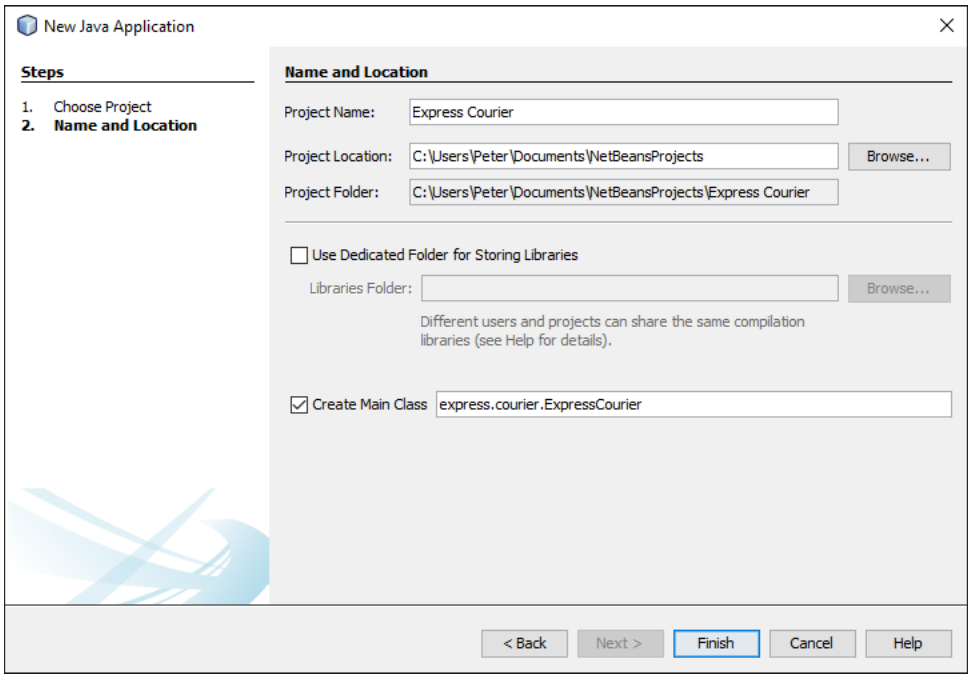
Visual Paradigm

How to Generate Java from UML Class Diagram in NetBeans?

Written Date : February 22, 2010

You can perform round-trip engineering in NetBeans, to keep Java source code and class model in sync. In this tutorial, we will see how to create a class model in NetBeans and eventually generating source code from model.

1. Create a Java project *Express Courier* in NetBeans.



New Java Application

Steps
1. Choose Project
2. **Name and Location**

Name and Location

Project Name: Express Courier

Project Location: C:\Users\Peter\Documents\NetBeansProjects

Project Folder: C:\Users\Peter\Documents\NetBeansProjects\Express Courier

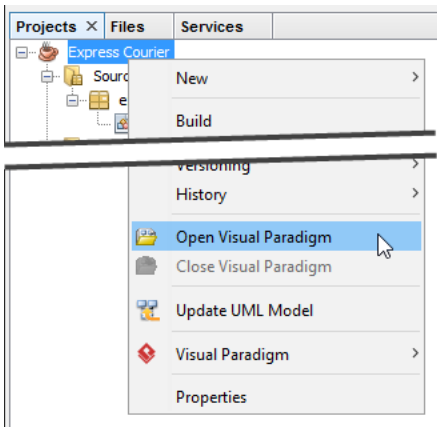
☐ Use Dedicated Folder for Storing Libraries

Libraries Folder:

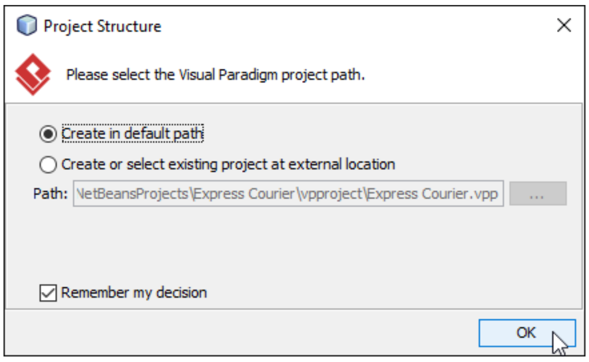
Different users and projects can share the same compilation libraries (see Help for details).

