**ECCS 1621 – Programming 2**

**Lab 1 – Introduction to Java, Graphical User Interfaces, and the IntelliJ IDE (Version 2019.3)**

Welcome to Lab 1. Please make sure you read the *entire* lab document. There are many new concepts. We have included explanations. This material is fair game on the midterm and final exam. Do not just cut and paste.

1. The GuessingGame application will allow a user an unlimited number of opportunities to guess a number between 0 and 99, inclusive. While not making for exciting game play, this is a good example program.
2. Start IntelliJ and create a new Java Application project called GuessingGame. First Click on “Create New Project”. (You can also create a new project via opening the “file” menu and select “New” then “Project…”:

A screenshot of a cell phone

Description automatically generated

A pop-up dialog titled “New Project” will now appear. Make sure that “Java” is selected on the left, then click on the “Next” button.

A screenshot of a social media post

Description automatically generated

1. The next screen asks about using templates just click “next” button.
2. The pop-up now becomes the ‘New Java Application’ dialog.
   1. Use “GuessingGame” as the project name.
   2. For the project location, please use the “…” button on the and navigate to the folder/subfolder you are keeping all the files for this course.
   3. Click on the “Finish” button.

A screenshot of a social media post

Description automatically generated

1. On the left side, you will see the contents of the entire project. Click on the **arrow** first listing/folder (which will be where you saved your files).

A screenshot of a social media post

Description automatically generated



1. Create the file for the graphical user interface (GUI) by right-clicking on the ‘src’ folder entry, then selecting ‘New’ followed by ‘Swing UI Designer…’ and finally “GUI Form”.

Graphical user interface, text, application

Description automatically generated

1. This will cause another dialog box to appear. Use ‘GuessingGameForm’ as the name of the form and as the name of the class. Click on the “Finish” button to create the Form.

A screenshot of a cell phone

Description automatically generated

1. You will now see two areas of interest; first, the “Design” view for the JFrame that will contain our GUI components. Second, the “Palette” containing the various “widgets” that can be dragged and dropped onto our design:

A screenshot of a social media post

Description automatically generated

1. Design the GUI for the JPanel of GuessingGame by dragging the following components from the Palette to the locations indicated by the figure below. You will need:
   1. one Label associated with the input Text Field,
   2. one Label to display the results,
   3. one Text Field for the input, and
   4. one Button for starting a new game.

When completed, the GUI should look similar to this (put do not worry if it is not exactly the same):

A screenshot of a computer

Description automatically generated

1. Once all of the components are placed on the Design window, their properties can be edited in several ways. One of the easiest ways is to select the component by clicking on it, then changing its properties by accessing the ‘Properties’ window to the left. To practice this, click on “Label”.

To change a property, the safest approach is to click on the “Value” area for each property. You will need to click on the checkbox for “Show expert properties” to make all the changes.

**A screenshot of a social media post

Description automatically generated**



At this time, change the following properties:

**Field name:** change from empty to **inputLabel**

**Font:** change to Arial Bold 24**.**

You will need to click on the folder to get to the font menu (circles in picture on the left). Make sure you click the checkboxes for “Font”, “Style”, and “Size” to change the defaults.

**A screenshot of a cell phone

Description automatically generatedA screenshot of a cell phone

Description automatically generated**



**Foreground:** change to blue by changing the RGB to 0, 0, 255.

A picture containing screenshot

Description automatically generated

**Horizontal Alignment:** use the dropdown to select RIGHT.

Before:

**A screenshot of a cell phone

Description automatically generated**

After:

Graphical user interface, application

Description automatically generated

**Label For:** use the dropdown to select jTextField1.

A screenshot of a cell phone

Description automatically generated

**Text:** click to change to “Guess the value:”

A screenshot of a social media post

Description automatically generated

(Note: Some properties can be changed by clicking on the component itself)

1. Finish the work on customizing the Text Field widget by going to the Properties window and changing the font to Arial Regular 24 and the change the field name to “inputTextField”.
2. Select the jLabel2 label and edit its text to read “The Guessing Game”. Change the font to Tahoma Bold 36, then center this label under the label – text field combination that appears above it. Change the field name to “resultLabel”.

