**Course Project**

**DeVry University**

**College of Engineering and Information Sciences**

**Course Number: CIS355A**

# Module 5: Read and Write to Files

# Objectives

* Write data from the GUI to a file.
* Read data from the file and display the data on the GUI application.
* Use a three-tier architecture to handle large development projects.

We can add Customer objects to our application and display the details as desired. This application looks great and it is very useful. The only problem is that the application must run 24 hours per day, every day of the year. If our application closes, on purpose or by accident, we will lose all our data. All our customers’ information will be gone.

How can we make our data persistent so we will have our data even if the application is turned off? We need to write our data to a file or, more commonly, to a database. This week, we are going to learn to read and write to a file. This way, if we turn off our application, we can turn it back on the next day and simply load our data (or set it to automatically load).

In industry, applications are created using a three-tier architecture. This design style is critical so teams of developers can work together on the same application. The tiers are as follows.

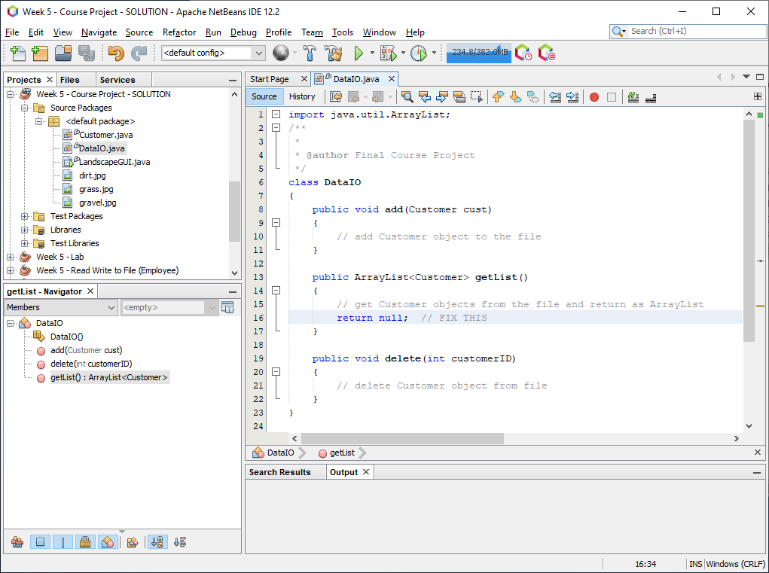
* Presentation: GUI classes read info from the user and displays info to the user.
  + The GUI classes should do all of the communication with the user.
  + Never use System.out.println or JOptionPane in your other classes.
* Business rules handle data processing and organization for the application.
  + Classes to build objects or do calculations are created here.
* Data storage use a separate class to read and write to the file or database.

Let’s make our data persistent.

# Steps

1. Open your Visio Class diagram from our design phase. We need to create the DataIO class. Remember, DataIO stands for data input and output. We will use this class to get data from the file as well as write data out to the file. Right-click on the default package and create a new class called DataIO. We do not need the database credentials at the top of the diagram, but we do want to add the methods from the bottom section of our UML class diagram.  
     
   

Skip this middle part for now.

1. Once you finish, the DataIO class should look something like this.  
     
   
2. Let’s create the add( ) method so we can write the Customer object to the file. In order to write data to a file, we need to open the file, write the data, and then close the file. When you write to the file, the drive can be full, or we may have read only permission (and not write permission). Because errors can take place when we write data, we need throw the error so the GUI can show the error to the user. Remember, the GUI classes should be the only classes to communicate with the user. Once you finish writing the code, your add( ) method should look something like this.

public void add(Customer cust) throws IOException

{

// open or create the file object using true to "APPEND" the data

BufferedWriter outfile = new BufferedWriter(new FileWriter("Customers.txt", true));

// write the object's data to the file on one line using # as separators

outfile.write(cust.getName());

outfile.write("#" + cust.getAddress());

outfile.write("#" + cust.getYardType());

outfile.write("#" + cust.getLength());

outfile.write("#" + cust.getWidth());

outfile.write("#" + cust.getTotalCost());

outfile.newLine(); //"enter" key

//close the file

outfile.close();

}

1. Let’s test the code. Save All and then open your GUI class. Click on the information tab. Right-click on the Submit Order button and choose Events 🡪 Action 🡪 actionPerformed. Right-click on the submitOrder( ) method and choose Navigate 🡪 Go To Source. Right after the code that displays the customer’s details, write the code to save the Customer object to the file. In other words, create a DataIO object and call its add( ) method. Remember, you need to put the code in a try-catch block in case something goes wrong and we need to show the error to the user. Once you finish writing the code, it should look something like this.

private void submitOrder()

{

if (validateInputs() == false)

{

return; // end the method if validation failed

}

Customer cust = createCustomer();

customerList.addElement(cust);

txaOrderInfo.setText(cust.getDetails());

try

{

DataIO data = new DataIO(); // create DataIO object

data.add(cust);

// reset for the next customer

reset();

//move to the client orders tab

tabMain.setSelectedIndex(2);

}

catch (IOException ex)

{

JOptionPane.showMessageDialog(this, "Error: " + ex.getMessage(),

"File IO Error", JOptionPane.ERROR\_MESSAGE);

