

Criterion C: Development

List of techniques:

1. Adding data to databases wrapped in an ArrayList
2. Deleting data from databases wrapped in an ArrayList
3. Searching for specific data from databases wrapped in an ArrayList
4. Overloading
5. Inheritance
6. Encapsulation
7. Parsing a string into integers
8. Implementing a hierarchical composite data structure, such as the TaxRulesDb, which consists of default tax ruleset, and custom tax ruleset. Both rulesets are instances of tax ruleset. The custom tax ruleset extends the tax ruleset by having a client ID.
9. Use of additional libraries (for arraylist in java.util.ArrayList and GUI design with SWT in org.eclipse.swt.SWT).
10. ArrayLists

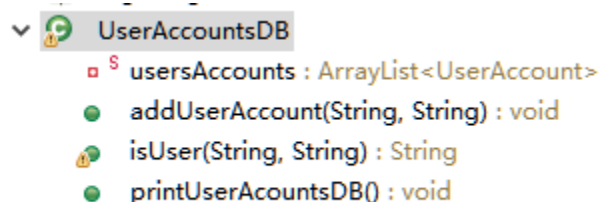
This program is written in the language of java, developed on the Eclipse IDE, with the windowBuilder addon based on swt library for ease of GUI development.

The program is to provide a personal income tax calculator. Given the extreme complexity of tax rules in the real world, the current development cycle is aimed to provide a framework with simplistic tax rules, which is expandable for later development.

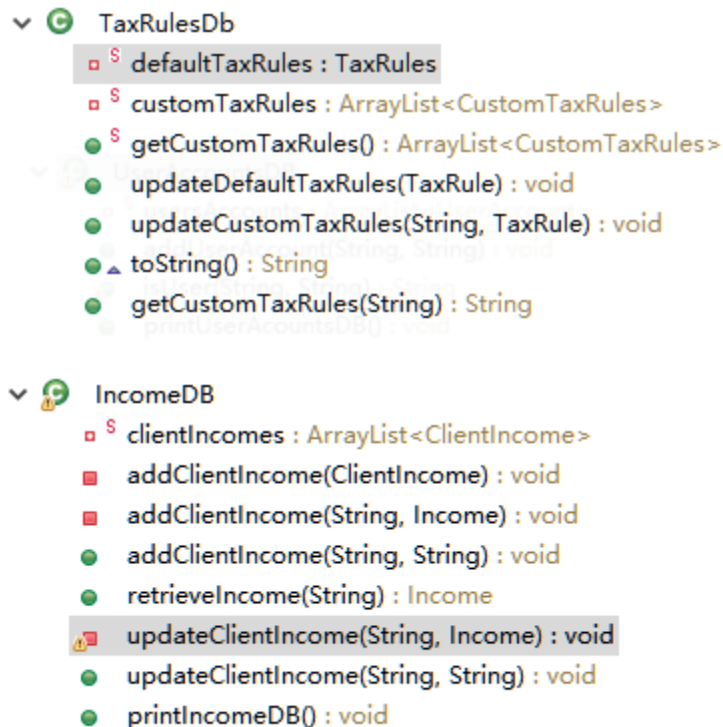
Persistent Data encapsulated in classes

For expandability, all the functions are encapsulated into classes, down to the persistent data involved. Ideally, the persistent data should be managed by a production-grade SQL database, such as Oracle or MySQL. However, at the initial stage, persistent data is stored in ArrayLists, which is included in standard Java. Besides, the databases are encapsulated in classes, so that it is easy to move them into a SQL database.

There are three databases (i.e.DB hereafter) involved: UserAccountsDB, IncomeDB, and TaxRuleDB.



