Client-side GUI

A simple Swing-gui for searching for proudcts

Working from a sketch to a rough GUI

- We make a list of the features / requirements
- We'll then start with a sketch of how a GUI for searching products could look
- We show how one could build such a gui with Swing
- We talk about where data comes from
- We talk about how to fake data
- We talk about how to be general so that we can switch to real data later

GUI functional requirements

- Display products in a table format
- Allow the user to filter products using input fields
  - min_price, max_price, min_alcohol, max_alcohol
- The GUI should update itself when the user enters filter values
  - Not implemented in this mock-up version, we'll focus on the GUI presentation
  - We'll even fake data for the GUI and the data won't change
  - But we'll create a Query object encapsulating a filtered request

Sketch

Title bar

Name	Price	Alcohol	Something else
Carlsberg	9.00	5.6	XYZ
Wine-oh-tintoh	37.00	10.5	BLEH
Loch Lomond	400.00	43.0	Haddock

Minimum price:		Maximum price:	
Minmum alcohol:		Maximum alcohol:	

What components can you find?

What components can you find?

- Window (JFrame)

What components can you find?

- Window (JFrame)
- Two areas
  - Table area
  - Input area

What components can you find?

- Window (JFrame)
- Two areas (JPanel)
  - Table area
  - Input area
- Table (JTable)
  - With table data (JTableModel)

What components can you find?

- Window (JFrame)
- Two areas (JPanel)
  - Table area
  - Input area
- Table (JTable)
  - With table data (JTableModel)
- Labels (JLable)
  - Four of them

What components can you find?

- Window (JFrame)
- Two areas (JPanel)
  - Table area
  - Input area
- Table (JTable)
  - With table data (JTableModel)
- Labels (JLable)
  - Four of them
- Text boxes for input (JTextField)
  - Four of them

What components can you find?

- Window (JFrame)
- Two areas (JPanel)
  - Table area
  - Input area
- Table (JTable)
  - With table data (JTableModel)
- Labels (JLable)
  - Four of them
- Text boxes for input (JTextField)
  - Four of them
  - They should update model (which updates the GUI)

GUI class layout

- instance variable declarations
- constructor (which could call)
  - init() - initialize the components
  - show() - make the GUI visible
- Helper methods
  - addListeners() - Add listeners to the text fields
  - newFilter() - Creates and filters the products using the text field values

Instance variables

```
private JFrame frame; // this is the actual window
private JPanel panel; // a panel is a surface to put other components on
private JPanel form;
private JTable table; // A table
// input fields for searching
private JTextField minAlcoField;
private JTextField maxAlcoField;
private JTextField minPriceField;
private JTextField maxPriceField;

private List<Product> products; // (to initiate the table model)
private ApiAccess api; // For talking to the REST API
```

JTableModel - We've written one for you

```
public class ProductTableModel extends AbstractTableModel {
  private String[] columnNames = { "Name", "Alcohol", "Price",
                                   "Volume", "cl alc per SEK"};
  private Object[][] data;

  public ProductTableModel(List<Product> products) {
    // initiate the data from the list...
  }

  // some overridden methods for column count, name, values etc
}
```

Idea behind JTableModel

- We can handle a List<Product> and present it as the data for a table
- We can replace the JtableModel when we filter on new products
- Focus now becomes how to filter data based on text field input
- Swing class becomes rather small
- about 126 lines of code in the GUI class
- about 60 lines of code in the table model class

How to fake the API code

Our design uses interfaces for flexibility:

- ApiAccess - interface for talking to the API (which can be a fake one)
- ApiAccessFactory - gives us an implementation (which can be fake)
- Query - interface encapsulating the query to send to an API
  - Holds a list of Param objects
    - A Param encapsulates a key-valu pair
- QueryFactory - gives us a Query implementation
  - can be a query for our REST API Servlet
  - could also be a query used for SQL or something else

Query

We'll let a `Query` represent a list of key-value pairs for constraints on the products to fetch. A `Param` represents such a key-value pair.

This works well for both queries to the REST API and for other types of queries, such as the `WHERE` clause of a `SELECT` statement.