A screenshot of a social media post

Description automatically generated

1. Finally, edit the text on the Button widget by first clicking on the widget so that it receives the “focus” (as in center of attention, which is denoted by the gold border now shown by the selected widget), then double-click and type in “New Game”.

A screenshot of a social media post

Description automatically generated

1. Modify the font to Tahoma 16 Regular/Plain. Change the field name to “newGameButton”.
2. You should now have a frame that looks similar to the following:

A screenshot of a social media post

Description automatically generated

1. As a general rule, we always want to design the GUI before writing any code, as a good IDE such as IntelliJ will generate a lot of the code for us in terms of the boilerplate needed to display and operate the GUI. We can see the code already generated by clicking on the “Source” view tab underGuessingGameForm.java (or simply switching tabs):

A screenshot of a cell phone

Description automatically generated

1. Go back to the “Design” view. Right click on either the text field or the button, the IDE will help you to create an event (specifically, an ActionEvent) to be associated with that component. Select “Create Listener”. Then select “ActionListener”. Click the “OK” button. And you will see code similar to that in the picture below.

A screenshot of a social media post

Description automatically generated A screenshot of a cell phone

Description automatically generatedA screenshot of a social media post

Description automatically generated A screenshot of a cell phone

Description automatically generated

1. At this time, please repeat using the textbook or button you did not select in step 17. Note that IntelliJ is now presenting you with the Source view, with the cursor positioned at the point where you need to enter code that instructs the program what to do when an Action Event (such as when someone enters a value, then presses the “Enter” key) is generated by this widget:

A screenshot of a cell phone

Description automatically generated

Before we can add the code for these widgets, we must first include the code for generating random numbers.

1. The next step is to create a separate class for the random number; an instantiation of this class will hold the value to be guessed and the class will define all of the behaviors associated with this value, such as checking to see if a guessed value is the same as the random value being guessed. To create this class right-click on the ‘src’ folder entry and select ‘New…’, then ‘Java Class…’.

A screenshot of a cell phone

Description automatically generated

1. When the “New Java Class” dialog box appears, enter ‘RandomValue’ in the ‘Class Name:’ field.

A screenshot of a cell phone

Description automatically generated

1. Press the “Enter” key. This creates a new file called RandomValue.java; it is within this file that we shall place the code containing all of the functionality associated with the randomly-generated value to be guessed.

A screenshot of a social media post

Description automatically generated

1. The following contains the entire source code for the RandomValue class. All of the code generated by IntelliJ is displayed in *italics* whereas the code to be entered in by you, the programmer, is displayed in **bold**. The following pages explain what’s going on. Please note that normally the code would be properly commented; this is not the case being shown in this example.

