CIT 149: Java I
Chapter 8
Lab 3

For this lab we will complete #12 on page 650, which creates a GUI application. This has several, separate classes, most of which are quite short. There will be two interfaces, two classes that implement the interfaces, a class that sets up the structure of the JFrame, and a class that runs the GUI application. Since both interfaces are written the same, I will give you one of them with these directions in order to save time and typing.

MessageDecoder Interface

This interface will be used to decode message.

1. Open a new document window in TextPad and save the file as MessageDecoder.java.
2. This interface will have only a few lines of code. Type:

```
public interface MessageDecoder
{
    public String decode(String cipherText );
}
```

   - If you read the directions for #9 you would have seen that this interface would have a single abstract method. All methods within an interface are abstract so you do not need to include abstract within the method header for interfaces.

3. Compile the interface.

MessageEncoder Interface

1. Using the MessageDecoder interface as an example, create the MessageEncoder interface. The only differences between the two is:
   - The method is encode instead of decode
   - Its parameter is a String named plainText
2. Compile the interface.

ShuffleCipher class

This class will implement both of the interfaces we created.

1. Open a new document window in Textpad and save the file as ShuffleCipher.java.
2. Type the class header and opening brace, implementing both interfaces:

```
public class ShuffleCipher implements MessageEncoder, MessageDecoder
{
```

3. Type the code that will declare a private integer named shufflesToDo

4. Type the constructor method header and opening brace with a parameter of an integer named *n*.
5. Within the constructor method set the variable shufflesToDo to equal the variable *n* that was passed to this method.
6. Close the method.
7. Our next step is to create a method named encode. Type:

```
public String encode(String plainText)
{
   String cipherText = plainText;
   for(int i=0; i<shufflesToDo; i++)
   {
      cipherText = shuffle(cipherText);
   }
   return cipherText;
}
```

   - The for loop in this method passes values of a String to the shuffle() method as long as the value of *i* is less than the value of the variable *shufflesToDo.* Recall that the value of *shufflesToDo* is based on what is passed to the constructor method when an instance of this class is created.
   - This method will return the value of *cipherText* to the point where this method is invoked.

8. Our next method is the shuffle() method. This is the method that will be passed the value of *cipherText* in the for loop of the encode() method. Type:

```
private String shuffle(String s)
{

    String shuffled = "";
   int mid = s.length()/2;
```

```
 String first = s.substring(0, mid);
String second = s.substring(mid, s.length());

for(int i=0; i<first.length(); i++)
{
    shuffled = shuffled + first.charAt(i) + second.charAt(i);
}

//If the length of the message is odd; add in the last character
if(second.length() > first.length())
   shuffled = shuffled + second.charAt(second.length()-1);

return shuffled;
}
```

- This method receives the value of a String from the encode method
- The String variable is set to an initial value of an empty string
- The integer *mid* is both declared and constructed, given the value of the length of the String passed to the method, dividing that length by 2.
- The String variable first is both declared and constructed, given the value returned by the String class' substring() method which is passed the value of 0 for its beginning index and *mid* for its ending index. In other words it will take the value of the first character of *s* and end with the character of *s* with an index number of *mid,* however don't confuse this because if *mid* is 3 then it would take on the character of the first through 4th. Each character in a String variable is counted as index numbers. The first character is at index number 0, and so on.
- The String variable of second is both declared and constructed, given the value of what is returned by the substring() method. This method is passed the values of *mid* as its starting point and the length of *s* as its ending index.
- The for loop assigns a value to the variable shuffled.
- The if statement checks to see if the length of the variable *second* is greater than the length of the variable *first*.

9. Our last two methods are easily understood so I will not give any explanation. Type:

```
public String decode(String cipherText)
{
    String plainText = cipherText;
   for(int i=0; i<shufflesToDo; i++)
   {
      plainText = unshuffle(plainText);
   }
   return plainText;
```

```
}

private String unshuffle(String s)
{
    String unshuffled = "";
    for(int i=0; i<s.length(); i+=2)
        unshuffled = unshuffled + s.charAt(i);

    for(int i=1; i<s.length(); i+=2)
        unshuffled = unshuffled + s.charAt(i);

    return unshuffled;
}
```

10. Close and compile the class.

SubstitutionCipher class

This class also implements the two interfaces.