☑ Create Main Class express.courier.ExpressCourier

2. Right-click on the project node in **Projects** window and select **Open Visual Paradigm** from the popup menu.



3. You may be prompted to specify the location of your Visual Paradigm project. In this case, simply select **Create in default path** and click **OK** to proceed.


Project Structure

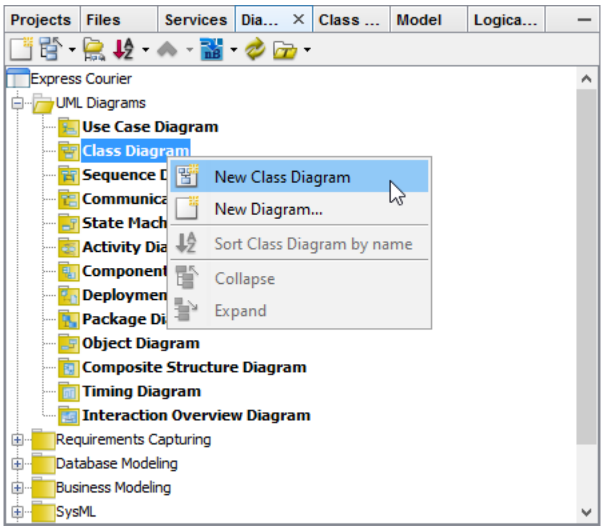
Please select the Visual Paradigm project path.

(●) Create in default path
( ) Create or select existing project at external location

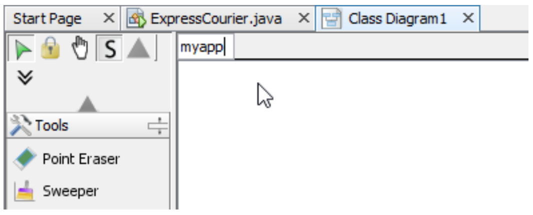
Path: \NetBeansProjects\Express Courier\vpproject\Express Courier.vpp

[✓] Remember my decision

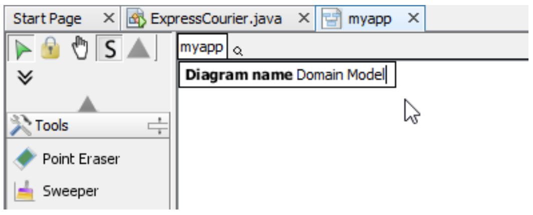
4. In **Diagram Navigator**, right-click on **Class Diagram** and select **New Class Diagram** from the popup menu.



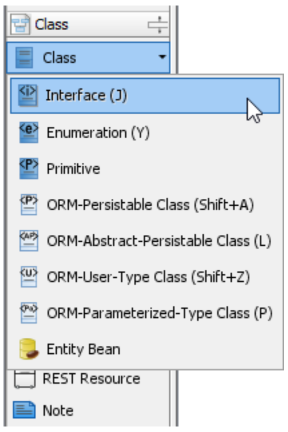
5. A new diagram is created. You asked to enter a package header on top of the diagram. Enter *myapp* and press **Enter**.



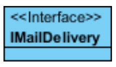
6. You are asked to provide the diagram name. Enter *Domain Model* and press **Enter**.