Query parameters example: `min_price=40&max_price=100`

WHERE clause example: `price >= 40 AND price <= 100`

Using an interface for Query allows for both kinds of implementations.

How to build a Query from the GUI

Using the addParam(Param) method of a Query, we can build a Query from the GUI by looking at the text entered into the JTextFields:

```
Query query = QueryFactory.getQuery();
if (!"".equals(minAlcoField.getText())) {
  query.addParam(new Param("min_alcohol", minAlcoField.getText()));
}
if (!"".equals(maxAlcoField.getText())) {
  query.addParam(new Param("max_alcohol", maxAlcoField.getText()));
}
if (!"".equals(minPriceField.getText())) {
  query.addParam(new Param("min_price", minPriceField.getText()));
}
if (!"".equals(maxPriceField.getText())) {
  query.addParam(new Param("max_price", maxPriceField.getText()));
}
```

The ApiAccess object can use that Query in fetch()

```
private ApiAccess api; // instance variable


// in e.g. constructor:
api = ApiAccessFactory.getApiAccess();


// in e.g. a Listener for the text fields:
products = api.fetch(query);
// Update the JTable:
table.setModel(new ProductTableModel(products));
```

Listener for the `JTextFields`

A DocumentListener reacts to change, remove and update of a document.

A JTextField has a Document. We can add a DocumentListener to the JTextFields.

Code for adding listeners to the text fields

```
private void addListeners() {
  for (JTextField textField : textFields()) {
    textField.getDocument()
      .addDocumentListener(new DocumentListener() {
          public void changedUpdate(DocumentEvent e) {
            newFilter(); // updates the model
          }
          public void insertUpdate(DocumentEvent e) {
            newFilter(); // updates the model
          }
          public void removeUpdate(DocumentEvent e) {
            newFilter(); // updates the model
          }
        });
  }
}
```

A small example building a Query from a GUI

For a small example, se the files

examples/QueryBuilder.java (main class)

examples/TextFieldDemo.java (GUI class)

(Links will be added)

```
// QueryBuilder.java
package examples;

public class QueryBuilder {
  public static void main(String[] args) {
    javax.swing.SwingUtilities.invokeLater(new Runnable() {
        public void run() {
          TextFieldDemo gui = new TextFieldDemo();
        }
      });
  }
}
```

```
// TextFieldDemo.java
package examples;

import java.awt.*;
import java.awt.event.*;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import javax.swing.*;
import javax.swing.event.*;

import se.itu.systemet.domain.Product;
import se.itu.systemet.rest.ApiAccess;
import se.itu.systemet.rest.ApiAccessFactory;
import se.itu.systemet.rest.Param;
import se.itu.systemet.rest.Query;
import se.itu.systemet.rest.QueryFactory;
```

