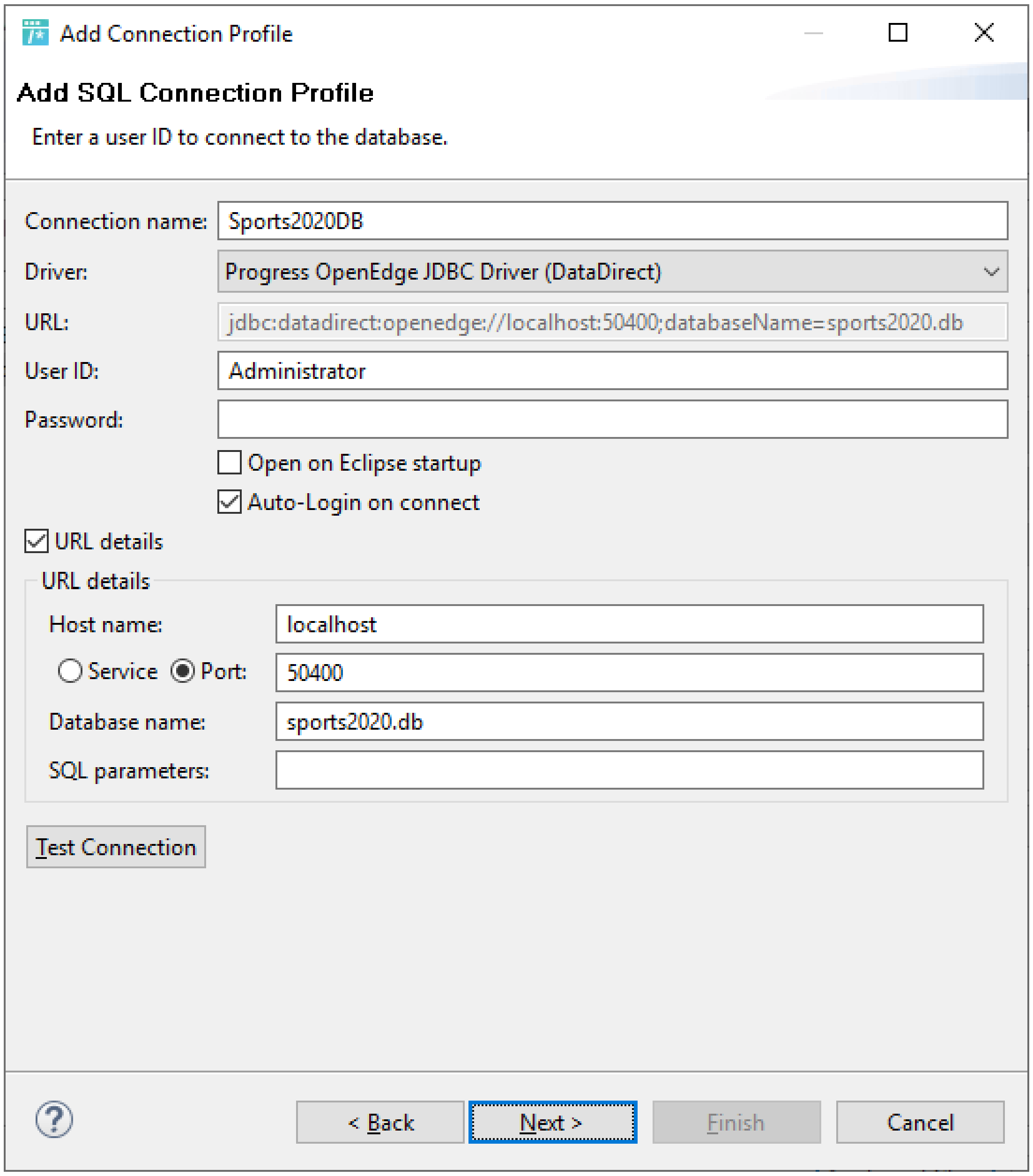
Connection name: Sports2020DB
Physical name: C:\progress_education\OpenEdge\IntroOOP\db\sports2020.db
Description:
Logical name:
Host name: localhost
Service/Port: 50400
User ID:
Password:
Aliases:
Group:
Other parameters:

11. Click **Next** in the **Add OpenEdge Database Connection** dialog box.
12. Click **Next** in the **Define a SQL connection** dialog box to keep the defaults.



Name	URL	Driver	Database	Host	Service/Port	Use

13. Click **Next** in **Add SQL Connection Profile**.



Add Connection Profile

Add SQL Connection Profile

Enter a user ID to connect to the database.

Connection name:	Sports2020DB
Driver:	Progress OpenEdge JDBC Driver (DataDirect)
URL:	jdbc:datadirect:openedge://localhost:50400;databaseName=sports2020.db
User ID:	Administrator
Password:	

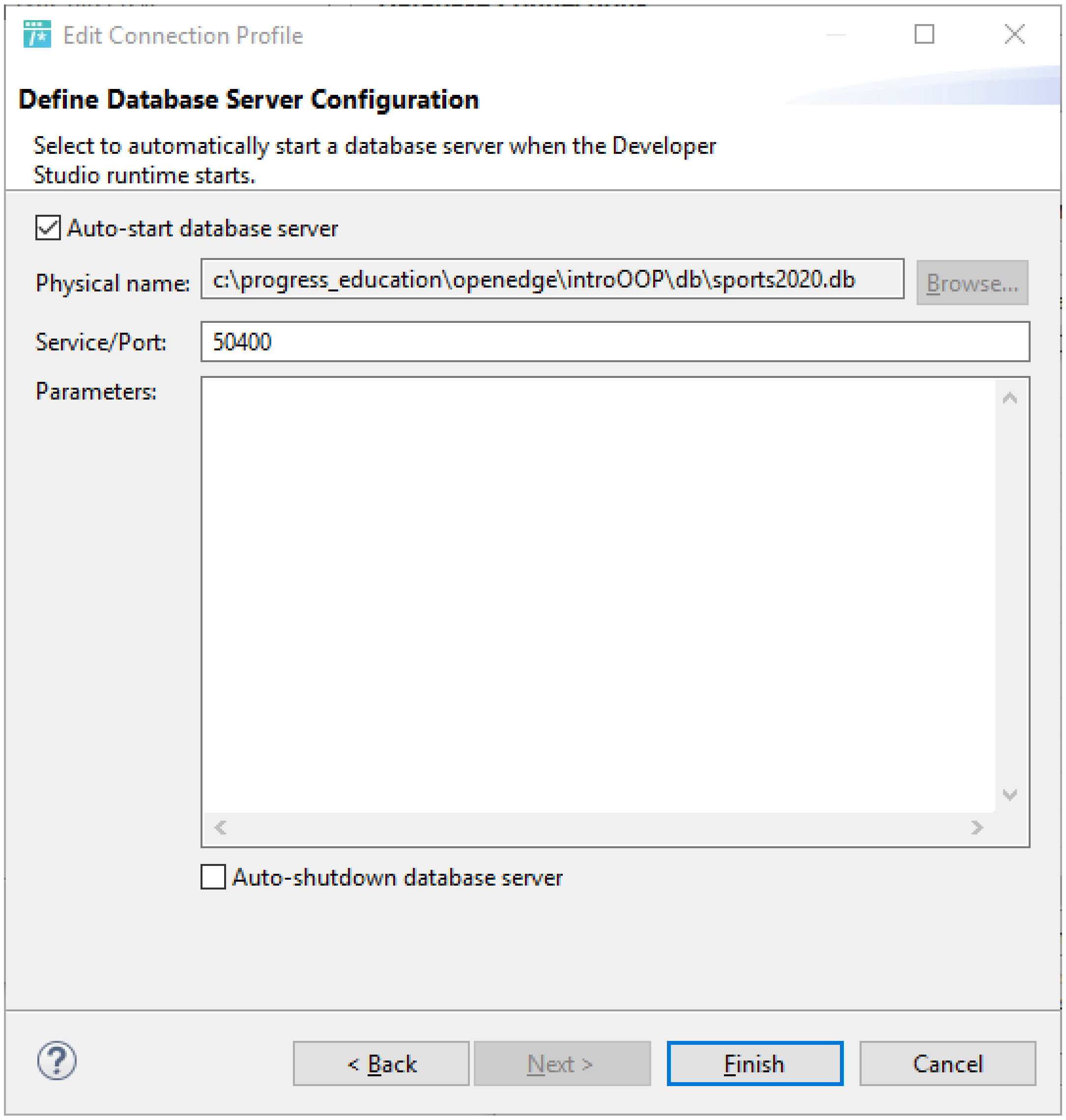
☐ Open on Eclipse startup
☑ Auto-Login on connect
☑ URL details

URL details

Host name:	localhost
○ Service ● Port:	50400
Database name:	sports2020.db
SQL parameters:	

14. Click **Next**.

15. In the **Defined Database Server Configuration** dialog box, select **Auto-start database server**.



Edit Connection Profile

Define Database Server Configuration

Select to automatically start a database server when the Developer Studio runtime starts.

☑ Auto-start database server

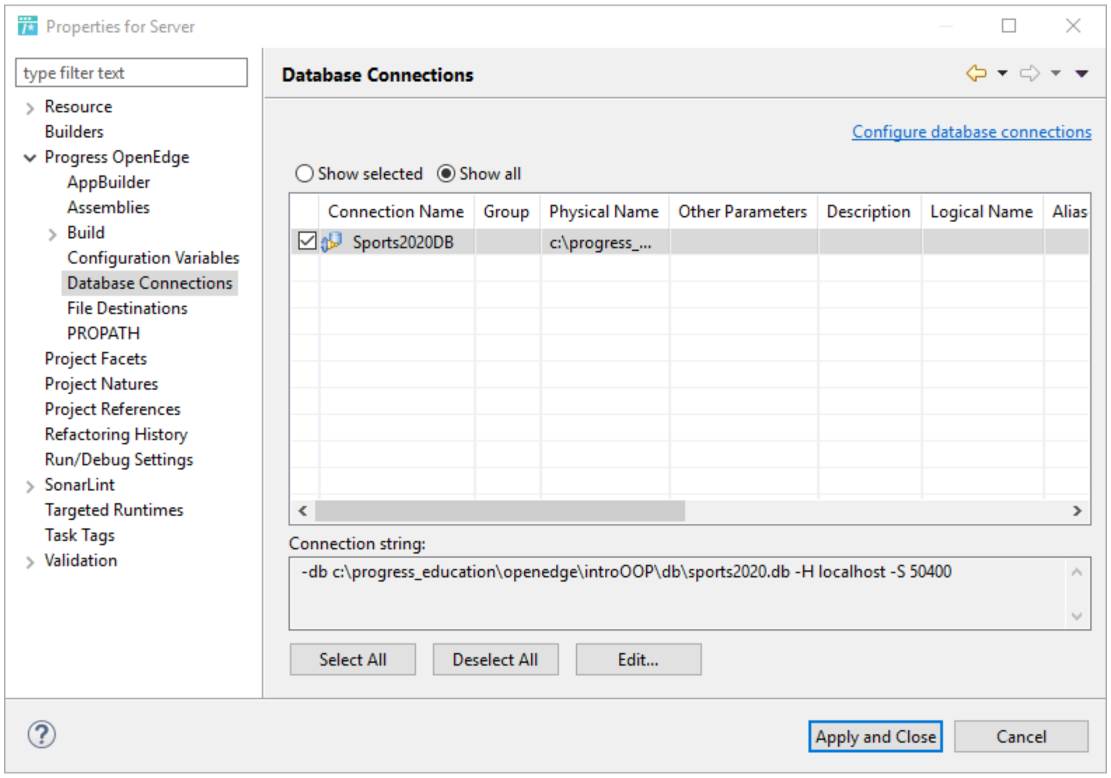
Physical name: c:\progress_education\openedge\introOOP\db\sports2020.db

Service/Port: 50400

Parameters:

☐ Auto-shutdown database server

16. Click **Finish** in the **Define Database Server Configuration** page.
17. Click **Apply and Close**.
18. Check the **Sports2020DB** checkbox in the **Database Connections** page.



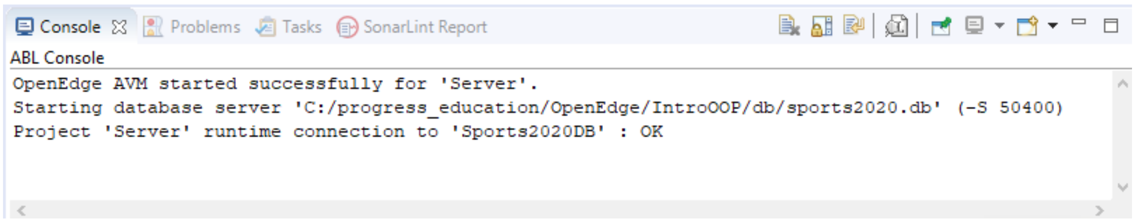
Database Connections

	Connection Name	Group	Physical Name	Other Parameters	Description	Logical Name	Alias
✓	Sports2020DB		c:\progress_...				

Connection string:

```
-db c:\progress_education\openedge\introOOP\db\sports2020.db -H localhost -S 50400
```

19. Click **Apply and Close**.
20. In the **Console** view, the following message appears:



```
OpenEdge AVM started successfully for 'Server'.
Starting database server 'C:/progress_education/OpenEdge/IntroOOP/db/sports2020.db' (-S 50400)
Project 'Server' runtime connection to 'Sports2020DB' : OK
```

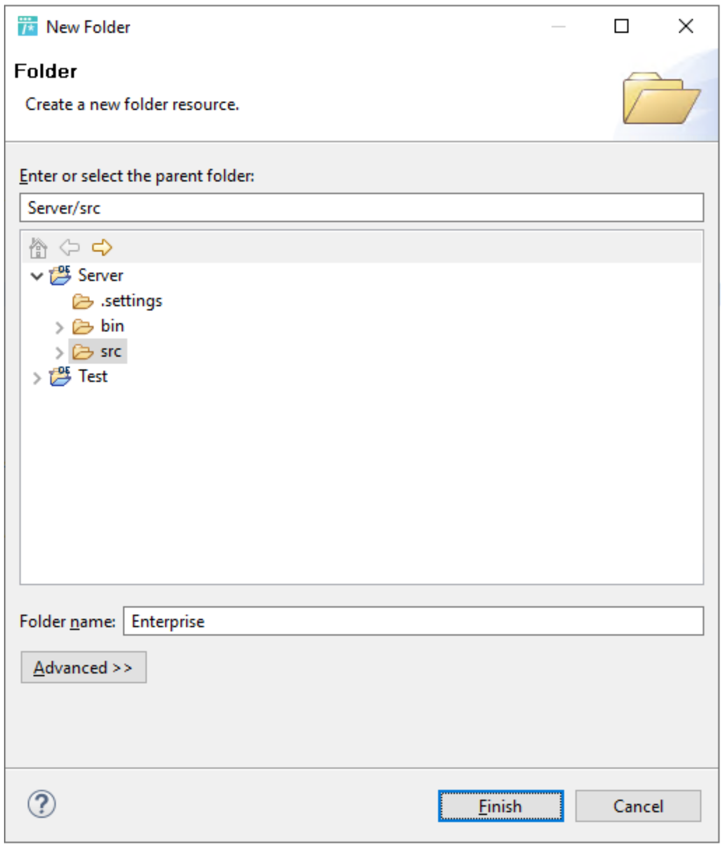
Wrap-up

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.

Solutions: Try It 2.1, Define classes

Part 1: Create the Emp class

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**.


New Folder

Folder

Create a new folder resource.

Enter or select the parent folder:

Server/src

Folder name: Enterprise

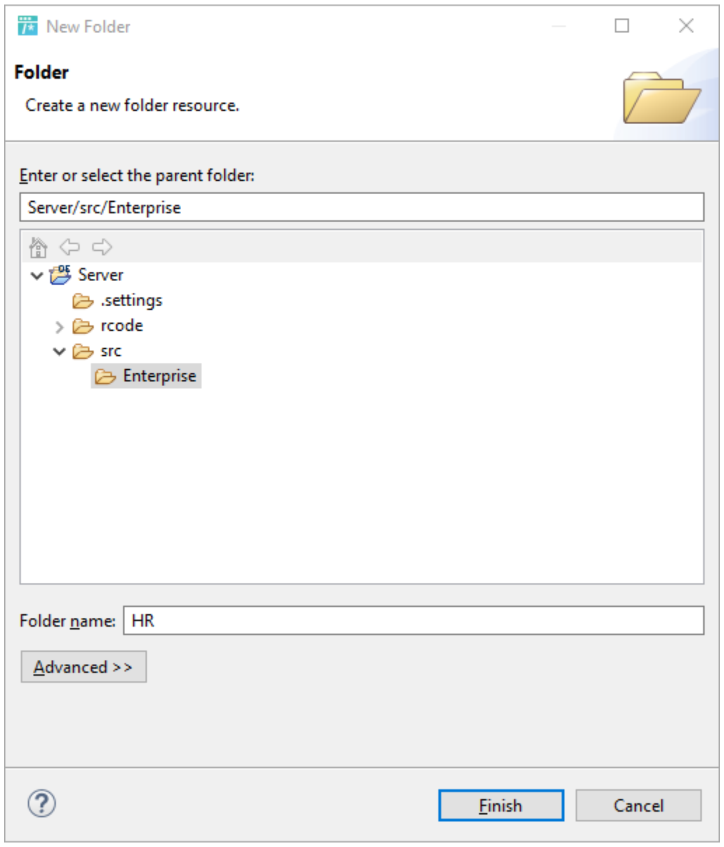
d. Click **Finish**.

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**.



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.

Tip: Use the **New ABL class** wizard.

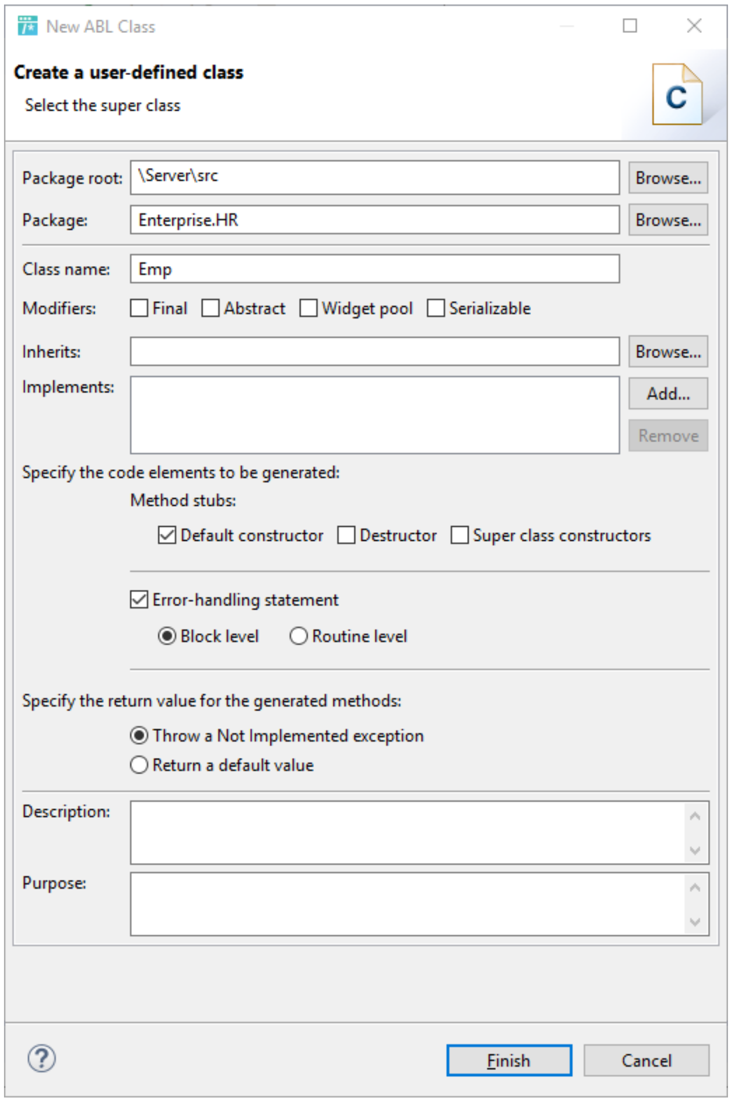
a. Right-click **HR**.

b. Select **New** > **ABL Class**.

c. Enter *Emp* for the **Class name**.

d. Select **Default constructor**.

You can optionally add information to the **Description**.



New ABL Class

Create a user-defined class

Select the super class

Package root: \Server\src

Package: Enterprise.HR

Class name: Emp

Modifiers: ☐ Final ☐ Abstract ☐ Widget pool ☐ Serializable

Inherits:

Implements:

Specify the code elements to be generated:

Method stubs:

☑ Default constructor ☐ Destructor ☐ Super class constructors

☑ Error-handling statement

● Block level ○ Routine level

Specify the return value for the generated methods:

● Throw a Not Implemented exception

○ Return a default value

Description:

Purpose:

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.
```

Part 2: Define data members for the Emp class

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*

Tip: Use the **Add Property** wizard. (Alt-Shift +Y)

a. In the editor, place the cursor anywhere in the source file.

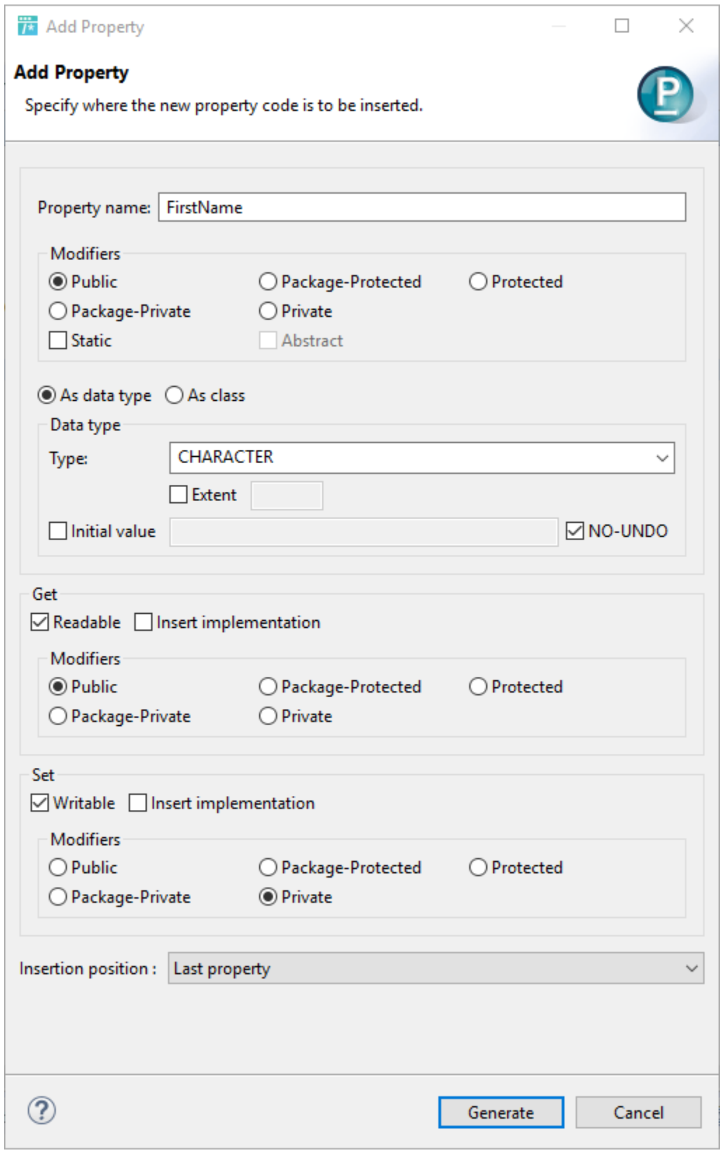
b. Right-click and then select **Source** > **Add Property**. The **Add Property** wizard opens.

c. Enter *FirstName* for the **Property name**.

d. Select the *CHARACTER* from the **Data type** drop-down list.

e. Select *Private* for the **Set accessor**.

f. Select *Last property* from the **Insertion position** drop-down list.



Add Property

Add Property

Specify where the new property code is to be inserted.

Property name: FirstName

Modifiers: [✓ Public] Package-Protected Protected Package-Private Private ☐ Static ☐ Abstract

[✓ As data type] As class

Data type

Type: CHARACTER

☐ Extent

☐ Initial value ☑ NO-UNDO

Get

☑ Readable ☐ Insert implementation

Modifiers: [✓ Public] Package-Protected Protected Package-Private Private

Set

☑ Writable ☐ Insert implementation

Modifiers: Public Package-Protected Protected Package-Private [✓ Private]

Insertion position : Last property

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
  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*.

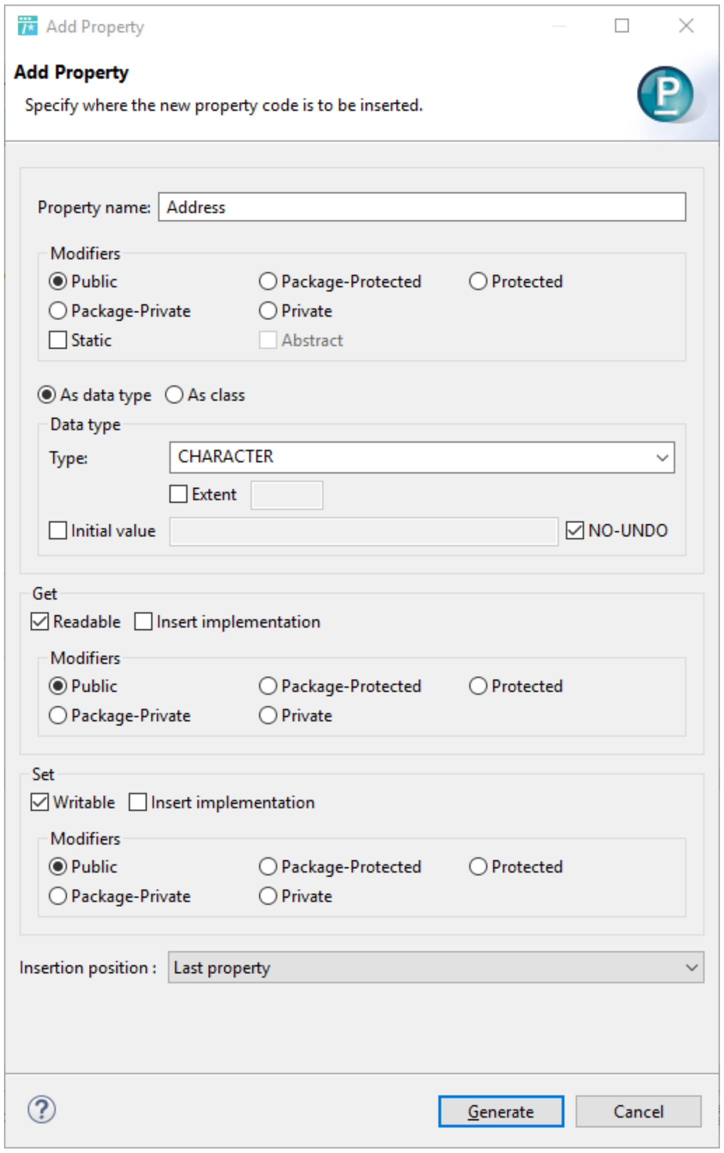
a. In the editor, place the cursor anywhere in the source file.

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* from the **Data type** drop-down list.

e. Select *Last property* from the **Insertion position** drop-down list.



Add Property

Add Property

Specify where the new property code is to be inserted.

Property name: Address

Modifiers: [✓ Public] Package-Protected Protected Package-Private Private Static Abstract

[✓ As data type] As class

Data type

Type: CHARACTER

Extent

Initial value [✓ NO-UNDO]

Get

[✓ Readable] Insert implementation

Modifiers: [✓ Public] Package-Protected Protected Package-Private Private

Set

[✓ Writable] Insert implementation

Modifiers: [✓ Public] Package-Protected Protected Package-Private Private

Insertion position : Last property

Generate Cancel

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.

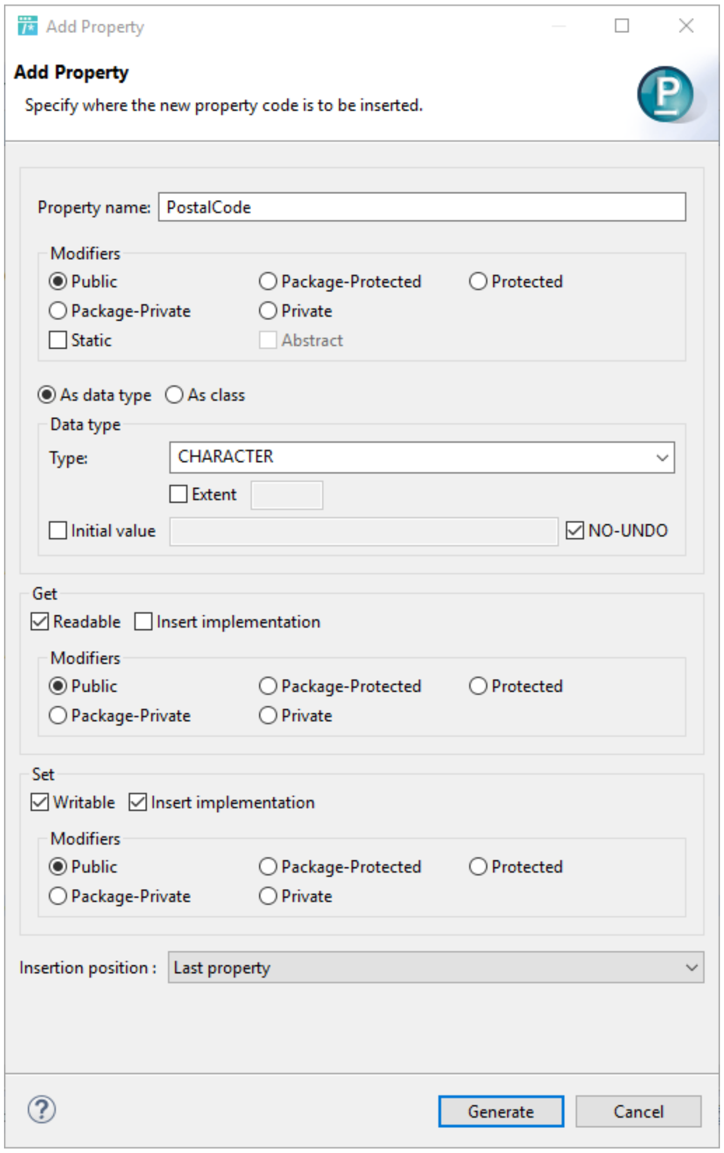
   b. Right-click and then select **Source** > **Add Property**. The **Add Property** wizard opens.

   c. Enter *PostalCode* for the property as an array of size **Property** name.

   d. Select the *CHARACTER* from the **Data type** drop-down list.

   e. Select *Insert implementation* for **Set**.

   f. Select *Last property* from the **Insertion position** drop-down list.


Add Property

Add Property

Specify where the new property code is to be inserted.

Property name: PostalCode

Modifiers: (● Public) ○ Package-Protected ○ Protected ○ Package-Private ○ Private ☐ Static ☐ Abstract

(● As data type) ○ As class

Data type

Type: CHARACTER

☐ Extent

☐ Initial value ☑ NO-UNDO

Get

☑ Readable ☐ Insert implementation

Modifiers: (● Public) ○ Package-Protected ○ Protected ○ Package-Private ○ Private

Set

☑ Writable ☑ Insert implementation

Modifiers: (● Public) ○ Package-Protected ○ Protected ○ Package-Private ○ Private

Insertion position : Last property

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 *Phones3*. It will be *public* property as an array 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.

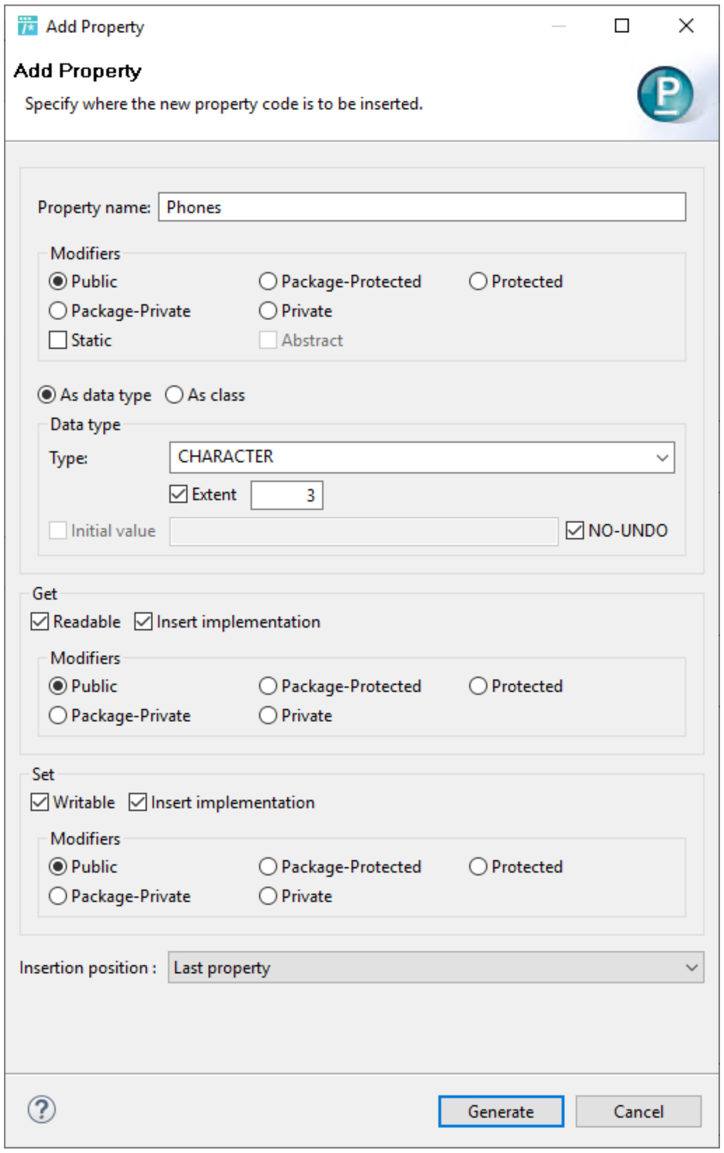
   d. Select *CHARACTER* as the **Type**.

   e. Select **Extent** and then enter *3* in 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. position.



Add Property

Add Property

Specify where the new property code is to be inserted.

Property name: Phones

Modifiers: [✓ Public] Package-Protected Protected Package-Private Private ☐ Static ☐ Abstract

[✓ As data type] As class

Data type

Type: CHARACTER

☑ Extent 3

☐ Initial value ☑ NO-UNDO

Get

☑ Readable ☑ Insert implementation

Modifiers: [✓ Public] Package-Protected Protected Package-Private Private

Set

☑ Writable ☑ Insert implementation

Modifiers: [✓ Public] Package-Protected Protected Package-Private Private

Insertion position : Last property

Generate Cancel

i. Click **Generate**.

The generated code should be as follows:

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

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

define public property Phones as character extent 3 no-undo

Part 3: Define methods for the Emp class

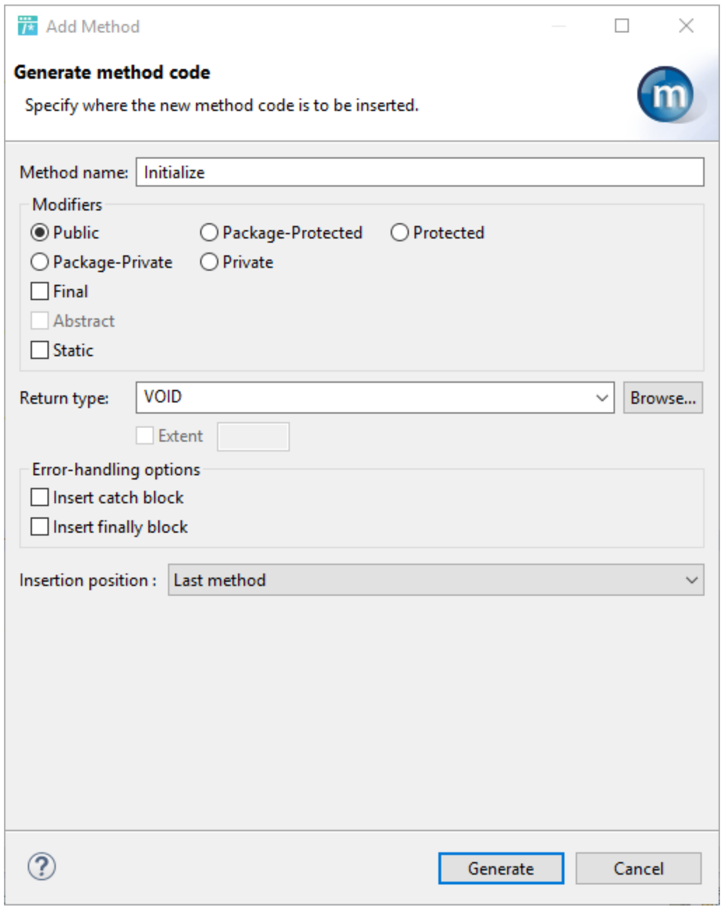
1. Define the *public Initialize()* method, which returns *void*. The input parameters for this method will be values for each of the data members of the class.

a. Place the cursor anywhere in the source file.

b. Right-click and then select **Source** > **Add Method**(Alt-Shift-M)

c. Enter *Initialize* in the **Method name** field.

d. Select *Last method* from the **Insertion position** drop-down list.



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

f. Modify the code 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 ):

    return.

end method.
```

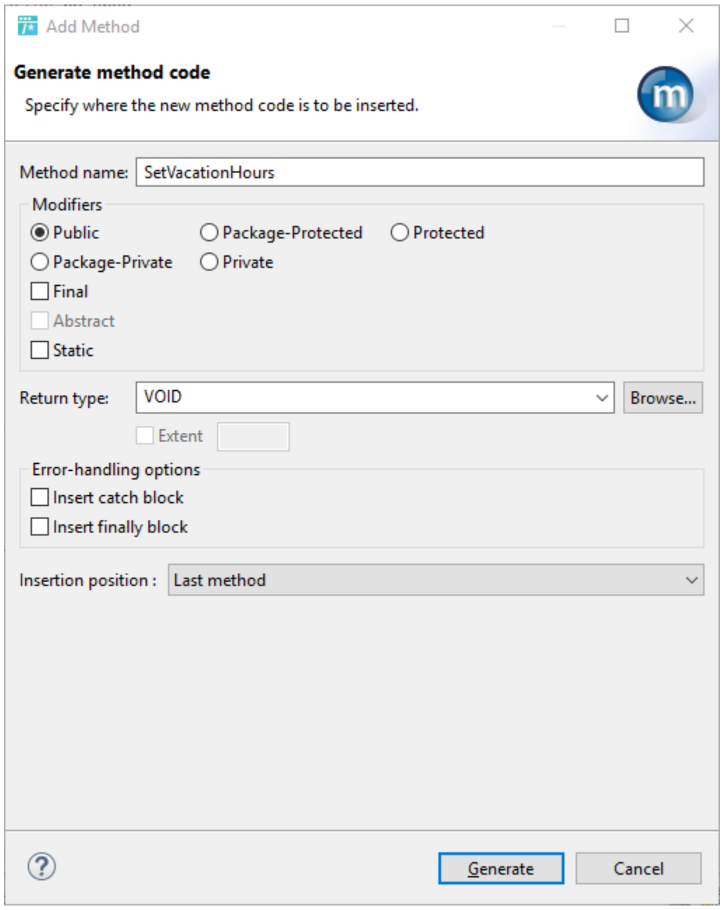
2. Define the *public SetVacationHours()* method, which returns *void*. It has a single *input* parameter, *pHours* of type *integer*.

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* from the **Insertion position** drop-down list.


Add Method

Generate method code

Specify where the new method code is to be inserted.

Method name: SetVacationHours

Modifiers: [✓ Public] Package-Protected Protected Package-Private Private; Final ☐ Abstract ☐ Static ☐

Return type: VOID

Extent ☐

Error-handling options: Insert catch block ☐ Insert finally block ☐

Insertion position : Last method

e. Click **Generate**.

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

g. Enter the parameter as:

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

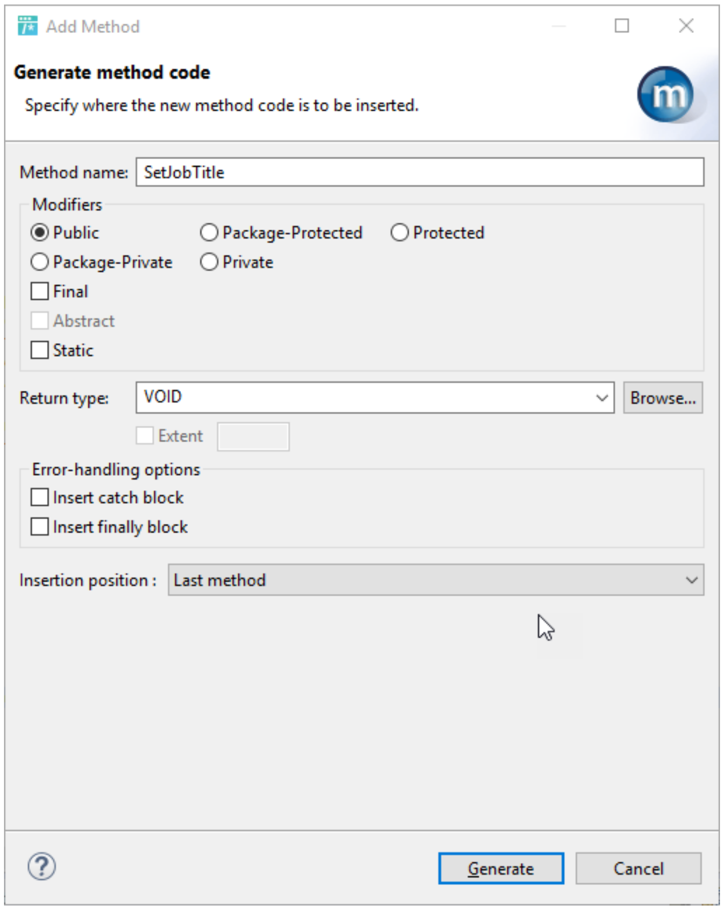
3. Define the *public SetJobTitle()* method, which returns *void*. It has a single input parameter, *pJobTitle*, of type *character*.

a. Place the cursor anywhere in the source file.

b. Right-click and then select **Source** > **Add Method**.

c. Enter *SetJobTitle* in the **Method name** field.

d. Select *Last method* from the **Insertion position** drop-down list.



Method name: SetJobTitle

Modifiers: [✓ Public] Package-Protected Protected Package-Private Private

Final, Abstract, Static: unchecked

Return type: VOID

Extent

Error-handling options: Insert catch block (unchecked), Insert finally block (unchecked)

Insertion position : Last method

e. Click **Generate**.

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

g. Enter the parameter as:

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

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

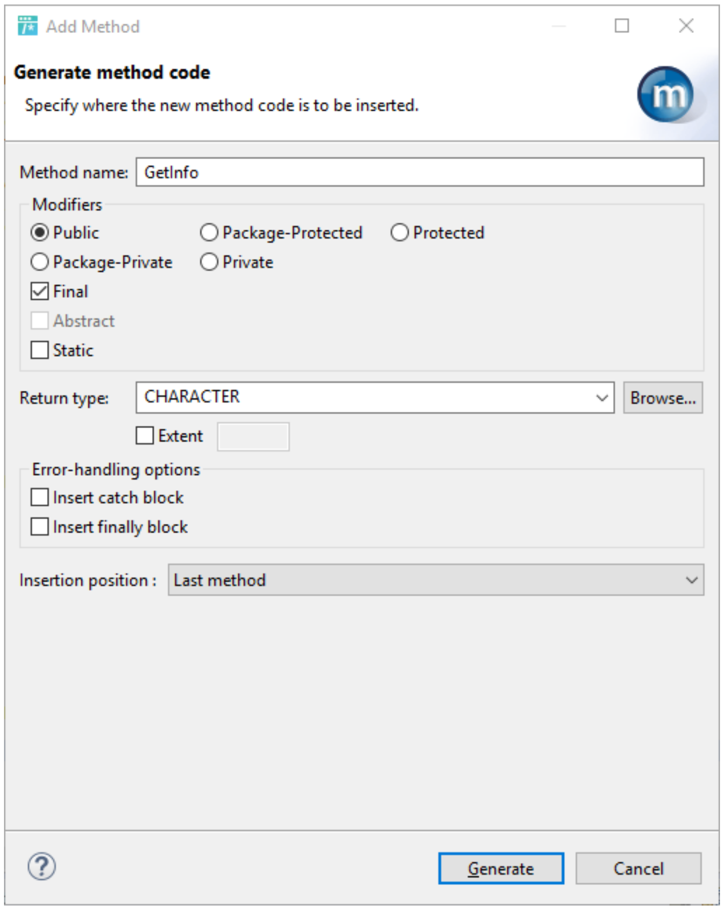
   a. Place the cursor anywhere in the source file.

   b. Right-click and then select **Source** > **Add Method**.

   c. Enter *GetInfo* for the **Method name**.

   d. Select *CHARACTER* for the **Return type**.

   e. Select *Last method* from the **Insertion position** drop-down list.



Method name: GetInfo

Modifiers: [✓ Public] Package-Protected Protected Package-Private Private

[✓ Final] Abstract Static

Return type: CHARACTER

Extent

Error-handling options: Insert catch block, Insert finally block

Insertion position : Last method

f. Click **Generate**.

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.

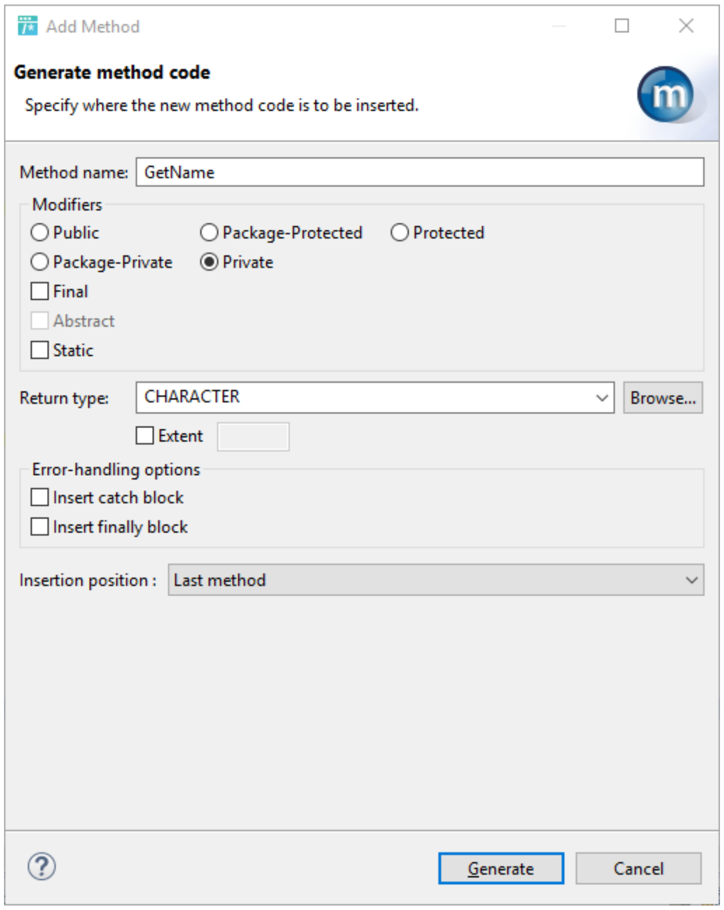
b. Right-click and then select **Source** > **Add Method**.

c. Enter *GetName* for the **Method name**.

d. Select *Private* for the **Modifier**.

e. Select *CHARACTER* for the **Return type**.

f. Select *Last method* from the **Insertion position** drop-down list.



Generate method code

Specify where the new method code is to be inserted.

Method name: GetName

Modifiers

Public
Package-Protected
Protected
Package-Private
[✓ Private]
Final
Abstract
Static

Return type: CHARACTER

Extent

Error-handling options

Insert catch block
Insert finally block

Insertion position : Last method

g. Click **Generate**.

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.

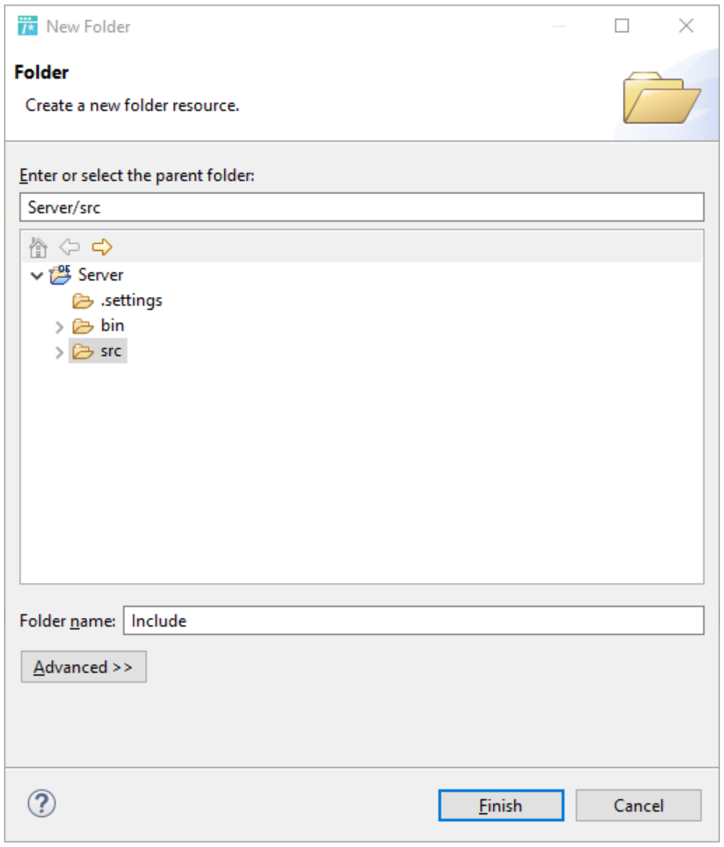
Part 4: Add the include file for the ttEmployee temp-table

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**.