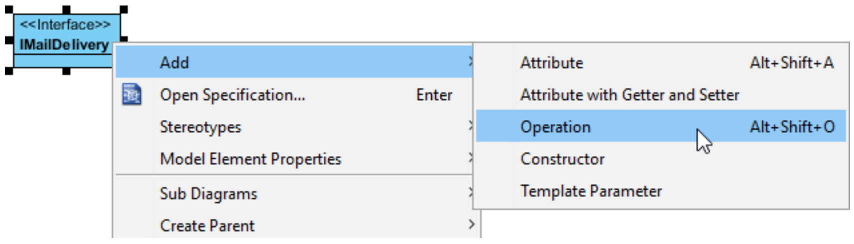
7. Click on the down arrow button near the shape selection **Class** in diagram toolbar, and select **Interface**.


Class
Class
Interface (J)
Enumeration (Y)
Primitive
ORM-Persistable Class (Shift+A)
ORM-Abstract-Persistable Class (L)
ORM-User-Type Class (Shift+Z)
ORM-Parameterized-Type Class (P)
Entity Bean
REST Resource
Note

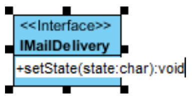
8. Click on the diagram to create an interface class and name it as *IMailDelivery*.



9. Create operations in *IMailDelivery*. Right-click on the class *IMailDelivery* and select **Add** > **Operation** from the popup menu.

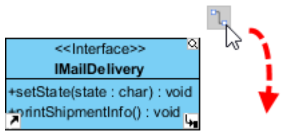


10. Enter *setState(state : char) : void* to create a public operation *stateState* with parameter *state* and return *void*.

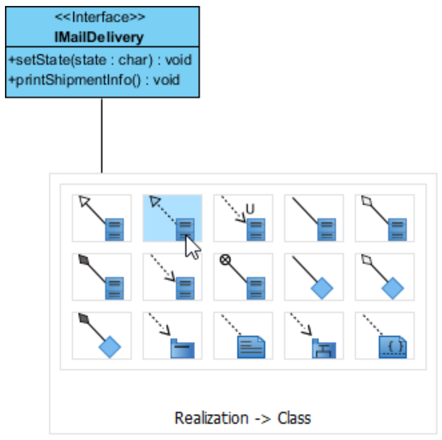


11. Press **Enter** to create another operation. Name it *printShipmentInfo() : void*. Click on diagram to confirm editing.

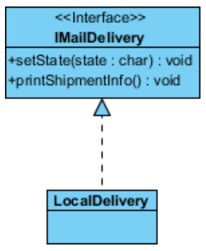
12. We need to create a class for local delivery which inherits *IMailDelivery*. Move the mouse pointer over interface *IMailDelivery*, press on the **Resource Catalog** icon and drag downwards.



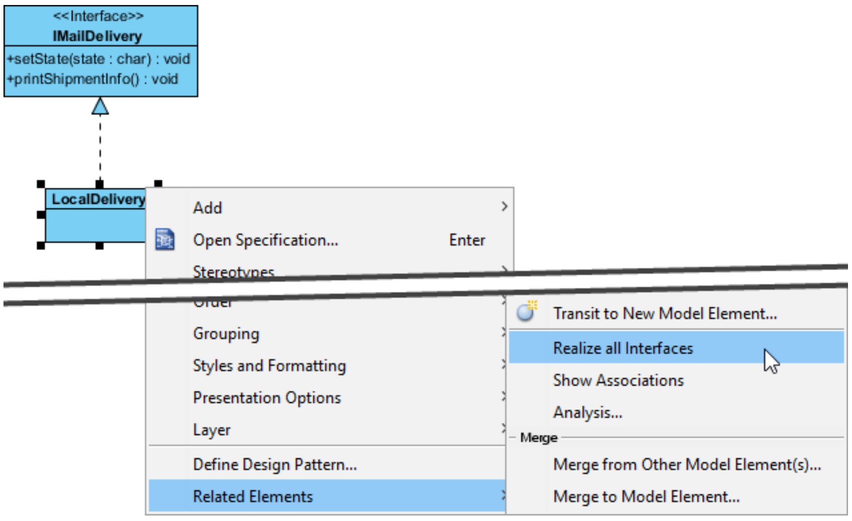
13. Release the mouse button. Select **Realization -> Class** in Resource Catalog.



