

# Reverse Engineering Sequence Diagram from Java Source Code

[Sequence diagram](#) can help to represent the interaction between objects in runtime. Since [Visual Paradigm for UML 8.0 \(VP-UML\)](#) enables you to reverse your Java source code into sequence diagram, you can gain a better understanding of Java source code by reading diagram instead of looking to a possibly thousand lines of source code.

August 16, 2010

User Rating: / 29

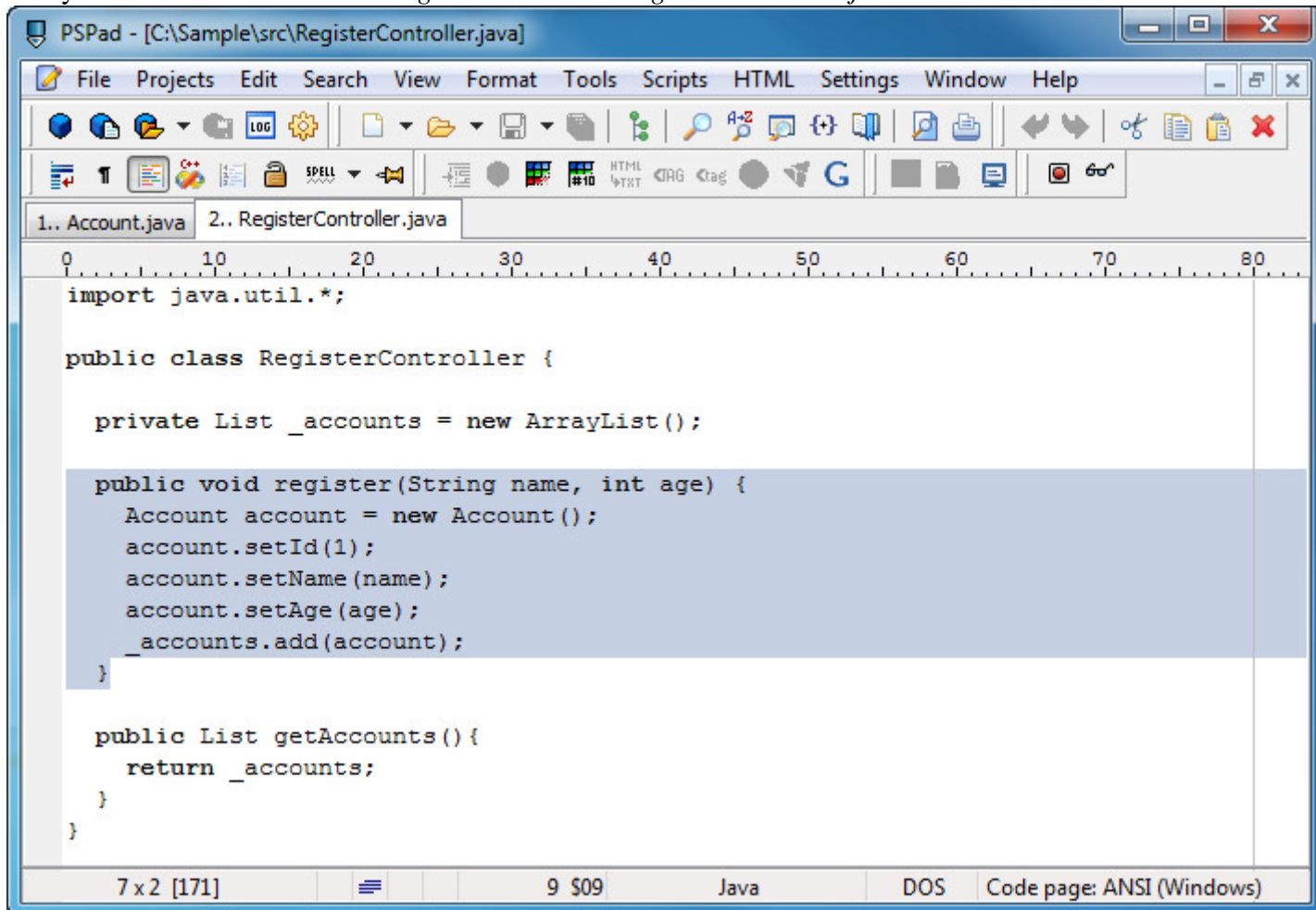
Views: 32,991

[PDF Link](#) [Add comments](#)

Edition: [Standard or above](#) ([Edition comparison](#))

---

1. Download *Sample.zip* of this tutorial and extract the zip file to any directory.
2. Study the source code. Read the *register* method in *RegisterController.java* to see how it works.