New Folder

Folder

Create a new folder resource.

Enter or select the parent folder: Server/src

Server
- .settings
- bin
- src

Folder name: Include

d. Click **Finish**.

Note: If you do not specify an access mode, the default access mode for a temp-table is `private`.

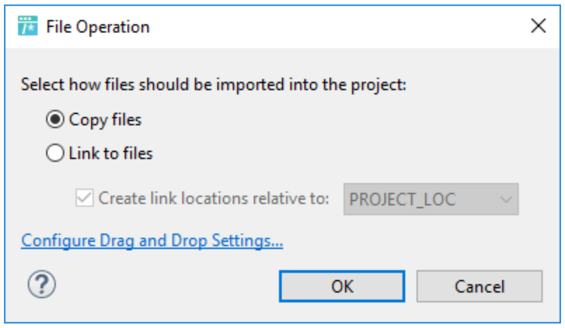
---

2. In the **Include** folder, import the *ttEmployee.i* file.

   a. In **Windows Explorer**, navigate to the `C:/progress_education/OpenEdge/introOOP/Exercises/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.


File Operation

Select how files should be imported into the project:

(●) Copy files

( ) Link to files

[✓] Create link locations relative to: PROJECT_LOC

   d. Click **OK**.

Part 5: Create the Dept class

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.

Tip: Use the **New ABL Class** wizard.

a. Select **Enterprise/HR**.

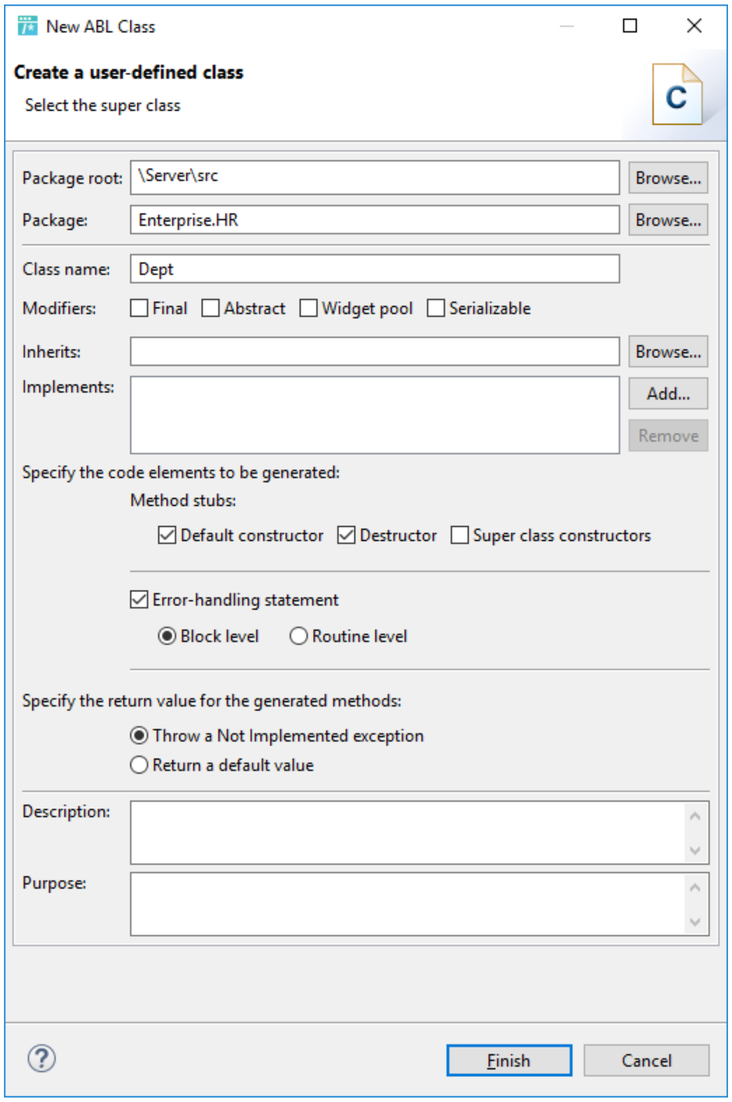
b. Right-click **HR**.

c. Select **New** > **ABL Class**.

d. Enter *Dept* for the **Class name**.

e. Select **Default constructor**.

f. Select **Destructor**.



New ABL Class

Create a user-defined class

Select the super class

Package root: \Server\src

Package: Enterprise.HR

Class name: Dept

Modifiers: ☐ Final ☐ Abstract ☐ Widget pool ☐ Serializable

Inherits:

Implements:

Specify the code elements to be generated:

Method stubs:

☑ Default constructor ☑ Destructor ☐ Super class constructors

☑ Error-handling statement

● Block level ○ Routine level

Specify the return value for the generated methods:

● Throw a Not Implemented exception

○ Return a default value

Description:

Purpose:

g. You can optionally add information to the **Description**.

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

The generated code should be as follows:

```
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.
```

Part 6: Define data members for the Dept class

1. Define these properties that 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.

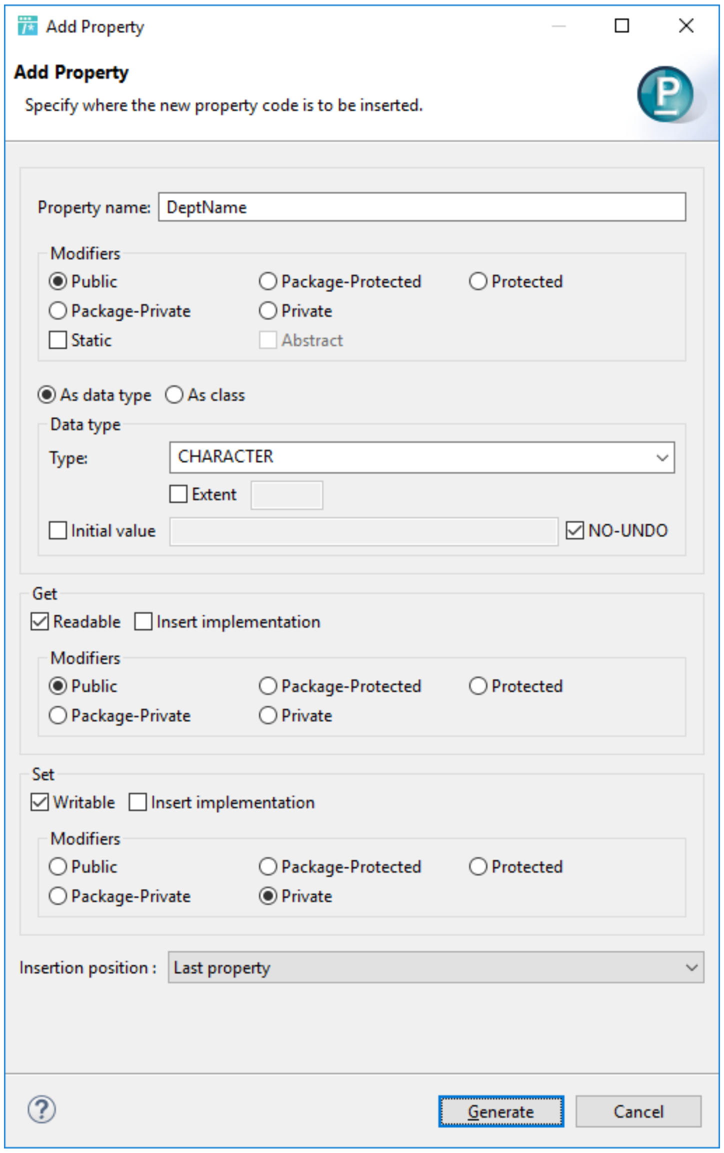
b. Right-click and then select **Source** > **Add Property**. The **Add Property** wizard opens.

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* from the **Insertion position** drop-down list.



Add Property

Add Property

Specify where the new property code is to be inserted.

Property name: DeptName

Modifiers

[✓ Public] Package-Protected Protected

Package-Private Private

☐ Static ☐ Abstract

[✓ As data type] As class

Data type

Type: CHARACTER

☐ Extent

☐ Initial value ☑ NO-UNDO

Get

☑ Readable ☐ Insert implementation

Modifiers

[✓ Public] Package-Protected Protected

Package-Private Private

Set

☑ Writable ☐ Insert implementation

Modifiers

Public Package-Protected Protected

Package-Private [✓ Private]

Insertion position : Last property

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 NumEmployees as integer no-undo
  get.
  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.

Part 7: Define a constructor and methods for the Dept class

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.

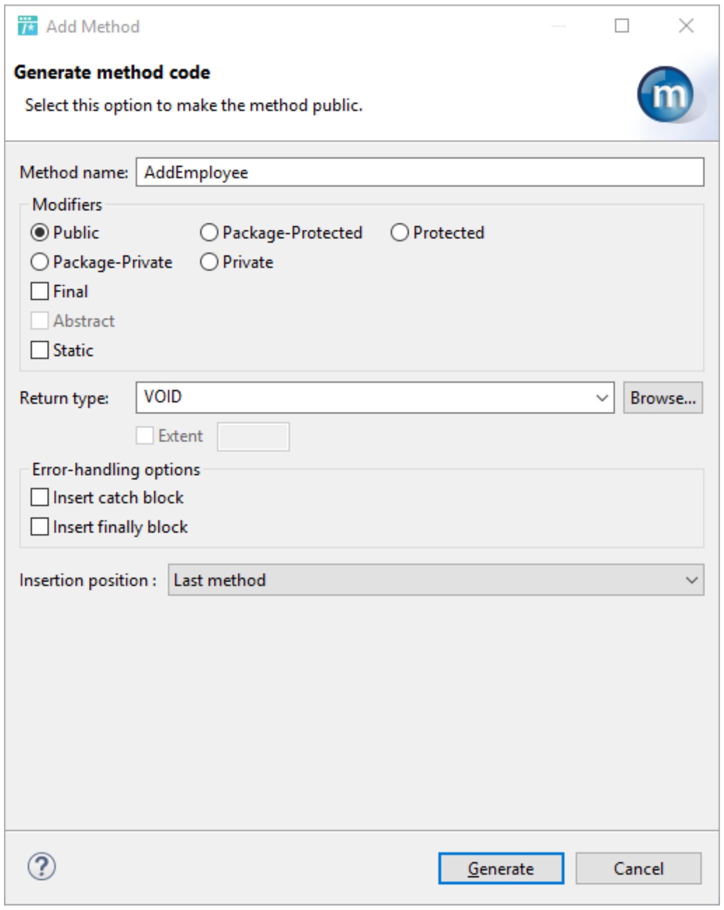
Now, you will define the *public AddEmployee()* method that returns *void*. It takes a single *input* parameter, *pEmployee*, which is a reference to an *Emp* instance.

   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* from the **Insertion position** drop-down list.


Add Method

Generate method code

Select this option to make the method public.

Method name: AddEmployee

Modifiers

- [✓ Public] Package-Protected Protected
- Package-Private Private
- ☐ Final
- ☐ Abstract
- ☐ Static

Return type: VOID

☐ Extent

Error-handling options

- ☐ Insert catch block
- ☐ Insert finally block

Insertion position : Last method

e. Click **Generate**.

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

g. Enter the parameter as:

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

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.

   a. Select all the code in the **AddEmployee()** method and copy.

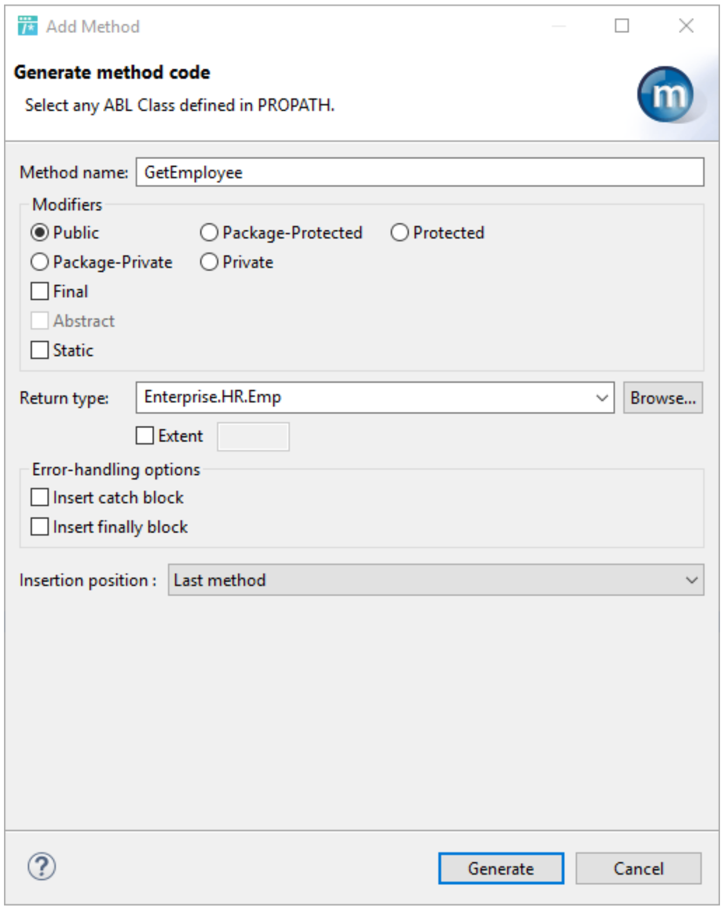
   b. Paste the code below the existing **AddEmployee()** method. You will see an error, because the methods are defined twice.

   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.
```

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 by searching for *Emp*.
   e. Select *Last method* from the **Insertion position** drop-down list.



Select any ABL Class defined in PROPATH.

Method name: GetEmployee

Modifiers: [✓ Public] Package-Protected Protected Package-Private Private; Final Abstract Static

Return type: Enterprise.HR.Emp

Extent

Error-handling options: Insert catch block; Insert finally block

Insertion position : Last method

   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 no-undo.
  return result.
end method.
```

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

Tip: 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.

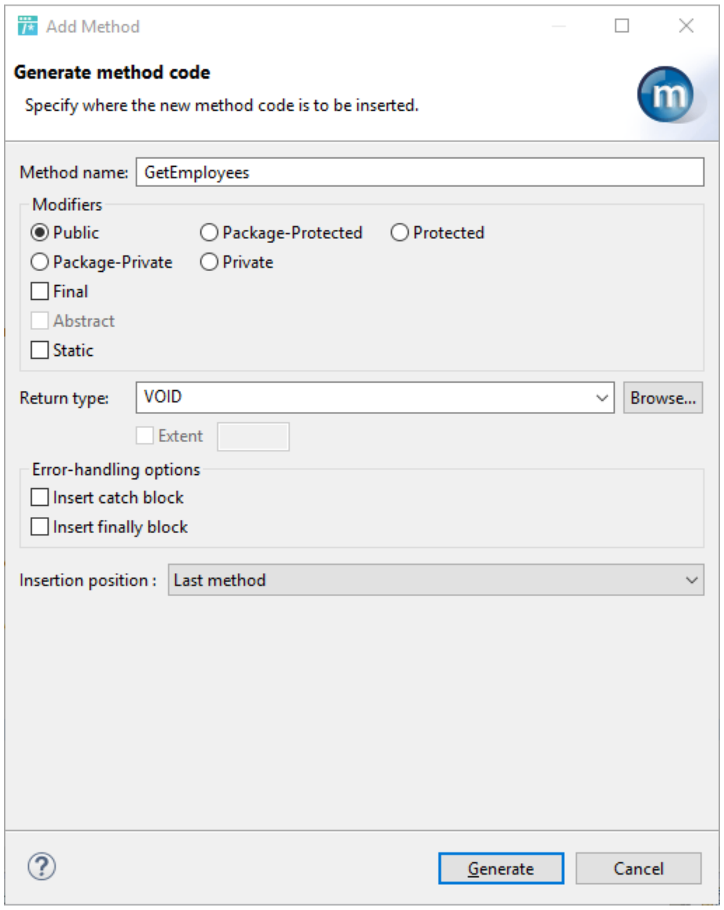
a. Place the cursor anywhere in the source file.

b. Right-click and then select **Source** > **Add Method**.

c. Enter *GetEmployees* for the **Method name**.

d. Select *VOID* for the **Return type**.

e. Select *Last method* from the **Insertion position** drop-down list.


Add Method

Generate method code

Specify where the new method code is to be inserted.

Method name: GetEmployees

Modifiers

[✓ Public] Package-Protected Protected

Package-Private Private

Final

Abstract

Static

Return type: VOID

Extent

Error-handling options

Insert catch block

Insert finally block

Insertion position : Last method

f. Click **Generate**.

g. Add the parameter information, which is output table *ttEmployee*.

The code should be as follows:

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

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

Solutions: Try It 3.1, Work with classes

Part 1: Implement the methods of the Emp class

1. Implement the code for the *Initialize()* method for the *Emp* class. Assign the value of the input parameter to each of the data members.

   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).
```

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.

Part 2: Implement the methods for the Dept class

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.

   Add this code to the constructor **after** the call to *super()*:

```
assign
   DeptName = pDeptName
   ExpenseCode = pExpenseCode
   DeptCode = pDeptCode.
```

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, 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.

   Add this code to the method:

```
define variable Empl as Emp no-undo.
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.

    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 for the class to delete all *ttEmployee* records and delete their corresponding *Emp* objects.

    Tip: 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:

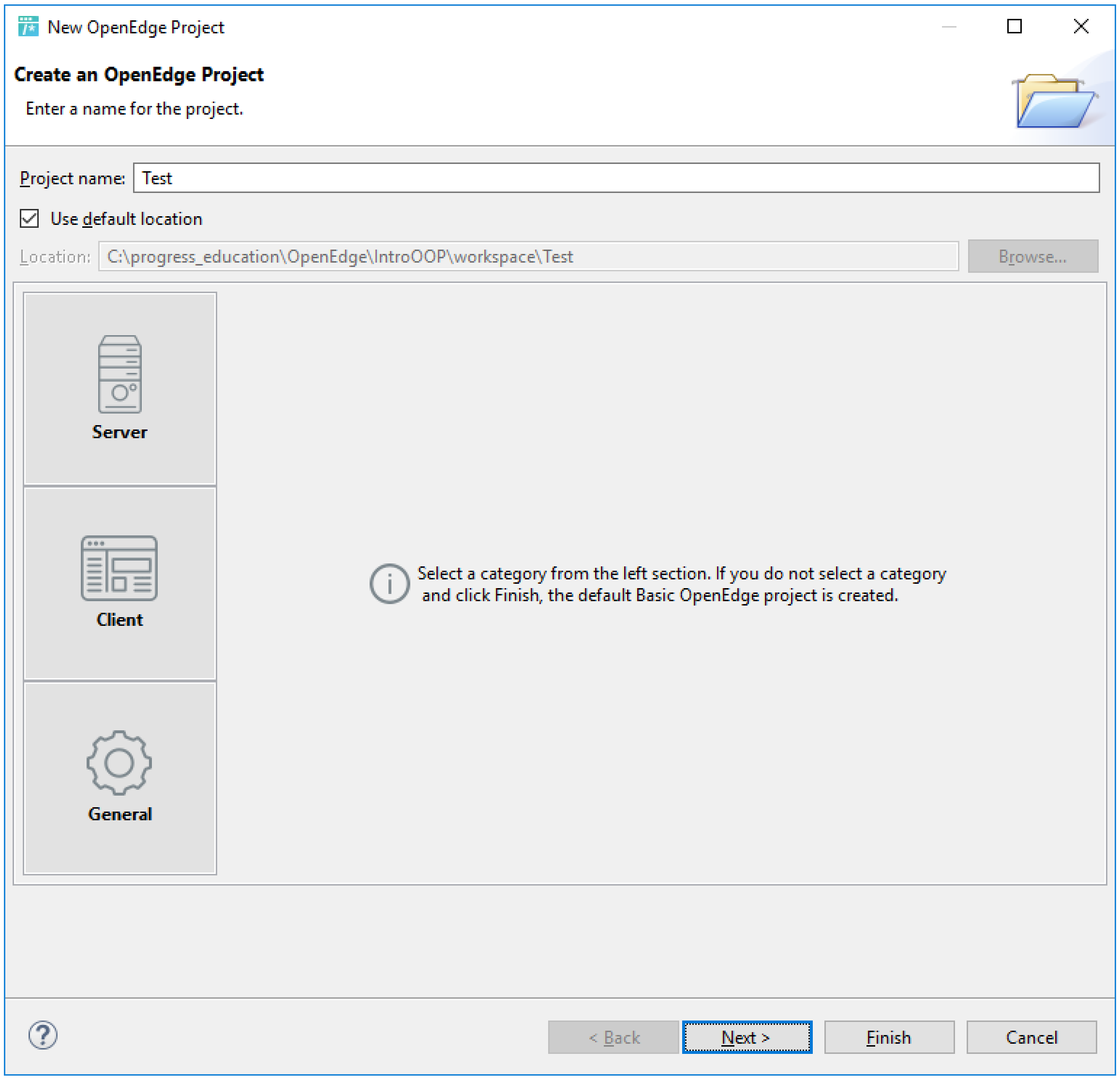
```
for each ttEmployee:
  delete object cast(ttEmployee.EmpRef,Emp).
  delete ttEmployee.
end.
```

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.

Solutions: Try It 3.2, Test classes

Part 1: Set up a Test project

1. Create a new OpenEdge Project named **Test** specifying *Sports2020DB* 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**


New OpenEdge Project

Create an OpenEdge Project

Enter a name for the project.

Project name: Test

☑ Use default location

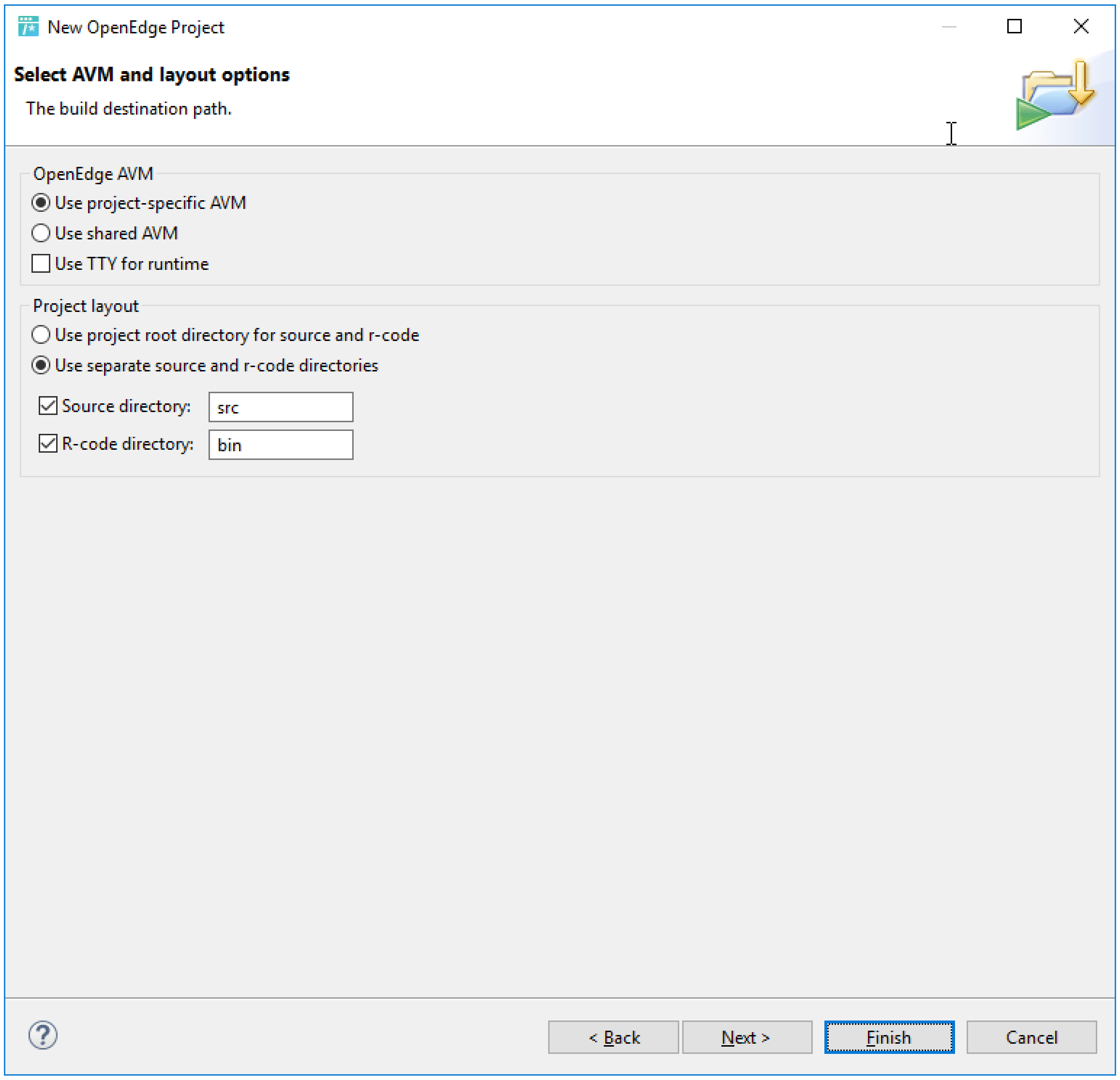
Location: C:\progress_education\OpenEdge\IntroOOP\workspace\Test

Select a category from the left section. If you do not select a category and click Finish, the default Basic OpenEdge project is created.

c. Click **Next**.

d. Select **Use separate source and r-code directories**.

e. Change **R-code directory** to *bin*.


