# README:

The goal of this document is to provide the reader with technical and functional information regarding the **ship code verifier** application.

## Technical Documentation:

Git repository:

<https://github.com/raffivegas/WillSon>

## Versions of important things:

java version "11.0.8" 2020-07-14 LTS

Java(TM) SE Runtime Environment 18.9 (build 11.0.8+10-LTS)

Java 8.0.2610.12

Eclipse IDE for Java Developers

Version: 2020-09 (4.17.0)

Build id: 20200910-1200

JUnit 5 Jupiter 5.7.0

## What does it do?

This application is intended to be run as an executable jar file which can run on most computers which have the latest JVM installed (or the Java version listed above in the 'Versions of important things' section of this document).

The application reads a series of zip code ranges from a file and outputs a condensed set of ranges. The acception criteria (of the story or epic) is as follows: when given a collection of 5-digit ZIP code ranges (each range including both their upper and lower bounds), the application must produce the minimum number of ranges required to represent the same restrictions as the input.

## How does it work?

The input file format is a tab delimited text file (.txt) with zipcodes within brackets and the number pair separated by a comma. For example, a valid file may contain the following:

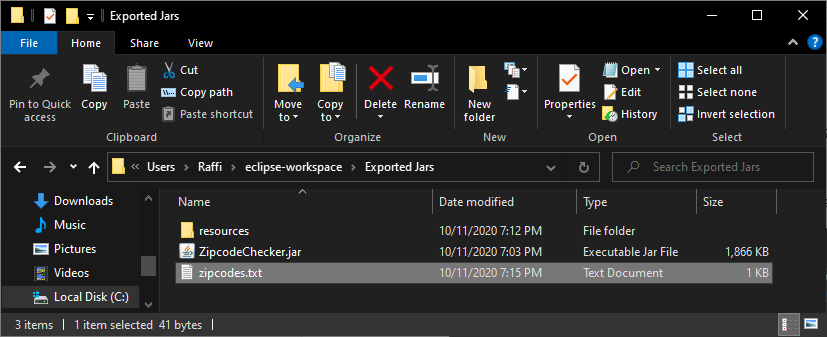
[94133, 94133] [94200,94299] [94226,94399]

As there is no file or input formatting validation implemented yet at this time, please ensure the file is formatted correctly with valid zipcode ranges.

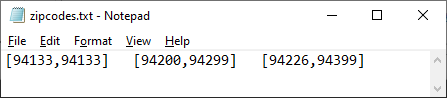
### Exporting and running the application:

After pulling the remote repository to local, open the project in Eclipse and export as an executable jar:

1. Right-click on ship code verifier project folder.
2. Left-click on Export...
3. Select the Runnable JAR File export wizard (you can search for it in the textbox).
4. Choose an available launch configuration.
5. Select and export destination and jar name (i.e. C:\Users\Raffi\eclipse-workspace\Exported Jars\ZipcodeChecker.jar)
6. Leave Library handling default (Extract required libraries...)
7. Click Finish
8. In the folder of the jar file, create a new text file. In this example, I will name the file zipcodes.txt.



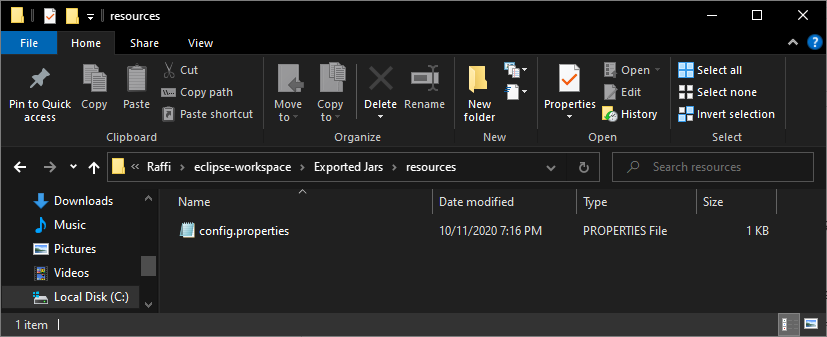
1. Edit the zipcodes.txt file and enter a tab delimited range of zipcodes as described earlier in this document.