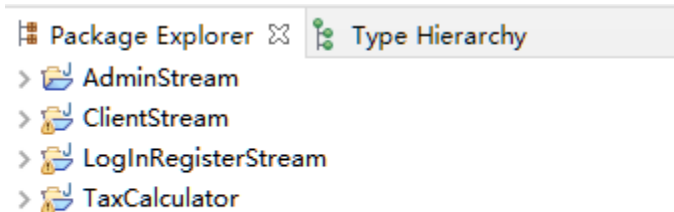
```
▼ UserAccountsDB
  □ S usersAccounts : ArrayList<UserAccount>
  ● addUserAccount(String, String) : void
  ● isUser(String, String) : String
  ● printUserAccountsDB() : void
```



Processes/Functions in several packages under the top project

The processes of the program are divided into three streams, each in a separate package, for ease of development and testing.

The three streams are defined in the classes of LoginRegisterStream, ClientStream, and AdminStream. The TaxCalculator class schedules among the three streams, and maintains the databases for all the streams.



Please note that the screenshot shows the three streams as separate projects, instead of being subjugated under TaxCalculator, for ease of testing. After testing at package levels is done, they are dragged under TaxCalculator for distribution.

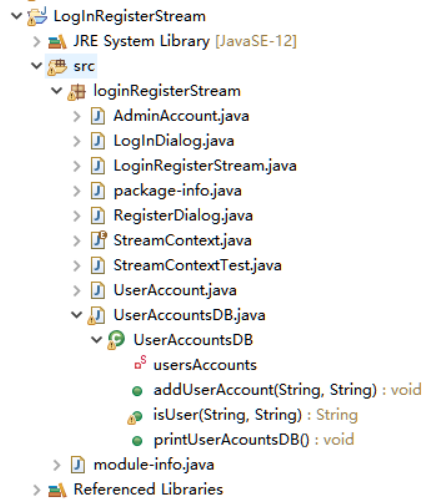
Outline of the whole process in TaxCalculator

LoginRegisterStream sets the global variable streamContext to one of the enum values of NOT_DEFINED, ADMIN_STREAM, or CLIENT_STREAM. Depending on the value of the streamContext, the TaxCalculator would exit, instantiate the AdminStream, or instantiate the

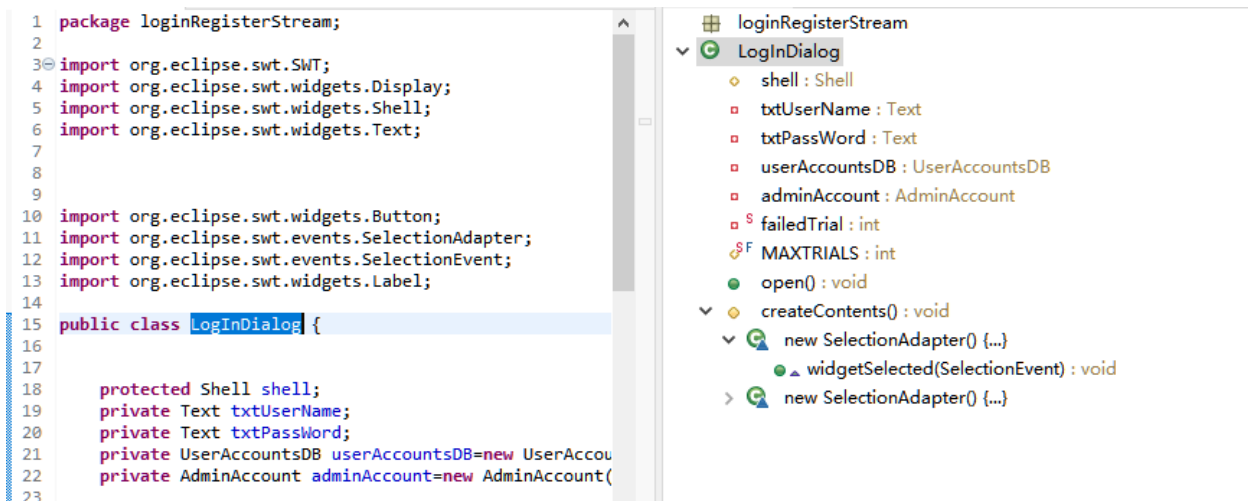
ClientStream, respectively. If a client logs in, LoginRegisterStream also sets the global variable userID, which ClientStream uses to retrieve data for the user identified by the userID.

