# Advanced Programming

Lab Sheet 20 – Swing Tables & Data

## Introduction

This lab gives you some additional practice in creating swing GUIs, this time using some UI components that were not discussed in the lecture. You are going to be creating an application that allows users to search data held in a SQLite database, viewing the results in a table. A large number of back-end business systems work in this manner, both in terms of desktop and web software.

The UK government publishes a dataset [1] detailing the number of single use carrier bags sold by large retailers since the introduction of charges for carrier bags in 2015. I have imported this data into a simple SQLite database, removing some unnecessary columns. You can download a copy of the database from moodle. The database contains a single table, with one row per retailer per tax year.

Your task is to create a swing application in which a user can enter the name of a retailer, click search, and view the number of bags sold by that retailer each year, along with a summary of the proceeds in a table.

## Architecting the Application

Keeping the code tidy on an application like this can be a little difficult. You will need code to access the database, code to create the UI, code to handle UI interaction, and code to represent the data being displayed inside the application. There are several popular code architectures for dealing with this type of application, notably Model View Controller (MVC) [2] and Model View ViewModel (MVVM) [3]. For this particular task, wouldn't recommend adopting any of these architectures strictly, and instead going with:

- A Database class, responsible for interacting with the SQLite database via JDBC, with methods that provide an easy to call signature (e.g. a method that accepts a company name string, and returns a List of BagSales records.
- A UI class, with a constructor that creates the UI, and with ActionListeners set up minimally to call separate methods in the class, for readability.
- A Main class, which will contain the application's main method, instantiating the UI class.
- A BagTableModel class to represent the data being displayed in the application's table, which we'll discuss in more detail later, along with a BagData POJO, representing the data in one row of the database or TableModel.

This architecture isn't all that far from MVVM, but we're doing a bit more work in the Model (Database) classes, and a bit less in the ViewModel (BagData & BagTableModel) classes than would be ideal for a true MVVM architecture.

## Performing Searches

You will implement this by adding an EventListener to your "Search" button in the GUI. When the user clicks the button, you will need to retrieve the text currently entered in the text field, and pass it to the database class's method for retrieving records by business name.

Because the database is just a single table that hasn't been optimised or normalised in any way, the SQL used will be a very simple SELECT, with the LIKE operator used to allow searches for "Tesco" to match the "Tesco Stores Ltd" in the database. Don't forget to use a parameterised query, so that a malicious user can't perform an SQL injection attack.

Your Database class should return data as an ArrayList of BagData objects. The BagData class is a simple POJO with properties, getters and setters for (e.g.) Company name, bags sold, proceeds, etc. Returning an ArrayList of these objects makes the next task easier...

## Displaying Data in a Table

The JTable component itself works in a similar manner to other JComponents. You can add it to a Container in the same way as a JButton or JfextField, and you can use the normal layout managers to control the way the table is displayed. The JTable does, however, require that the data it displays be passed to it in a particular format: a class that implements the TableModel interface, or extends the AbstractTableModel class (which implements it for us, in part). This is why I recommended using a BagTableModel class in the previous section.

### Extending AbstractTableModel

The TableModel interface requires implementers to have quite a few methods, most of which are to do with editing the contents of the table, which we don't need to implement. The AbstractTableModel features do-nothing implementations of these unnecessary methods, and is provided for our convenience when making TableModels. Our BagTableModel class will need an ArrayList of BagData objects to keep track of the data to be displayed in the table, while providing three methods required by its abstract parent:

- `public int getRowCount()`
- `public int getColumnCount()`
- `public Object getValueAt(int rowIndex, int columnIndex)`

The getRowCount() method will return the number of entries in the ArrayList, the getColumnCount() method will return a fixed integer: - the number of columns to display. The getValueAt() method will return one of the values of the BagData object in the ArrayList element specified by the rowIndex parameter. The value returned will depend on the columnIndex parameter, I'd recommend using 0= company name, 1 = bags sold, 2 = proceeds to start with.

You will also wish to implement the following, technically optional methods:

- `public String getColumnName(int column)`
- `public Class<?> getColumnClass(int columnIndex)`

The getColumnName() method will return the column heading associated with a particular column number. You can simply use some ifs or a switch statement for this, (e.g. 0 = "Company", 1 = "Bags Sold"). The getColumnClass() returns the class object associated with the datatype in the specified column. This method is simpler than it looks for this particular table: everything is a string, and so this method can always return String.class.

Your BagTableModel class will also need some means of setting the data in the ArrayList, I would recommend designing it such that the data is passed in via its constructor, but also providing a setData() method so to make it easy to update when the user performs searches.

### Using the TableModel in the JTable

You can create a BagTableModel with an empty ArrayList when creating the UI, and pass this object to the JTable in its constructor when setting up. This will mean that the application displays a blank table on start-up.

When the user performs a search, you will need to pass a fresh ArrayList (this time with data) to the BagTableModel, and call its (inherited) fireTableDataChanged() method, before doing the usual invalidate() and repaint() to request the UI updates.

If you would like to read some more comprehensive documentation on the use of JTables, the official Java documentation goes through the topic in detail here.

## Extension Tasks

There are a lot of ways in which you can extend your application. You could add some buttons to perform some pre-set searches (e.g. "Big Four" or "Big Six" supermarkets), you could add a button to view the top 5 single use bag sellers in a specified year, or you could add a report that displays the total number of single use bags sold by year over time, although this may involve fixing the database.

## References


1. Department for Environment, Food and Rural Affairs, "Single-use plastic carrier bags charge data for England", 2022, Available: https://www.data.gov.uk/dataset/682843a8-168c-4056-b6fe-741161a39f60/single-use-plastic-carrier-bags-charge-data-for-england
2. Mozilla Corportation, "MVC – MDN Web Docs Glossary: Definitions of Web-Related Terms", 2022, Available: https://developer.mozilla.org/en-US/docs/Glossary/MVC
3. D. Britch, N. Schonning, C. Dunn and J Osborne, "The Model-View-ViewModel Pattern", 2021, Available: https://docs.microsoft.com/en-us/xamarin/xamarin-forms/enterprise-application-patterns/mvvm