*public class RandomValue {*

**// code section 1 - instance variables**

**private java.util.Random rndGenerator = new java.util.Random();**

**private int randomValue;**

**public final static int NUMBER\_OF\_VALUES = 100;**

**// code section 2 - public constructor**

**public RandomValue() {**

**randomValue = rndGenerator.nextInt(NUMBER\_OF\_VALUES);**

**}**

**// code section 3 - getters and setters**

**public void setRandomValue() {**

**randomValue = rndGenerator.nextInt(NUMBER\_OF\_VALUES);**

**}**

**public void setRandomValue( int n ) {**

**if ( n < 0 || n >= NUMBER\_OF\_VALUES )**

**return; // take no action - value out of range**

**randomValue = n;**

**}**

**public int getRandomValue() {**

**return randomValue;**

**}**

**// code section 4 - inquirers**

**public boolean isCorrectValue( int value ) {**

**if ( value == randomValue )**

**return true;**

**return false; // fall-through is necessary to make IDE happy...**

**}**

**// compareTo: returns < 0 if random value is less than passed value,**

**// > 0 if random value is greater than passed value**

**// 0 if random value equals passed value**

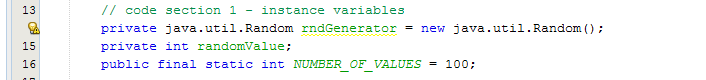
**public int compareTo( int value ) {**

**return randomValue - value;**

**}**

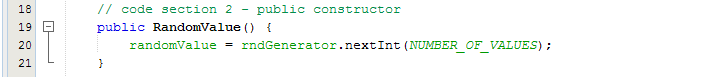
*}*

1. Explanation of code section 1: instance variables.



Instance variables are the variables, or data fields, that are created whenever an object is instantiated from the class. For this program we will need access to a random number generator, which Java conveniently provides as part of the java.util package. As **Random** is a class, we have to instantiate an object of this class in order to use it. The variable **randomValue** is used to store the value to be guessed. The constant **NUMBER\_OF\_VALUES** is an example of a symbolic constant; a counterexample would be to hardcode the value 100 into the program. This would then be considered a ‘magic number,’ as the value would be appearing without any understanding on the part of the person reading the code.

1. Explanation of code section 2: the constructor method.



All classes should have at least one constructor method defined. A constructor method is simply a method possessing the exact same name as that of the class. As the **Random** object already exists, all that is needed when the instantiation occurs is to generate the value to be guessed and store it in the variable **randomValue.**

Please note that the IntelligJ IDE supports code completion and built-in documentation; once the ‘.’ character for accessing a method of a class is entered, the IDE will display all of the methods associated with the class.

A screenshot of a social media post

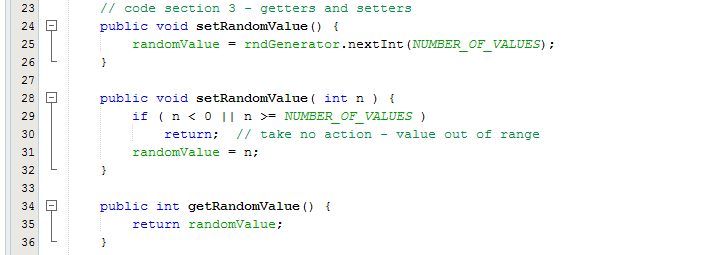
Description automatically generated

If you enter the first few letters of a method name, then the display is limited to just those methods starting with those letters. By clicking or highlighting on one of the listed elements and clicking the three dots in the bottom right corner, you can also get the documentation for that method. Shown below is the documentation for the **nextInt( )** method.

A screenshot of a cell phone

Description automatically generated

1. Explanation of code section 3: getters and setters.

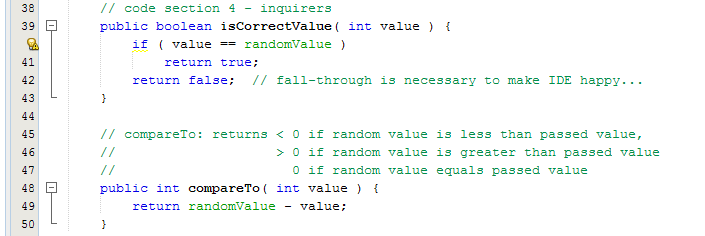


Getters (AKA accessors) and setters (AKA mutators) are those methods that are used to access the instance variables. Java programming has a simple convention for identifying these methods.

* 1. All getter methods start with the word ‘get’, followed by the name of the appropriate instance variable.
  2. All setter methods start with the word ‘set’, followed by the name of the appropriate instance variable.

For this application we have created two setters through use of an overloaded method (where two or more methods share a name but have different parameter lists). The setter where nothing is passed is the default; this method will generate a random number for us. The other setter has an integer passed to it; if we were to write a testing application to verify the correctness of this class, then this would be the method called. Note that this method verifies the correctness of the passed data before allowing it to be stored in the instance variable. Finally, there is but one getter, and it simply returns the value of the number to be guessed.

1. Explanation of code section 4: inquirers.



‘Inquirers’ are what I call the next – and final – group of methods. These are used to inquire the object with respect to the data contained therein. The **isCorrectValue** method is used to determine if the passed value is the correct answer. The **compareTo** method performs a simple subtraction to determine whether one value is less than, greater than, or equal to the other. Note the use of comments explaining the operation of this method. Later on we’ll utilize some IDE tools that will automatically generate comments ***and*** documentation for this purpose.

