

Developing Object-Oriented Systems: Refactoring an object model




The practical this week is an exercise in re-structuring a given Java application using Together Architect. The application is a simple commercial database: It provides interactive access to a database of customer names and details (in one panel of the window) and to a database of product names and details (in another panel of the window).

The given application comprises:

- A main class that is the "launch" mechanism: It creates the database and an interactive user interface window – both instances of *other classes*.
- A user interface class that offers the database functionality through drop-down choice lists, buttons and text fields. This is a `JFrame` window that contains two panels, one for the customer management and one for the product management.
- A database class: This contains arrays of strings for the customer names, customer details, product names and product details. The actual names of customers and products is fixed (five of each, for simplicity), but the details held about each are modifiable.

The key point, for this exercise, is that the database comprises two completely independent sub-databases. You are going to split up the database into two independent parts using a "delegation" architecture – that is, the database class itself will be retained as a "container", and the classes that depend on the database (the main and user interface classes) should not change at all...

1. Open the folder `CSCU9P6\Practicals in Courses` on the Divisional file server (mapped to drive K: in My Computer).
2. Open your T: drive. Somewhere convenient create a new folder for a Java project, call it `DualDatabase`. Inside it create a new folder called `src`.
3. Open and copy the contents of the folder `DualDatabase` on K: into the `src` folder that you created in step 2. Check and un-set the read-only attributes on the files if necessary.
4. As before, launch Together Architect via the **Start** icon and either type **Together** into the search box, and click on the **Together** that is found, or navigate via **All Programs** then **Borland Together** and then click on **Together**, close any project opened automatically, make sure that you are using the **Modelling** perspective, and create a new Together project for your `DualDatabase` folder using **File/New/Project/Modelling/Java Modelling Project**.
5. Compile, run and play and play with the application for bit. You select a customer or product name from either drop down list, and then either click to fetch the relevant details, or enter details into the text field and click to set new details.
6. Take a look around the code. The main class creates instances of the database and user interface classes (giving the user interface a reference to the database). All the display layout and event handling is in the user interface class. The database class contains arrays and access methods.
7. *As described above, the database class is actually two independent databases: there are arrays and access methods for customers and products – we assume that it simply grew like that, and we now realise that perhaps it should have been two separate classes, but it's rather late and we have already designed the rest of the system to deal with a single database object. The class is a candidate for splitting into two, retaining the main database class as a container that delegates service calls to its new components – see the next page.*