New OpenEdge Project

Select AVM and layout options

The build destination path.

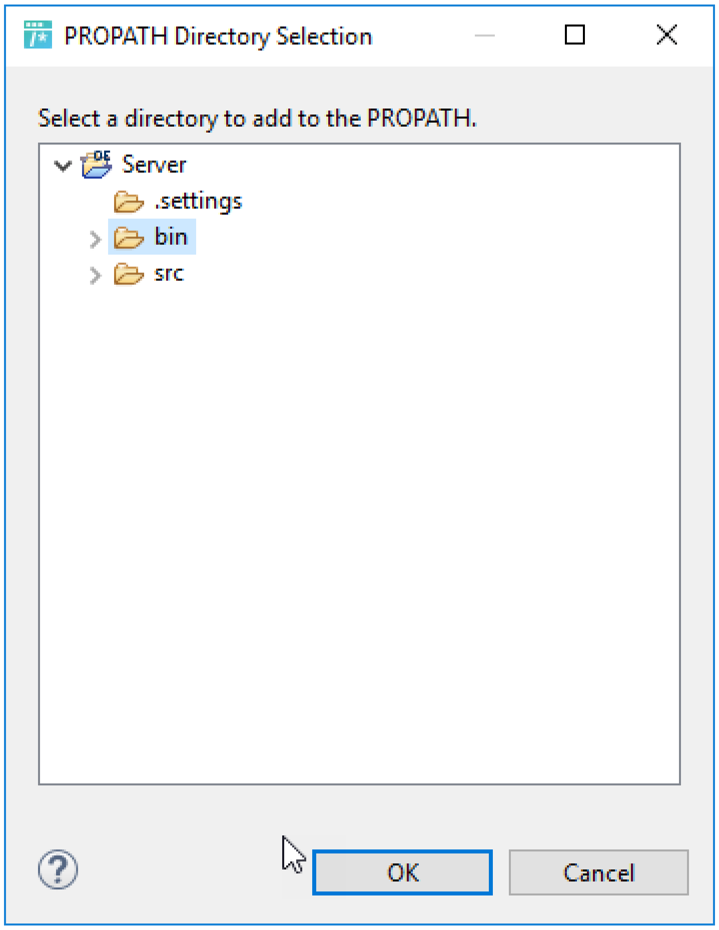
OpenEdge AVM
- [✓ Use project-specific AVM]
- Use shared AVM
- ☐ Use TTY for runtime

Project layout
- Use project root directory for source and r-code
- [✓ Use separate source and r-code directories]
- ☑ Source directory: src
- ☑ R-code directory: bin

f. Click **Next**.

g. Click **Add Workspace Directory**.

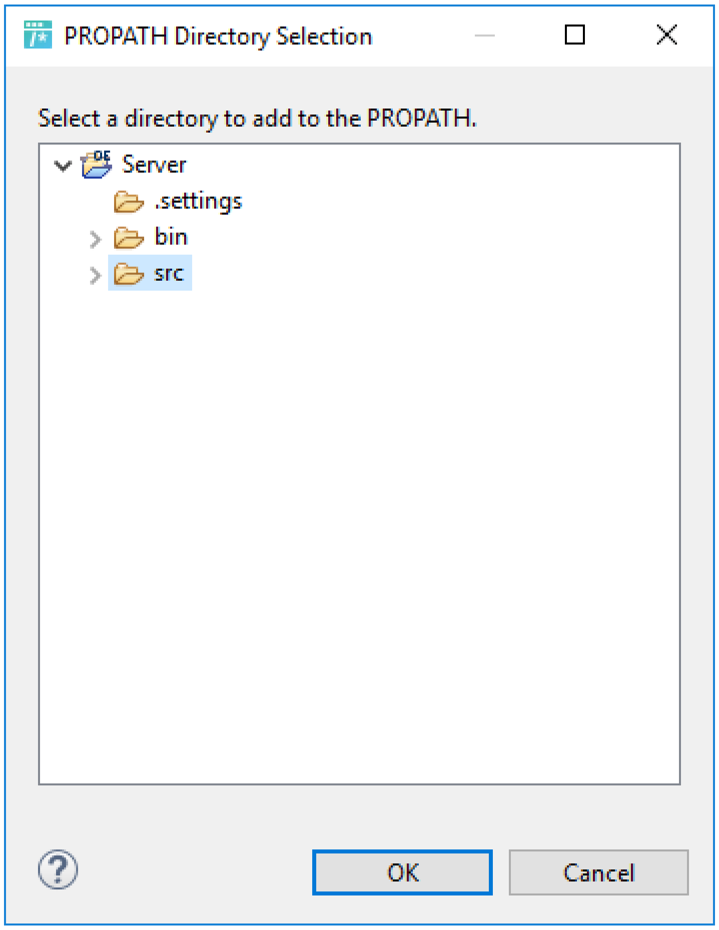
h. Select the **Server/bin** directory.


PROPATH Directory Selection
Select a directory to add to the PROPATH.
Server
.settings
bin
src
OK
Cancel

i. Click **OK**.

j. Click **Add Workspace Directory**.

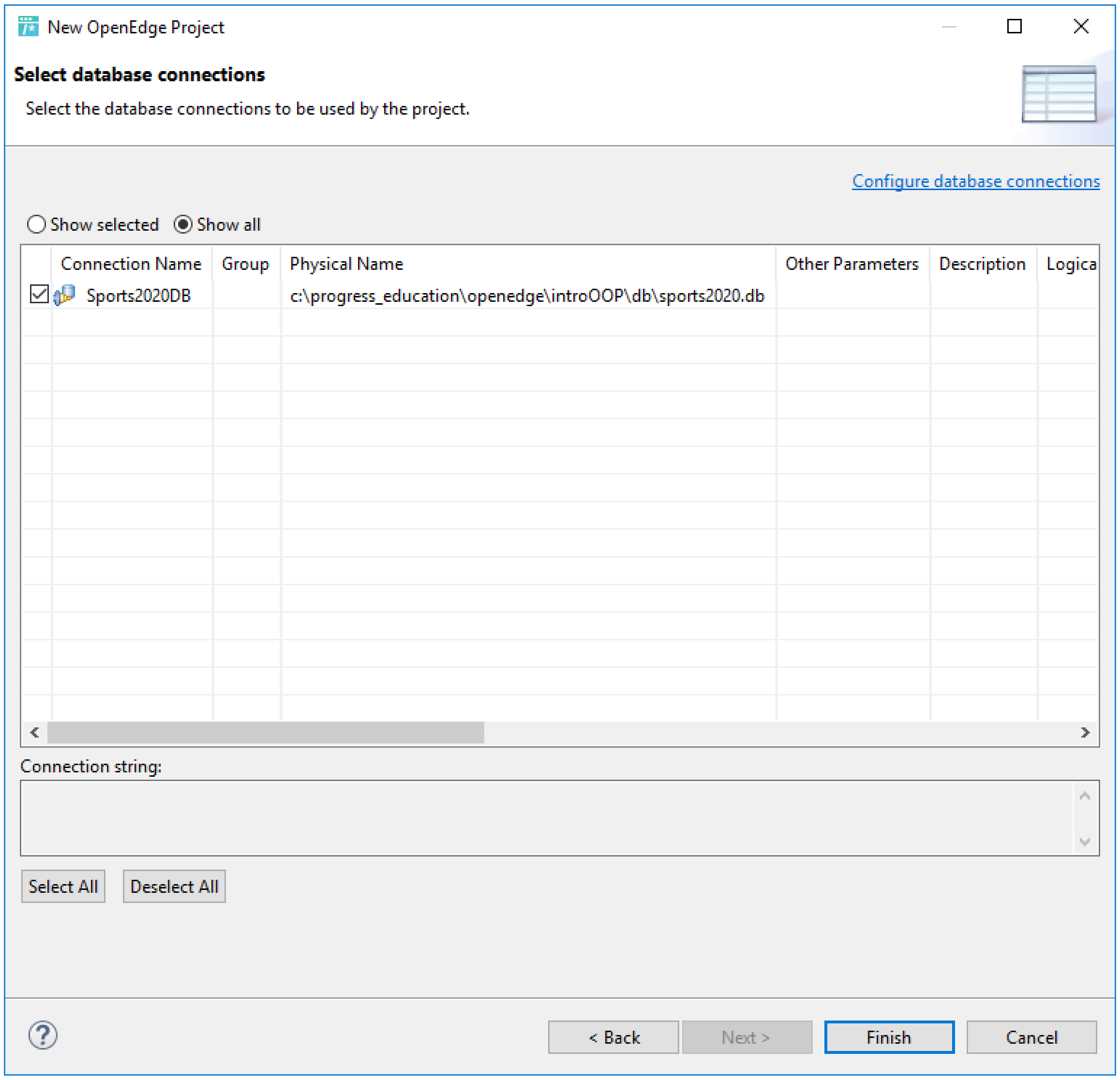
k. Select the **Server/src** directory.


PROPATH Directory Selection
Select a directory to add to the PROPATH.
Server
.settings
bin
src
OK
Cancel

l. Click **OK**.

m. Click **Next**.

n. Check the **Sports2020DB** database connection checkbox.



Select database connections

Select the database connections to be used by the project.

Configure database connections

( ) Show selected (●) Show all

	Connection Name	Group	Physical Name	Other Parameters	Description	Logica
✓	Sports2020DB		c:\progress_education\openedge\introOOP\db\sports2020.db			

Connection string:

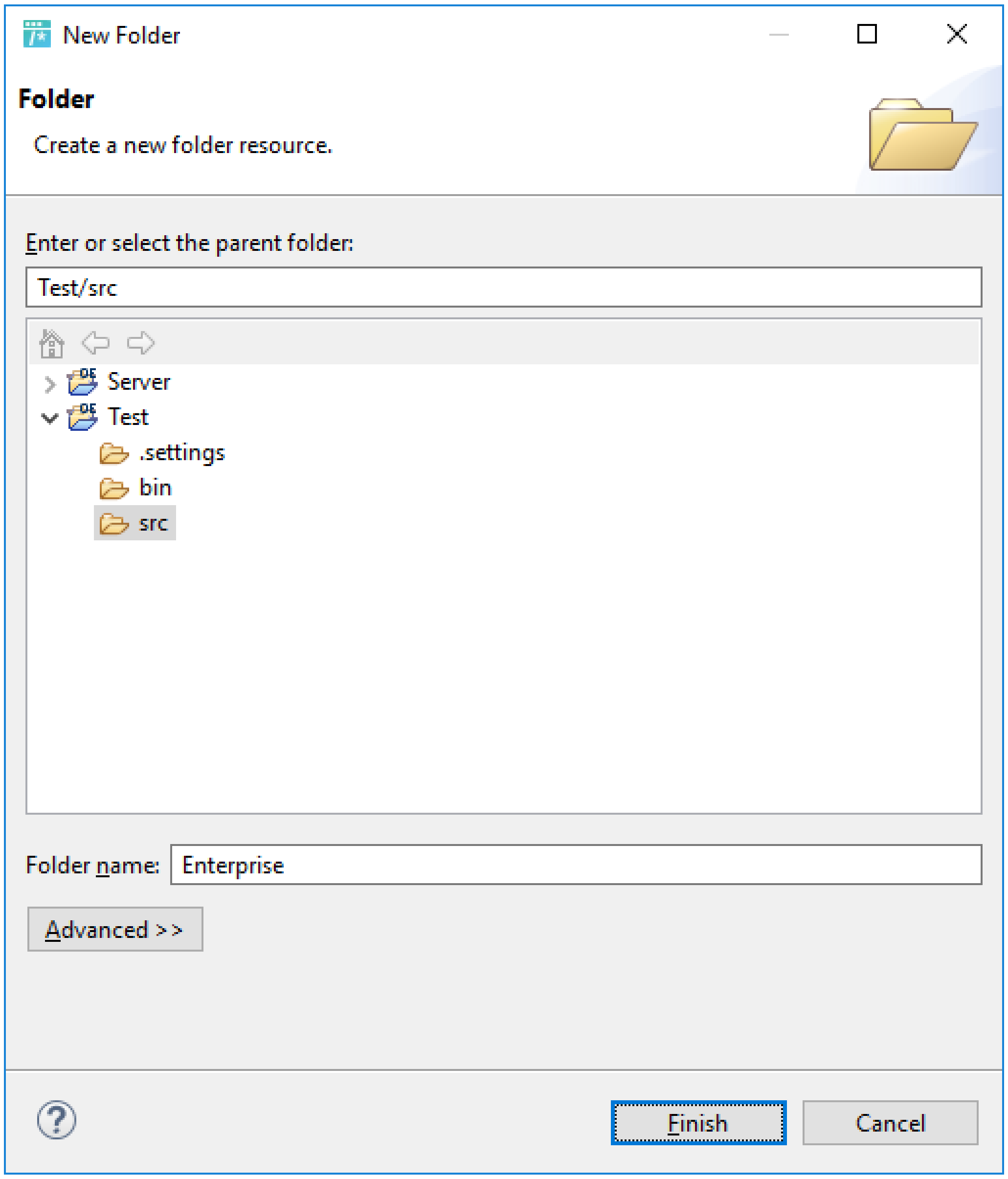
o. Click **Finish**.

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**.


New Folder

Folder

Create a new folder resource.

Enter or select the parent folder:

Test/src

Folder name: Enterprise

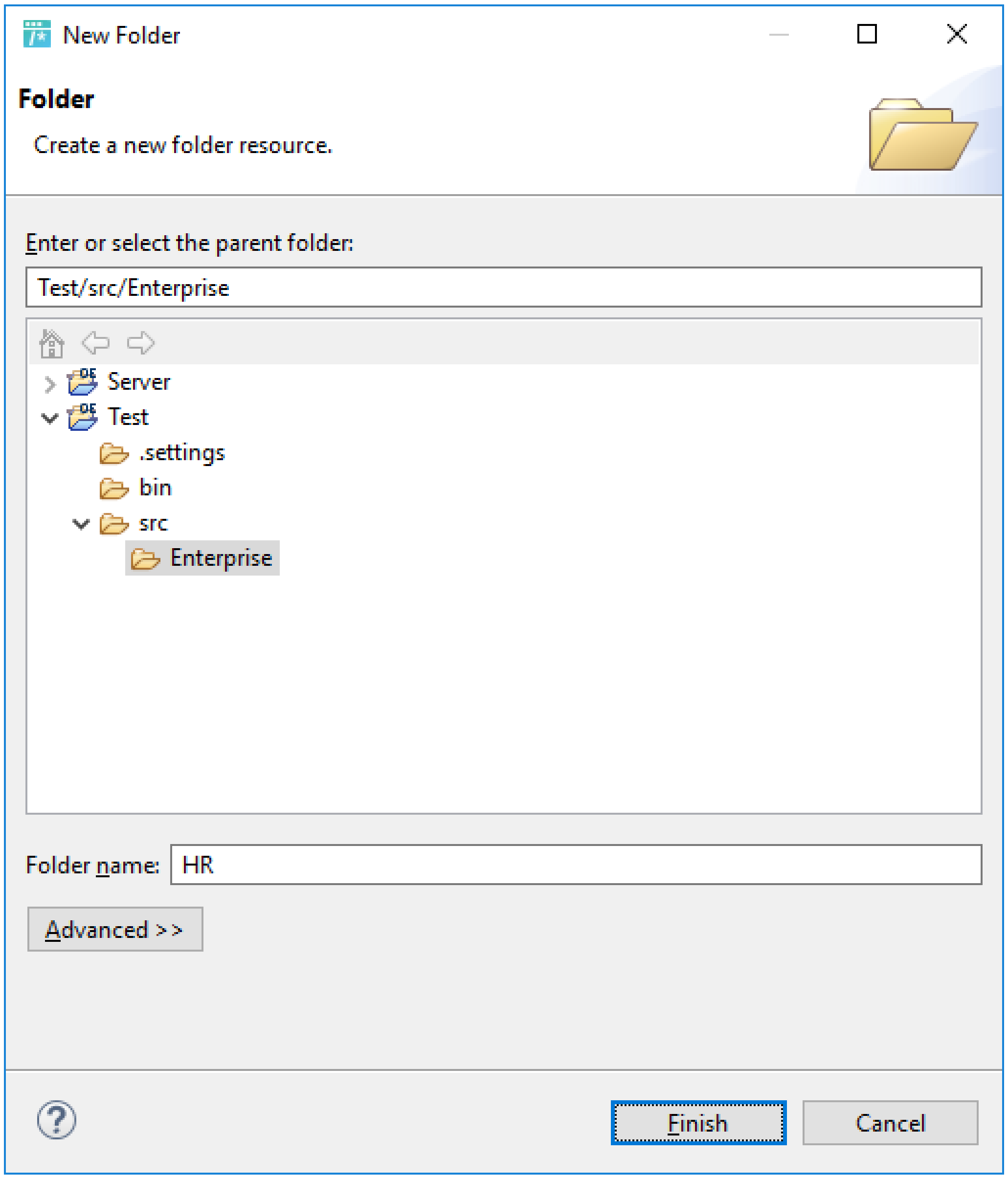
d. Click **Finish**.

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**.



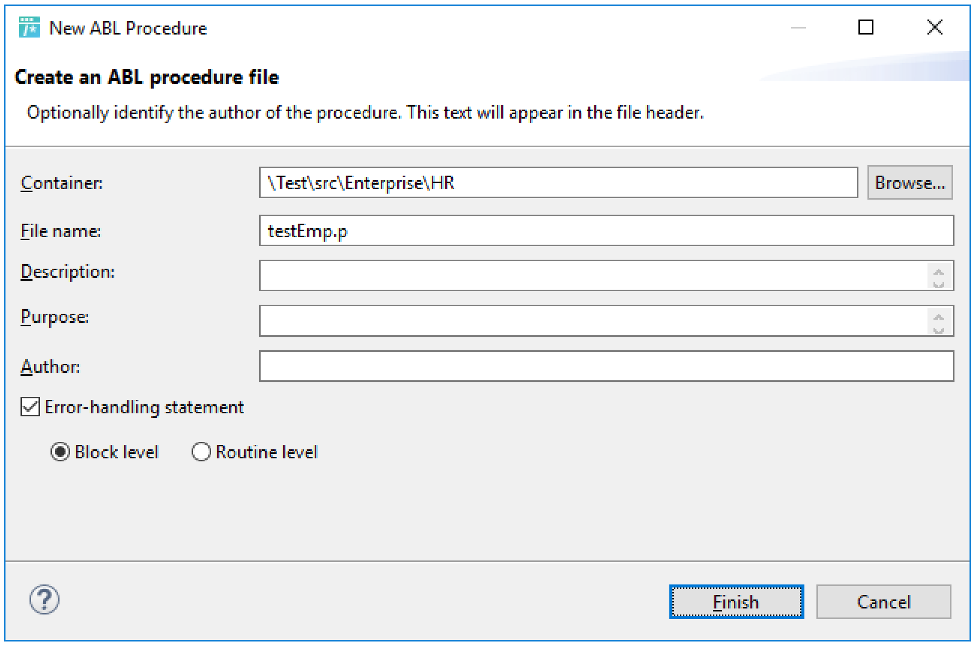
Enter or select the parent folder: Test/src/Enterprise

Folder name: HR

d. Click **Finish**.

Part 2: Write the test procedure for the Emp class

1. In the **Enterprise/HR** directory of the **Test** project, create a procedure named *testEmp.p*.
   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**.



New ABL Procedure

Create an ABL procedure file

Optionally identify the author of the procedure. This text will appear in the file header.

Container: \Test\src\Enterprise\HR

File name: testEmp.p

Description:

Purpose:

Author:

[✓] Error-handling statement

(●) Block level ( ) Routine level

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* that will hold a reference to an instance of *Emp*.

   Then add this statement in the definitions section:

```
define variable EmpInstance as Emp no-undo.
```

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 `testEmp.out`. It will be located in the `/progress_education/openedge/IntroOOP/log` directory.

Tip: Use the full pathname of 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").
```

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:

```
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:

```
/* 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 */
delete object EmpInstance.
```

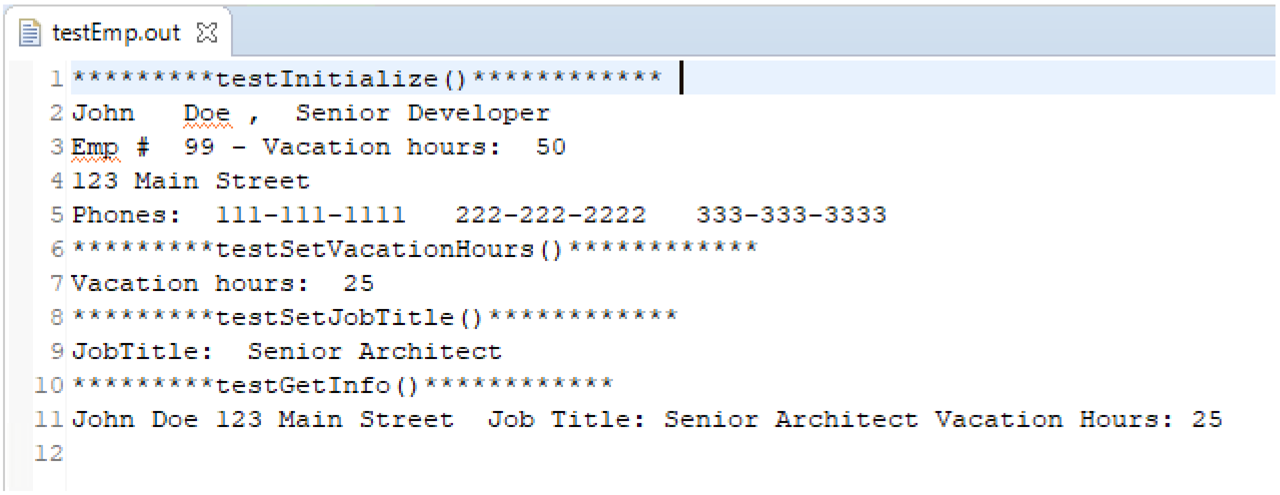
16. Save this file. Ensure that it compiles without errors.

Part 3: Test the Emp class

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 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



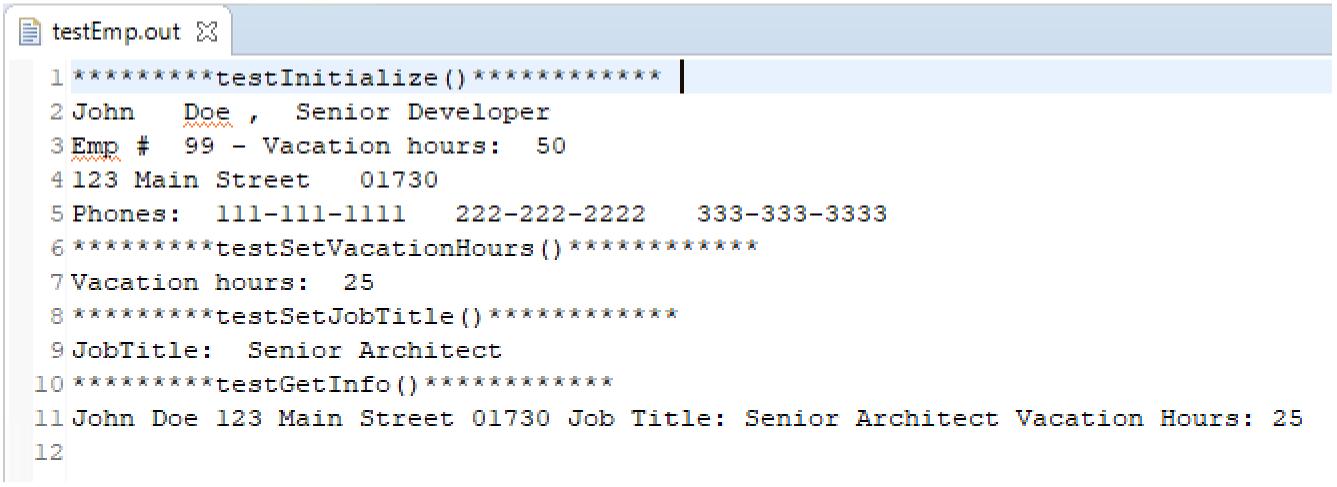
```
*********testInitialize()************
John   Doe ,  Senior Developer
Emp #  99 - Vacation hours:  50
123 Main Street
Phones:  111-111-1111   222-222-2222   333-333-3333
*********testSetVacationHours()************
Vacation hours:  25
*********testSetJobTitle()************
JobTitle:  Senior Architect
*********testGetInfo()************
John Doe 123 Main Street  Job Title: Senior Architect Vacation Hours: 25
```

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.