LoginRegisterStream with LoginDialog and RegisterDialog

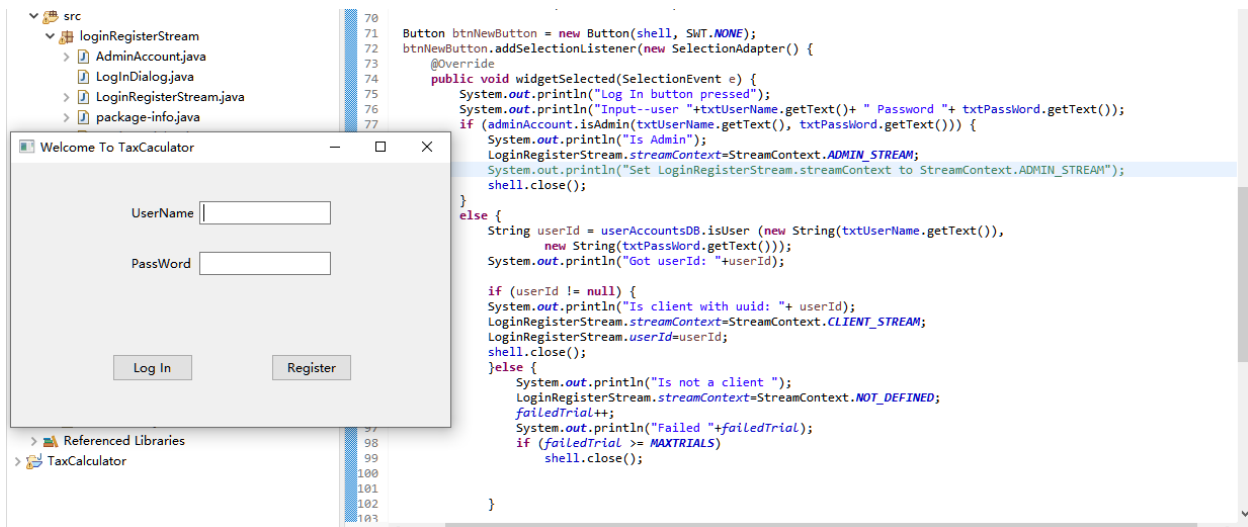
The LoginRegisterStream contains two dialogs of LoginDialog and RegisterDialog. _____



The two dialogs of LoginDialog and RegisterDialog were developed with the help of the WindowBuilder in the IDE and import the swt package.

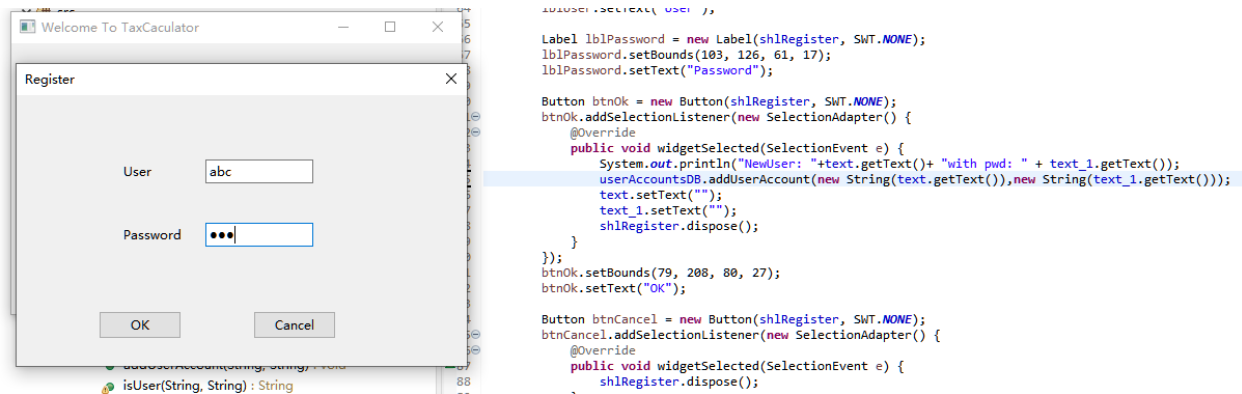


The failedTrial shown is to remember the failures of login attempts. At MAXTRIALS shown, the LoginDialog would close, while setting the global variable of streamContext to NOT_DEFINED.



