Running the Program:

Locate the folder that contains the .jar file in your file explorer. Once you’ve located the file double click it to start the program.

Running the Program From the Command Line

**java -jar GroupProj-1.0-SNAPSHOT-jar-with-dependencies**

The way you would add another language into this project:

1) Create an abstract class that implements GeneralConvertCreateDDL

- This means it will have the methods

- Initialize - which initializes the bounds of your table

- getTable(int numFigure) - returns the table at specified index

- getField(int numFigure) - returns the field at specified index

- Make sure it has a constructor(tables, fields)

- Constructor calls initialize() method, and sets the tables and fields.

- Other methods to add to your abstract class:

- public abstract String getDatabaseName() - this return the name of the database

- public abstract String getProductName() - this is the type of database being used

- public abstract String getSQLString() - runs the createDDL(), then returns a string of the StringBuffer createDDL() makes

- public abstract void createDDL() - This is the method that is responsible for building the sql file

2) Make a class that "EXTENDS" the abstract class you just made

- This means it will have instance variables

- String databaseName

- String[] strDataType = {"VARCHAR","BOOL","INT","DOUBLE"}

- StringBuffer sb

- This means it will have the methods

- Constructor with parameters ( YOUR\_TABLE\_CLASS, YOUR\_FIELD\_CLASS )

- Blank constructor to set output directory

- public void createDDL() - this creates the database

- public int convertStrBooleanToInt(String input) - checks if input == true

- public String generateDatabaseName() - helper method for createDDL

- public String getDatabaseName() - helper method that returns type of DB

- public String getProductName() - helper method that returns type of product

- public String getSQLString() - creates the DDL and returns the Stringbuffer toString()

3) Make New Field and Table Objects

- This means implement GeneralTable, for table object, and will have methods

- public int getNumFigure - Gets num figure instance variable

* public String getName - Gets name instance variable
* public void addRelatedTable(int relatedTable) - sets the related tables arraylist to the

table corresponding with the integer

* public int[] getRelatedTablesArray() - gets the array instance variable of related tables
* public int[] getRelatedFieldsArray() - get the array instance variable of related fields
* setRelatedField(int index, int relatedValue) - sets the relation between fields
* getNativeFieldsArray() - returns the instance variable native fields
* addNativeField(int value) - takes the integer and add's it to the natvive fields arraylist
* moveFieldUp(int index) - move the specified index up one position in the arraylist
* moveFieldDown(int index) - move the specified index down one position in the arraylist
* makeArrays() - Makes the arrays of native fields and tables by converting arraylists to

int[]

* toString() - Makes the toString a user friendly readable string

- This means implement GeneralField, for field object

* public int getNumFigure() - Gets the length of the field
* public String getName() - Gets the name of the field
* public int getTableID() - Gets the table's ID
* public void setTableID(int value) - Sets the tables ID
* public int getTableBound() - gets the tables boundary
* public void setTableBound(int value) - sets the tables boundary
* public int getFieldBound() - Gets the field boundary
* public void setFieldBound(int value) - sets the fiels boundary
* public boolean getDissallowNull() - gets the disallow value
* public void setDisallowNull(boolean value) - sets the disallow value
* public boolean getIsPrimaryKey(boolean value) - checks if value is the primary key
* public void setIsPrimarykey(boolean value) - sets this field as a primary key
* public String getDefaultValue() - gets the default value of the field
* public void setDefaultValue(String value) - sets the default value of the field
* public int getVarcharValue() - gets the varchar value
* public void setVarcharValue(int value) - sets the varchar value = value
* public int getDataType() - gets the data type of the field
* public void setDataType(int value) - sets the data type of the field
* public String toString() - toString value for the field, in a human readable format

4) Make a file that sets up your tables and fields, for an example, look at "CreateFieldsAndTables.java"

5) Make a file parser for your input file

- You'll need to add a method for parsing your file type ( Example: parseNoSQLField() )

- You'll need to add a method for getting your Tables, and another for getting your fields

6) Make your connector, for determining endpoints of your database

- Implement GeneralConnector

- You must also need a constructor for your connector, ExampleConnector(String inputString)

- This will be a place where you parse the input string for data you store it in your instance variables for your file parsers and createDDL classes to use

Adding a new input file type to the GUI:

1. First, you will need to go into the **EdgeConvertGUI** java file and find the constructor. Look for the string array **supportedFileTypes** and change the array size to match the total number of file types you want to support. You will also need to add the file extension to the array as a string.

Example: **supportedFileTypes[0] = "edg";**

1. Next, we will be creating an **ExampleFileFilter** for the new input type. At the top of the program where the declarations are, look for the declarations for **ExampleFileFilter** and add the new type as a variable.

Example: **private static ExampleFileFilter effEdge, effSave, effClass;**

1. Next, we need to initialize our new variable. Within the **createDTScreen()** method look for the section where **jfcInput** is initialized and add a line to initialize the new variable. Example: **effEdge = new ExampleFileFilter("edg","Edge Diagrammer Files");**
2. Next, we will need to add our new input type to the switch case found within the **EdgeMenuListener** class. Add a new case using the extension type string we used earlier.

Example:

**switch(response){**

**case "edg":**

**jfcEdge.addChoosableFileFilter(effEdge);**

**break;**

**}**

1. The “edg” case provides an example of what methods and variables need to be called/accessed to successfully load in the new input file. This includes:

* Create a file parser for your new input type (See step 5 in the previous section).
* Parsing the selected file with the file parser
* Using the file parser to get the tables and fields
* Calling the **populateLists()** method
* Calling **setEnabled(true)** on **jmiDTSave**, **jmiDRSave**, **jmiDTSaveAs**, **jmiDRSaveAs**, **jbDTDefineRelations**, **jbDTCreateDDL**, and **jbDRCreateDDL**.
* Truncating the file name and using the truncated file name to set the title for **jfDT** and **jfDR**.
* Set **dataSaved** **= true;**

Fixed Bugs:

* EdgeConvertGUI, EdgeConvertFileParser; NullPointerException occurs after selecting an invalid file type when doing File > Open Save File > (invalid file type) --Fixed by Will
* EdgeConvertGUI; NullPointerException occurs when the program is closed without saving changes --Fixed by Will
* EdgeConvertGUI; If user attempts to save changes to the edge file when that file already exists, even if user cancels the file gets overwritten --Fixed by Will
* EdgeConvertGUI; When attempting to cancel out of choosing an output path it won’t allow you to cancel and gets stuck on that screen -- Fixed by Campbell: CreateDDLButtonListener in EdgeConverterGUI
* EdgeConvertGUI; Removed , null’s on all the varargs errors to fix the compile of EdgeConverterGUI --Will
* MySQL file now compiles in the command line

Changes made to support refactoring:

I added 5 interfaces, one for fields, tables, ConvertDDL, and a connector. I also adjusted the classes to use these interface superclasses instead of using the specific class. An example of this is, when edge field is used, I replaced it with general field.

The GUI has been updated to make it easier to add new types of input files. The menu option “Open Edge File” has been changed to “Open Input File”, which when selected, will bring up all the possible input options that the user can choose from. For more information on implementing new input types please refer to the section “Adding a new input file type to the GUI”.

KNOWN ERRORS:

Setting the output definition file just doesn’t work in the jar. If you look at the error message, it will say “Class not found” but that class appears right above the error. Works in the java code if you just run the file RunEdgeConvert.java.