   Add the bold text to this property:

```
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.



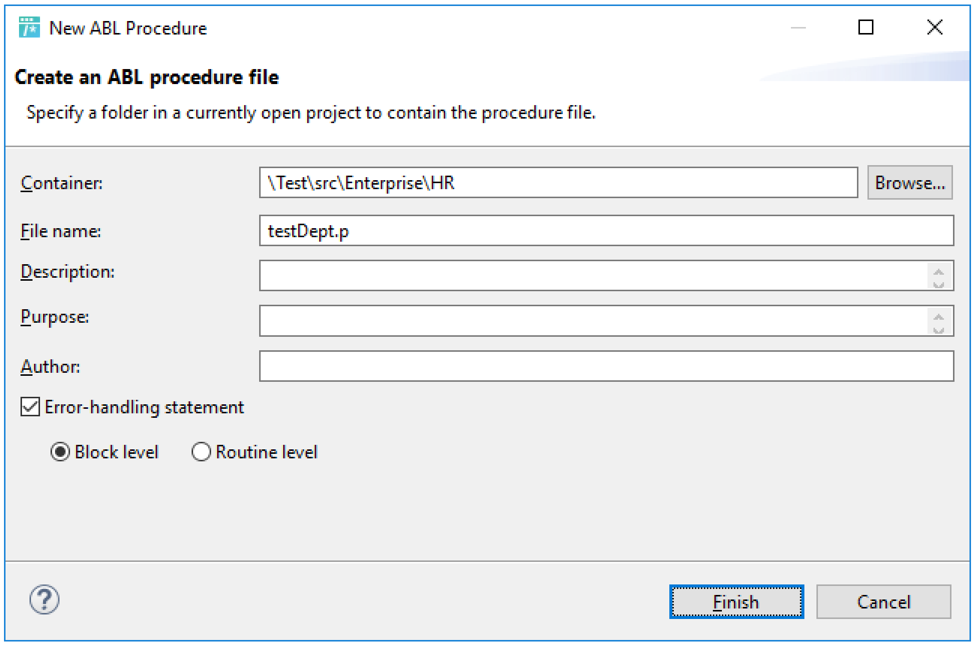
testEmp.out

```
*********testInitialize()************
John   Doe ,  Senior Developer
Emp #  99 - Vacation hours:  50
123 Main Street   01730
Phones:  111-111-1111   222-222-2222   333-333-3333
*********testSetVacationHours()************
Vacation hours:  25
*********testSetJobTitle()************
JobTitle:  Senior Architect
*********testGetInfo()************
John Doe 123 Main Street 01730 Job Title: Senior Architect Vacation Hours: 25

```

Part 4: Write 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.

1. In the **Test/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.


New ABL Procedure

Create an ABL procedure file

Specify a folder in a currently open project to contain the procedure file.

Container: \Test\src\Enterprise\HR

File name: testDept.p

Description:

Purpose:

Author:

[✓] Error-handling statement

[● Block level] ○ Routine level

e. Click **Finish**.

f. 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:

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

4. Define the following variables in 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.

   Tip: 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 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.

   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").
   ```

9. 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. 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. In the FOR EACH block, initialize the *Emp* instance by calling the *Initialize()* method, passing values from the current Employee buffer.

    Add this code as the next statement in the block:

```
EmpInstance:Initialize(Employee.EmpNum,
                           Employee.FirstName,
                           Employee.LastName,
                           Employee.Address,
                           Employee.PostalCode,
                           Phones,
                           Employee.VacationDaysLeft * 8,
                           Employee.Position).
```

14. In the FOR EACH block, add the created *Emp* instance to the *DeptInstance*.

    Tip: 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.

    Tip: 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:

```
assign
    Phones[1] = "111-111-1111"
    Phones[2] = "222-222-2222"
    Phones[3] = "333-333-3333".
```

16. 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.

    Tip: 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").
```

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.

Tip: 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:

```
/* database test case */
retrievedEmp = DeptInstance:GetEmployee("Luke", "Sanders").
if valid-object(retrievedEmp)
    then message "************testGetEmployee( ) for Luke Sanders ************"
         retrievedEmp:GetInfo().

/* manual test case */
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.

Tip: You will need to use the cast function to call 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.
```

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:

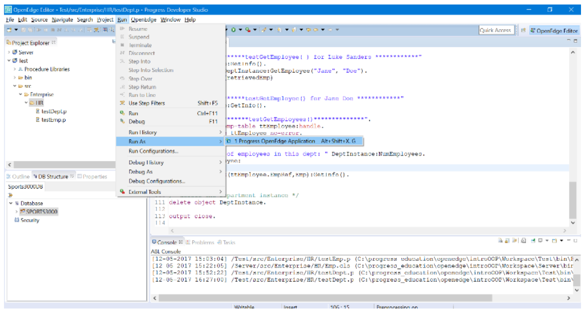
```
output close.
```

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

Part 5: Test the Dept class

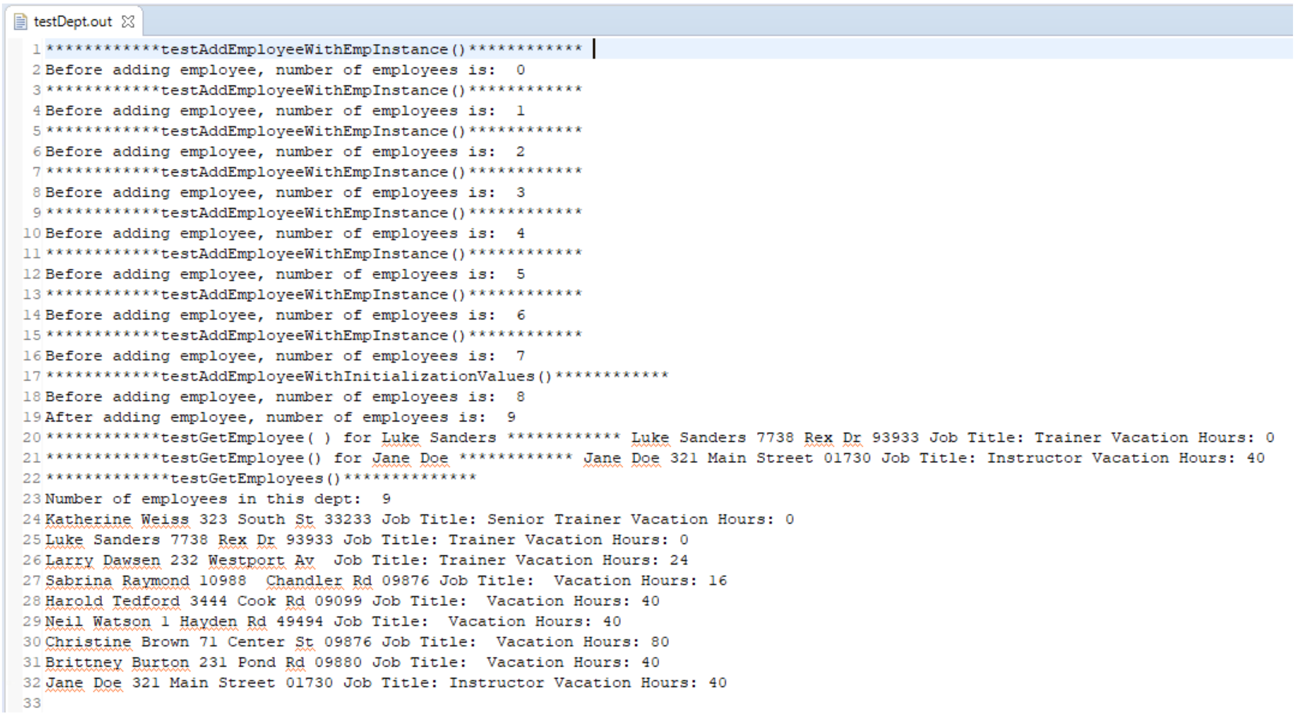
1. Run **testDept.p**. Does it run correctly and does the *Dept* class execute properly? View the output file.

   a. With **testDept.p** open in the **editor**, select **Run** > **Run as** > **Progress OpenEdge Application** to run *testDept.p*.



   b. Click **OK**.

2. Run *testDept.p*. Does it run correctly? If not, use the debugger to fix the problem, save your changes, and re-test until it runs correctly.



testDept.out

```
************testAddEmployeeWithEmpInstance()************
Before adding employee, number of employees is:  0
************testAddEmployeeWithEmpInstance()************
Before adding employee, number of employees is:  1
************testAddEmployeeWithEmpInstance()************
Before adding employee, number of employees is:  2
************testAddEmployeeWithEmpInstance()************
Before adding employee, number of employees is:  3
************testAddEmployeeWithEmpInstance()************
Before adding employee, number of employees is:  4
************testAddEmployeeWithEmpInstance()************
Before adding employee, number of employees is:  5
************testAddEmployeeWithEmpInstance()************
Before adding employee, number of employees is:  6
************testAddEmployeeWithEmpInstance()************
Before adding employee, number of employees is:  7
************testAddEmployeeWithInitializationValues()************
Before adding employee, number of employees is:  8
After adding employee, number of employees is:  9
************testGetEmployee( ) for Luke Sanders ************ Luke Sanders 7738 Rex Dr 93933 Job Title: Trainer Vacation Hours: 0
************testGetEmployee() for Jane Doe ************ Jane Doe 321 Main Street 01730 Job Title: Instructor Vacation Hours: 40
*************testGetEmployees()*************
Number of employees in this dept:  9
Katherine Weiss 323 South St 33233 Job Title: Senior Trainer Vacation Hours: 0
Luke Sanders 7738 Rex Dr 93933 Job Title: Trainer Vacation Hours: 0
Larry Dawsen 232 Westport Av  Job Title: Trainer Vacation Hours: 24
Sabrina Raymond 10988  Chandler Rd 09876 Job Title:  Vacation Hours: 16
Harold Tedford 3444 Cook Rd 09099 Job Title:  Vacation Hours: 40
Neil Watson 1 Hayden Rd 49494 Job Title:  Vacation Hours: 40
Christine Brown 71 Center St 09876 Job Title:  Vacation Hours: 80
Brittney Burton 231 Pond Rd 09880 Job Title:  Vacation Hours: 40
Jane Doe 321 Main Street 01730 Job Title: Instructor Vacation Hours: 40
```

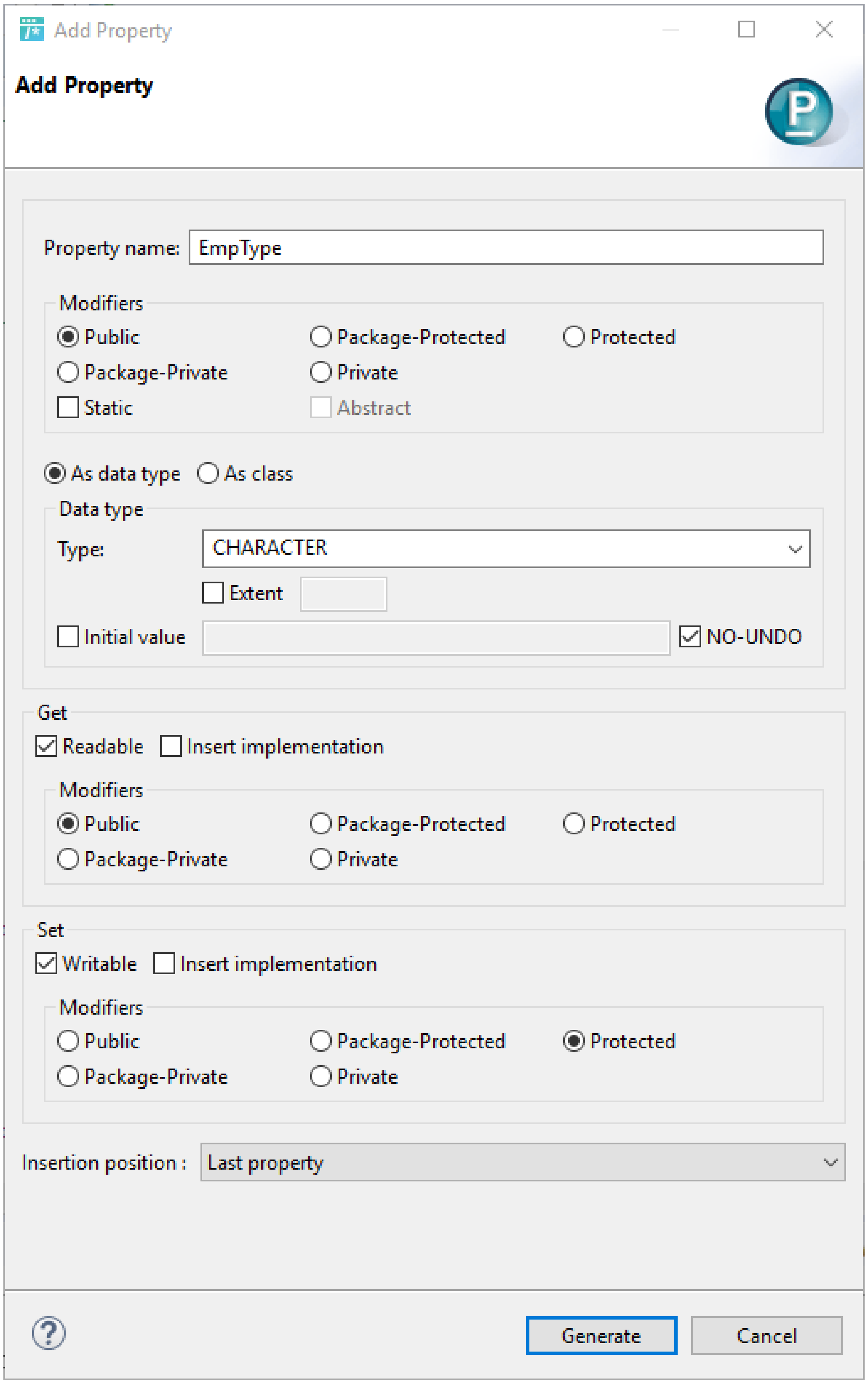
Solution: Try It 4.1, Use inheritance

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

1. At the end of the definition section of the *Emp* class, define the *EmpType* public property as a character with a protected setter.

   Tip: 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 modifiers**.
   f. Select *Protected* for the **Set modifiers**.
   g. Select *Last Property* for the **Insertion position**.



Add Property

Add Property

Property name: EmpType

Modifiers
- (●) Public ( ) Package-Protected ( ) Protected
- ( ) Package-Private ( ) Private
- [ ] Static [ ] Abstract

(●) As data type ( ) As class

Data type
- Type: CHARACTER
- [ ] Extent
- [ ] Initial value
- [✓] NO-UNDO

Get
- [✓] Readable [ ] Insert implementation
- Modifiers
  - (●) Public ( ) Package-Protected ( ) Protected
  - ( ) Package-Private ( ) Private

Set
- [✓] Writable [ ] Insert implementation
- Modifiers
  - ( ) Public ( ) Package-Protected (●) Protected
  - ( ) Package-Private ( ) Private

Insertion position : Last property

h. Click **Generate**.

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

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

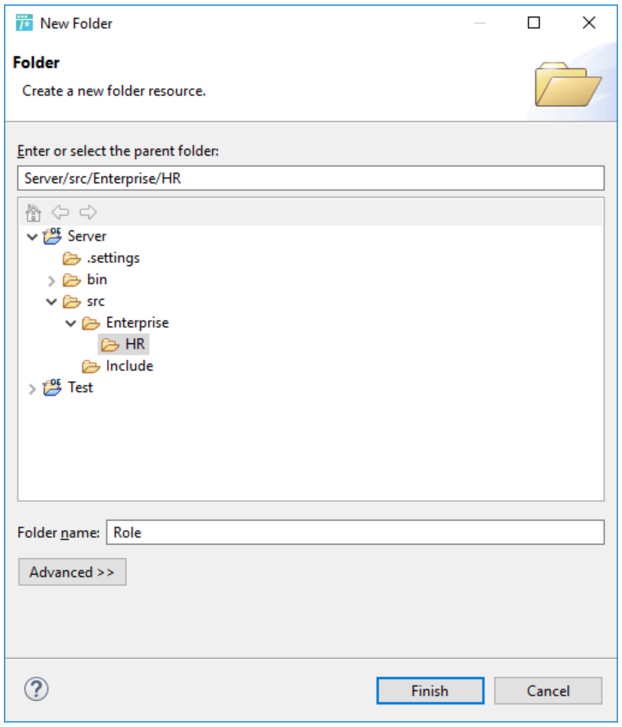
2. Change the *GetName()* method to *protected*.

```
method protected character GetName(  ):
```

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

Part 2: Create the Manager class

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**.


New Folder

Folder

Create a new folder resource.

Enter or select the parent folder:

Server/src/Enterprise/HR

Folder name: Role

   d. Click **Finish**.

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.

Tip: Use the **New ABL class** wizard.

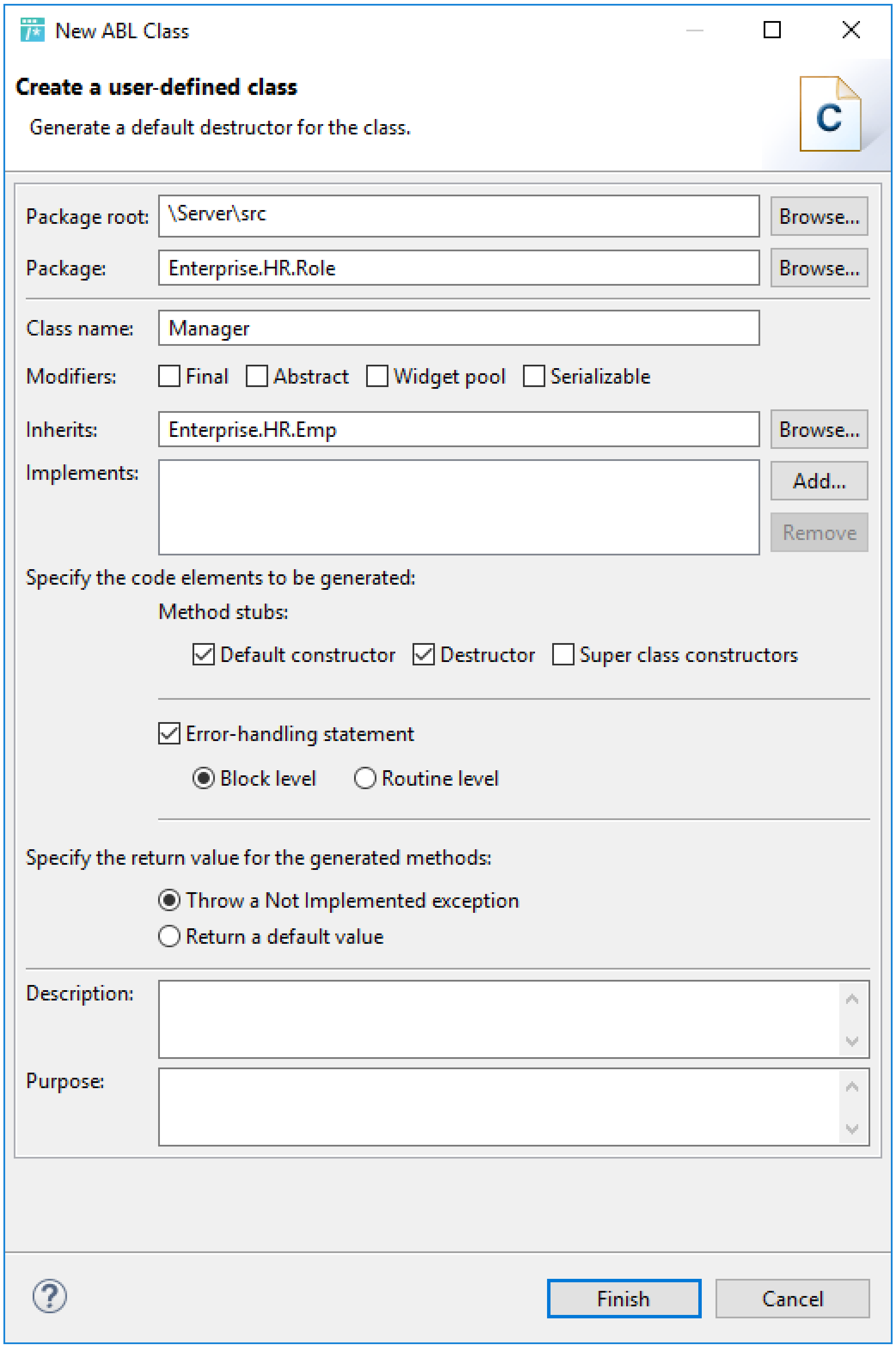
   a. Right-click **Role**.
   b. Select **New** > **ABL Class**.
   c. Enter *Manager* for the **Class name**.

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**.


New ABL Class

Create a user-defined class

Generate a default destructor for the class.

Package root: \Server\src

Package: Enterprise.HR.Role

Class name: Manager

Modifiers: ☐ Final ☐ Abstract ☐ Widget pool ☐ Serializable

Inherits: Enterprise.HR.Emp

Implements:

Specify the code elements to be generated:

Method stubs:

☑ Default constructor ☑ Destructor ☐ Super class constructors

☑ Error-handling statement

● Block level ○ Routine level

Specify the return value for the generated methods:

● Throw a Not Implemented exception

○ Return a default value

Description:

Purpose:

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.
```

Note: The constructor includes the call of the constructor in the super class by including the `super()` statement.

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.

   a. Add this statement at the beginning of the file before the error-handling statement:

```
using Enterprise.HR.Role.TeamMember.
```

Part 3: Define a constructor and a data member for the Manager class

1. Modify the default constructor to set the *EmpType property* to be *Manager*.

   a. Place your code after the `super ()` statement in the `constructor` for the `Manager` class.

```
constructor public Manager( ):
  super ().
  EmpType = "Manager".
end constructor.
```

2. Define the *private* temp-table by including the *ttEmployee* temp-table definition.

Tip: Add this statement in the definitions part of the class.

   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 (except for the error related to the reference to *TeamMember*).

Part 4: Define methods for the Manager class

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.

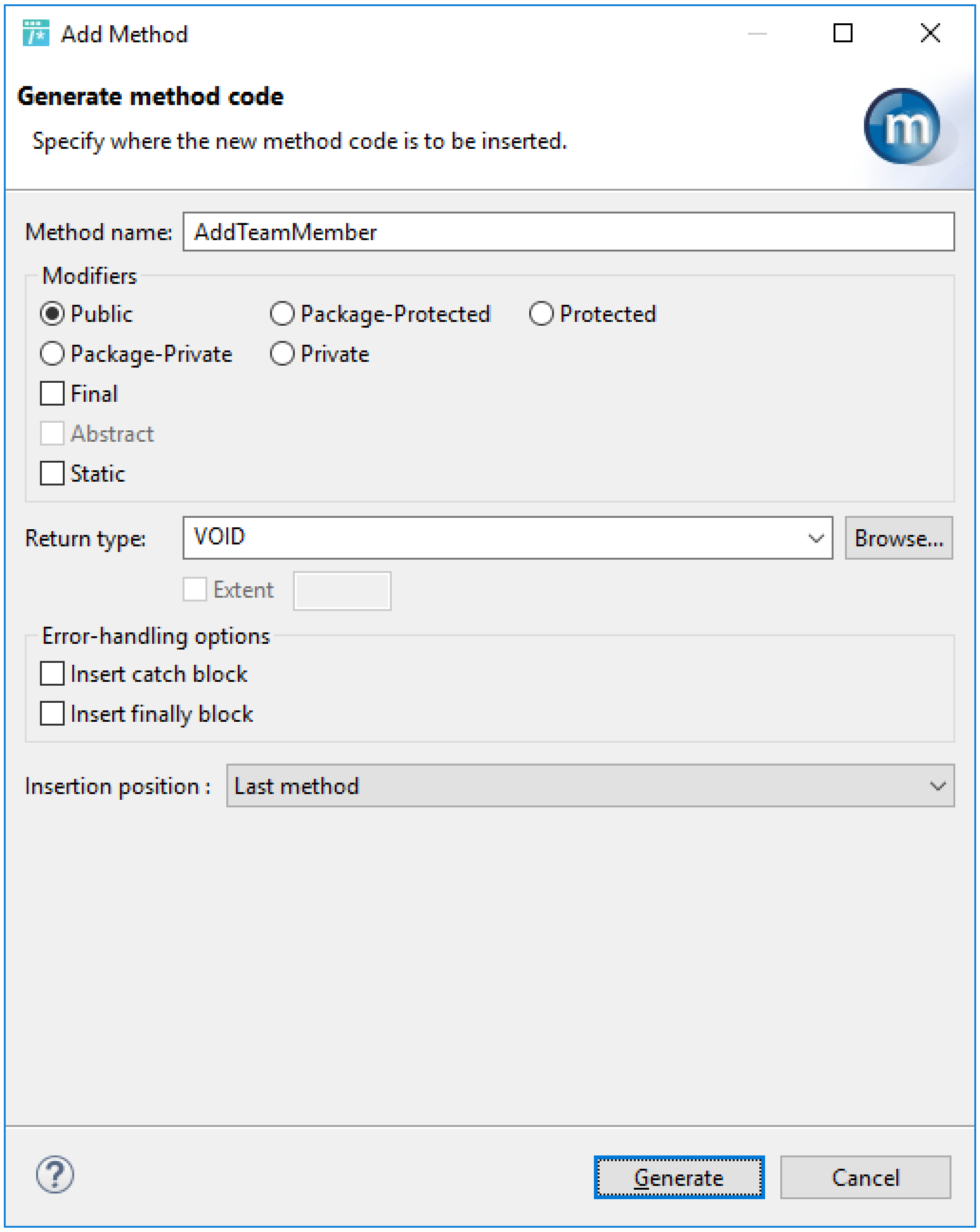
a. Place the cursor anywhere in the class.