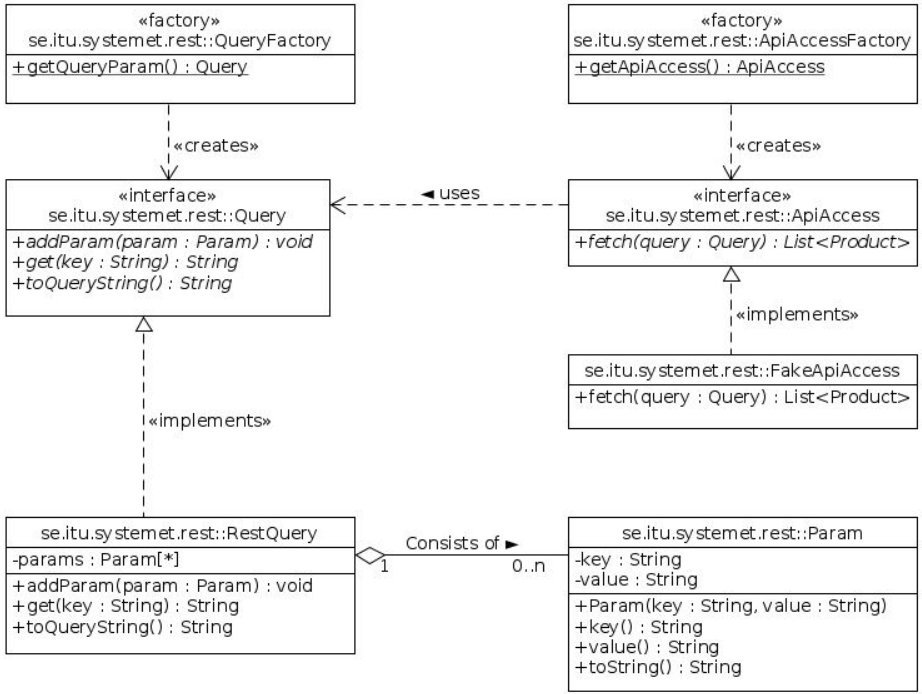
```
// TextFieldDemo.java - continued
public class TextFieldDemo {
  // Instance variables below - mostly Swing components for the UI
  private JFrame frame; // this is the actual window
  private JPanel form;
  private JTextField minAlcoField;
  private JTextField maxAlcoField;
  private JTextField minPriceField;
  private JTextField maxPriceField;
  private JLabel resultLabel;

  private List<Product> products;
  private ApiAccess api; // For talking to the REST API

  public TextFieldDemo() {
    api = ApiAccessFactory.getApiAccess();
    products = api.fetch(QueryFactory.getQuery());
    init(); // Initiate the components
    show(); // Show the frame
  }
```

```
// TextFieldDemo.java - continued
  private void init() {
    frame = new JFrame("Search products");
    frame.setLayout(new BorderLayout());
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    minPriceField = new JTextField(20);
    maxPriceField = new JTextField(20);
    minAlcoField = new JTextField(20);
    maxAlcoField = new JTextField(20);
    form = new JPanel(new GridLayout(2,4));
    form.add(new JLabel("Minimum alcolhol:"));
    form.add(minAlcoField);
    form.add(new JLabel("Maximum alcolhol:"));
    form.add(maxAlcoField);
    form.add(new JLabel("Minimum price:"));
    form.add(minPriceField);
    form.add(new JLabel("Maximum price:"));
    form.add(maxPriceField);
    resultLabel = new JLabel("Query result");
    frame.add(form, BorderLayout.CENTER);
    frame.add(resultLabel, BorderLayout.SOUTH);
    addListeners();
  }
```

```
// TextFieldDemo.java - continued
  private void addListeners() {
    for (JTextField textField : textFields()) {
      textField.getDocument()
        .addDocumentListener(new DocumentListener() {
            public void changedUpdate(DocumentEvent e) {
              newFilter();
            }
            public void insertUpdate(DocumentEvent e) {
              newFilter();
            }
            public void removeUpdate(DocumentEvent e) {
              newFilter();
            }
          });
    }
  }
```

```
// TextFieldDemo.java - continued
  /* Return the JTextFields as a List */
  private List<JTextField> textFields() {
    List<JTextField> textFields =
        Arrays.asList(minPriceField, maxPriceField, minAlcoField, maxAlcoField);
    return textFields;
  }

  private void show() {
    frame.pack();
    frame.setVisible(true);
  }
```

```
// TextFieldDemo.java - continued
  private void newFilter() {
    Query query = QueryFactory.getQuery();
    if (!"".equals(minAlcoField.getText())) {
      query.addParam(new Param("min_alcohol", minAlcoField.getText()));
    }
    if (!"".equals(maxAlcoField.getText())) {
      query.addParam(new Param("max_alcohol", maxAlcoField.getText()));
    }
    if (!"".equals(minPriceField.getText())) {
      query.addParam(new Param("min_price", minPriceField.getText()));
    }
    if (!"".equals(maxPriceField.getText())) {
      query.addParam(new Param("max_price", maxPriceField.getText()));
    }

    // If we wanted to fetch new products, we could do this:
    products = api.fetch(query);
    // Show the query string
    resultLabel.setText(query.toQueryString());

  }

} // end class
```



* The GUI creates a Query object
* The GUI adds Param objects to the Query
* The GUI creates an ApiAccess object
* The GUI sends the Query to an API using the ApiAccess object