**Course Project**

**DeVry University**

**College of Engineering and Information Sciences**

**Course Number: CIS355A**

# Module 4: Use ArrayList and Tabs, JList Controls

# Objectives

* Add Tabs to GUI application
* Add JList to GUI application
* Use ArrayList to process object arrays

Our GUI application is starting to look very nice! Our main GUI window is able to launch a second, “child” window. That is very cool!

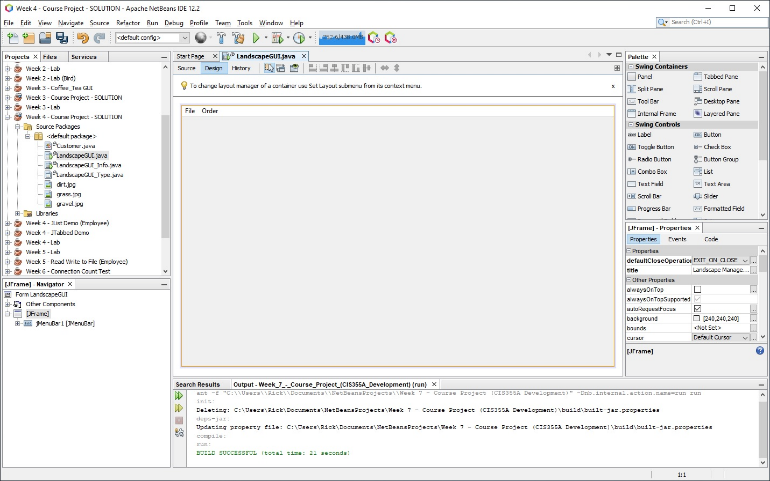
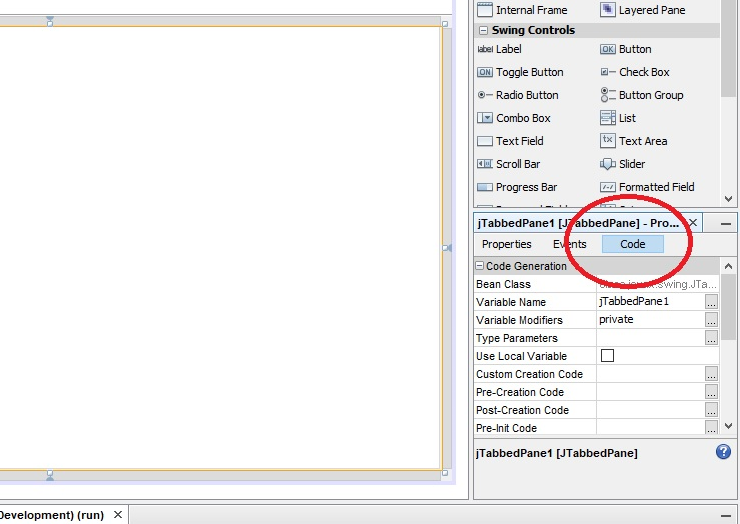
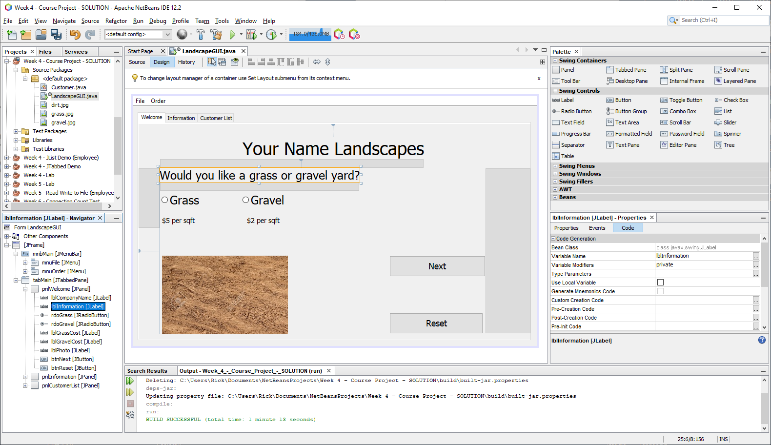
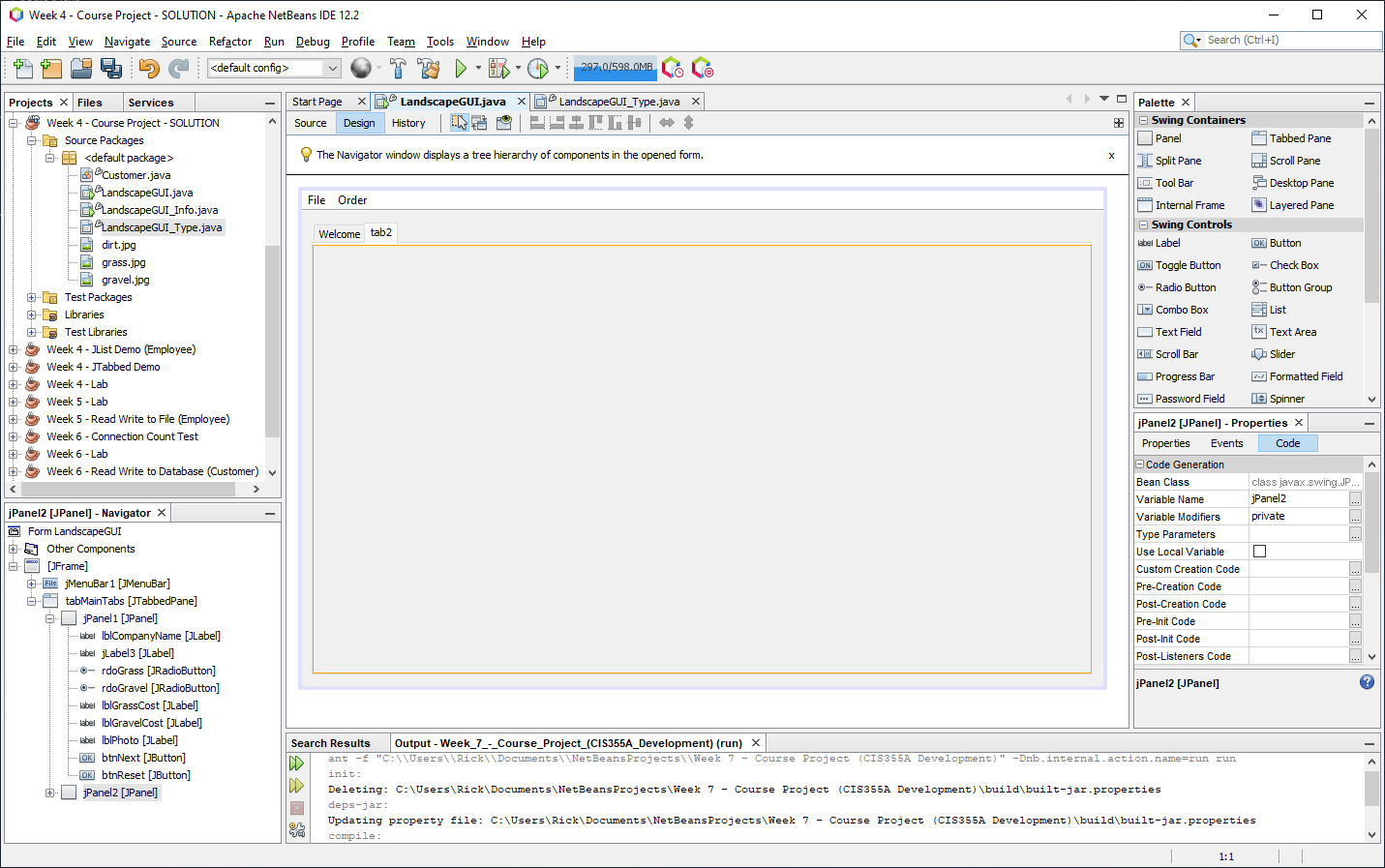
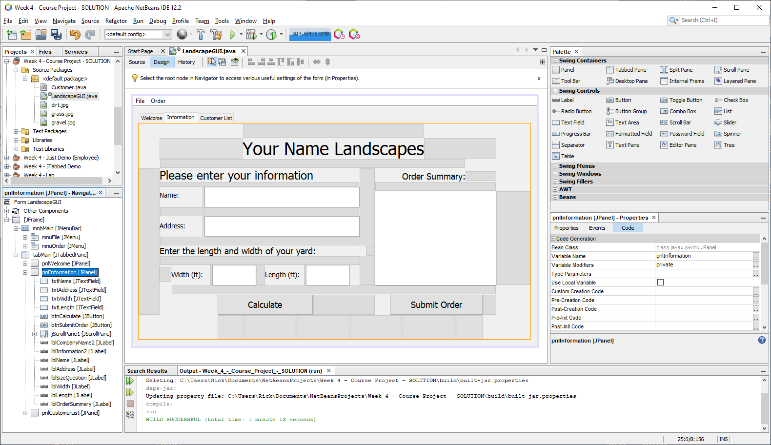
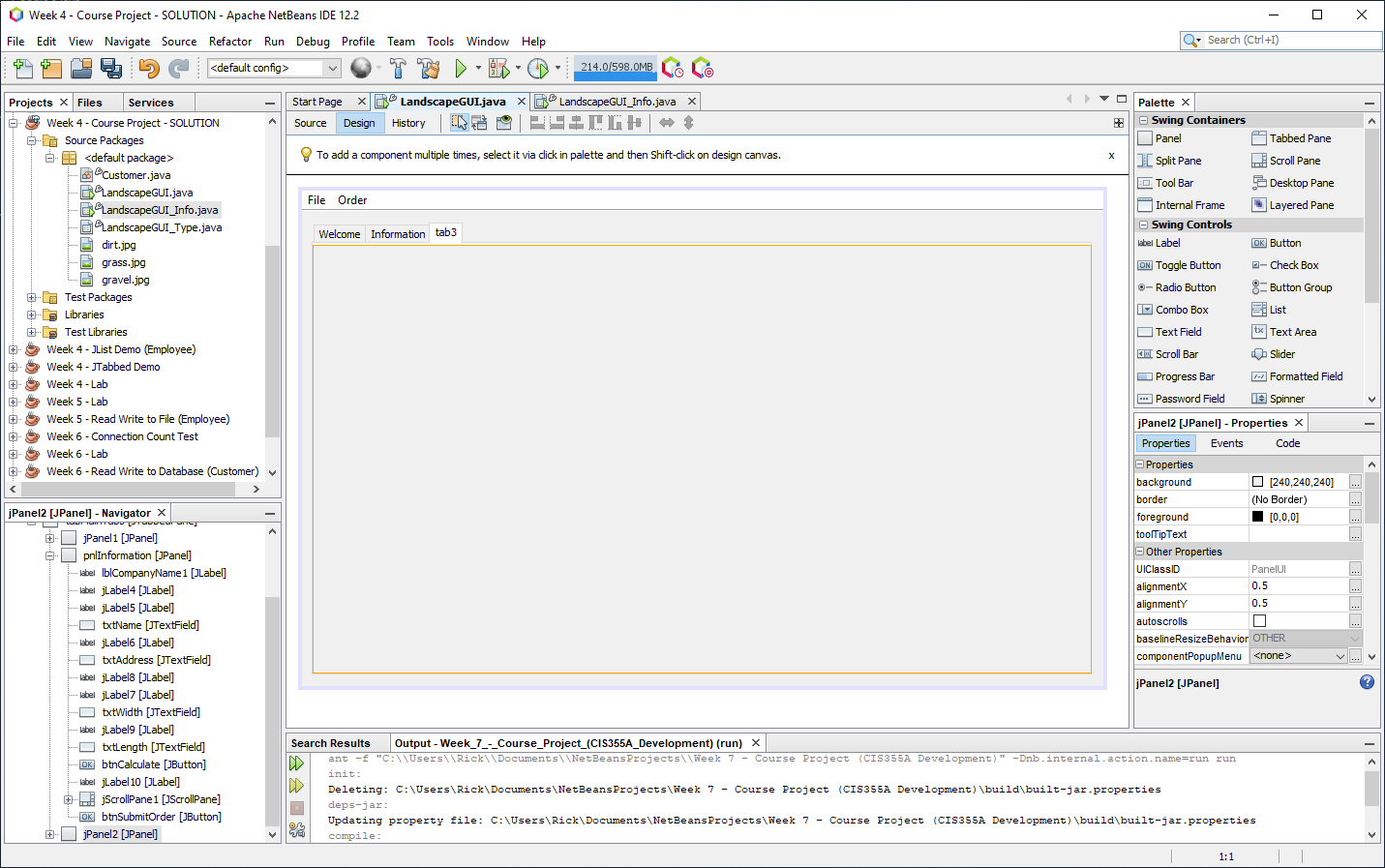
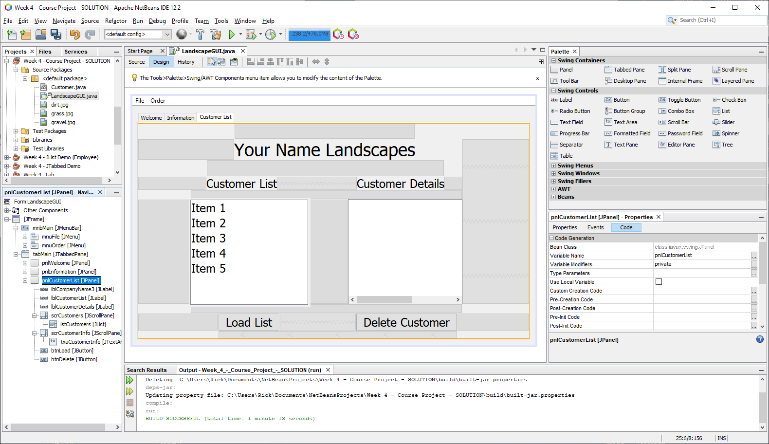
Let’s think about professional applications. When you use NetBeans and you open a second code file, does it launch out into a new window? No. The second code file simply opens in a second tab! Think about MS Excel. If you create a second spreadsheet in the same workbook, does the second spreadsheet launch out into a new window? No. The second spreadsheet in the workbook simply appears as a second tab with the tabs located at the bottom, left corner of the workbook!