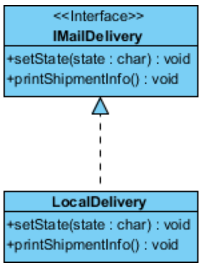
14. Name the class *LocalDelivery* and press **Enter** to confirm.



15. As the *LocalDelivery* class is implementing the interface *IMailDelivery*, we need to implement the operations defined in *IMailDelivery*. Right-click on class *LocalDelivery* and select **Related Element** > **Realize all Interfaces** from the popup menu.

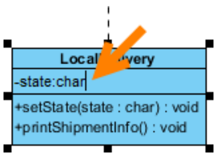


16. You can see that operations *setState* and *printShipmentInfo* are both inherited.


<<Interface>>
IMailDelivery
+setState(state : char) : void
+printShipmentInfo() : void
LocalDelivery
+setState(state : char) : void
+printShipmentInfo() : void

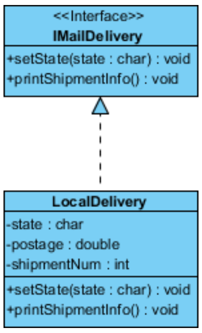
17. It is time to add attributes to classes. Right-click on class *LocalDelivery* and select **Add** > **Attribute** from the popup menu.

18. Enter *state : char* to name the attribute as *state*, and set the type as *char*.

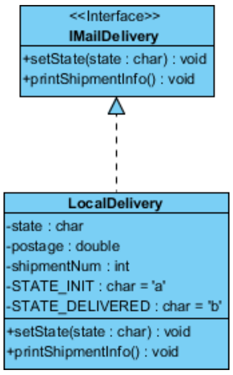


19. Press **Enter** to proceed to the next attribute. Enter *postage : double* as attribute name and type.

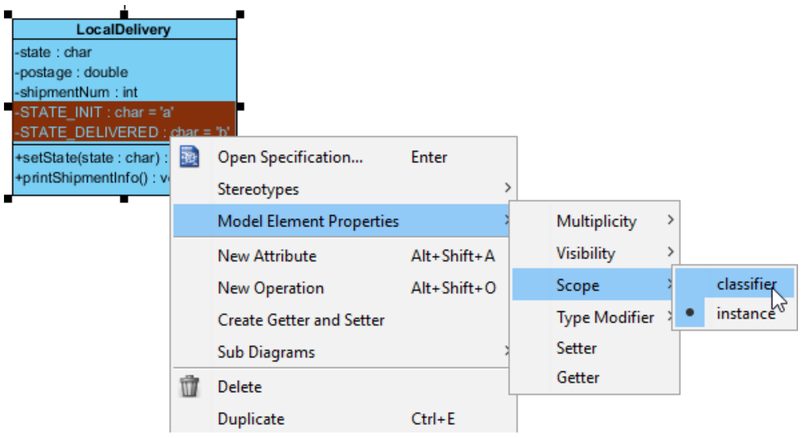
20. Again, press **Enter** and create attribute *shipmentNum : int*.


<<Interface>>
IMailDelivery
+setState(state : char) : void
+printShipmentInfo() : void
LocalDelivery
-state : char
-postage : double
-shipmentNum : int
+setState(state : char) : void
+printShipmentInfo() : void

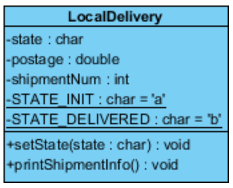
21. We need to have two constants for representing states. Press **Enter** to continue creating attribute. Name the attribute *STATE_INIT : char = 'a'*. This means to create an attribute named *STATE_INIT*, which is in *char* type and have *'a'* as default value. Press **Enter** to create another constant *STATE_DELIVERED : char = 'b'*. Up to now, the class should look like:


<<Interface>>
IMailDelivery
+setState(state : char) : void
+printShipmentInfo() : void
LocalDelivery
-state : char
-postage : double
-shipmentNum : int
-STATE_INIT : char = 'a'
-STATE_DELIVERED : char = 'b'
+setState(state : char) : void
+printShipmentInfo() : void

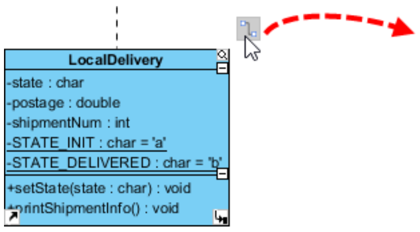
22. In order to declare both *STATE_* attribute as constants, select *STATE_INIT*, press the **Ctrl** key and select *STATE_DELIVERED* to make a multiple selection. Right-click on them and select **Model Element Propertes** > **Scope** > **classifier** from the popup menu. By doing so, both attributes will be static (in code level) and are underlined.



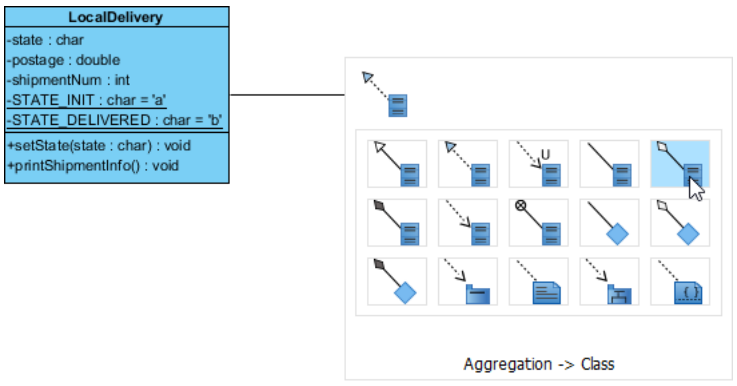
23. Again, right-click on the two attributes and select **Code Detail** > **final** from the popup menu. Click on the diagram to confirm editing. The class should now become:


LocalDelivery
-state : char
-postage : double
-shipmentNum : int
-STATE_INIT : char = 'a'
-STATE_DELIVERED : char = 'b'
+setState(state : char) : void
+printShipmentInfo() : void

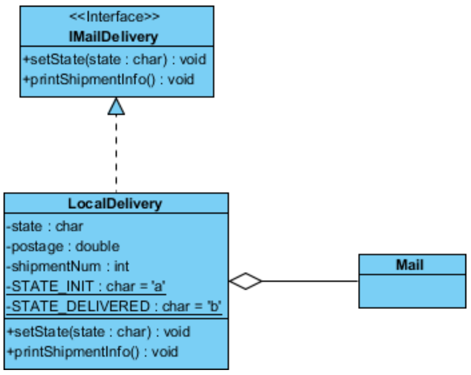
24. We need to create a new class *Mail* with association from class *LocalDelivery*. Move the mouse pointer to the class *LocalDelivery*. Press on the **Resource Catalog** icon and drag it out.


LocalDelivery
-state : char
-postage : double
-shipmentNum : int
-STATE_INIT : char = 'a'
-STATE_DELIVERED : char = 'b'
+setState(state : char) : void
+printShipmentInfo() : void

25. Select **Aggregation -> Class** in Resource Catalog.



26. Drag to the right of the class *LocalDelivery* and release the mouse button. Name the new class *Mail*.



<<Interface>>
IMailDelivery
+setState(state : char) : void
+printShipmentInfo() : void
LocalDelivery
-state : char
-postage : double
-shipmentNum : int
-STATE_INIT : char = 'a'
-STATE_DELIVERED : char = 'b'
+setState(state : char) : void
+printShipmentInfo() : void
Mail

