Amar Korac
CS304: Project

Part A: Getting Started

Question #A.1

In FileInputStream there is a total of 3 constructors and 9 methods. In FileOutputStream there is a total of 5 constructors and 7 methods. We would use a FileInputStream rather than the Scanner object because the scanner object cannot read inputs from a file specified. It can read only single line inputs that is given by the user. However, FileInputStream object can read a file and the multiple lines. In order to import FileInputStream and FileOutputStream we would write the following code; import java.io.FileInputStream, and import java.io.FileOutputStream.

Question #A.2

The ArrayList class is a resizable array. It is basically a dynamic sized array in Java that implements List interface. The difference between an array and ArrayList is that the size of an array cannot be modified. To add an element to an array you would have to create a new array (same goes for removing an element). However, in an ArrayList the elements can be added or removed whenever. Another difference is that an array can contain primitive data types as well as objects from a class, however ArrayList contains object entries, and not the primitive data types. ArrayList has angular brackets that is used to define generics. The idea for this is to allow types (int, string, etc.) to be a parameter to methods, classes, and interfaces. In order to import the array and ArrayList we would write the following code; import java.util.ArrayList, and import java.util.Arrays.

Question #A.3

The ZipEntry class is used to represent a zip file entry. ZipOutputStream class implements an output stream filter for writing files in the Zip file format. It allows for writing compressed zip archives which can contain one or several other files. In order to import zip we would write the following code; import java.util.zip

Question #A.4

JavaFX is a graphics and media package that enables people to design, create, test, etc. applications that operate across platforms. In short it is as GUI (Graphical User Interface) toolkit for Java. It consists of many components like buttons, text fields, tables, trees, charts etc. FileChooser is a part of JavaFX. It is used to invoke file open dialogs for selecting a single file. The DirectoryChooser shows a directory chooser dialogue which allows the users to select a specific directory. Opening a directory dialog may always result in the null file being returned. It also inherits Object class. To create a file chooser in an application, the class must have the initial directory. Javafx.stage.FileChooser class represents FileChooser.

Part B: The Zipper Class

Question #B.1

The names of the methods are the following: start with a return type void, handle with a return type void, createZip with a return type void, createZip with return type void, main with a return type void, launch with the return type void, showOpenDialog with return type int, showDialog with return type int, setHgap with return type int, getChicken with return type node, and addAll with return type Boolean.

Question #B.2

The main method is calling Application.launch(args). It will pass throught the command line arguments (args) to javafx.application.Application.launch which will open the JavaFX.

Question #B.3

To define a JavaFX main class, first you need to extend the Application class. Public void start(stage) is used. Stage is a window holding something specific. The purpose of the start method is to activate the GUI for the user to be able to select the folder they wish navigate through. There will be a total of 2 buttons. The first will select the directory and the other will select a file. Once selected, the appropriate handle method is executed.

Question #B.4

File is the parameter that should be passed. The method is void. And there is no return value.

Part C: The ZipFile Class

Question #C.1

It is part of the File class and it returns the absolute File object of the given pathname. The File object can also represent a directory. The instance variables are ArrayList<String>, String outputFile, String srcPath, and File file. They represent the list of files, the name of the output file, the location of the source file, and the name of the source file.

Question #C.2

The fileFileList returns the array of files and directories in the directory defined by this abstract path name. The method returns null if the abstract pathname does not denote a directory. It first checks to see if the current object is a file, if it's a file, the current file path is replaced with the new file path. Then if the length of the source path is less than the length of the current file path.

Question #C.3

*See code at the end of the document

Part F: Final Summary

Question #F.1

Time management is the key to all projects. Unfortunately, in my case work and travel got in the way of things. I disliked the idea of being late, so I bought a planner that allows me to organize my work and school at the same time. This way I can be up to date on everything. Being clear and concise went well for me. My experience in writing lab reports for undergrad helped a lot here.

Question #F.2

The most difficult part for me was probably question 3 of part C and finding the correct syntax to match what I was trying to execute.

Question #F.3

The most fun part of the lab was the research itself. I enjoy learning new things. On a few websites like W3 schools, they have interactive examples that people can try the code themselves.

```
import java.io.File;
import java.io.InputStream;
import java.io.FileOutputStream;
import java.util.ArrayList;
import java.util.zip.*;
import java.io.File;

  public class ZipFile
  {
     private ArrayList<String> fileList;
     private String outputFile;
     private String srcPath;
     private File file;

     public ZipFile(File file)
     {
        fileList = new ArrayList<String>();
        outputFile = file.getName() + ".zip";
        srcPath = file.getAbsolutePath();
        this.file = file;
     }

     public void fillFileList(File f)
     {
        if (f.isFile())
        {
           String currFilePath = f.getAbsoluteFile().toString();
           String relativeFilePath;
           if (this.srcPath.length() < currFilePath.length())
              relativeFilePath = currFilePath.substring(this.srcPath.length() + 1, currFilePath.length());
           else
              relativeFilePath = currFilePath.substring(this.srcPath.length(), currFilePath.length());

           fileList.add(relativeFilePath);
        }

        if (f.isDirectory())
        {
           File[] files = f.listFiles();
           for(File file : files){
              fillFileList(file);
           }
        }
     }

     public void makeZip(File dir)
     {
         byte b[] = new byte[1024];
     }
  }
```

```
import java.awt.Desktop;
import java.io.File;
import java.io.IOException;
import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;
import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.geometry.Insets;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.GridPane;
import javafx.scene.layout.Pane;
import javafx.scene.layout.VBox;
import javafx.stage.FileChooser;
import javafx.stage.DirectoryChooser;
import javafx.stage.Stage;
import java.util.*;
import java.util.zip.*;
import java.io.*;

public final class Zipper extends Application
{
   private Desktop desktop = Desktop.getDesktop();

   @Override
   public void start(final Stage stage)
   {
      stage.setTitle("File Chooser");

      final FileChooser fileChooser = new FileChooser();
      final DirectoryChooser directoryChooser = new DirectoryChooser();
      final Button openFileButton = new Button("Select File");
      final Button openDirectoryButton = new Button("Select Directory");

      openFileButton.setOnAction(new EventHandler<ActionEvent>()
      {
          @Override
         public void handle(final ActionEvent e)
         {
          File src = fileChooser.showOpenDialog(null);
            if (src != null)
                    createZip(src);
            else
               System.out.println("No File Selected.");
         }
      });

      openDirectoryButton.setOnAction(new EventHandler<ActionEvent>()
      {
```

```
            @Override
            public void handle(final ActionEvent e)
            {
              File src = directoryChooser.showDialog(null);
                if (src != null)
                    createZip(src);
                else
                    System.out.println("No Directory Selected.");
            }
        });

        final GridPane inputGridPane = new GridPane();

        GridPane.setConstraints(openFileButton, 0, 0);
        GridPane.setConstraints(openDirectoryButton, 1, 0);
        inputGridPane.setHgap(6);
        inputGridPane.setVgap(6);
        inputGridPane.getChildren().addAll(openFileButton, openDirectoryButton);

        final Pane rootGroup = new VBox(12);
        rootGroup.getChildren().addAll(inputGridPane);
        rootGroup.setPadding(new Insets(12, 12, 12, 12));

        stage.setScene(new Scene(rootGroup));
        stage.show();
    }

    private static void createZip(File src)
    {
        ZipFile zipFile = new ZipFile(src);
        zipFile.fillFileList(src);



    }

    public static void main(String[] args) {
        Application.launch(args);

    }
}
```