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**.


Add Method

Generate method code

Specify where the new method code is to be inserted.

Method name: AddTeamMember

Modifiers

[✓ Public] Package-Protected Protected

Package-Private Private

☐ Final

☐ Abstract

☐ Static

Return type: VOID

☐ Extent

Error-handling options

☐ Insert catch block

☐ Insert finally block

Insertion position : Last method

f. Click **Generate**.

g. Enter the input parameter as *ptm* with a type *TeamMember*.

   Your code should appear as follows:

```
method public void AddTeamMember(input ptm as TeamMember)
 return.
end method.
```

2. Define the public *GetManagersEmployees()* method that has an output parameter, which is the *ttEmployee* table, and returns *void*.

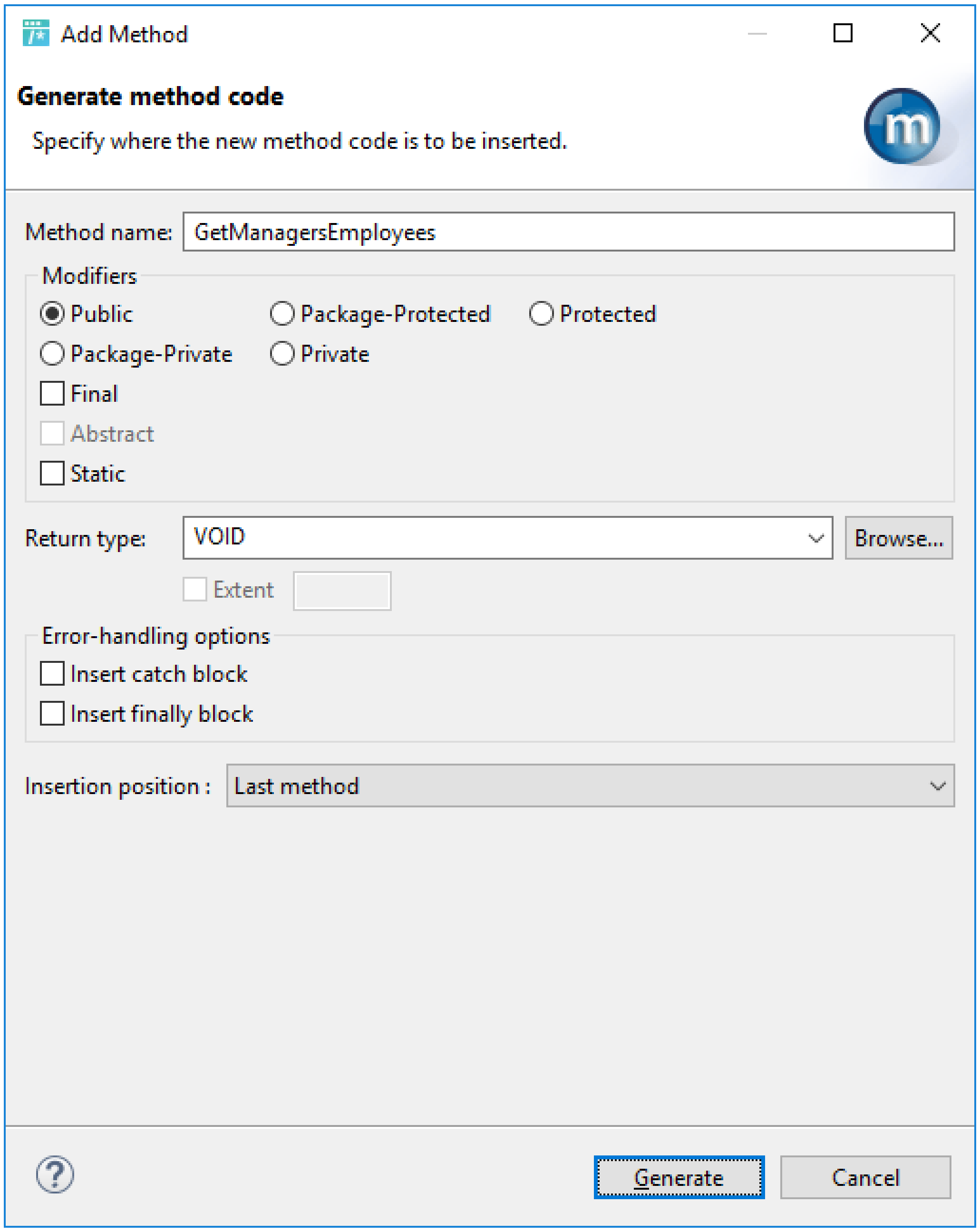
   a. Place the cursor anywhere in the class.

   b. Right-click and then select **Source** > **Add Method**.

   c. Enter *GetManagersEmployees* for the **Method name**.

   d. Select *VOID* as the **Return type**.

   e. Select *Last method* for the **Insertion position**.



Generate method code

Specify where the new method code is to be inserted.

Method name: GetManagersEmployees

Modifiers

[✓ Public] Package-Protected Protected

Package-Private Private

Final

Abstract

Static

Return type: VOID

Extent

Error-handling options

Insert catch block

Insert finally block

Insertion position : Last method

f. Click **Generate**.

g. Enter the output parameter as table *ttEmployee*.

Your code should appear as follows:

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

3. Define the *public* method *GetInfo()*, which returns *character* and takes no parameters. It will override the method in the *Emp* class.

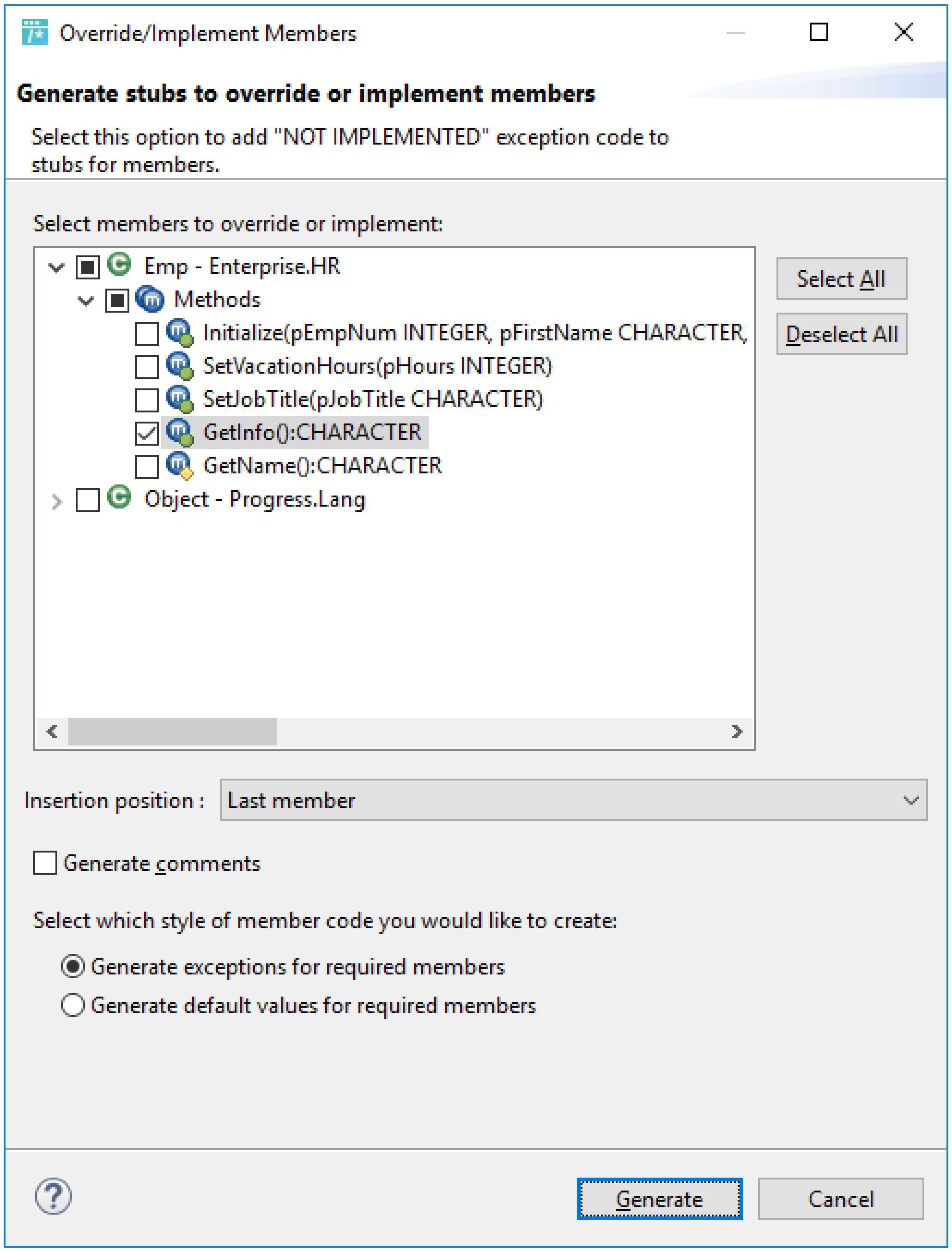
Tip: Use the Override/Implement Members wizard.

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():CHARACTER** method in the **Emp - Enterprise.HR** class.

d. Select *Last member* for the **Insertion position**.


Override/Implement Members

Generate stubs to override or implement members

Select this option to add "NOT IMPLEMENTED" exception code to stubs for members.

Select members to override or implement:

- [■] Emp - Enterprise.HR
  - [■] Methods
    - [ ] Initialize(pEmpNum INTEGER, pFirstName CHARACTER,
    - [ ] SetVacationHours(pHours INTEGER)
    - [ ] SetJobTitle(pJobTitle CHARACTER)
    - [✓] GetInfo():CHARACTER
    - [ ] GetName():CHARACTER
- [ ] Object - Progress.Lang

Insertion position : Last member

[ ] Generate comments

Select which style of member code you would like to create:

(●) Generate exceptions for required members
( ) Generate default values for required members

e. Click **Generate**.

The code should appear as follows:

```
method override public character GetInfo():
   return super:GetInfo().
   end method.
```

4. Save your file. Ensure that it compiles without errors. (except for the errors related to the *TeamMember* reference)

Part 5: Implement the methods of the Manager class

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.

   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()* 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*.
   - Replace the return statement `return super:GetInfo().` with *return result*.

     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 for the class to delete all *ttEmployee* records.

   This method should now appear as follows:

```
destructor public Manager ( ):
  for each ttEmployee:
    delete ttEmployee.
  end.

end destructor.
```

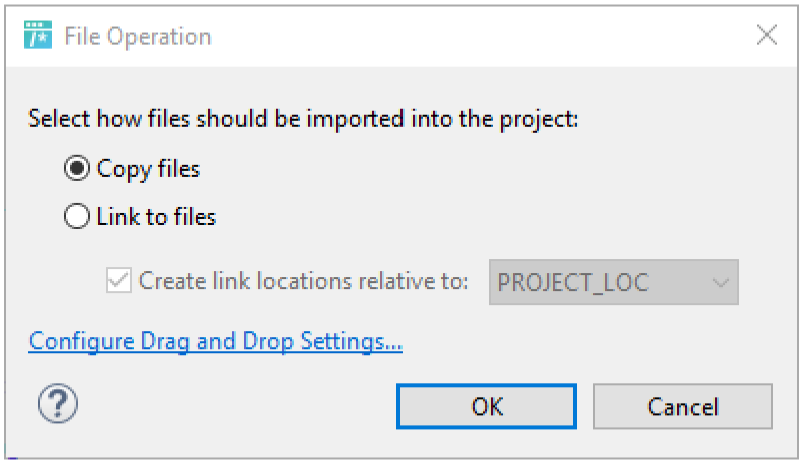
4. Note that you need not implement the *GetManagersEmployees()* method. ABL will automatically return the *ttEmployee* temp-table. Save your file. It will have compilation errors because the *TeamMember* class has not yet been created. If you see other syntax errors not related to *TeamMember*, correct them.

Part 6: Import the TeamMember and Dept classes

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/Exercises/Lesson04` directory and select the `TeamMember.cls` file.

     a. Drag and drop the `TeamMember.cls` file into the **Role** folder.

     b. Ensure that **Copy files** is selected.


File Operation

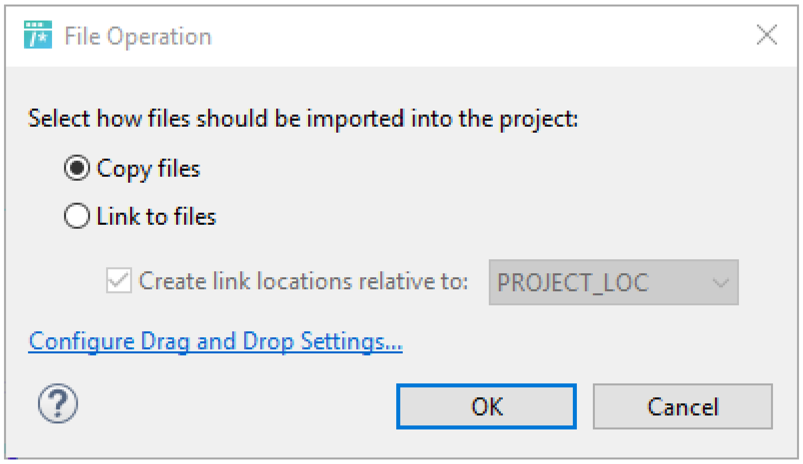
Select how files should be imported into the project:

(●) Copy files
( ) Link to files

[✓] Create link locations relative to: PROJECT_LOC

   - b. Click **OK**.

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.
   - a. In **Windows Explorer**, navigate to the `/progress_education/OpenEdge/introOOP/Exercises/Lesson04` 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.

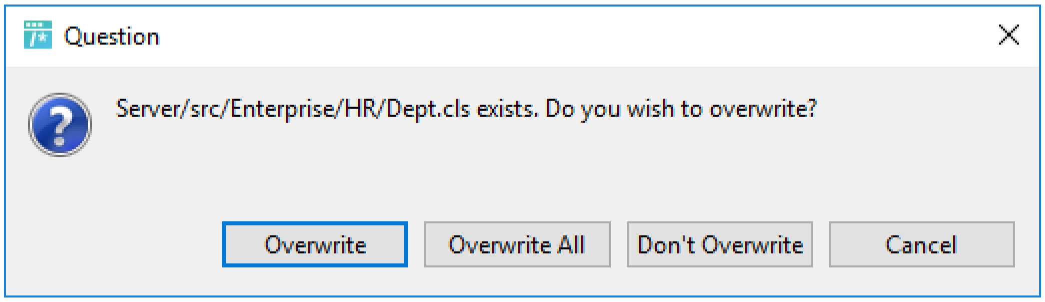

File Operation

Select how files should be imported into the project:

(●) Copy files
( ) Link to files

[✓] Create link locations relative to: PROJECT_LOC

   - d. Click **OK**.
   - e. When asked if you want to overwrite the `Dept.cls` file, respond **Overwrite**.

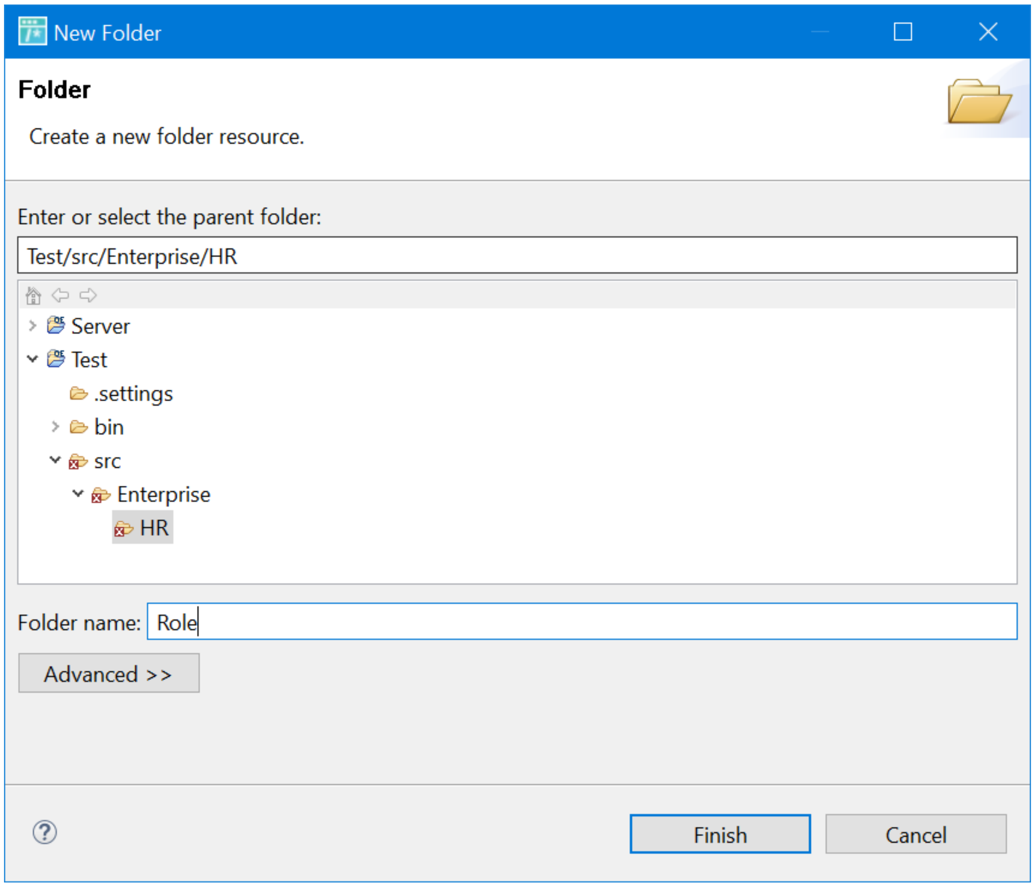

Server/src/Enterprise/HR/Dept.cls exists. Do you wish to overwrite?

4. Clean and rebuild the workspace. You should have no compilation errors.

Note: You will see an error in the **Test** project due to replacement of the `Dept.cls` file which can be ignored for now.

Part 7: Test the inheritance hierarchy

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**.



Folder

Create a new folder resource.

Enter or select the parent folder:

Test/src/Enterprise/HR

- Server
- Test
  - .settings
  - bin
  - src
    - Enterprise
      - HR

Folder name: Role

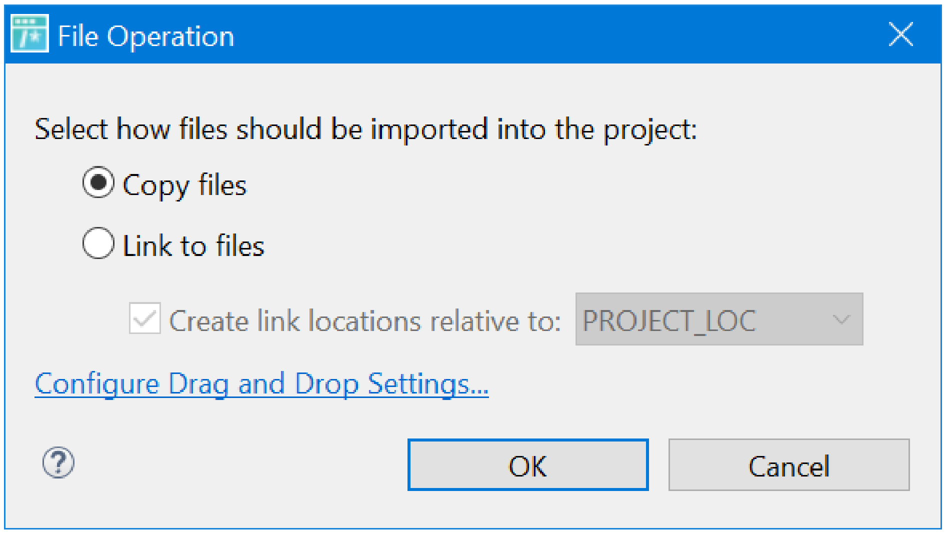
d. Click **Finish**.

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/Exercises/Lesson04` 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.



File Operation

Select how files should be imported into the project:

[✓ Copy files]

Link to files

Create link locations relative to: PROJECT_LOC

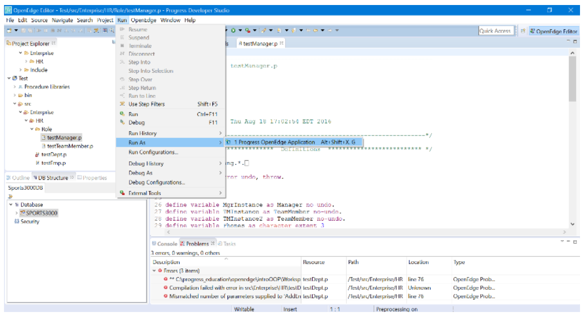
Configure Drag and Drop Settings...

OK Cancel

d. Click **OK**.

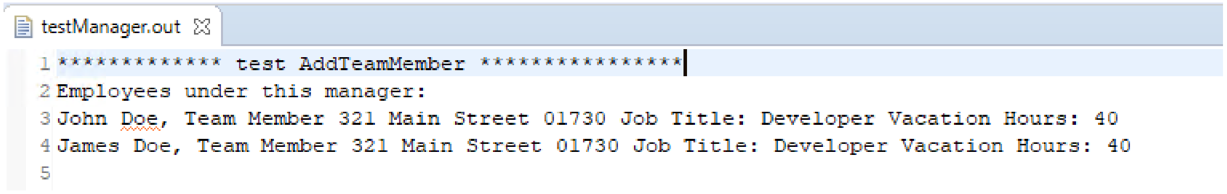
3. Run **testManager.p**. Does it run correctly and was the *Manager* information added to the output file? View the output file.

a. With *testManager.p* open in the editor, click the **Run** > **Run As** > **Progress OpenEdge Application**.



b. Click **OK**.