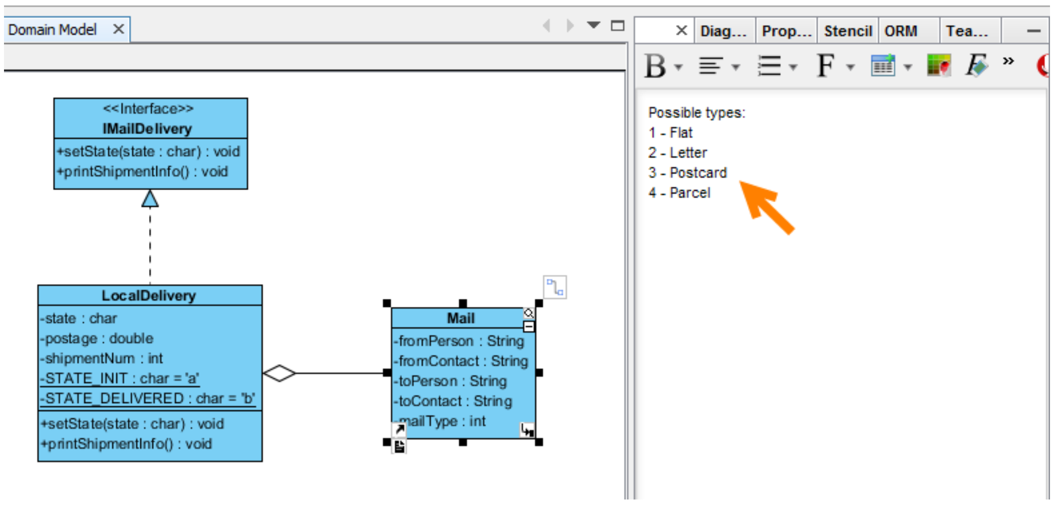
27. Follow the previous steps to create attributes in class *Mail*.

Class	Attributes	Operations
Mail	fromPerson : String fromContact : String toPerson : String toContact : String mailType : int	-

28. We need to describe the attribute *mailType* by listing the possible values. Select *mailType.*

29. Activate the **Description** pane. If you cannot find it on the screen, or if you had closed that, open it via the **View** menu. In the **Description** pane, enter the following:

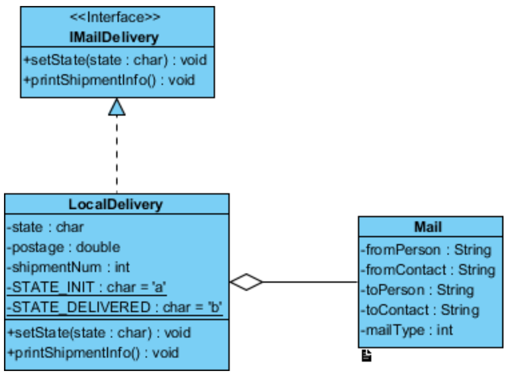
Possible types:
1 - Flat
2 - Letter
3 - Postcard
4 - Parcel



Possible types:
1 - Flat
2 - Letter
3 - Postcard
4 - Parcel

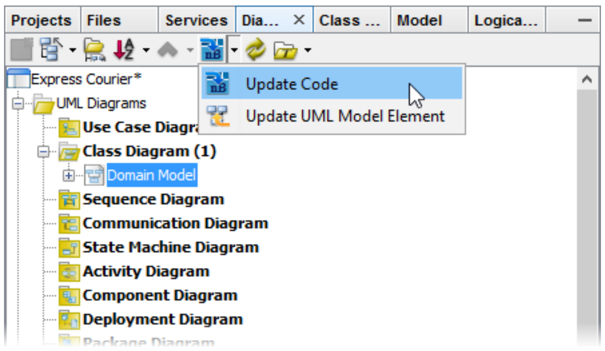
The description for classes, attributes, operations and parameters will become comments in source code to be generated.