We need to use tabs in our applications when appropriate. Based on our Visio Wireframe design, our current application would look better with tabs! Let add some tabs to make our application easier to navigate.

In addition, let’s show all of our customers in a JList in a separate tab. This way, we can click on the customer’s name and see the customer’s order details.

I have to tell you that we will have a problem once we finish the JList. When we close our application, all of our customer data will be lost. The JList will only hold the data while the application is running. How can we fix this problem and make our data “persistent”? We will have to wait until next week when we learn to read and write to files!

# Steps

1. Open your course project. Open the Source Packages. Right-click on the LandscapeGUI.java file in the Projects pane (left side window). Choose Copy and then right-click it again and choose Paste 🡪 Refactor Copy. In the “New Name” field, enter “LandscapeGUI\_Type” and then click the Refactor button.
2. Close the LandscapeGUI\_Type.java tab at the top. Then, double-click on the LandscapeGUI.java file on the Projects pane to open it. Confirm that the LandscapeGUI.java file is open by reading the tab at the top. Delete all of the GUI components except the menu, including the image, radiobuttons, labels, and command buttons. When you finish, the form should look something like this:  
     
   
3. Check the main menu bar of the menu bar on the Navigator windows. If the main menu bar is still called jMenuBar1, go to the Properties window and click on the Code tab. Change the Variable Name to **mnbMain** .
4. Look at the Palette of controls at the top, right corner. If the Swing Containers section is closed, click the plus sign (“+”) to open it. Drag a “**Tabbed Pane**” (JTabbedPane) and drop it on your frame. Grab the corners and resize the Tabbed Pane so that it almost fills the entire frame. Go to the Properties window and click on the Code tab.  
     
   
5. Change the “Variable Name” for the JTabbedPane to **tabMain**. The first three letters tell us that this object is a “tabbed pane”. The rest of the name describes the object. Notice that every “word” in the name starts with a capital letter. This notation is called “Hungarian Notation” and many companies follow this standard when naming GUI components.
6. How do you create tabs in the JTabbedPane? Easy! You simply drag and drop Panels for each tab. Look at your Visio Wireframe diagram. We will need three tabs. Let’s do one tab at a time and rename each tab as we go along. Drag a Panel (JPanel) and drop it on the center of your JTabbedPane. Notice that “tab1” appears at the top now. Click on the **center** of the Panel to select it. Then, go to the Properties on the right side and scroll down to “Tab Title”. Change the Tab Title to “**Welcome**” to match our Visio Wireframe design. Notice that the tab name changed. Now, click on the Code tab in the Properties area. Change the Variable Name to **pnlWelcome**.
7. We need to copy all of our components from the LandscapeGUI\_Type.java frame to the pnlWelcome tab. Double-click on LandscapeGUI\_Type.java to open it. Select all of the components. Go back to the LandscapeGUI.java window. Right-click on the center of the Panel and paste the components to the Panel. Remember to change the Variable Name on the components using Hungarian notation. When you finish, it should look something like this:  
     