1. In the folder of the jar file, create a new folder named 'resources', for example:

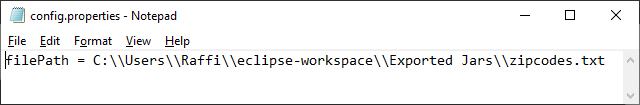
C:\Users\Raffi\eclipse-workspace\Exported Jars\resources

1. Inside the resources folder, either create or copy a config.properties file (copy from the Eclipse project's resources folder).



1. Edit the config.properties file and add a key value pair for the path to the input file (in this case zipcodes.txt) as such:

filePath = C:\\Users\\Raffi\\eclipse-workspace\\Exported Jars\\zipcodes.txt



replacing the folder names with your own.

To run the application and process the zipcodes, double-click on the jar file in the root folder. The application will display a window containing the results of the processing. The application may also be run within Eclipse, with the output displaying in the console. **IF RUNNING WITHIN ECLIPSE**, edit the config.properties file and update it with the correct path to your zipcode input file. If the application does not display anything or displays something incorrectly, please verify the input file format looks correct.

## Objects, object breakdown, code structure, and purpose

### Object and breakdown:

In the IDE/project, the application starting point is the ShipCheckerMain class located in /src/logic.

ShipCheckerMain is the main class which starts the logic calls within the application. It won’t run the rest of the application unless there is a valid file in the expected location.

Also in /src/logic, there is a FileTools class and a ZipcodeCruncher class.

There are two interfaces in src/interfaces, IIOHandler and IPairHandler.

FileTools implements the IIOHandler interface.

ZipcodeCruncher implements the IPairhandler interface.

Unit tests are in /src/tests using the Junit testing library. Junit-jupiter-5.7.0.jar must be added to the referenced libraries in the project.

Unit test coverage covers the primary methods in the FileTools and ZipcodeCruncher classes.

/src/utils/ contains the ConfigReader and StringPair utility classes.

ConfigReader is a static class which reads the /resources/config.properties file.

StringPair holds a pair of strings.

### Structure

The IIOHandler is an interface created to facilitate the validation of the input, to parse the input and return a list, and to prepare a list and send the output. There are 3 interface methods:

* List<T> parsePackage();
* void validateInput();
* void sendPackage(List<StringPair> finalList);

The FileTools class implements IIOHandler.

* Filetools’ parsePackage method reads values from a file and parses them to a list of StringPairs.
* Filetools’ validateInput method verifies a file exists and the file is not empty.
* Filetools’ sendPackage method takes a list containing strings and sends it to its destination (outputs to screen).

A generic interface was implemented to enable scalability for future changes if different Input/Output vehicles are implemented. The program currently works with an input file and outputs a message to the screen. In the future, there may be an input received from a messaging service, and it may be desired to output to a queue or to a file. For this reason, the IIOHandler interface exists, so in the future, classes like MessageHandler may exist which implement the IIOHandler interface and they’ll have methods to validate the input (validateInput), parse the received input (parsePackage), and send the package (sendPackage), regardless of the input and output object types (files or xml messages).

The IPairHandler is an interface created to facilitate the common business functions for a given input. There are 2 interface methods:

* List<T> sortPairs(List<T> myList);
* List<T> findFinalPairs(List<T> mySortedList);

The ZipcodeCruncher.java class implements IIPairHandler.

* ZipcodeCruncher’s sortPairs method takes a list of StringPairs and returns a sorted list of StringPairs. The sort is done using the value on the left of each pair, in ascending order.
* ZipcodeCruncher’s findFinalPairs method takes a sorted list of StringPairs and returns a cleaned up list of zipcode ranges (containing the minimum number of pairs to represent the original list).

The assumption is the input will always be a pair of something. The interface will allow the application to handle other types of pairs in the future, such as area codes, countries, or phone numbers, rather than zipcodes. So there may be an AreacodeCruncher class in the future which would implement the IPairHandler interface and it would have a sortPairs function and a findFinalPairs function. There may be a CountryCruncher or StateCruncher which might have a range of countries or states which can or can’t be shipped to, and they’d have a sortPairs and findFinalPairs implementation.

Both interfaces are generic to be able to work with other pair type classes which may come later, such as CountryPair, IntPair, CharPair, etc.

## Tests

The naming convention for the tests lists the name of the class followed by the method name being tested followed by what it should achieve.

README v0.0.2 - Raffi Tchakmakjian, 10/12/2020