A screenshot of the PSPad text editor window. The title bar reads "PSPad - [C:\Sample\src\RegisterController.java]". The menu bar includes File, Projects, Edit, Search, View, Format, Tools, Scripts, HTML, Settings, Window, and Help. The toolbar contains various icons for file operations, editing, and development. Below the toolbar, two tabs are visible: "1.. Account.java" and "2.. RegisterController.java". The main text area shows the source code of the RegisterController class. The code includes an import statement for java.util.\*, a class declaration, a private ArrayList field, and two public methods: register and getAccounts. The register method is currently selected with a blue highlight. The status bar at the bottom shows "7 x 2 [171]", a list icon, "9 \$09", "Java", "DOS", and "Code page: ANSI (Windows)".

```
import java.util.*;

public class RegisterController {

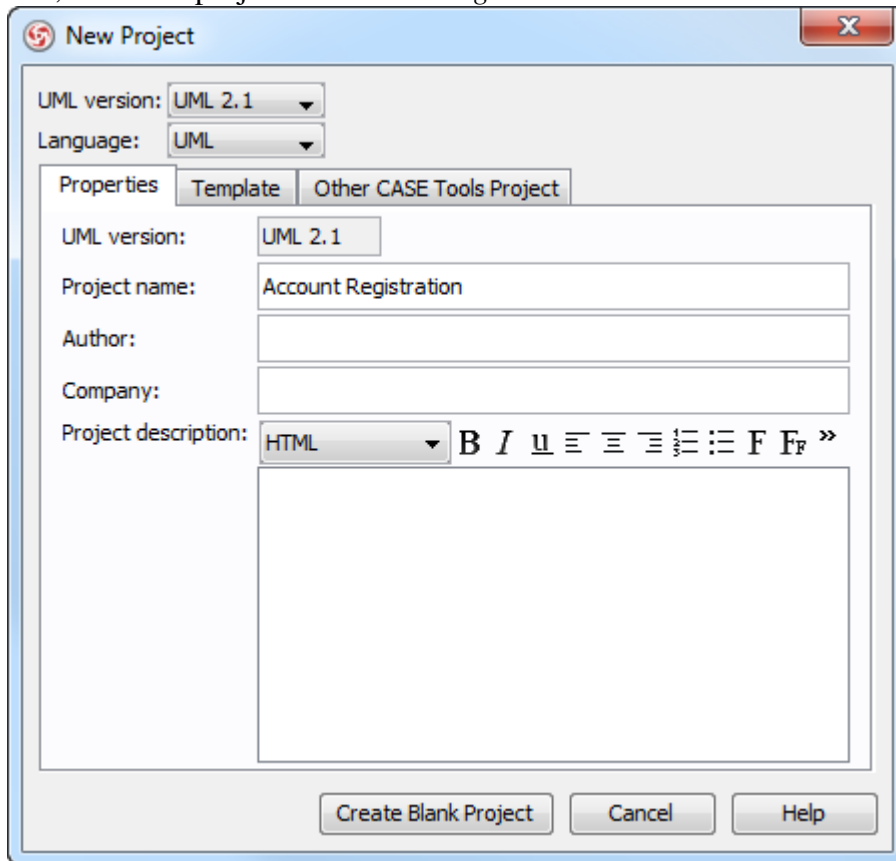
    private List _accounts = new ArrayList();

    public void register(String name, int age) {
        Account account = new Account();
        account.setId(1);
        account.setName(name);
        account.setAge(age);
        _accounts.add(account);
    }

    public List getAccounts() {
        return _accounts;
    }
}
```

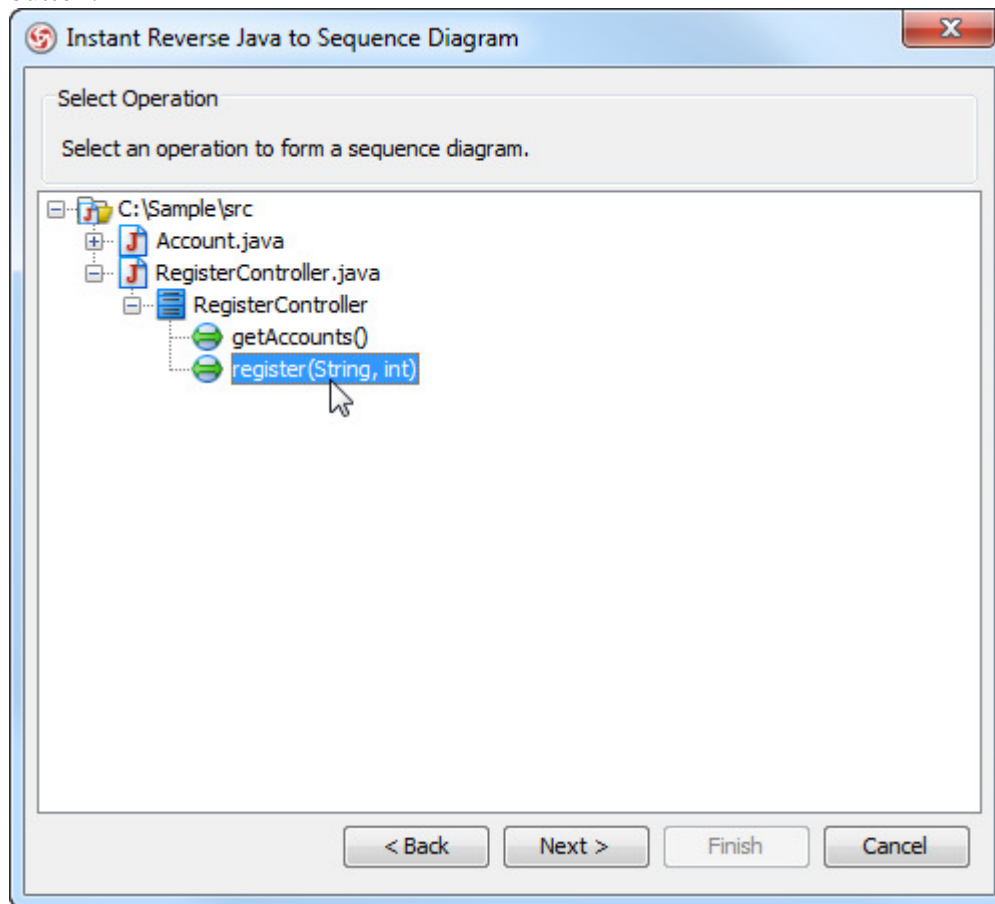
3. Start VP-UML.

4. Create a new project by selecting **File > New Project** from the main menu. In the **New Project** dialog box, name the project as *Account Registration* and click **Create Blank Project** button.