   
8. Add the radiobuttons to the **btgYardType** ButtonGroup. Go to the Properties for the **rdoGrass** and the **rdoGravel** radiobuttons and change the buttonGroup property for each one to **btgYardType**.
9. We need to create our second tab. Drag a Panel (JPanel) and drop it to the right of the **first tab** (Welcome tab). It is important that you drop is on the tab row at the top and not inside of the first panel. You are doing it correctly if you get an orange dashed line around the JTabbedPane area. If it did not work, simply hit Undo or Ctrl-Z and do it again! If you did it correctly, it should look like this:  
     
   
10. Click on the center of the new Panel. Change the Tab Title to **Information** to match our Visio Wireframe design. Then, click on the Code tab in the Properties area and change the Variable Name to **pnlInformation**.
11. We need to copy all of our components from the LandscapeGUI\_Info.java frame to the **pnlInformation** tab. Double-click on LandscapeGUI\_Info.java to open it. Select all of the components. Go back to the LandscapeGUI.java window. Right-click on the center of the Panel and paste the components to the Panel. Remember to change the Variable Name on the components using Hungarian notation. When you finish, it should look something like this:  
      
    
12. Review your Visio Wireframe diagram. Notice that we need to add a third tab. Drag a Panel (JPanel) and drop it to the right of the **second tab** (Information tab). It is important that you drop is on the tab row at the top and not inside of the first panel. You are doing it correctly if you get an orange dashed line around the JTabbedPane area. If it did not work, simply hit Undo or Ctrl-Z and do it again! If you did it correctly, it should look like this:  
      
    
13. Click on the center of the new Panel. Change the Tab Title to **Customer List** to match our Visio Wireframe design. Then, click on the Code tab in the Properties area and change the Variable Name to **pnlCustomerList**.
14. Review the third page (Page-3) of your Visio Wireframe diagram. We need a label at the top with the company name. You can copy the company name label from one of your other tabs and paste it to the top of this third tab. Then, drag two labels and change the text of the first one to “**Customer List**” and the text of the second one to “**Customer Details**”. Remember to change the Variable Name property to follow Hungarian Notation. Place the labels to match your Visio Wireframe design.
15. Drag a **List** (JList) and place it on the left side. Move it and resize it to match your Visio Wireframe design. Change the Variable Name to **lstCustomers**. Drag a Text Area (JTextArea) and place it on the right side. Move it and resize it as needed. Change the Variable Name to **txaCustomerInfo**. Drag two Buttons (JButton) and place them at the bottom below each column. Change the text on the first button to “**Load List**” and rename the first button to **btnLoad** using the Code tab on the right side. Change the text on the second button to “**Delete Customer**” and rename the button to **btnDelete**. Change the Font property on each component as necessary. Remember to change the Variable Name on the components using Hungarian notation. Once you finish, it should look something like this:  
      
    
16. Now that the GUI is done, let’s finish the code. Go to your first panel (Welcome panel) by clicking on the Welcome tab. Right-click on the **Next** button and choose Events 🡪 Action 🡪 actionPerformed. The code should verify that a yard type was selected and then it should move to the next tab. Replace the code in the actionPerformed event for the btnNext button with this code:  
      
    // confirm that a selection was made then move to next tab  
    if (rdoGrass.isSelected() == false && rdoGravel.isSelected() == false)

{

JOptionPane.showMessageDialog(this, "Please select the type of yard.",

"Error", JOptionPane.ERROR\_MESSAGE);

}

else

{

tabMain.setSelectedIndex(1);

}

1. Go back to your designer. Let’s add code to the Reset button. Right-click on the Reset button and choose Events 🡪 Action 🡪 actionPerformed. Add this code that will reset all input fields so you will be ready to add another customer’s information:  
     
   reset( );
2. This code will call the reset( ) method. Add or update the reset( ) code to the bottom of your code window (scroll to the very bottom). Remember to stay inside the class. In other words, be sure that there is a closing curly brace at the very bottom that closes the class:  
     
   tabMain.setSelectedIndex(0);