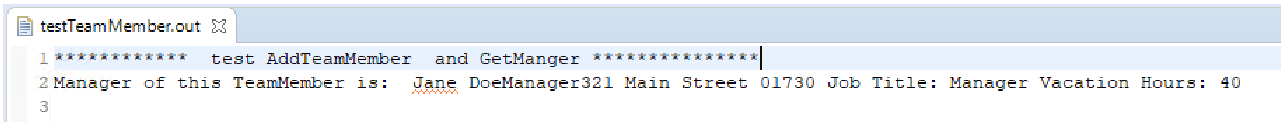
Note: You will receive a message that the project has errors. This is because the `testdept.p` file needs to be updated later. It does not impact the current test.



testManager.out

```
************** test AddTeamMember ****************
Employees under this manager:
John Doe, Team Member 321 Main Street 01730 Job Title: Developer Vacation Hours: 40
James Doe, Team Member 321 Main Street 01730 Job Title: Developer Vacation Hours: 40

```

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



testTeamMember.out

```
************ test AddTeamMember and GetManger ***************
Manager of this TeamMember is:  Jane DoeManager321 Main Street 01730 Job Title: Manager Vacation Hours: 40

```

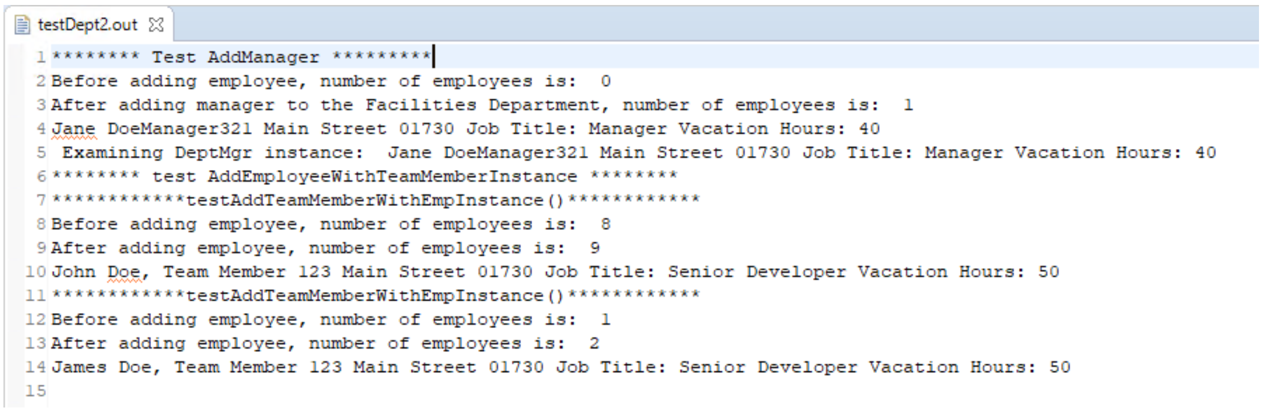
5. If the files did not run as expected, use the debugger to fix the problem, save your changes, and retest until it runs correctly.
6. In the **Test/Enterprise/HR** directory, import an ABL procedure file **testDept.p**. View the file to see how each method is defined for testing.

   Note: You will replace the existing **testDept.p** file.

   ---

   1. In **Windows Explorer**, navigate to the */progress_education/OpenEdge/introOOP/Exercises/Lesson04* directory and select the **testDept.p** file.
   2. Drag and drop the **testDept.p** file into the **HR** folder.
   3. Ensure that **Copy files** is selected.
   4. Reply **Overwrite** when asked if you want to overwrite 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 **Run As** > **Progress OpenEdge Application**.
   c. The output looks like this:



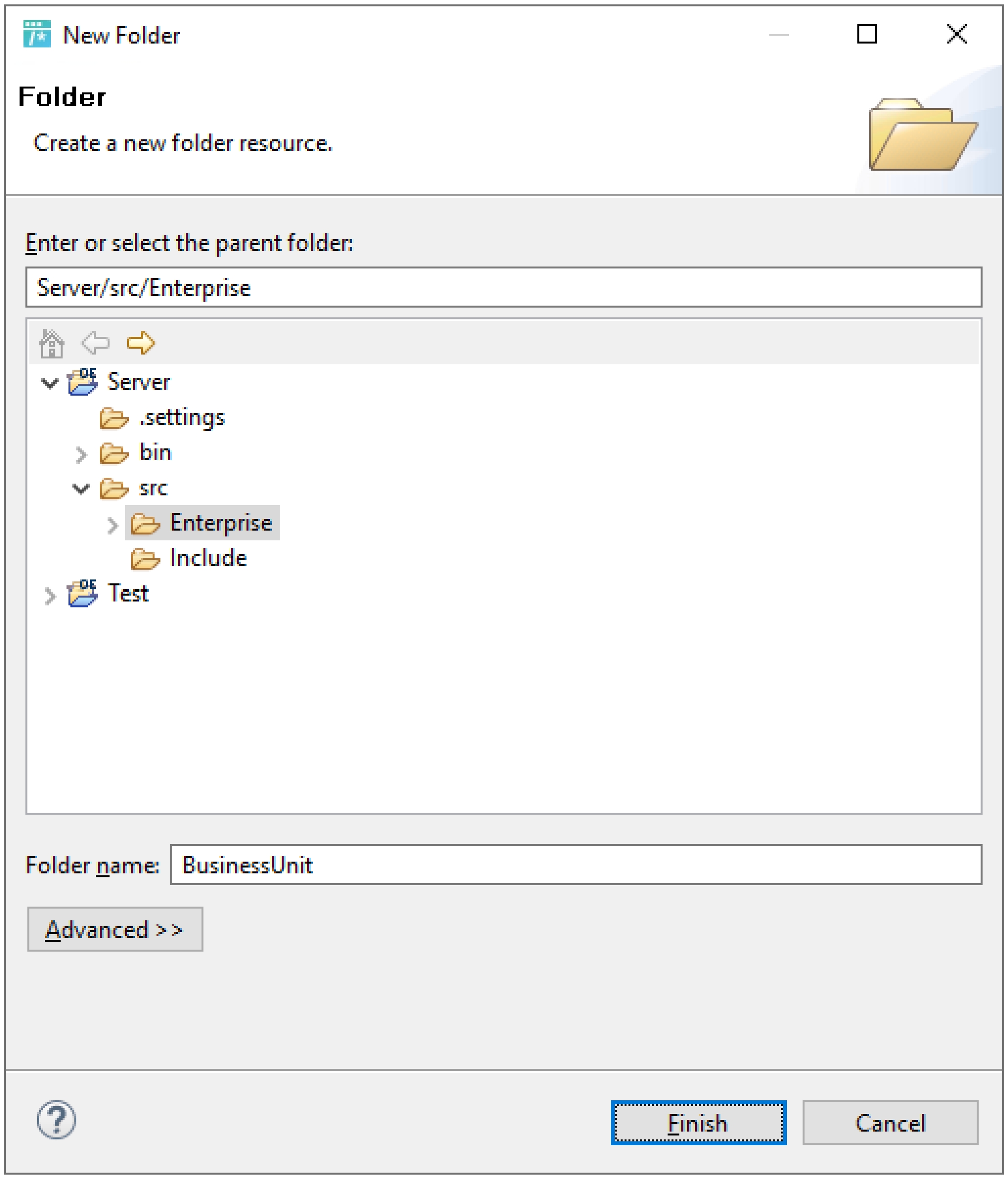
```
******** Test AddManager *********
Before adding employee, number of employees is:  0
After adding manager to the Facilities Department, number of employees is:  1
Jane DoeManager321 Main Street 01730 Job Title: Manager Vacation Hours: 40
 Examining DeptMgr instance:  Jane DoeManager321 Main Street 01730 Job Title: Manager Vacation Hours: 40
******** test AddEmployeeWithTeamMemberInstance ********
************testAddTeamMemberWithEmpInstance()************
Before adding employee, number of employees is:  8
After adding employee, number of employees is:  9
John Doe, Team Member 123 Main Street 01730 Job Title: Senior Developer Vacation Hours: 50
************testAddTeamMemberWithEmpInstance()************
Before adding employee, number of employees is:  1
After adding employee, number of employees is:  2
James Doe, Team Member 123 Main Street 01730 Job Title: Senior Developer Vacation Hours: 50

```

8. If it does not run correctly, use the debugger to fix the problem, save your changes, and retest until it runs correctly.

Solutions: Try it 5.1, Use an interface class

Part 1: Create the IBusiness Unit interface class

1. In **Project Explorer**, in the **Server** project, create a new folder named *BusinessUnit* under the **src/Enterprise** folder.
   a. Right-click **src** > **Enterprise** folder.
   b. Select **New** > **Folder**.
   c. Enter *BusinessUnit* for the **Folder name**.


New Folder

Folder

Create a new folder resource.

Enter or select the parent folder:

Server/src/Enterprise

Folder name: BusinessUnit

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 define a set of properties and methods for the *Company* and *Franchise* classes.

Tip: 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**.

f. The **IBusinessUnit.cls** file opens in the editor.

```
using Progress.Lang.*.

interface Enterprise.BusinessUnit.IBusinessUnit:
end interface.
```

4. 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.

interface Enterprise.BusinessUnit.IBusinessUnit:
end interface.
```

Part 2: Define data members for the IBusinessUnit interface class

Define the data members for the *IBusiness Unit* interface class.

1. Define these properties as *public* with a `get` but no `set` method:
   - *Name* as *character*
   - *NumDepartments* as *Integer*

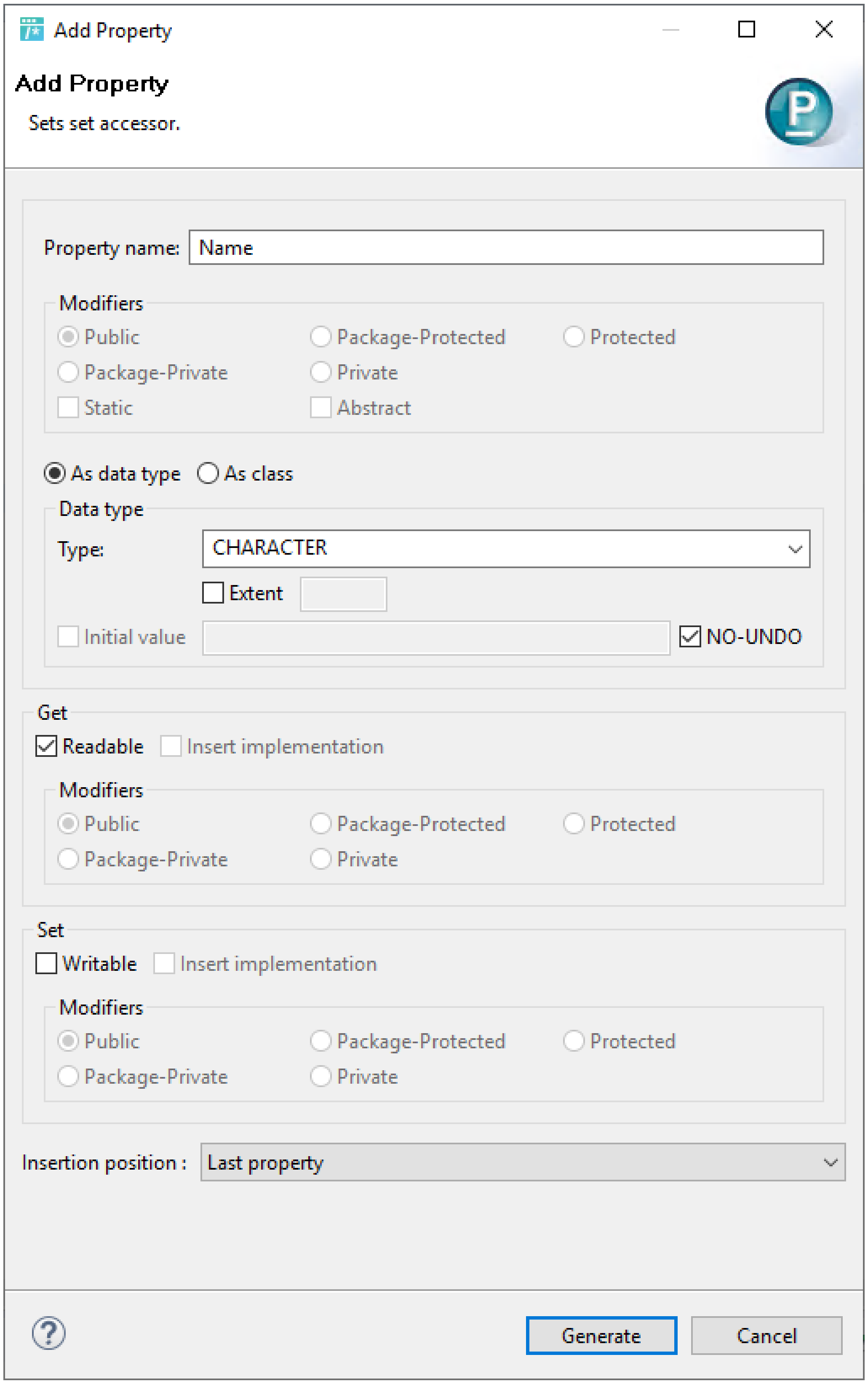
   a. Right-click anywhere in the *IBusinessUnit.cls* file

   b. Select **Source** > **Add Property**. The **Add Property** wizard opens.

   c. Enter *Name* for the **Property name**.

   d. Select the *CHARACTER* from the **Type** the drop-down list.

   e. Select *Last Property* for the **Insertion position**.



Add Property

Add Property

Sets set accessor.

Property name: Name

Modifiers

[✓ Public] Package-Protected Protected
Package-Private Private
Static Abstract

[✓ As data type] As class

Data type

Type: CHARACTER

Extent

Initial value

[✓ NO-UNDO]

Get

[✓ Readable] Insert implementation

Modifiers

[✓ Public] Package-Protected Protected
Package-Private Private

Set

Writable Insert implementation

Modifiers

[✓ Public] Package-Protected Protected
Package-Private Private

Insertion position : Last property

Generate Cancel

   f. Click **Generate**.
   g. Repeat these steps for the definition of the *NumDepartments* property, making sure you select the *integer* type.

      The generated code should appear as follows:

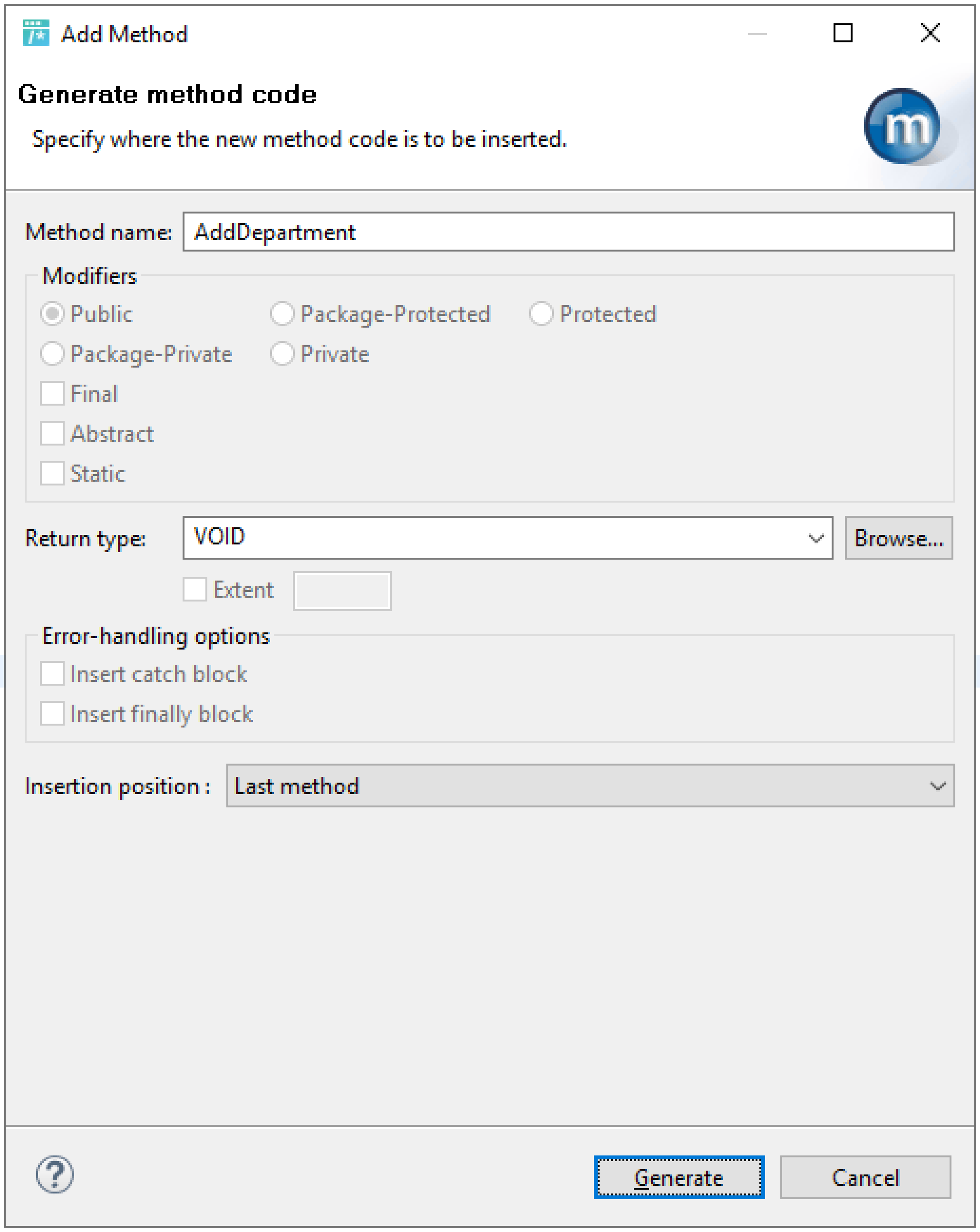
```
define public property Name as character no-undo
  get.
define public property NumDepartments as integer no-undo
  get.
```

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

Part 3: Define methods for the IBusinessUnit interface class

Define the methods for the *IBusiness Unit* interface class.

1. Define the public *AddDepartment()* method with return type *void*. The input parameter for this method will be *pDept* of type *Dept*.
   a. Place the cursor anywhere in the file.
   b. Right-click and select **Source** > **Add Method**.
   c. Enter *AddDepartment* as the **Method name**.



Method name: AddDepartment

Modifiers: [✓ Public] Package-Protected Protected Package-Private Private
Final
Abstract
Static

Return type: VOID
Extent

Error-handling options:
Insert catch block
Insert finally block

Insertion position : Last method

d. Click **Generate**.

e. Specify input parameter *pDept as Dept*.

The definition of this method should be as follows:

```
method public void AddDepartment(input pDept as Dept).
```

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 select **Source** > **Add Method**.

   c. Enter *GetDepartment* for the **Method name**.

d. Select *Dept (Enterprise.HR.Dept)* for the **Return type**.

Tip: Use the **Browse** button then search for *Dept* to enter Dept(Enterprise.HR.Dept).

e. Click **Generate**.

f. Specify input parameter *pDeptCode as character*.

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.

The definition of this method should be as follows:

```
method public integer NumberEmployees().
```

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

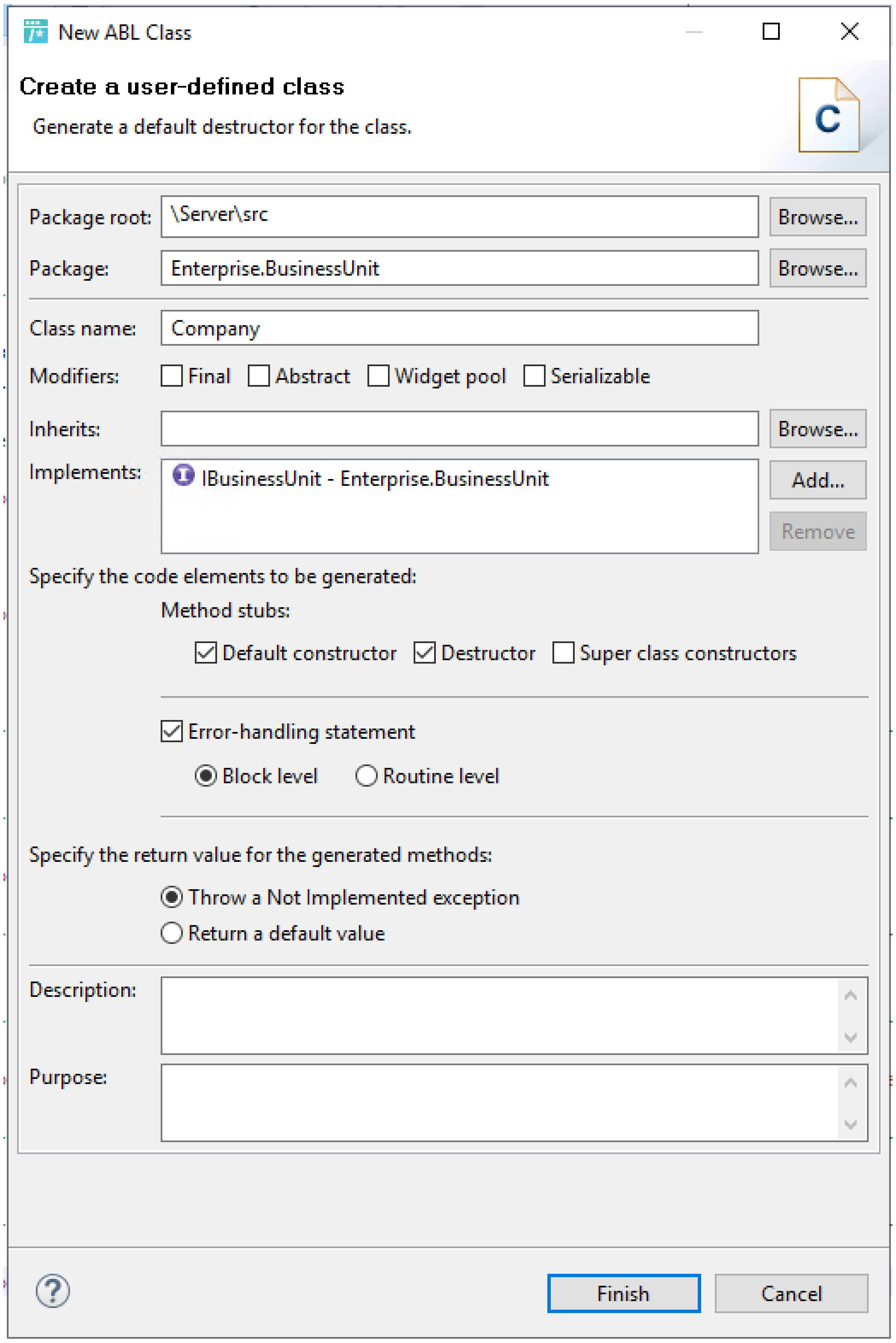
Part 4: Create the Company class

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

1. In **Project Explorer**, navigate to the **Include** sub-folder under the **src** folder.
2. In the **Include** folder, import the `ttDepartment.i` file.
   a. In **Windows Explorer,** navigate to the `/progress_education/OpenEdge/introOOP/Exercises/Lesson05` 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.

Tip: 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*.



New ABL Class

Create a user-defined class

Generate a default destructor for the class.

Package root: \Server\src

Package: Enterprise.BusinessUnit

Class name: Company

Modifiers: ☐ Final ☐ Abstract ☐ Widget pool ☐ Serializable

Inherits:

Implements: IBusinessUnit - Enterprise.BusinessUnit

Specify the code elements to be generated:

Method stubs:

☑ Default constructor ☑ Destructor ☐ Super class constructors

☑ Error-handling statement

● Block level ○ Routine level

Specify the return value for the generated methods:

● Throw a Not Implemented exception

○ Return a default value

Description:

Purpose:

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

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.
    private set.

  define public property NumDepartments as integer no-undo
    get.
    private 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 ):
    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 that 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"}
      ```

Part 5: Implement a constructor, a destructor, and methods for the Company class

Modify the constructor and the destructor, and define methods for the *Company* class.

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*.
   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.

d. 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.

Tip: 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.

Tip: 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:

```
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*.

Tip: 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.

Part 6: Import the Franchise class

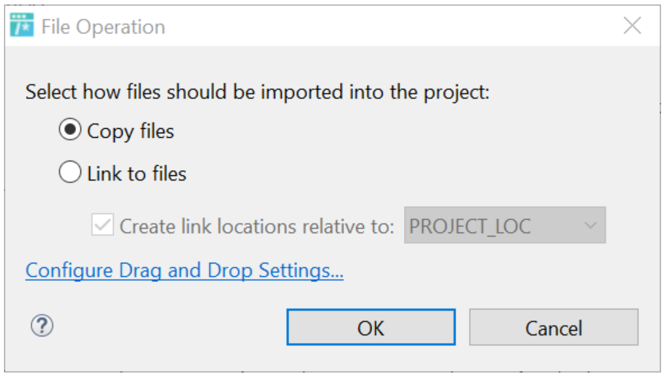
In this part of the Try It, you import the **Franchise.cls** file. This class also implements the *IBusinessUnit* class.

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/Exercises/Lesson05` 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.



File Operation

Select how files should be imported into the project:

(●) Copy files
( ) Link to files

☑ Create link locations relative to: PROJECT_LOC

Configure Drag and Drop Settings...