1. Now that the class for the random number has been completed, we have to add functionality to the GUI so that the program actually does something. First, click on the tab for ‘GuessingGameForm’ so that you have the Source display of the GUI visible. We want to create event handlers for both the text field and the button; before we can do this, however, the code shown in bold font below needs to be inserted into GuessingGameJFrame.java:
   1. Place code insert 1 just after the declaration of the class:

*public class GuessingGameForm extends javax.swing.JPanel{*

**// code insert 1 - instance variables of the main class**

**private int numberOfGuesses = 0;**

**private RandomValue randomValue;**

* 1. Place code insert 2 **INSIDE** the GuessingGameJFrame() method

*/\*\* Creates new form GuessingGame \*/*

*public GuessingGameForm() {*

**// code insert 2 - initialization**

**randomValue = new RandomValue();**

**jButton1.setEnabled(false);**

*}*

1. Explanation of code insert 1:
2. *public class GuessingGameForm extends javax.swing.JPanel{*
4. **// code insert 1 - instance variables of the main class**
5. **private int numberOfGuesses = 0;**
6. **private RandomValue randomValue;**

We need to have instance variables within the main method (the execution of the program will cause the instantiation – this will be shown later in the quarter) for keeping track of the number of guesses and for the instantiation of the **RandomValue** object. This code is placed after the class definition and before the constructor method.

1. Explanation of code insert 2:
2. *public GuessingGameForm() {*
3. **// code insert 2 - initialization**
4. **randomValue = new RandomValue();**
5. **jButton1.setEnabled(false);**
6. *}*

Here we are adding code to the constructor method ***after*** the call to initComponents (which instantiates all of the widgets displayed on the GUI). This is used to perform the instantiation of the **randomValue** object and to disable the new game button.

1. Next, we need to enter the code that performs the event handling.

Please enter the following code as shown in bold font:

*inputTextField.addActionListener(new ActionListener() {*

*@Override*

*public void actionPerformed(ActionEvent e) {*

**numberOfGuesses++;**

**int guess = Integer.parseInt( inputTextField.getText() );**

**if ( randomValue.isCorrectValue(guess) ) {**

**String guessStr = " guess"; // note space before word**

**if ( numberOfGuesses > 1 )**

**guessStr = " guesses";**

**resultLabel.setText("Correct in " + numberOfGuesses + guessStr );**

**inputTextField.setEnabled(false); // no more inputs until new game**

**newGameButton.setEnabled(true);**

**return;**

**}**

**// if here then incorrect guess...**

**int difference = randomValue.compareTo(guess);**

**if ( difference > 0 ) // then random value > guess**

**resultLabel.setText( guess + " is too low" );**

**else**

**resultLabel.setText( guess + " is too high" );**

**inputTextField.setText( "" ); // clear the input field**

*}*

1. Again, please note that IntelliJ offers auto-completion as you enter in the names of objects; that is, once the dot operator is entered, you are presented with a list of the instance members of the associated class:

Auto-completion will work for both built-in and self-defined classes. Note that there is no documentation provided if no ‘Javadoc’ for this method was found. ‘Javadoc’ is how Java programs are documented; it uses special comments and tags so that this documentation can be readily recognized by an IDE. We will soon have a Javadoc lab so that you’ll get first-hand experience as to how to construct such documentation.

1. Other features to note are that IntelliJ will show you the parameter type to be entered and will attempt to fill in the parameter list of method calls for you:

A screenshot of a social media post

Description automatically generated

1. Next, add code for the new game button event handler, by entering the code shown below in bold font:

*newGameButton.addActionListener(new ActionListener() {*

*@Override*

*public void actionPerformed(ActionEvent e) {*

**numberOfGuesses = 0;**

**randomValue.setRandomValue();**

**inputTextField.setText( "" );**

**resultLabel.setText("Can you guess my new value?");**

**inputTextField.setEnabled(true); // allow a new round of guessing**

**newGameButton.setEnabled(false);**

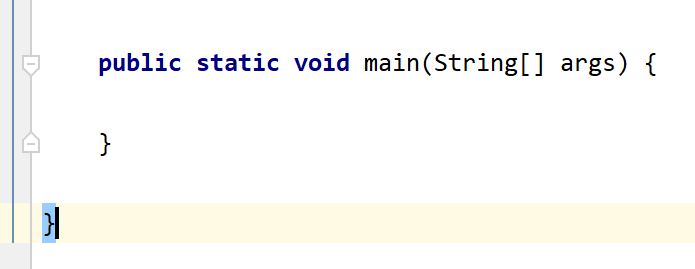
*}*

1. We need to add the main method to the code. Go to the GuessingGameForm.java file. Scroll to the bottom of the file. More your curse the line before the final “}” symbol at the end of the file. Type psvm. Press enter.