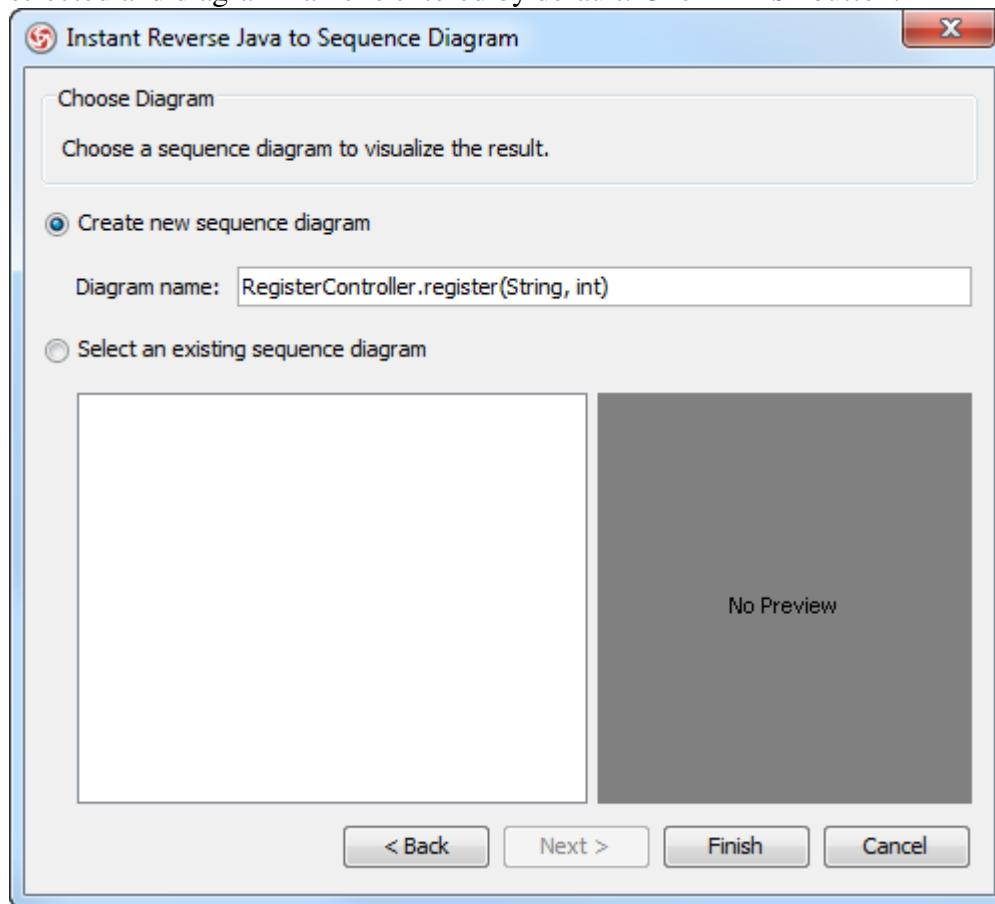


5. Select **Tools > Instant Reverse > Java to Sequence Diagram...** from the main menu.
6. In the **Instant Reverse Java to Sequence Diagram** dialog box, click on **Add Source Folder...** button.
7. Select the extracted source folder *src*. Click **Next** button.

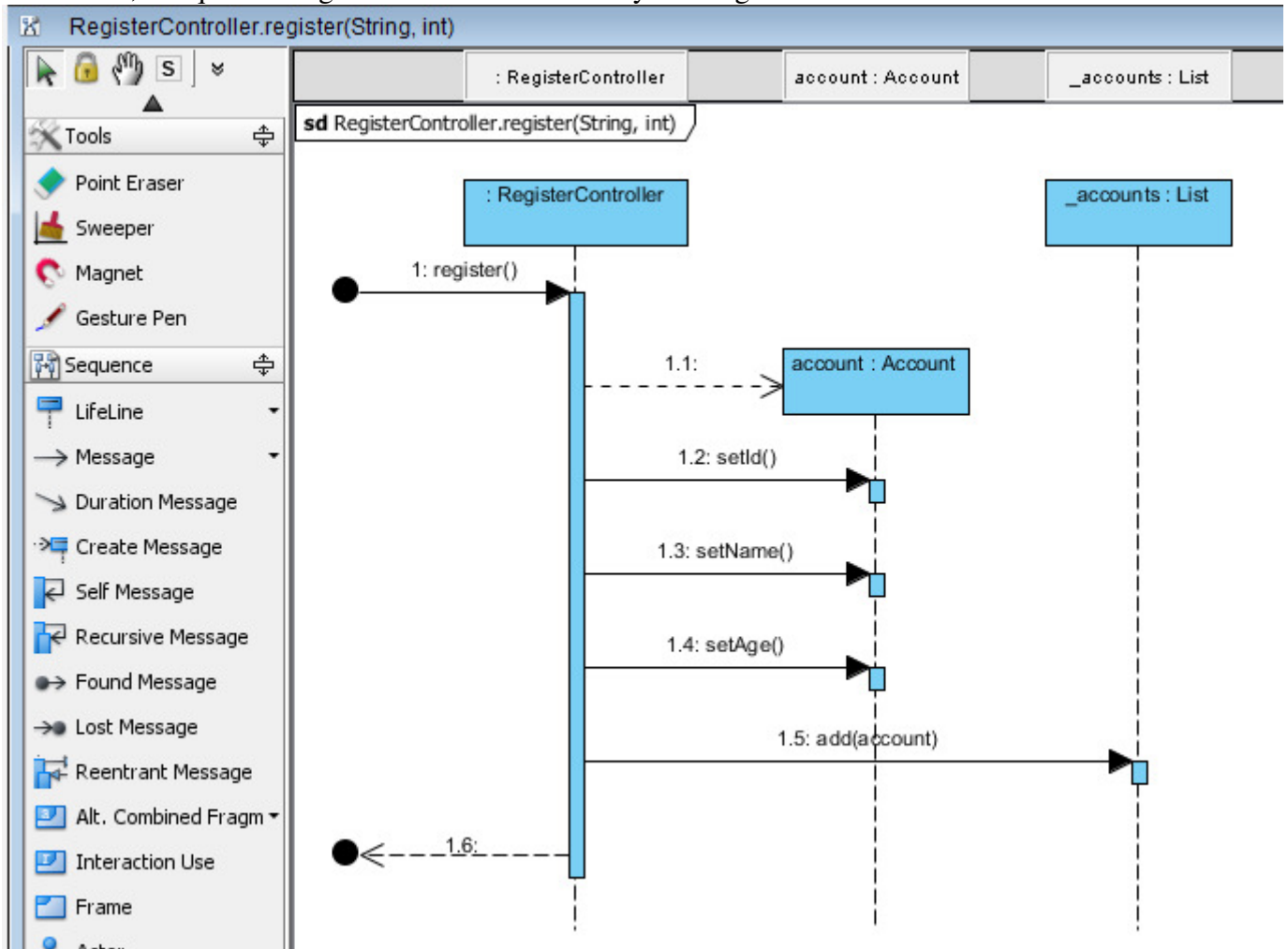
8. Select the method to visualize. Select **src > RegisterController.java > reister(String,int)**. Click **Next** button.