1. Open a new document window in Textpad and save the file as SubstitutionCipher.java.
2. By this time you should be able to read the code and tell what is going on. Ask questions if you do not understand any part of the following. Type:

```
public class SubstitutionCipher implements MessageEncoder, MessageDecoder
{

    private int shift;

    /**

     * Creates a new instance of SubstitutionCipher

     */


public SubstitutionCipher(int n)
{

    shift = n;

}
```

```
    public String encode(String plainText)
    {
        String cipherText = "";
        for(int i=0; i<plainText.length(); i++)
        {
            Character c = plainText.charAt(i);
            cipherText = cipherText + codeCharacter(c);
        }
        return cipherText;
    }

    private Character codeCharacter(Character c)
    {
        return (char)(c + shift);
    }

    // This is for the project
    public String decode(String cipherText)
    {
        String plainText = "";
        for(int i=0; i<cipherText.length(); i++)
        {
            Character c = cipherText.charAt(i);
            plainText = plainText + decodeCharacter(c);
        }
        return plainText;
}
```

```
    private Character decodeCharacter(Character c)
    {

        return (char)(c - shift);

}
```

3. Close and compile the class.

CoderFrame class

This class will set up the appearance of the GUI application. However, unlike previous GUI applications this one will not run. You will see what I mean shortly.

1. Open a new document window in Textpad and save the file as CoderFrame.java.
2. Type the following import statements:

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
```

- These are the minimum number of import statements you will include when creating a frame.

3. Our next step is to type the class header and opening brace. Type:

```
public class CoderFrame extends JFrame implements ActionListener
{
```

- Here we extends the JFrame class which will become this class' superclass.
- We will create components that we want our program is listener for. A class will not listen for such things as buttons until a listener is created for it. When we implement the ActionListener interface the program knows to listen for some event.

4. We create two constants by typing:

```
public static final int WIDTH = 500;
public static final int HEIGHT = 500;
```

- recall that you cannot change the value of a constant while the program is running.

5. Next we want to create several components for our frame. Type:

```
private JButton encodeButton;
private JButton decodeButton;
private JButton shuffleButton;
private JButton subButton;
```

```
private JTextField keyTextField;
 private JTextField messageTextField;
private JLabel outputLabel;
```

- Here we create 4 buttons (declare), 2 text fields and a label. Notice in parentheses I have declare. This is different from constructing an object. Declaring simply tells what type of object and gives it an identifier. Constructing an object constructs it as a new object. You must construct an object before adding it to a frame. This is often done within a constructor method.

6. We also want to create a MessageEncoder and MessageDecoder object by typing:

```
private MessageEncoder encoder;
private MessageDecoder decoder;
```

7. Our constructor method will construct our frame. First type the method header and opening brace:

```
public CoderFrame()
{
```

8. We need to set the size of the frame and make certain that the program stops running when the closing button is pressed. Type:

```
setSize(WIDTH, HEIGHT);
setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
```

- Using the setDefaultCloseOperation() method ensures that the program truly stops running. Within this the program continues to run in the background even though it may not appear so. This ensures that it stops. You can also completely stop it by closing the command prompt. However, in the real world such applications are made into executable files.

9. We know must construct our objects that were declared earlier. Type:

