Lab 11 – Sending Email

1 Sending the sale receipt confirmation email

It is always a good idea to use a fake server when testing email code in case you accidentally send something silly to a real server which might get you or your entire company added to a black-list. It also gives you the opportunity to review and check the completed emails before you actually send them to people.

The following Java based fake SMTP server works very well and should work on all platforms:

`http://nilhcem.github.io/FakeSMTP/`

The easiest way to send email in Java is to use a combination of *JavaMail*, and *commons-email*.

`https://javaee.github.io/javamail/`
`http://commons.apache.org/proper/commons-email/`

JavaMail is very powerful, but has a complicated and verbose API. *commons-email* is a facade that makes JavaMail much easier to use.

The *commons-email* user guide has the code that you need to send a simple message. You should use *localhost* as the host name, and 2525 as the port (make sure that you use also 2525 when starting FakeSMTP too). You will **not** need to use an authenticator, or SSL, since FakeSMTP does not require either, and there is no point in making things more complicated than they need to be at this point.

You should first create a main method that sends a hard-coded email message to test that you are able to send simple messages before trying to integrate the email code into the *check out* process of your web application.

For the real message, you should extract the details from the customer, sale, and sale item objects to create a suitable message that describes the contents of the sale for the customer.

The Gradle dependency that you need is:

```
compile group: 'org.apache.commons', name: 'commons-email', version: '1.5'
```

All of the information that you need is in the user guide for commons-email at the site linked above.

2 Threading the email sending

Connecting to an SMTP server can often take a while, particularly if you are using an off-site email provider. We should insulate our web application from delays caused by a slow connection, otherwise the users will be waiting for a long time for the checkout process to complete before they are redirected to the next page.

The way to do this is to wrap the email code in a thread.

There are many way to do this in Java. The easiest way is to use a **`CompletableFuture`**, and run it using the **`runAsync`** method:

```
CompletableFuture.runAsync(() -> {
   // code to send email goes here
});
```

Once you are happy that your email sending code is working you can copy it into the in your Jooby sale module so that it sends the email after saving the sale in the database.

When a customer checks out their shopping cart the email should be send to their registered email address.