Please note that the code in “btnNewButton.addSelectionListener(new SelectionAdapter() {})” is connected with the clicking event of the “Log In” button.

The Register button selection would open the RegisterDialog.

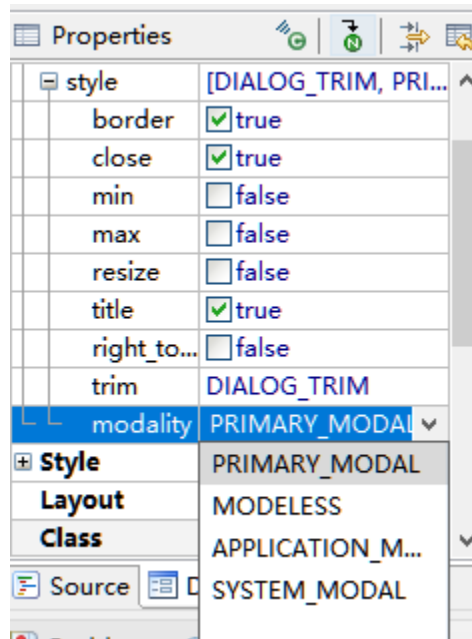


On invocation of the RegisterDialog, the LoginDialog is deactivated, until the RegisterDialog is exited. The password input is also masked. Both of these features are readily done in the WindowBuilder GUI. For instance, the Deactivation of LoginDialog is achieved by setting the modality of the RegisterDialog to PRIMARY_MODAL, which corresponds to code auto-generated by the WindowBuilder.

```

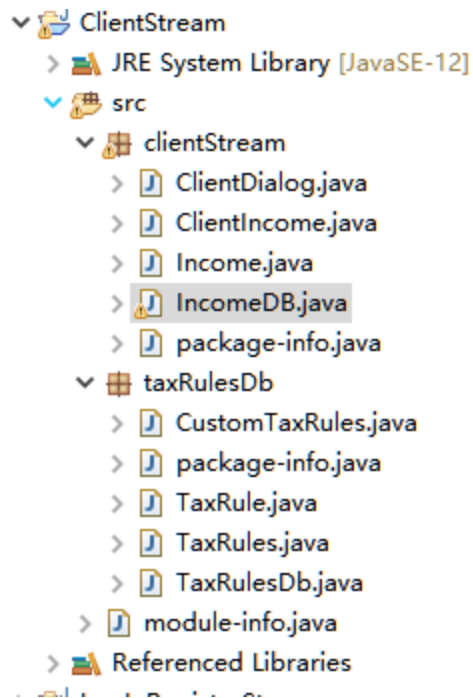
/**
 * Create contents of the dialog.
 */
private void createContents() {
    shlRegister = new Shell(getParent(), SWT.DIALOG_TRIM | SWT.PRIMARY_MODAL);
    shlRegister.setSize(450, 300);
    shlRegister.setText("Register");
}