// clear the radiobutton selection

btgYardType.clearSelection();

// put the image back to "Dirt"

lblPhoto.setIcon(new ImageIcon(

this.getClass().getResource("dirt.jpg")) );

txtName.setText("");

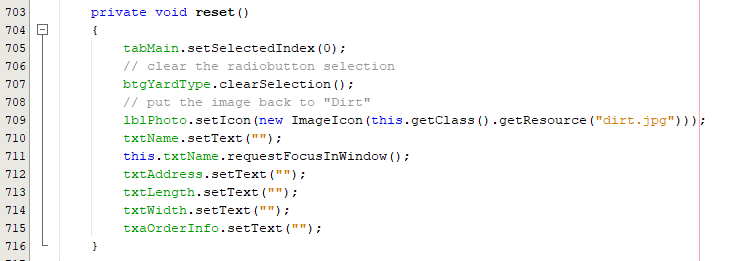
this.txtName.requestFocusInWindow();

txtAddress.setText("");

txtLength.setText("");

txtWidth.setText("");

txaOrderInfo.setText("");

1. When you finish adding the code, it should look something like this image. Your line number will probably be different than the line numbers in the image.  
     
   
2. If your code indentation gets messed up, remember to click Source 🡪 Format on the menu at the top of NetBeans. Go to your Design window and click on the second tab (Information tab). Right-click on the Calculate button and choose Events 🡪 Action 🡪 actionPerformed. Add this code to validate the inputs, create a Customer object, and set the text for the txaOrderInfo textarea:

// validate the inputs

if (validateInputs() == false)

{

return; // end the method if validation failed

}

// create the Customer object and show the information

Customer cust = createCustomer();

txaOrderInfo.setText( cust.getDetails() );

1. This code calls two methods, which we can add to the bottom. Add the validateInputs( ) and createCustomer( ) methods to the bottom.

private boolean validateInputs()

{

// get inputs from user using the textboxes

String sName = txtName.getText();

String sAddress = txtAddress.getText();

String sWidth = txtWidth.getText();

String sLength = txtLength.getText();

// confirm that a radiobutton has been selected

if (btgYardType.getSelection() == null)

{

JOptionPane.showMessageDialog(this,

"Error. Please select the type of yard.",

"Error", JOptionPane.ERROR\_MESSAGE);

return false;

}

// make sure we have a name for the order

if (sName.isBlank())

{

JOptionPane.showMessageDialog(this, "Enter a Name",

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

txtName.requestFocusInWindow();

return false;

}

// make sure we have an address for the order

if (sAddress.isBlank())

{

JOptionPane.showMessageDialog(this, "Enter a Address",

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

txtAddress.requestFocusInWindow();

return false;

}

if (sAddress.length() < 6)

{

JOptionPane.showMessageDialog(this, "Address must be six characters or more.",

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

txtAddress.requestFocusInWindow();

return false;

}

if (sWidth.isBlank())

{

JOptionPane.showMessageDialog(this, "Enter a Width",

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

txtWidth.requestFocusInWindow();

return false;

}

// check to see if the width is a valid number

try

{

Double.parseDouble(sWidth);

}

catch (NumberFormatException e)

{

JOptionPane.showMessageDialog(this, "Width must be a number",

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

txtWidth.setText("");

txtWidth.requestFocusInWindow();

return false;

}

if (Double.parseDouble(sWidth) <= 0)

{

JOptionPane.showMessageDialog(this, "Width must be greater than 0",

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

txtLength.setText("");

txtLength.requestFocusInWindow();

return false;

}

if (sLength.isBlank())

{

JOptionPane.showMessageDialog(this, "Enter a Length",

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

txtLength.requestFocusInWindow();

return false;

}

try

{

Double.parseDouble(sLength);

}

catch (NumberFormatException e)

{

JOptionPane.showMessageDialog(this, "Length must be a number",

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

txtLength.setText("");

txtLength.requestFocusInWindow();

return false;

}

if (Double.parseDouble(sLength) <= 0)

{

JOptionPane.showMessageDialog(this,

"Length must be must be greater than 0",

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

txtLength.setText("");

txtLength.requestFocusInWindow();

return false;

}

// all is good so return true

return true;

}

private Customer createCustomer()

{

String name = txtName.getText();