A screenshot of a social media post

Description automatically generated

The main method will be automatically created and your code should look something like this:



The main method will include a couple lines of code:

**public static void** main(String[] args) {  
 JFrame myFrame = **new** JFrame(**"The Guessing Game!"**); *//Makes the frame* myFrame.setContentPane(**new** GuessingGameForm().**panel1**); *//attaches the panel to the frame*

*//sets up what happens with the frame is closes* myFrame.setDefaultCloseOperation(JFrame.***EXIT\_ON\_CLOSE***);

myFrame.pack(); *//put everything in the frame* myFrame.setVisible(**true**); *//allows us to see the frame*}

1. We are now done with entering code. To run the application, we must first build the project (a fancy way of saying the code needs to be compiled). In IntelliJ, this is done by selecting ‘Build Project’, either by using the Ctrl + F11 function key, clicking on the hammer icon, or (as shown below) using the ‘Run’ menu.

A screenshot of a cell phone

Description automatically generated

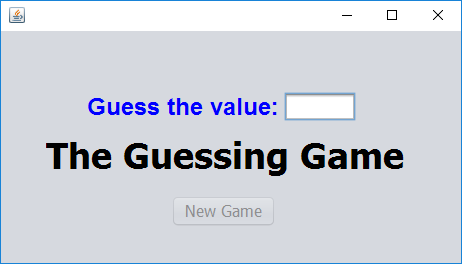
1. Assuming no compilation errors, we can now run the application either using the ‘Run’ menu. Select the GuessingGameForm (basically this will be the class that contains the main method).

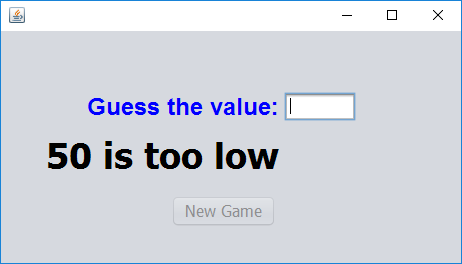
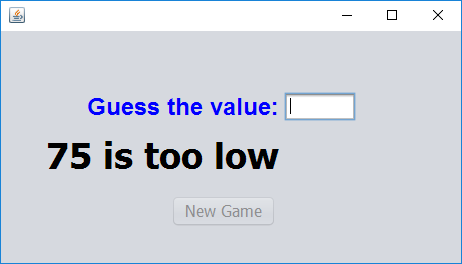
A screenshot of a cell phone

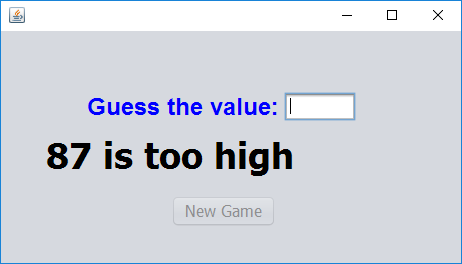
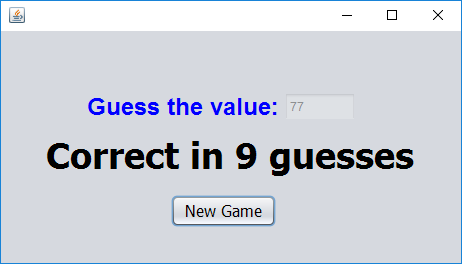
Description automatically generatedA screenshot of a cell phone

Description automatically generated

1. Congratulations! You have made a Java application, so start guessing!



1. Play the game a couple of times. Do you need to make any adjustments regarding font and/or component sizes? If so, then do so, making sure that all items are clearly displayed throughout the conduct of the game.
2. Just for fun, determine what the maximum number of guesses should be in order to correctly guess the value.
3. Finally, create 3 screenshots to demonstrate your completed game. Submit your 3 screenshots and your .java files on Moodle.