```
encodeButton = new JButton("Encode");
decodeButton = new JButton("Decode");
shuffleButton = new JButton("Shuffle Code");
subButton = new JButton("Substitution Code");
```

    - You cannot assign an object, such as a JButton, to a frame unless you construct it. If you attempt to do so, the program may compile but when you run it you will get an error of

NullPointerException.

10. To make certain that our program listens for the press of our buttons we have to add a listener to them. Type:

    ```
    encodeButton.addActionListener(this);
    decodeButton.addActionListener(this);
    shuffleButton.addActionListener(this);
    subButton.addActionListener(this);
    ```

11. We also have to construct our MessageEncoder and MessageDecoder objects. Type:

    ```
    encoder = new SubstitutionCipher(5);
    decoder = new SubstitutionCipher(5);
    ```

    - Notice that these are declared as new SubstitutionCipher objects? This allows us to access methods from within both classes.

12. We both declare and construct a new label. Type:

    ```
    JLabel keyLabel = new JLabel("Enter an integer key here: ");
    ```

    - Since we declared this label within the constructor method itself, we cannot reference it from another method.

13. We construct our text fields, and the one label that was declared at the beginning of the program. Type:

    ```
    keyTextField = new JTextField("5");

    messageTextField = new JTextField("Enter your message here");
    outputLabel = new JLabel("                ");
    ```

    - Notice that for the outputLabel we gave it a value of an empty String? We can change this so that it holds a true String value later.

14. Next we set the layout manager of our frame. By default, JFrames have a layout manager of BorderLayout. In this case we want all our components to display left to right. If one line is filled up objects will be moved to the next line. Type:

```
setLayout(new FlowLayout());
```

15. So that each displays within our frame we have to add them to the frame. The order in which they are added determines how they are displayed within the frame, going from left to right. Type:

```
add(encodeButton);
add(decodeButton);
add(shuffleButton);
add(subButton);

add(keyLabel);
add(keyTextField);

add(messageTextField);
add(outputLabel);
```

16. Close the constructor method.
17. Our final method is the actionPerformed() method. This method is required should you implement the ActionListener interface. Type:

```
public void actionPerformed(ActionEvent e)
{
```

18. A series of if/else statements will check to see which button is pressed and what is to occur when it is. Type:

```
if(e.getActionCommand().equals("Encode") )
{
   String message = messageTextField.getText();
   outputLabel.setText(encoder.encode(message));
}
else if(e.getActionCommand().equals("Decode") )
{
   String message = messageTextField.getText();
   outputLabel.setText(decoder.decode(message));
}
else if(e.getActionCommand().equals("Shuffle Code") )
{
   String keyString = keyTextField.getText().trim();
   int key = Integer.parseInt(keyString);
   encoder = new ShuffleCipher(key);
```