```

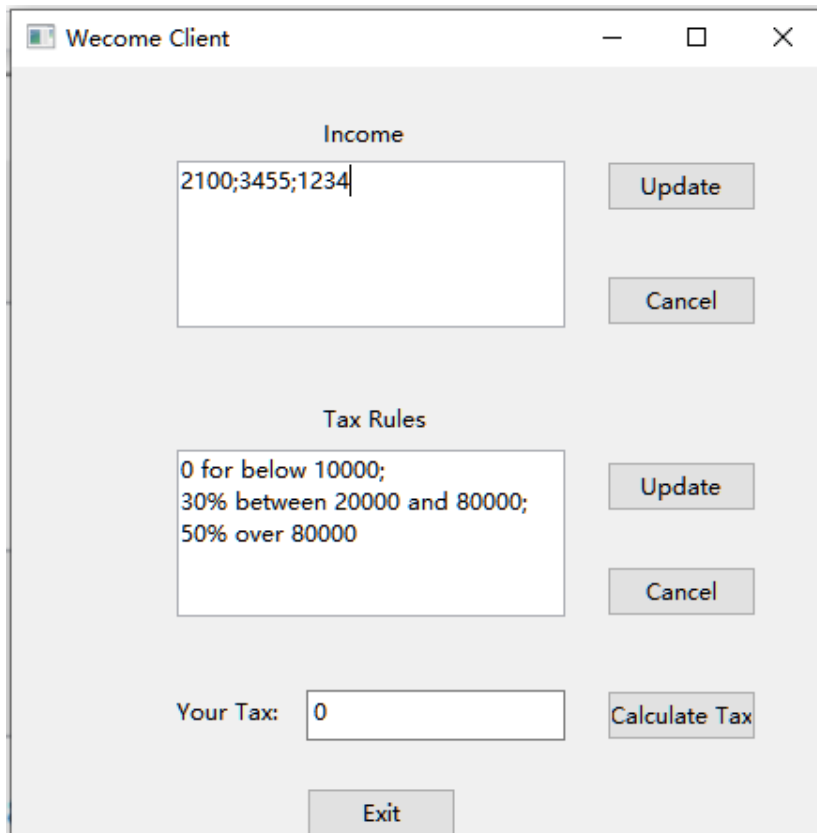


ClientStream calculating tax for a client based on IncomeDB and TaxRuleDB

The ClientStream calculates tax based on the client's income and tax rules are retrieved respectively from IncomeDB and TaxRuleDB. For the initial stage, the IncomeDB and TaxRuleDB are inside the ClientStream package, for the sake of convenience. Eventually, they will be moved into another DB package together with UserAccountsDB. Assembling all the DB classes into one package will facilitate portability into production-grade DB.



The ClientDialog shows the calculated tax.



The two update buttons will take the income and tax rules in the text boxes besides them and update them into IncomeDB and TaxRulesDB respectively.

```

    Button btnNewButton = new Button(shell, SWT.NONE);
    btnNewButton.addSelectionListener(new SelectionAdapter() {
        @Override
        public void widgetSelected(SelectionEvent e) {
            String s = styledText.getText();
            System.out.println("incometext box is" + s);

            incomeDB.updateClientIncome(clientId, styledText.getText());
            System.out.println("incomeDb is");
            incomeDB.printIncomeDB();

        }
    });

```

AdminStream still under development

Besides showing some statistics, the major work in AdminStream is to transfer human readable tax rules into backend rules.

```

5 public class TaxRule {
6     private String readableRule;
7     private String backendRule;
8

```

The backend rule is to do the actual tax calculation. At the current stage, the tax calculation is hard coded into the “public int apply(Income alIncome)” method of TaxRule. Ideally, the AdminDialog in AdminStream should allow administrators to translate the readableRule into backEndRule.

Overloading examples

There are two constructors for TaxRule, to accommodate the situation when a client inputs only the human readable rule, which the administrator will translate later.

```

9 public TaxRule(String r) { // only readable rules
10     readableRule = r;
11     backendRule = r;
12 }
13
14 public TaxRule(String r, String b) {
15     readableRule = r;
16     backendRule = b;
17 }

```

The IncomeDB has three addClientIncome methods, which build on top of each other.

```
private void addClientIncome(ClientIncome e) {  
    clientIncomes.add(e);  
}  
  
private void addClientIncome(String userId, Income income) {  
    addClientIncome(new ClientIncome(userId, income));  
}  
  
public void addClientIncome(String userId, String incomeString) {  
    System.out.println ("Before adding income for "+userId+incomeString);  
    Income i = new Income(incomeString);  
    System.out.println(i.toString());  
    addClientIncome(userId, i);  
}
```

Word Count: 789