9. You need to select a diagram to visualize the interaction. The **Create new sequence diagram** option is selected and diagram name is entered by default. Click **Finish** button.



10. As a result, a sequence diagram is formed. Let's study the diagram.



When a person invokes *RegisterController*'s register method (message: 1), it creates an account object (message: 1.1). After that, the controller sets the id, name and age to the account object (message 1.2, 1.3, 1.4), and adds itself to the account list (message: 1.5). The invocation ends with a return (message 1.6).

# Java Code Generation from Class Diagram

VP-UML supports [generating Java source code from UML class model](#), and keeping the code and model synchronized. In this tutorial we will develop a simple [class diagram](#), generate code from it, modify the code and update the changes back to UML class model.

May 10, 2010

User Rating: / 25

Views: 20,153

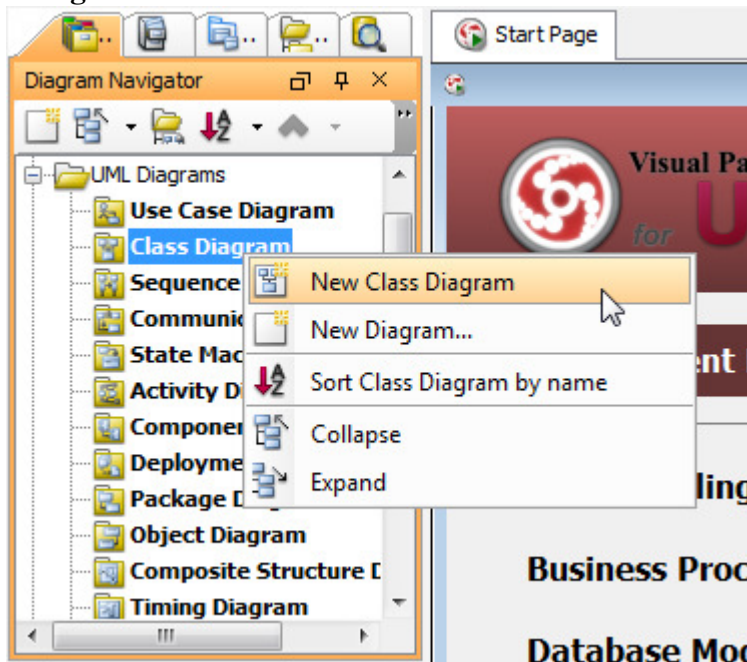
[PDF Link](#)

[Add comments](#)

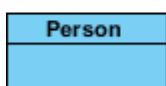
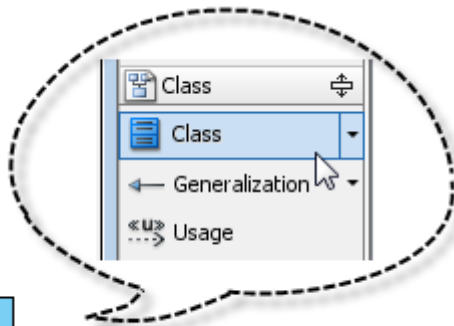
Edition: [Standard or above](#) ([Edition comparison](#))

---

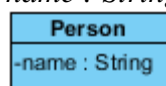
1. Start VP-UML in a new workspace.
2. Create a class diagram and name it as *Employee*. You can create a class diagram through the **Diagram Navigator**.



3. Select **Class** from diagram toolbar and click on class diagram to create a class. Name it as *Person*.



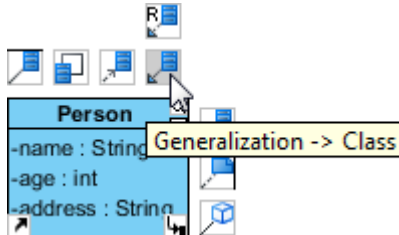
4. Create attributes in it. Right click on *Person* and select **Add Attribute** from the popup menu. Enter *name* : *String* to name the attribute as *name* and set type as *String*.