```
        decoder = new ShuffleCipher(key);
    }
    else if(e.getActionCommand().equals("Substitution Code") )
    {
        String keyString = keyTextField.getText().trim();
        int key = Integer.parseInt(keyString);
        encoder = new SubstitutionCipher(key);
        decoder = new SubstitutionCipher(key);
    }
```

19. You should be able to read what is occurring. If you do not understand this, please ask questions.
20. Close the actionPerformed() method and the class and compile the class. If you have any errors, fix them and recompile.

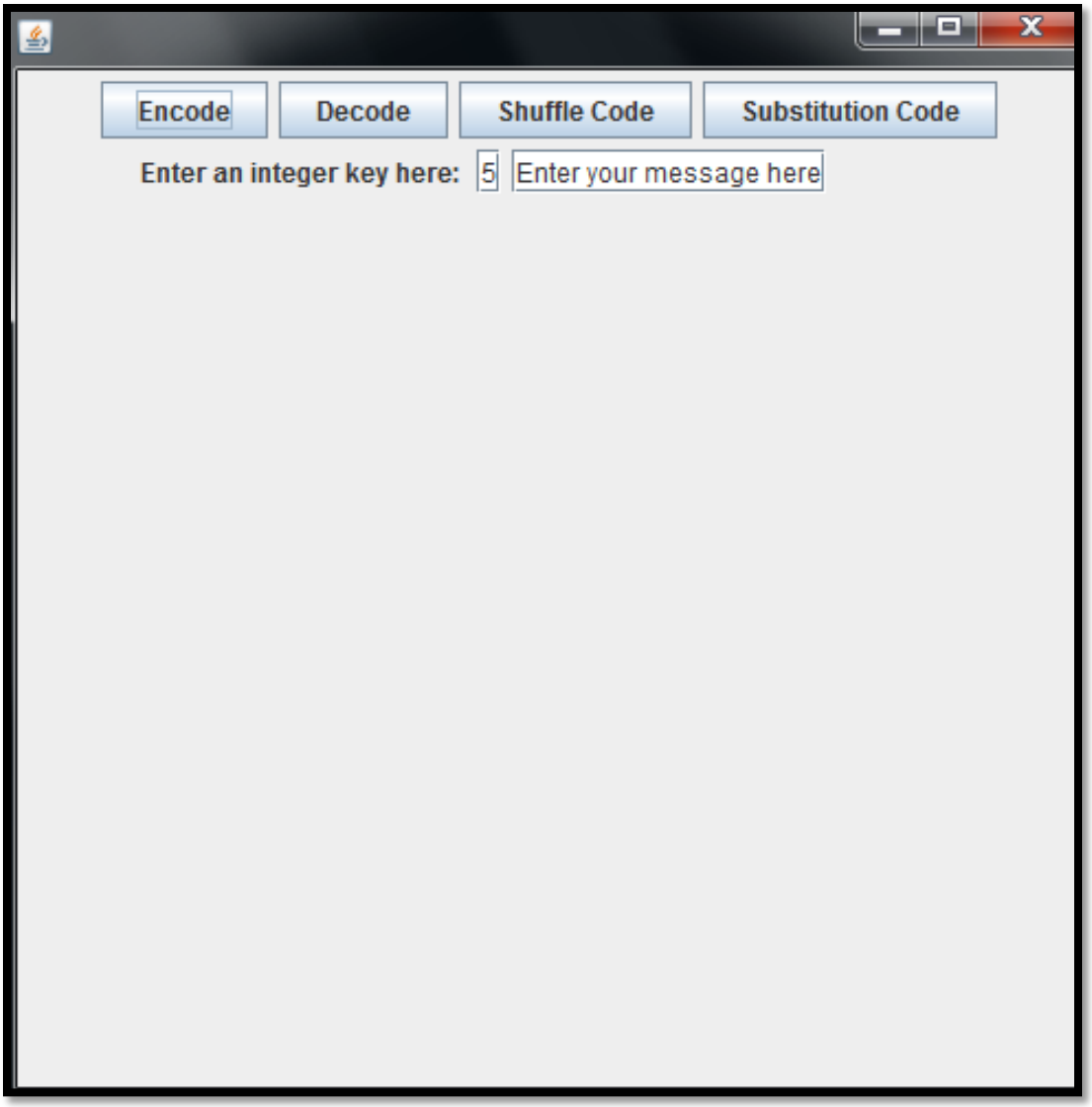
ShowCoder class

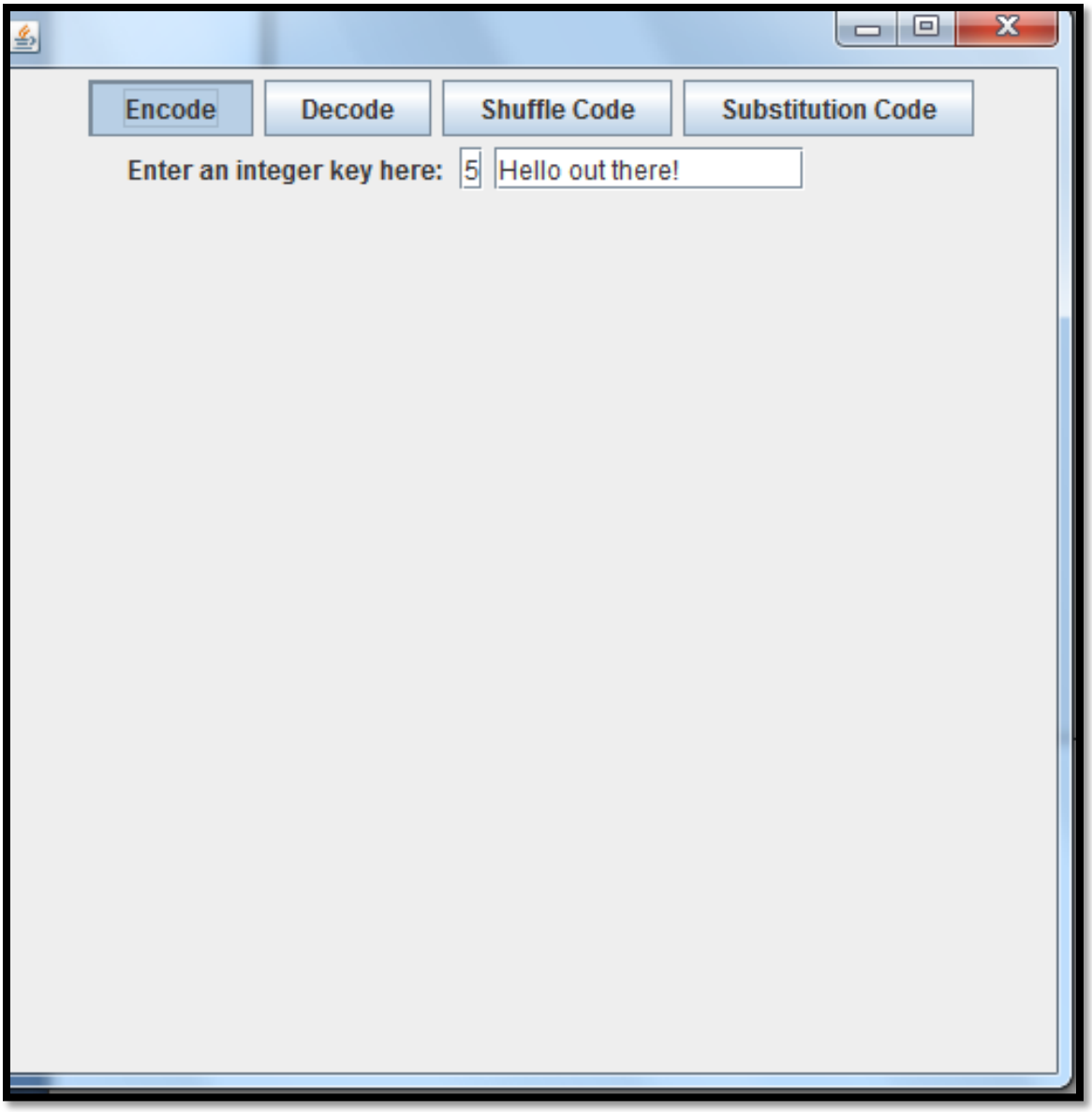