8. **In the class diagram**, select the Database class.
9. Using the *right* mouse button on the *class name*, “clone” the class **two times** using the **Clone** option in the pop-up menu. You can also clone an item by holding the **Ctrl** key down while you click-and drag it. The clones are automatically placed in new Java files, and given unique class names which you can change immediately or later, but their internal details are exact copies of the Database class.
10. If the classes are overlapping on the diagram, then you can either drag them to better locations, or let Together lay out the diagram for you either by selecting **Layout All** from the layout options available via the button that looks like this  in the toolbar just above the diagram pane, or via the **Diagram** menu, **Layout, Layout All**. You can also alter the zoom setting using these buttons just above the diagram pane  if you want to see more of the diagram at once.
11. Now carry out the following steps to complete the refactoring:
 - **In the class diagram**, rename the new classes to CustomerDB and ProductDB (click on the class name and then type the new name). Note how Together renames the classes’ constructors automatically, and how it also renames the Java files for the new classes. CustomerDB is going to hold everything to do with customers, ProductDB everything to do with products, and the original Database class is going remain only as a container for an instance of CustomerDB and an instance of ProductDB.
 - **In the class diagram** remove the *irrelevant attributes* and *operations* from both CustomerDB and ProductDB: **Select and delete all product related items from CustomerDB, and vice versa.**
 - **In the code editor pane**, you should *delete the irrelevant (and now incorrect!) lines in the constructors of the two new classes*: The customer database should not set up product data, nor vice versa. The lines to delete are probably highlighted red as syntax errors if you correctly removed the appropriate array attributes at the previous step!
 - **In the class diagram**, remove *all the attributes* from Database – since that data is now held in the two *new classes*. **Note** that you *must not* delete the **operations** from Database, as they are public and are referred to by UserInterface.
 - **In the class diagram**, add *aggregation associations* between the Database class and each of CustomerDB and ProductDB (that is, so that CustomerDB and ProductDB are *aggregated into* Database): There are two ways to do this, try both:
 - Add a plain association from Database to CustomerDB (by selecting the **Association** button on the palette on the left of the class diagram, then dragging from Database to CustomerDB), and then right-mouse click the association and select **Link type** from the pop-up menu, then **Aggregation**;
 - Select the **Link by Template** button from the palette to the left of the class diagram , drag from Database to ProductDB, and then choose **Aggregation** from the selection offered in the **Apply Template** dialogue.
 - With these extra associations, Together can now do a better job of laying out the class diagram automatically, so try **Layout All** again if you wish.
 - Together has automatically added two attributes to the code for Database for new the aggregation links – find them in the *code editor pane*. Together will have given them rather strange names starting with `lnk` – it uses those names to indicate to itself that they are not be displayed as actual attributes on the class diagram. Change their names to something sensible (such as `theCustomerDB`, `theProductDB`) in *either* the code editor pane *or* the **Properties** pane for the *link* (try both ways) – they will then appear as explicit attributes in the class diagram. [To activate the **Properties** pane for a link just click it.]

- **Via the class diagram**, use the **Properties** pane to set an *initial value* for the two new attributes in class `Database`: One should be a `new CustomerDB()` and the other a `new ProductDB()`: references to the new instances will be stored in the relevant attributes: To do this: simply click on an *attribute in the class diagram* and its properties will be displayed in the **Properties** pane, then locate the line for the *initial value* property, and type the expression to be assigned to the attribute, such as `new CustomerDB()` – with no semicolon after.
- **In the code editor pane**, alter the bodies of *all the methods* in `Database` to *delegate the calls* to the appropriate methods in the instances of `CustomerDB` and `ProductDB`: The basic idea is that each method `m` in `Database` will have a simple method body calling `theCustomerDB.m` or `theProductDB.m` as appropriate – you will need to be careful with parameters, and the results of non-void methods. Their entries in the class diagram will remain unchanged.

Note: Further, the `Database` constructor now has nothing to do as the arrays are elsewhere, so you can either delete it, or delete all the statements in its body.

- Check that everything works properly by compiling and running ...
- Identify the three statements in the `main` method of the `Main` class that set up the user interface window. Duplicate them so that you have *two* (or more) user interface windows on display, but sharing their database – you should be able to inspect and change the shared customer and product details through either/any window.

Checkpoint: Now show your refactored `DualDatabase` application to a demonstrator: Show them that after updating, say, a customers' details in one interface window, those details can then be fetched in another interface window. Explain how that works.

12. [Optional] An alternative way of carrying out the factorization above in `Together` is to create two new (empty) classes for `CustomerDB` and `ProductDB`, and to move/copy the components from within `Database` to the new classes by *dragging* (*Control-drag* to give a *copy* of what is being dragged). The string arrays can literally be *moved*, but the methods are best *copied* as you need to retain them as services in `Database`. You could try this, starting with a fresh copy of `DualDatabase`. [Observation: *Cloning*, as above, is a very convenient tool!]
13. [Optional] Start again with a fresh copy of `DualDatabase`, and try a refactoring in which you split the `Database` class into two separate classes, *without retaining Database as a delegating container* (so the `main` class *must instantiate two database objects and hand them both to the user interface*, which has extensive ramifications...). Clone `Database` once, rename both clones, delete the irrelevant parts from each, and go from there... Reflect on your experience...

That's all

SBJ January 2019