30. Up to now, the diagram should look like:

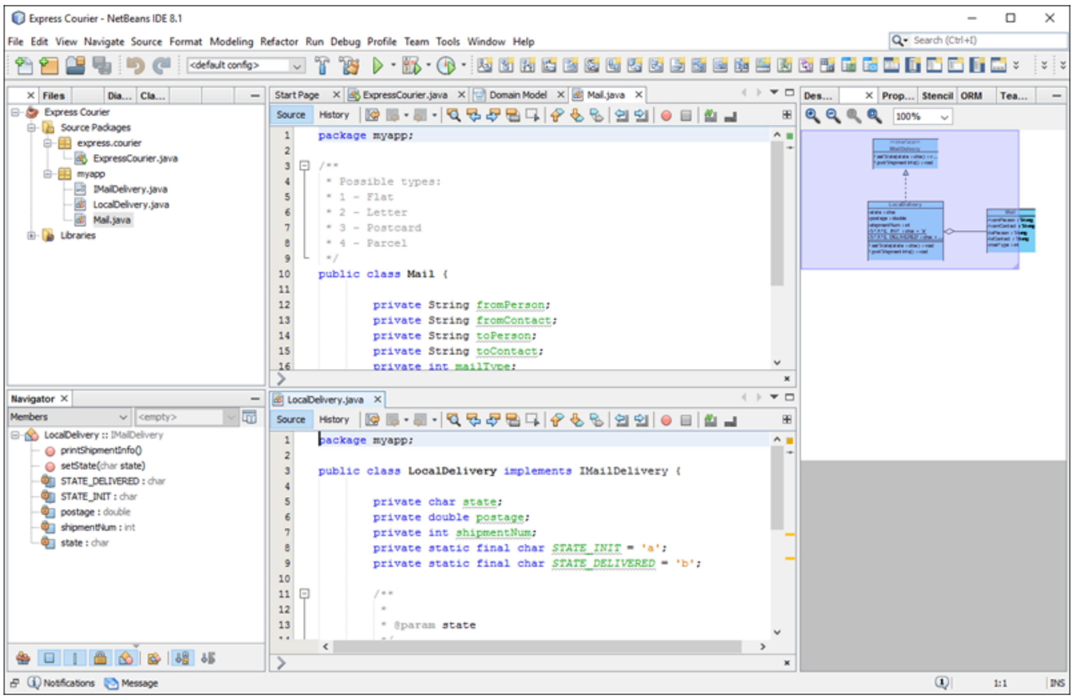


31. Save your work via the File menu.

32. Now comes the code generation. Select the menu **Update Code** on **Diagram Navigator**.



33. Check the **Projects** window. You should see a list of generated file. You can open them to fill in the code body.



```
package myapp;

/**
 * Possible types:
 * 1 - Flat
 * 2 - Letter
 * 3 - Postcard
 * 4 - Parcel
 */
public class Mail {

        private String fromPerson;
        private String fromContact;
        private String toPerson;
        private String toContact;
        private int mailType;
```

```
package myapp;

public class LocalDelivery implements IMailDelivery {

        private char state;
        private double postage;
        private int shipmentNum;
        private static final char STATE_INIT = 'a';
        private static final char STATE_DELIVERED = 'b';

        /**
         *
         * @param state
```

34. This is the end of the tutorial. Instead of closing NetBeans now, you may try something more by editing the code like to add, rename or delete class, attributes and operations, and click the **Update UML Model** button on toolbar, and observe the changes that will make in the class model. Enjoy!

Related Links

- Tutorial - Working with Hibernate in NetBeans
- Tutorial - Perform UML Modeling in NetBeans
- User's Guide - NetBeans Integration

Visual Paradigm

Visual Paradigm home page
(https://www.visual-paradigm.com/)

Visual Paradigm tutorials
(https://www.visual-paradigm.com/tutorials/)