String address = txtAddress.getText();

double width = Double.parseDouble(txtWidth.getText());

double length = Double.parseDouble(txtLength.getText());

String yardType = "";

double totalCost = 0.0;

if (rdoGrass.isSelected())

{

yardType = "Grass";

totalCost = GRASS\_PER\_SQFT \* width \* length;

}

else if (rdoGravel.isSelected())

{

yardType = "Gravel";

totalCost = GRAVEL\_PER\_SQFT \* width \* length;

}

else

{

JOptionPane.showMessageDialog(this,   
 "Error. Please select a yard type.");

totalCost = 0.0;

}

Customer cust = new Customer(0, name, address, yardType,   
 length, width, totalCost);

return cust;

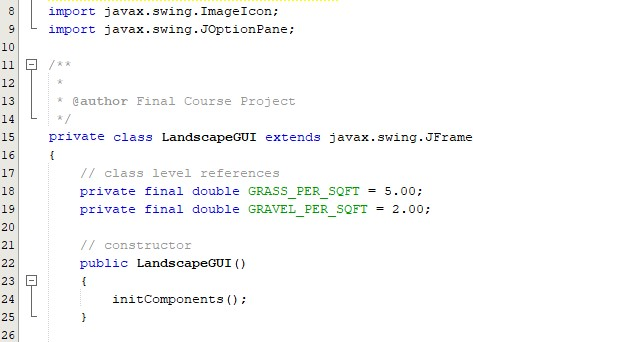
}

1. You will notice two errors in the createCustomer( ) code. The code uses constants, which is a very good idea! Let’s create the constants at the very top of the code window. Be sure the code is inside the class and above the constructor.

// class level references

private final double GRASS\_PER\_SQFT = 5.00;

private final double GRAVEL\_PER\_SQFT = 2.00;

1. When you finish, the top code should look something like this screenshot. Your lines numbers may be different and you should have **your** name as the @author.  
     
   
2. Go to your Design window. Right-click on the Submit Order button and choose Events 🡪 Action 🡪 actionPerformed. Add or confirm that this code is in the event method:

submitOrder( );

1. Let’s add the submitOrder( ) method to the bottom of the code window. Remember to stay in the class!

private void submitOrder()

{

if (validateInputs() == false)

{

return; // end the method if validation failed

}

Customer cust = createCustomer();

customerList.addElement(cust);

txaOrderInfo.setText(cust.getDetails());

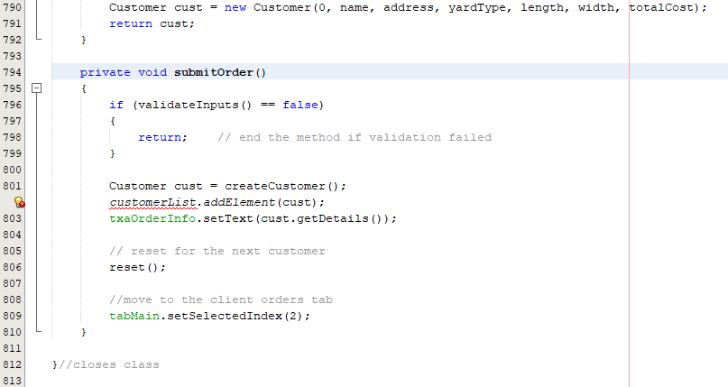
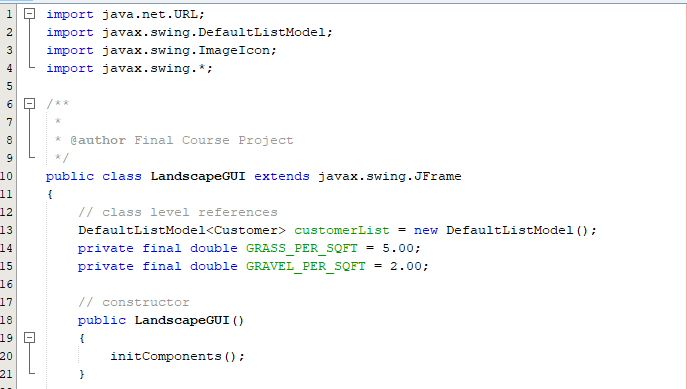
// reset for the next customer

reset();

//move to the client orders tab

tabMain.setSelectedIndex(2);

}

1. When you finish, it should look something like this screenshot. Notice that we have an error indicator under the customerList object. This is expected since we have not set up our JList yet. We will set up the JList on the next step!  
     