2. Clean and rebuild the **Server** project.

Part 7: Test the classes

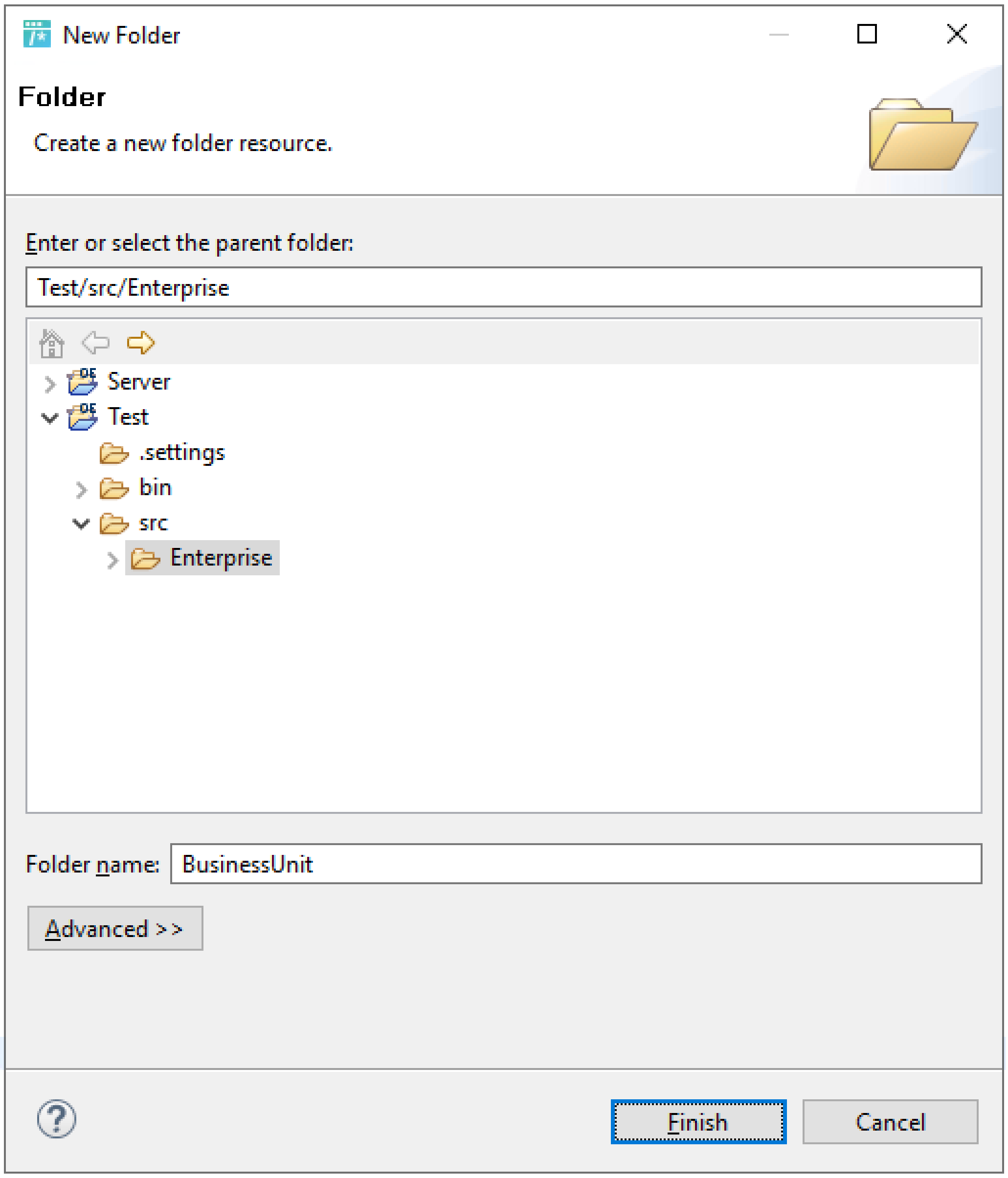
In this part of the Try It, import the *testCompany.p* and *testFranchise.p* procedure files for testing the *Company* and *Franchise* classes. Then, you test them one by one.

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.

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**.

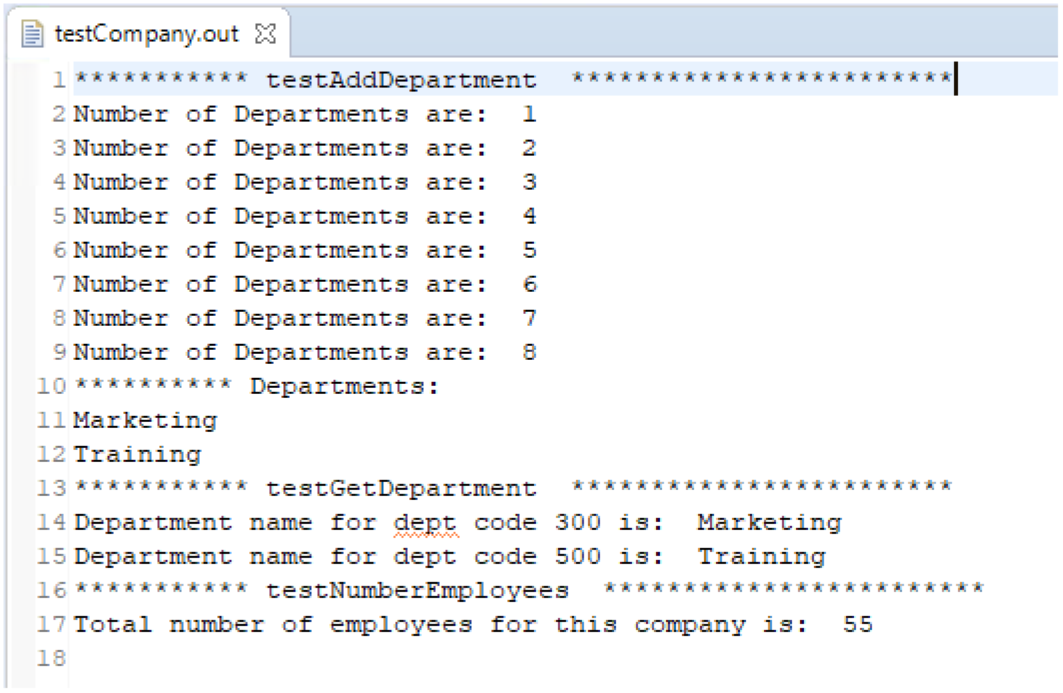
2. In the **Enterprise/BusinessUnit** directory of the **Test** project, import the procedure files named `testCompany.p` and `testFranchise.p`.
   a. In **Windows Explorer**, navigate to the `/progress_education/OpenEdge/introOOP/Exercises/Lesson05` directory and select the `testCompany.p` and `testFranchise.p` 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** > **Run As** > **Progress OpenEdge Application**.

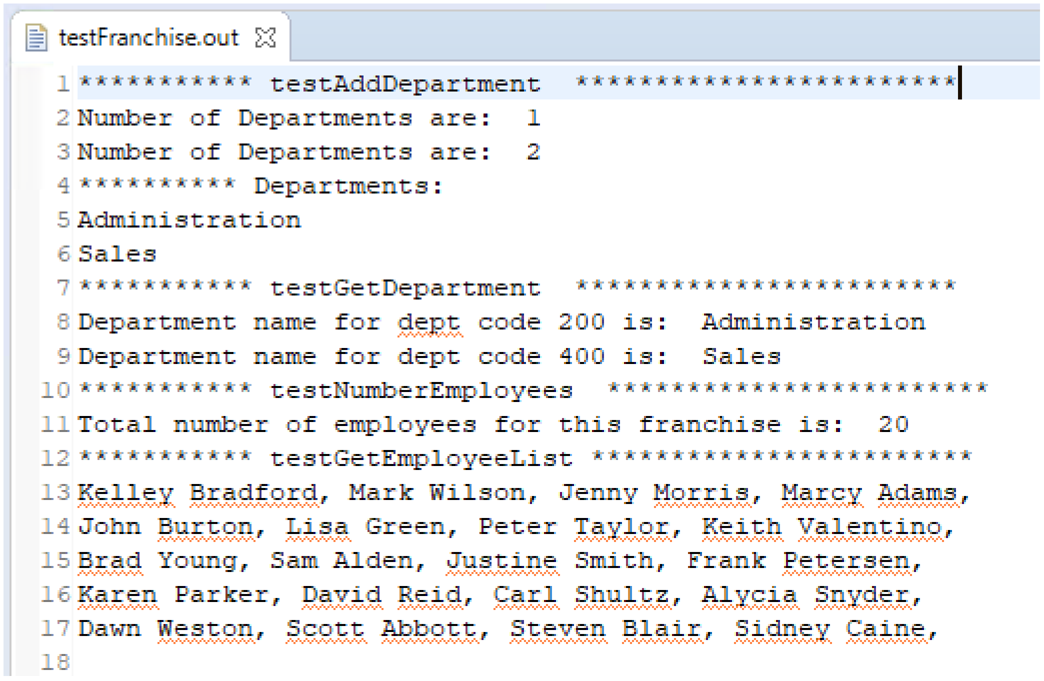
   b. Review the output.



```
*********** testAddDepartment  ************************
Number of Departments are:  1
Number of Departments are:  2
Number of Departments are:  3
Number of Departments are:  4
Number of Departments are:  5
Number of Departments are:  6
Number of Departments are:  7
Number of Departments are:  8
********** Departments:
Marketing
Training
*********** testGetDepartment  ************************
Department name for dept code 300 is:  Marketing
Department name for dept code 500 is:  Training
*********** testNumberEmployees  ************************
Total number of employees for this company is:  55
```

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** > **Run As** > **Progress OpenEdge Application**.



```
*********** testAddDepartment  ***********************
Number of Departments are:  1
Number of Departments are:  2
********** Departments:
Administration
Sales
*********** testGetDepartment  ***********************
Department name for dept code 200 is:  Administration
Department name for dept code 400 is:  Sales
*********** testNumberEmployees  ***********************
Total number of employees for this franchise is:  20
*********** testGetEmployeeList ***********************
Kelley Bradford, Mark Wilson, Jenny Morris, Marcy Adams,
John Burton, Lisa Green, Peter Taylor, Keith Valentino,
Brad Young, Sam Alden, Justine Smith, Frank Petersen,
Karen Parker, David Reid, Carl Shultz, Alycia Snyder,
Dawn Weston, Scott Abbott, Steven Blair, Sidney Caine,
```

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.

Solutions: Try It 5.2, Use events

Part 1: Define and publish an event in the Manager class

1. In **Project Explorer**, open the *Manager* class and define a new public event *DischargeEmployee*.

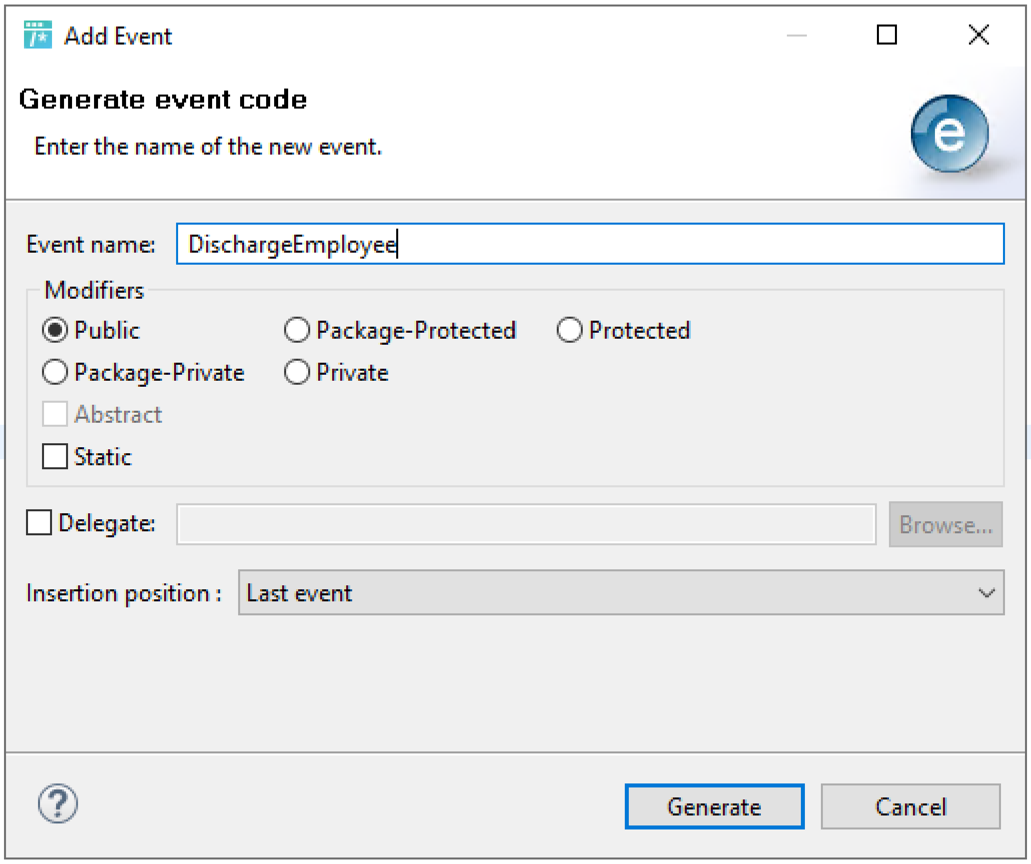
   Tip: Use the **Add Event** wizard.

   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**.


Add Event

Generate event code

Enter the name of the new event.

Event name: DischargeEmployee

Modifiers

● Public ○ Package-Protected ○ Protected

○ Package-Private ○ Private

☐ Abstract

☐ Static

☐ Delegate:

Insertion position : Last event

e. Click **Generate**.

The generated code should appear as follows:

```
define public event DischargeEmployee signature void ().
```

2. In the *DischargeEmployee* event, define the following input parameters.

- *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).
```

3. Define the *public DischargeTeamMember()* method with return type *void*.

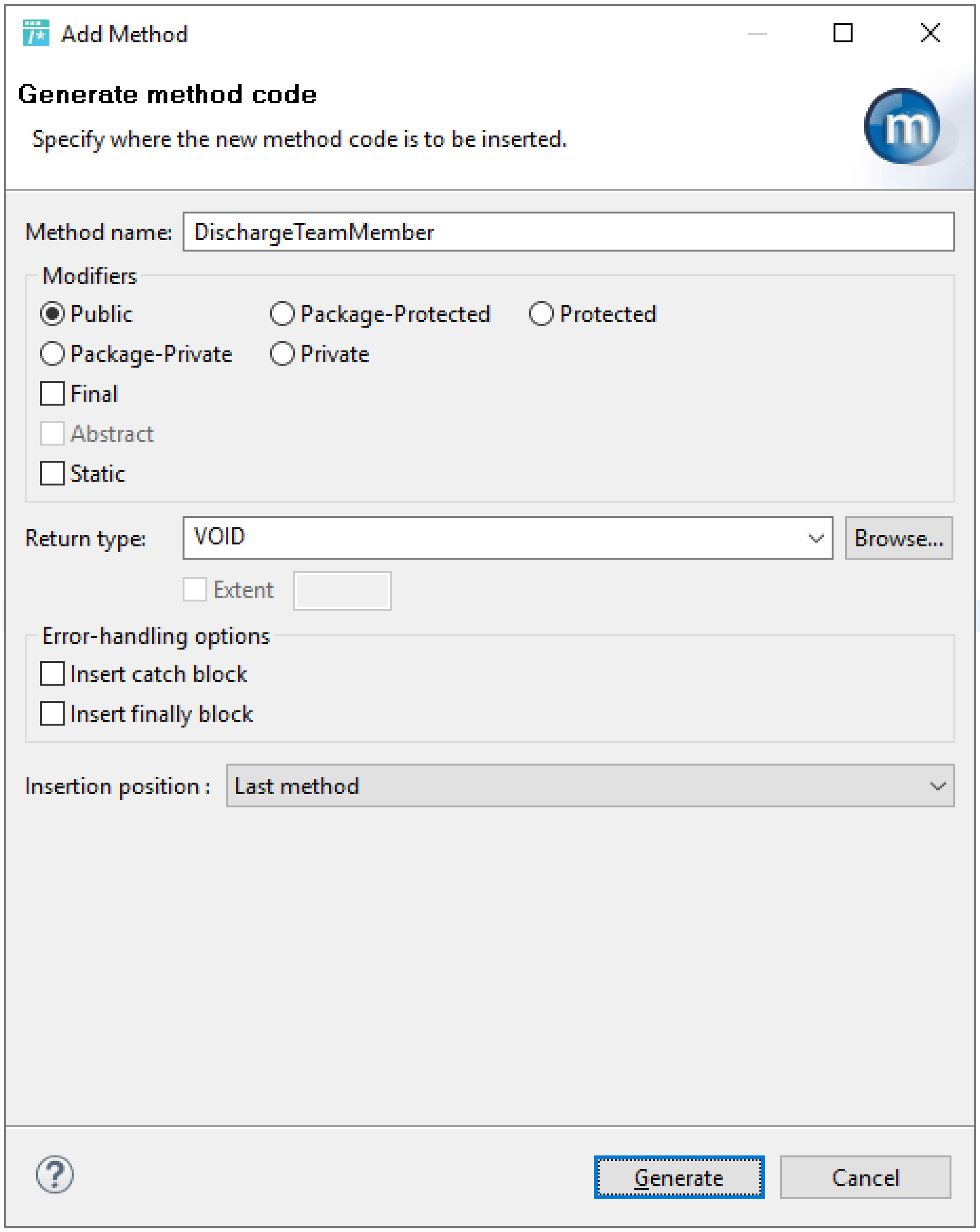
Tip: Use the **Add Method** wizard.

a. Place the cursor anywhere in the file.

b. Right-click and then select **Source** > **Add Method**.

c. Enter *DischargeTeamMember* for the **Method name**.

d. Select *void* for the **Return type**.



Generate method code

Specify where the new method code is to be inserted.

Method name: DischargeTeamMember

Modifiers

[✓ Public] Package-Protected Protected

Package-Private Private

☐ Final

☐ Abstract

☐ Static

Return type: VOID

☐ Extent

Error-handling options

☐ Insert catch block

☐ Insert finally block

Insertion position : Last method

e. Click **Generate**.

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*

  The definition of this method should be as follows:

  ```
  method public void DischargeTeamMember
    (input pFirstname as character, pLastname as character):
    return.
  end method.
  ```

5. Modify the public *DischargeTeamMember()* method to publish the *DischargeEmployee* event.

   The definition of this method should be 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.

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.

1. In **Project Explorer**, open the *Dept* class.
2. At the end of the *AddManager()* method, add a statement to subscribe to the *DischargeEmployee* event specifying the *DischargeEmployee_Handler* method.

   Tip: You will see an error in this statement because you have not yet created the *DischargeEmployee_Handler()* method.

   The definition of this method should be as follows:

   ```
   DeptMgr:DischargeEmployee:subscribe(DischargeEmployee_Handler).
   ```

3. Define the public *DischargeEmployee_Handler()* method with return type *void*.

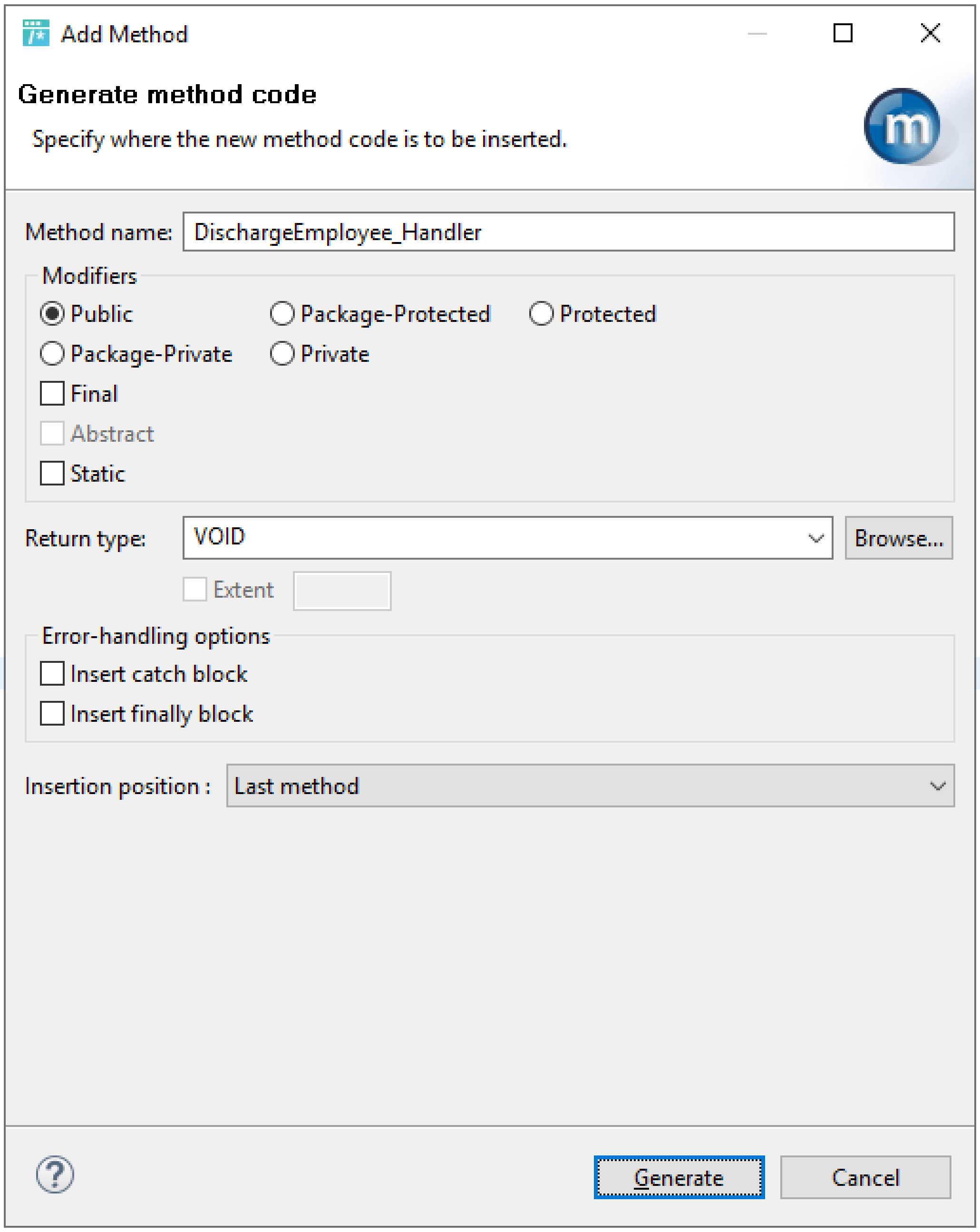
   Tip: Use the **Add Method** wizard.

   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 **Method name**.

   d. Select *Last method* for the **Insertion position**.


Add Method

Generate method code

Specify where the new method code is to be inserted.

Method name: DischargeEmployee_Handler

Modifiers

(● Public) ○ Package-Protected ○ Protected

○ Package-Private ○ Private

☐ Final

☐ Abstract

☐ Static

Return type: VOID

☐ Extent

Error-handling options

☐ Insert catch block

☐ Insert finally block

Insertion position : Last method

e. Click **Generate**.

The generated code should appear as follows:

```
method public void DischargeEmployee_Handler ():
  return.
end method.
```

4. To the public *DischargeEmployee_Handler()* method add the following input parameters.

- *pFirstname* as *character*
- *pLastname* as *character*

The code you add should appear as follows:

```
method public void DischargeEmployee_Handler
    (input pFirstName as character, input pLastName as character):
    return.
end method.
```

5. Modify the public *DischargeEmployee_Handler()* method to find the discharged employee in the *ttEmployee* temp-table based on the first and last name.

   The code you add should appear as follows:

```
find ttEmployee where ttEmployee.FirstName = pFirstName
     and ttEmployee.LastName = pLastName no-error.
```

6. 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.

Tip: 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.

Tip: 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.
```

7. In the *do* block, add a statement to delete the *ttEmployee* temp-table record that was found.

```
delete ttEmployee.
```

8. In the *do* block, add a statement to decrease the employee count by 1.

```
NumEmployees = NumEmployees - 1.
```

9. In the *do* block, add a statement to close the output file.

```
output close.
```

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

Part 3: Test 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.

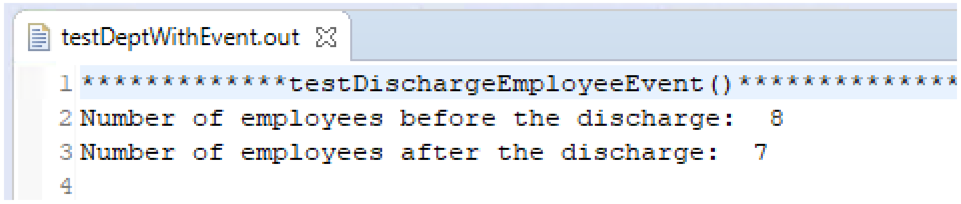
1. In the **Enterprise/HR** folder of the **Test** project, import the procedure file named `testEvents.p`.

1. In **Windows Explorer**, navigate to the `/progress_education/OpenEdge/introOOP/Exercises/Lesson05` directory and select the `testEvents.p` file.
2. Drag and drop the *testEvents.p* file into the **Enterprise/HR** folder.
3. Ensure that **Copy files** is selected.
4. 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** > **Run As** > **Progress OpenEdge Application**.

3. Review the results. If the file did not run as expected, use the debugger to fix the problem, save your changes, and retest until it runs correctly.



```
*************testDischargeEmployeeEvent()**************
Number of employees before the discharge:  8
Number of employees after the discharge:  7
```