This class will actually run the program, which means it will contain the main method. Our last class created the structure of the frame but could not be run due to the main method not being included. When should you include the main method in the same class where the frame was set up? This depends on whether another you can reuse the contents of the frame for another program or not. It also, is often a preference between programmers. I will type the entire code for this class since it is quite short:

1. Open a new document window in Textpad and save the file as ShowCoder.java.
2. Type the following code for this class:

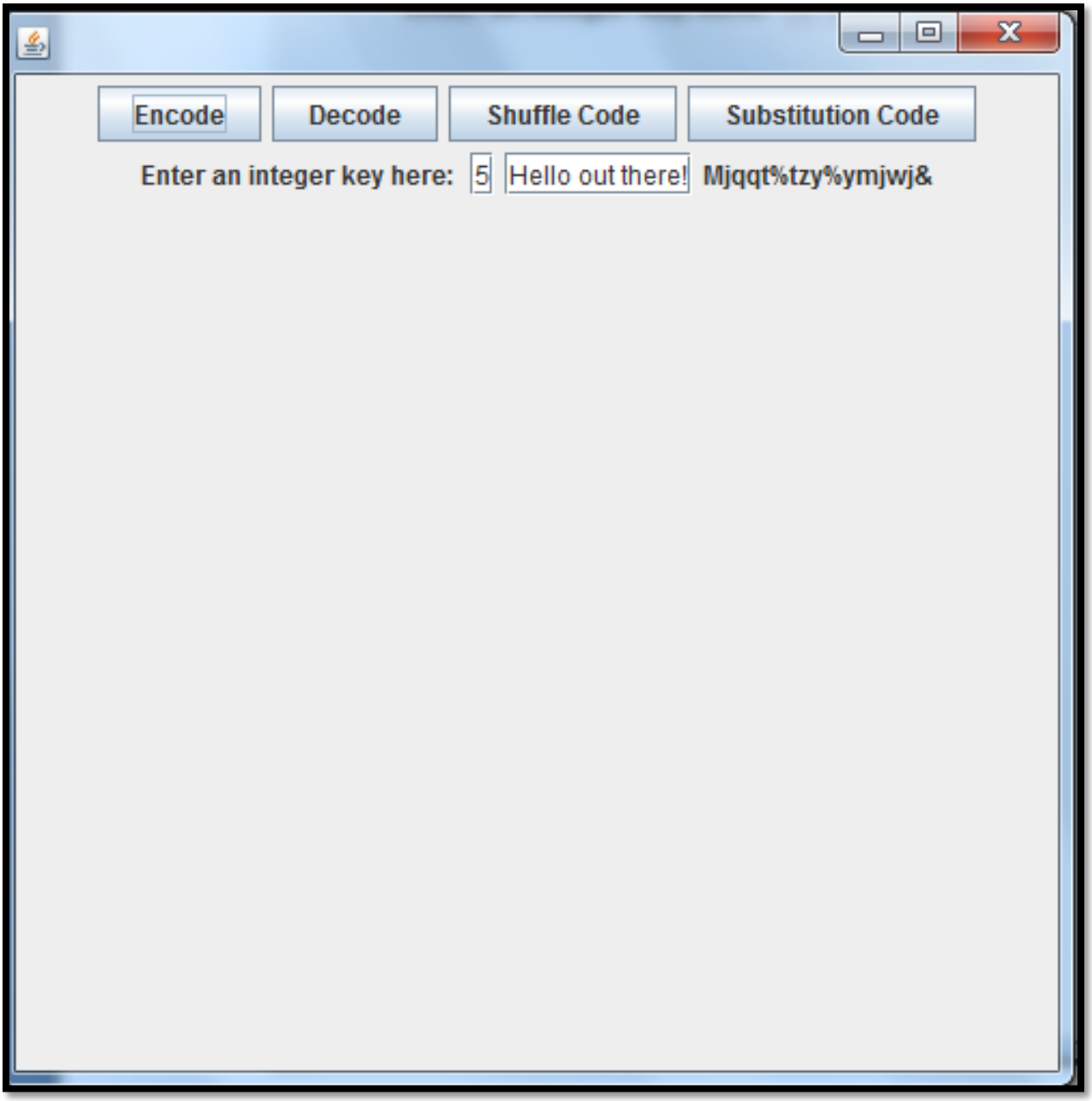
```
public class ShowCoder
{
    public static void main(String[] args)
    {
        CoderFrame gui = new CoderFrame();
        gui.setVisible(true);
    }
}
```

3. Compile the program, fix any errors if necessary, and run the program. Here are some screenshots of the program in action.


Encode
Decode
Shuffle Code
Substitution Code
Enter an integer key here:
5
Enter your message here



I hold down on the Encode button.



The message is encoded. A different coding will occur for the other buttons.

1. Compress the following files into a single zip file and submit to the appropriate drop box.

   CoderFrame.java
   CoderFrame.class
   MessageDecoder.java
   MessageDecoder.class
   MessageEncoder.java
   MessageEncoder.class

ShowCoder.java
ShowCoder.class
ShuffleCipher.java
ShuffleCipher.lcass
SubstitutionCipher.java
SubstitutionCipher.class