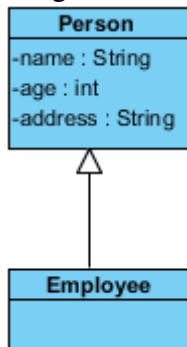
5. Repeat step 4 to create 2 more attributes:

Name	Type
age	int
address	String

6. Now, create a sub-class from *Person*. Move the mouse pointer to class *Person* and press on the resource icon **Generalization -> Class**.



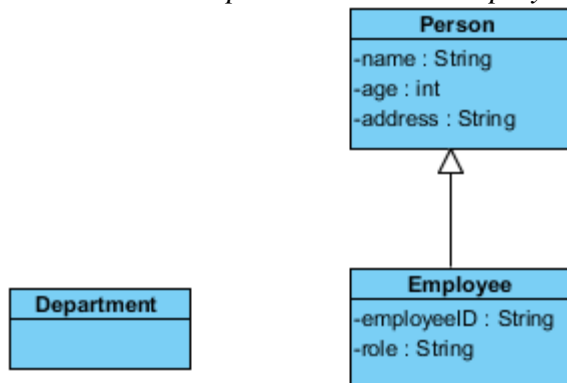
7. Drag downwards and release the mouse button. Name the class as *Employee*.



8. Add the following attributes to the *Employee* class:

Name	Type
employeeID	String
role	String

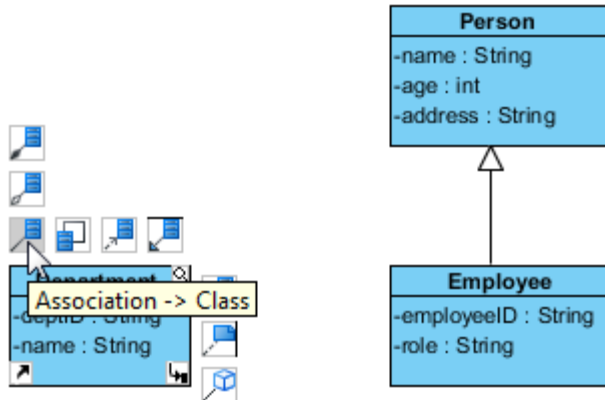
9. Create a class *Department* next to *Employee* class, like this:



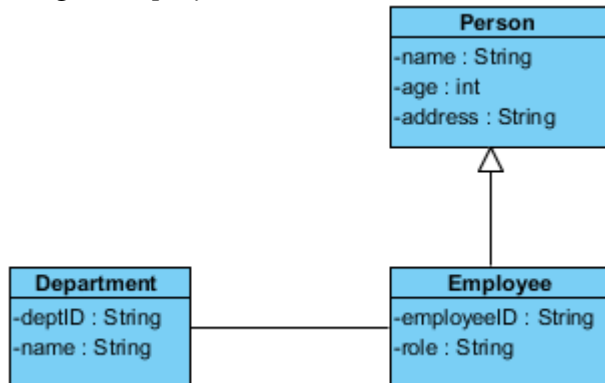
10. Add the following attributes to the *Department* class:

Name	Type
deptID	String
name	String

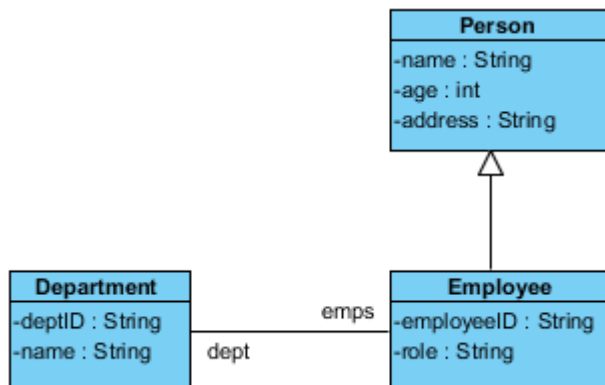
11. Associate *Department* with *Employee*. Move the mouse pointer to class *Department* and press on the resource icon **Association -> Class**.