   
2. Go to the Design view and click on the “Customer List” tab (3rd tab). The left side has a List (JList) and the right side has a Text Area (JTextArea). Let’s set up the Text Area first. Click on the center of the Text Area to select it. Then, go to the Properties pane and change the Font to 18 pt or 24 pt. Next, click on the Code tab at the top of the Properties pane. Change the Variable Name to **txaCustomerInfo**.
3. The JList is a container that holds an array of objects called a DefaultListModel. The DefaultListModel is very much like an ArrayList. It can hold as few as zero items or up to thousands of items. You can easily add one item at a time and the DefaultListModel will automatically resize to accommodate the new items as needed. To set up a JList, you need to take three steps: 1) create a DefaultListModel, 2) add the DefaultListModel to the JList container, 3) change the data type for the JList.  
   1. Go to the top of the code window and create a DefaultListModel that works with Customer objects and call it customerList. Then, you can do the import for the DefaultListModel.  
      DefaultListModel<Customer> customerList = new DefaultListModel( );  
        
      It should look like this:  
      
4. Now, you need to add the DefaultListModel to the JList. Go to the Design window. Click one time on the center of the JList to select it. Go to the Properties pane on the right side. Change the font to 18 pts or 24 pts to match the txaCustomerInfo (JTextArea). Then, click on the Code tab at the top of the Properties pane. Change the Variable Name to **lstCustomers**. In addition, you must change the Type Parameters to <Customer> . The arrow brackets are required and the datatype must match our class name for our objects. Finally, click on the Properties tab. Go to the **model** property and click on the ellipsis on the right side. Click the dropdown at the top and choose “**Custom code**”. In the textbox, type the name of your DefaultListModel, which is **customerList**. Notice that this line will set the model for the JList to whatever you put in the textbox. Now, click the OK button to close the dialog.
5. Let’s create the Event handlers for our JList and buttons! Right-click on the JList (lstCustomers) and choose Events 🡪 ListSelection 🡪 valueChanged. Add this code to the event method:

// get selected item’s index number

int index = lstCustomers.getSelectedIndex();

// if something was selected, show the object’s details

if (index > -1)

{

Customer cust = customerList.getElementAt(index);

txaCustomerInfo.setText(cust.getDetails());

}

1. Now, right-click on the Load List button (btnLoadList) and choose Events 🡪 Action 🡪 actionPerformed. Add this line of code to the event method:

JOptionPane.showMessageDialog(this, "Method is incomplete.");

1. Right-click on the Delete Customer button (btnDelete) and choose Events 🡪 Action 🡪 actionPerformed. Add this code to the event method:

// get the index for the selected item

int index = lstCustomers.getSelectedIndex();

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

if (index > -1)

{

customerList.remove(index);

txaCustomerInfo.setText("");

}

1. Let’s clean up a bit. Go to the Projects pane on the left side. Delete the two GUI files that we do not need anymore. Right-click on the **LandscapeGUI\_Info.java** file and choose **Delete**. Click **OK** to confirm. Now, right-click on the **LandscapeGUI\_Type**.java file and choose **Delete**. Click **OK** to confirm.
2. Let’s center our form. Go to the Source (code) tab for the LandscapeGUI.java. Scroll to the very top and then find you constructor. Center the form by adding this.setLocationRelativeTo(null). When you finish, the constructor should look like this:

// constructor

public LandscapeGUI()

{

initComponents();

// center the form

this.setLocationRelativeTo(null);

}

1. When you run your application, you will notice that the tabbing does not work properly in the Information tab. You can change the “tab order” very easily. Go to Design view and click on the Information tab. Click on the first component (txtName) and then, in the nextFocusableComponent, choose the next component that you want to gain focus when you press tab. Go to each component that you want in the “tab order” and set it’s nextFocusableComponent property.
2. Click on the **Order** menu at the top. Select the **Submit Order** menu item. Then, right-click on the **Submit Order** menu item and choose Events 🡪 Action 🡪 actionPerformed. Change the code to:

submitOrder( );

1. Run your application and test it. Is everything working? Nice!

# Course Project Deliverables for Week 4

* **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 4 Object Arrays – Your Name.zip