}

}

1. Create two or three Customer objects and hit the submit order button for each one. Open the Windows file explorer. Navigate to the project’s root folder. Do you notice a file called Customers.txt? Double-click on the file and you will see that your customer’s information has been saved to the file.
2. Go to your DataIO class. In order to read data from a file, we need to open the file, read the data using a while loop, and then close the file. When you read from a file, the drive can be corrupted, or we may not have permission to read the file. Because errors can take place when we read data, we need throw the error so the GUI can show the error to the user. Remember, the GUI classes should be the only classes to communicate with the user. Once you finish writing the code, your getList( ) method should look something like this.

public ArrayList<Customer> getList() throws IOException

{

// get Customer objects from the file and return as ArrayList

//create an arraylist

ArrayList<Customer> customers = new ArrayList<Customer>();

// read data from the file

BufferedReader inbuffer = new BufferedReader(new FileReader("Customers.txt"));

StringTokenizer tokens;

//get first line

String inputString = inbuffer.readLine();

while (inputString != null)

{

//break the line into pieces using Tokenizer

tokens = new StringTokenizer(inputString, "#");

//fields are name#address#yardType#length#width#totalCost

String name = tokens.nextToken();

String address = tokens.nextToken();

String yardType = tokens.nextToken();

double length = Double.parseDouble(tokens.nextToken());

double width = Double.parseDouble(tokens.nextToken());

double totalCost = Double.parseDouble(tokens.nextToken());

// create Customer object and add it to the ArrayList

Customer cust = new Customer(0, name, address, yardType, length,

width, totalCost);

customers.add(cust);

//read next line

inputString = inbuffer.readLine();

}

// close the pipe to the file once the records have been read

inbuffer.close();

// return the ArrayList

return customers;

}

1. Save All and then switch over to your GUI. Click on the customer list tab. Right-click on the load list button and choose Events 🡪 Action 🡪 actionPerformed. Write this method call in the event method.

loadCustomers( );

1. Let NetBeans create the method for you or scroll to the bottom and create the method by hand. Now, write code that creates a DataIO object and calls the getList( ) method. The getList( ) method returns an ArrayList. Clear the DefaultListModel and the txaOrderInfo. Add the ArrayList data to the DefaultListModel. When you finish, the loadCustomers( ) code should look something like this.

try

{

DataIO data = new DataIO(); // create DataIO object

ArrayList<Customer> customers = data.getList();

// clear out the DefaultListModel and textarea

customerList.clear();

txaOrderInfo.setText("");

// copy each object from the ArrayList over to the DefaultListModel

for( int i = 0; i < customers.size(); i++ )

{

customerList.addElement(customers.get(i));

}

}

catch (IOException ex)

{

JOptionPane.showMessageDialog(this, "Error: " + ex.getMessage(),

"File IO Error", JOptionPane.ERROR\_MESSAGE);

}

1. When we add a customer, only the latest customer information shows up. How can we fix this issue? Update the submitOrder( ) method. Go to the try block and, if the customer is added to the file, then the loadCustomers( ) method should run. When you finish updating the submitOrder( ) method, the try block should look something like this.

try

{

DataIO data = new DataIO(); // create DataIO object

data.add(cust);

loadCustomers(); // load all customers

// reset for the next customer

reset();

//move to the client orders tab

tabMain.setSelectedIndex(2);

}

1. We need to be able to delete a customer. Go to your DataIO class. Change the method header so we can delete the customer based on the customer’s name. In your delete( ) method, get all of the data using the getList( ) method. Delete the old file so the records are not duplicated. Finally, write all of the records to the file except the one that matches the customer to delete. When you finish, the delete( ) method should look something like this.

public void delete(String deleteName) throws IOException

{

// get all records

ArrayList<Customer> customers = getList();

// delete the old file

File oldFile = new File("Customers.txt");

if( oldFile.exists() == false )

throw new IOException("File does not exist!");

oldFile.delete();

// write "good" records to the file

for (Customer cust : customers) // foreach loops are cool!

{

if( deleteName.equalsIgnoreCase(cust.getName()) == false )

{

// good record so write it to the file

add(cust);

}

}

}

1. Let’s update our delete customer button event code in the GUI. Go to the code and get the selected Customer object instead. Then, create a DataIO object and delete the customer based on the customer’s name. Finally, load the current customers using the loadCustomers( ) method. When you finish, your code should look something like this.

try

{

// get the selected object

Customer old = lstCustomers.getSelectedValue();

// if something is selected, delete it and clear the details textarea

if (old != null)

{

DataIO data = new DataIO();

data.delete(old.getName()); // get the name only

txaCustomerInfo.setText("");

loadCustomers();

}

}

catch (IOException ex)

{

JOptionPane.showMessageDialog(this, "Error: " + ex.getMessage(),

"DataIO Error", JOptionPane.ERROR\_MESSAGE);

}

1. Here is an optional challenge for you. Test your application right now and you will see that the customer list is blank when you first start the application. You have to click the load list button to load the customers. If you want the customer list to automatically load when the application is started, how would you do that step?
2. Run your application and test it, including the menu. Are you having fun? It’s amazing that you can create such cool GUI applications.

# Course Project Deliverables for Week 5

* Close NetBeans so your zip file will have the latest code. Then, go to the project folder. Right-click on the project folder and choose Send to 🡪 Compressed (zipped) file. Then, rename the zipped file to include your name. Submit the zipped file on Canvas.
  + Week 5 File IO – Your Name.zip