12. Drag to *Employee* class and release the mouse button.

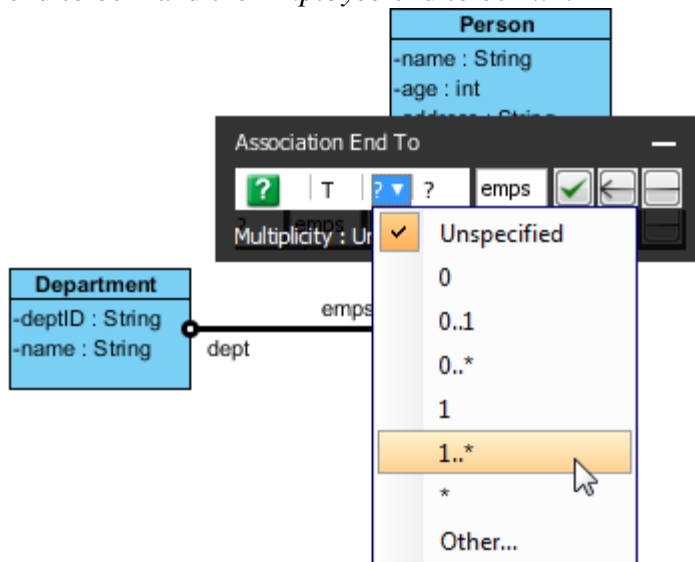


13. Name the roles of both ends of association. Double click at the end of association to name it.

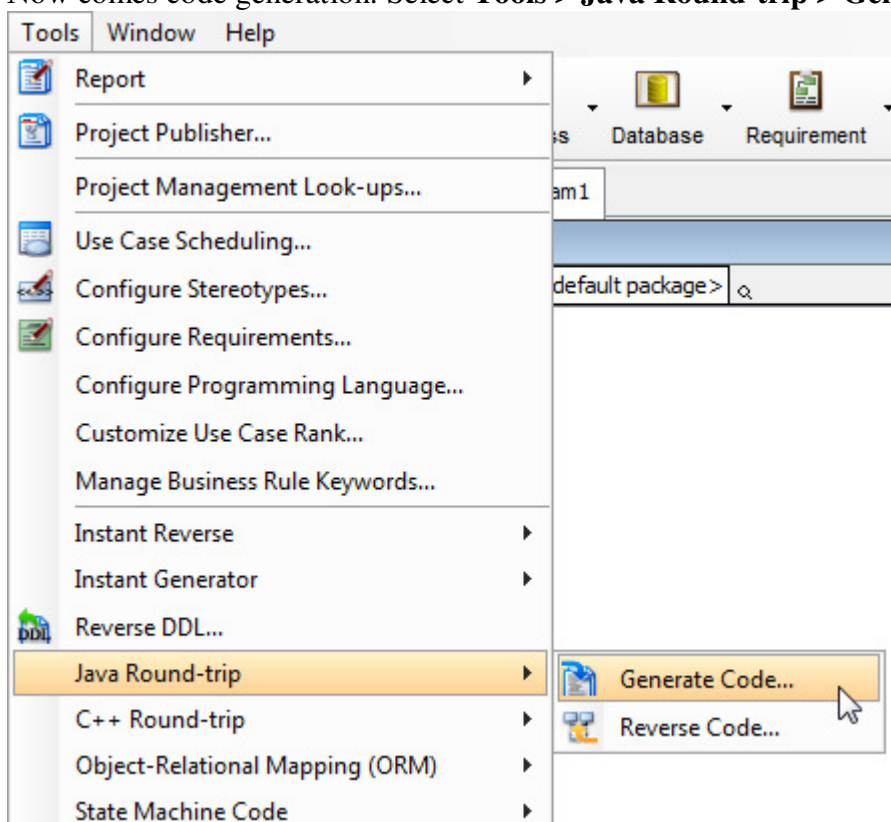




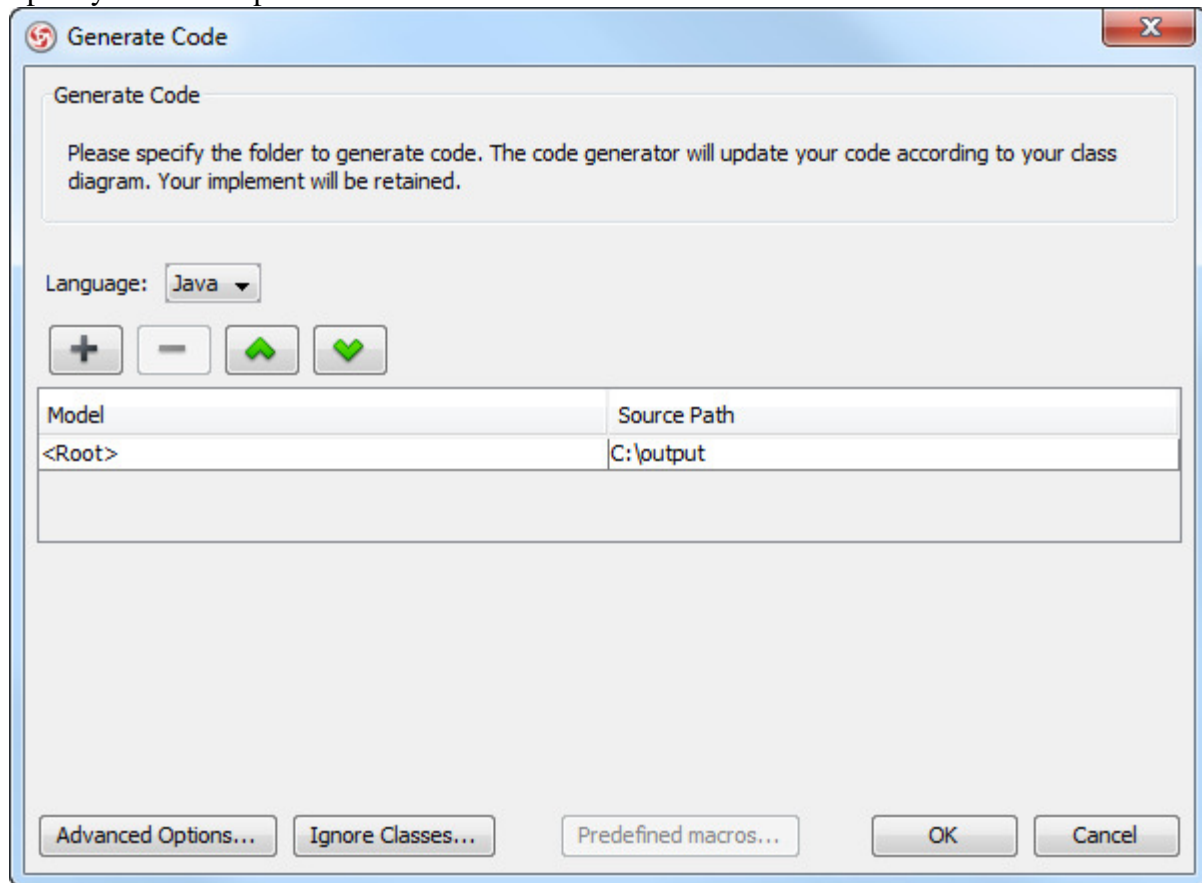
14. Double click at the association end again and specify the multiplicity of both ends. Set the *Department* end to be 1 and the *Employee* end to be 1..\*.



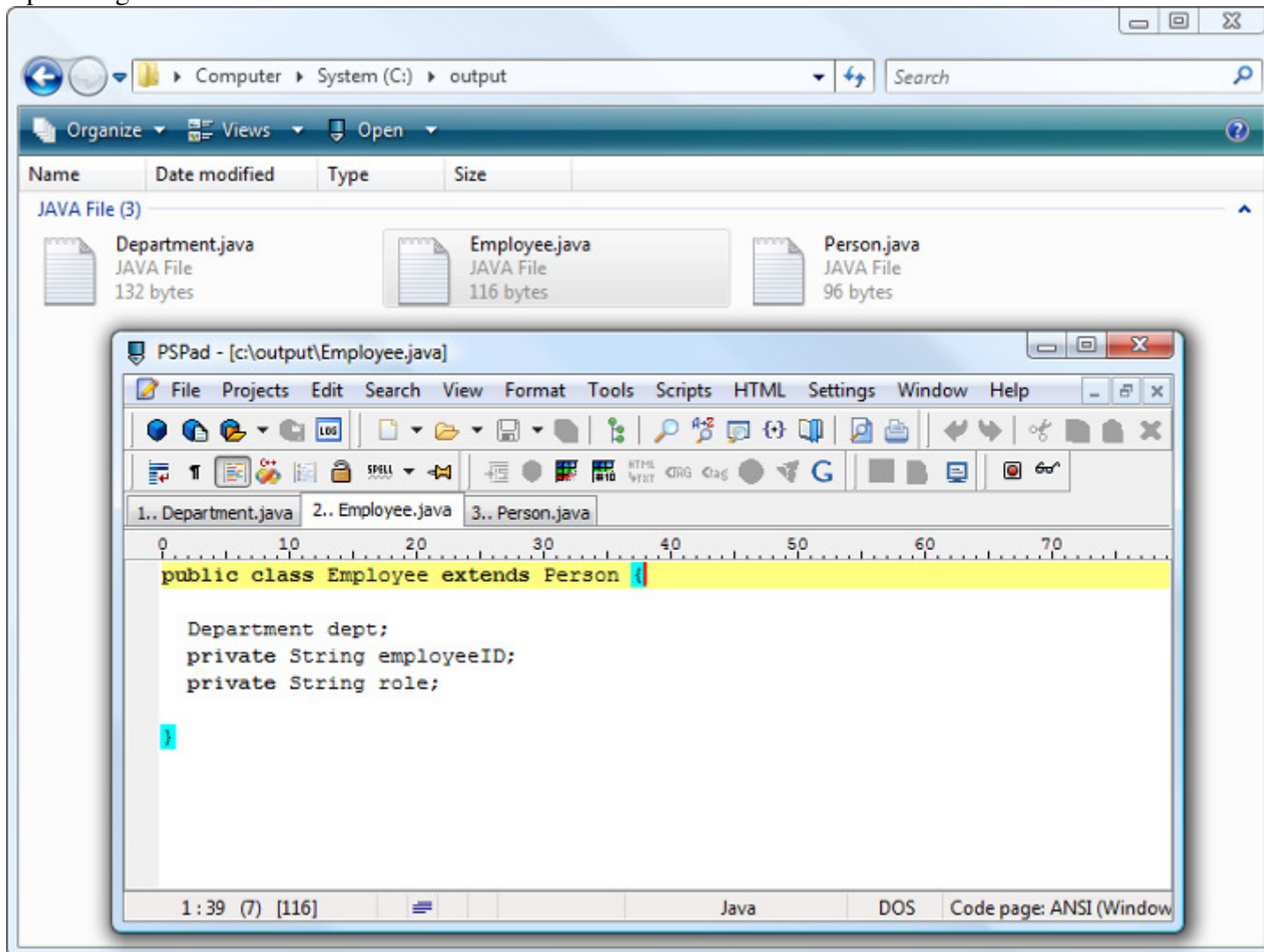
15. Now comes code generation. Select **Tools > Java Round-trip > Generate Code...** from the main menu.



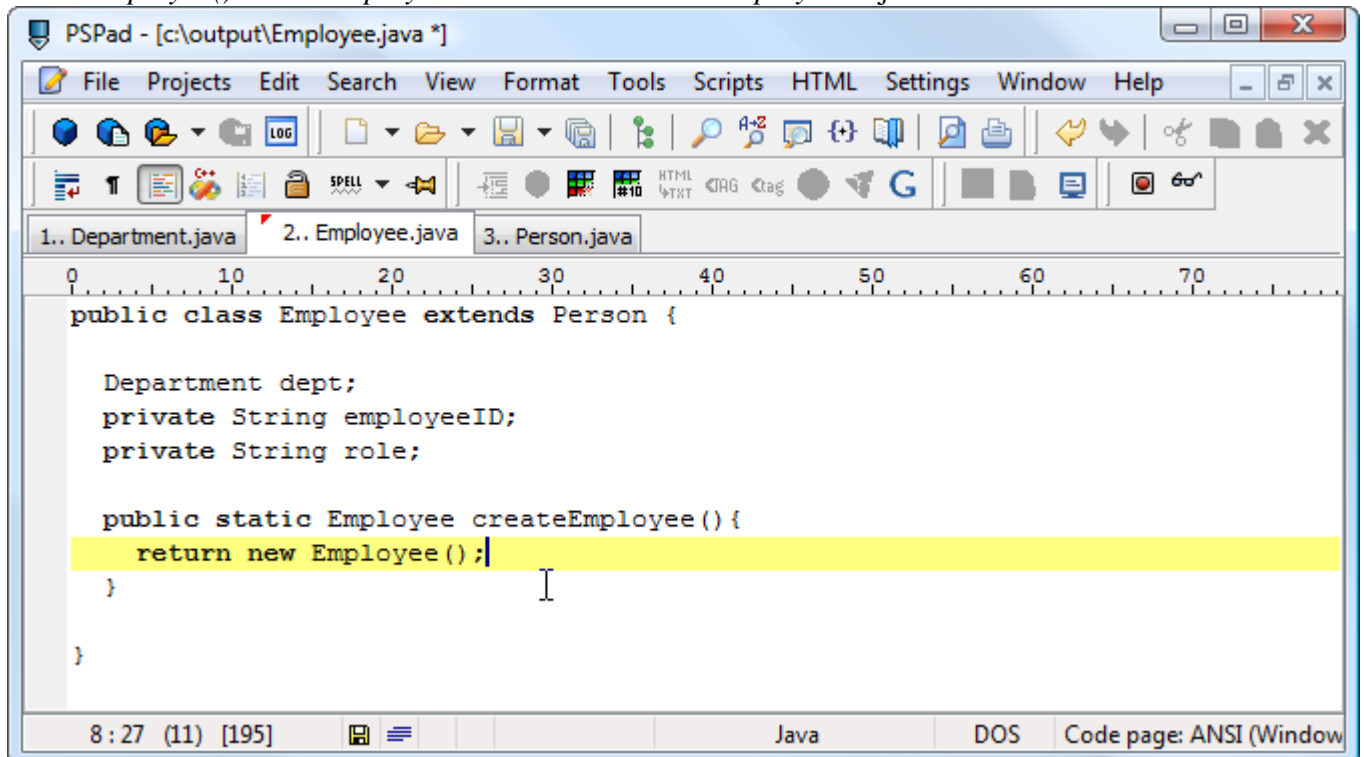
16. Specify the source path and click **OK**.



17. Open the generated code files.

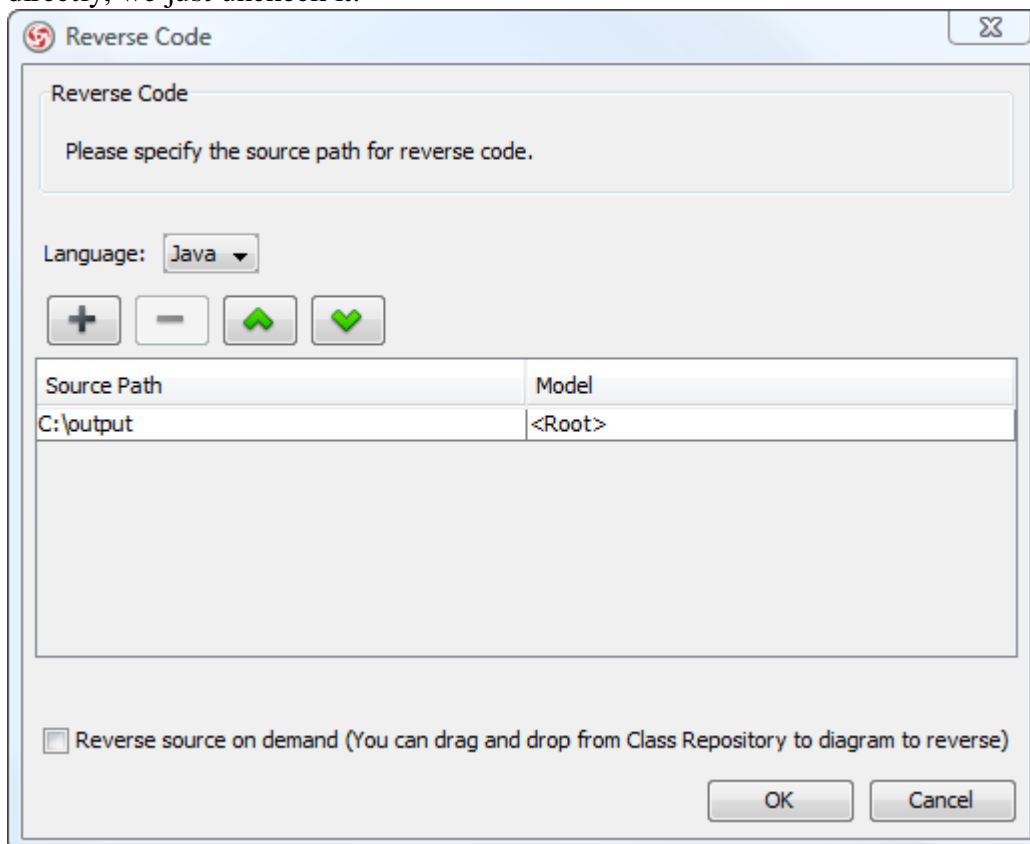


18. Let's edit the code file. After that, we will update the changes back to model. Add a static operation *createEmployee()* in the *Employee* class. It returns and *Employee* object.



```
public class Employee extends Person {  
  
    Department dept;  
    private String employeeID;  
    private String role;  
  
    public static Employee createEmployee() {  
        return new Employee();  
    }  
  
}
```

19. Save the file and go back to VP-UML.  
20. Select **Tools > Java Round-trip > Reverse Code...** from the main menu.  
21. In the **Reverse Code** dialog box, uncheck **Reverse source on demand** and click **OK**. This option will cause an index tree to be created instead of reversing files directly. Since we want to update the class model directly, we just uncheck it.



Reverse Code

Please specify the source path for reverse code.

Language: Java

+ - ↕ ✓

Source Path	Model
C:\output	<Root>

☐ Reverse source on demand (You can drag and drop from Class Repository to diagram to reverse)

OK Cancel

22. You can see that the operation *createEmployee()* is created in *Employee* class.

