Concordia University
Comp 249 – Fall 2016
Object-Oriented Programming II
Assignment 3 – Due 11:59 PM - Friday November 11, 2016

Purpose: The purpose of this assignment is to allow you practice Inheritance, Polymorphism, Exception Handling, and File I/O, as well as other previous object oriented concepts.

There are two independent parts in this assignment, Part I and Part II.

Part I

Post Canada is not happy with their current software that keeps track of activities of a delivery truck transporting packages from one city to another for subsequent local delivery by other trucks. You accepted to take the challenge as a programmer. Your first project is to write a new software application. Certain simplifying assumptions have been made to allow you to concentrate on OOP concepts rather than the complete realism of the simulation.

Implementation

Implement the following classes: `Truck`, `Package`, `Letter`, `Box`, `Crate`, `Metal Crate` and `Wood Crate`, and `CargoTest` to simulate the long-distance transportation of packages by a truck.

Your solution must conform to the following:

- A set of classes, corresponding to `Truck`, `Package`, `Letter`, `Box`, `Crate`, `Metal Crate` and `Wood Crate`, must be designed, declared and implemented.
- The `Truck` class should contain the driver's name, the originating and destination cities, its gross and unloaded weight, the number of packages and additional information about each package that it carries. You must use an array of type Package. You may assume that trucks can carry a maximum of 200 packages.
  Trucks will be assigned a driver, loaded, weighed before and after loading, driven and unloaded. In addition, each truck has a gross income equal to the total shipping costs of the packages.

- The `Package` class is the abstract base class for a 'packages' hierarchy. Each package has a tracking number, weight and shipping cost, based on its weight. The tracking number encodes the originating and destination cities as well as the type of package. If the least significant digit of the tracking number is a 0, the package is a letter whose weight is given in ounces. Similarly if the unit's digit is a 1, 2 or 3 then the package is a box, wooden crate or metal crate, respectively, whose weight is given in pounds. We are not concerned with the source and destination information for each package, so we will not examine the tracking number for this information.

- Shipping costs are computed as follows.
  - Letters cost $.05/ounce up to a maximum of 32 ounces (2 pounds).
  - Boxes cost $2/lb up to a maximum of 40 pounds.
  - Wooden crates cost $2.50/lb up to a maximum of 80 pounds.
  - Metal crates cost $3.00/lb up to a maximum of 100 pounds.

- Your load method must check for packages that are too heavy. If so, then throw an exception and print out that package's information (package type, tracking number, weight). Of course, don't load that package. If the package is ok to load, "get some space" for it and load it. NOTE: All Exceptions thrown will be of your own type – `PackageException`.

- If you determine a package is not one of the ones your truck is allowed to carry, throw an exception and print out the information (unknown package, tracking number, weight). NOTE: All Exceptions thrown will be of your own type – `PackageException`.

- If the truck is full, throw an exception and then printout the information (package type, tracking number, weight) along with the fact that it was not loaded because the truck is full. NOTE: All Exceptions thrown will be of your own type – `PackageException`.

- When packages are unloaded the package information is printout, and deducted from the truck load.

- You must handle weight conversion in two ways:
  - For package:
    This can be implemented using two methods `toOunces` and `toPounds`, that will accept a double as a parameter. You will implement them in `Package`, or the subclasses, it is entirely up to you.
  - For truck
    This can be implemented using two methods `toKilograms` and `toPounds`, that will accept a double as a parameter. You must print out the weight of the vehicle in kilograms and in pounds.

To test your program you are to write the `CargoTest` code class which behaves in the following way:

- Instantiate a truck, with its driver, unloaded weight and both the originating and destination cities
- Load the truck with packages.
- Determine the gross income earned by shipping of the cargo.
- Weigh the truck (AFTER it has been completely loaded).
- Drive the truck to its destination – this will simply be nice output to the screen that states the truck and its driver is driving from originating city to destination city.
- Unload the truck.

– Presents the following menu to the user, hence possible actions:

```
What would you like to do?
    1. Start a cargo
        a. Driver name
        b. Unload weight(kg; lb)
        c. Originating city
        d. Destination city
    2. Load the truck with packages
        a. Package tracking number
        b. Package weight(oz; lb)
        c. Package shipping cost
    3. Unload a package
    4. The number of packages loaded
    5. The gross income earned by shipping of the cargo
    6. Weight the truck(after it has been completely loaded)
    7. Drive the truck to destination
    0. To quit
Please enter your choice and press <Enter>:
```

Part I Evaluation

You will be evaluated mostly on the implementation of the classes, the relationship (is-a vs has-a) between the classes, abstract classes, methods' implementation, methods' overriding and the access, rights of the class members, as well as exception handling.

Required documents

- UML diagram.
- Source code in Java.
- Javadoc file in HTML.
- Test cases for all the functionalities of the menu options.

Note- you need to hard code the array of packages, to validate when you try to load more than the allowed 200 packages.

Part II

Warships ….

The **Warship** class has the following attributes: a *serial number* (long type), a *name* (String type), a *creation year* (int type), an *owner country* (String type), a *price* (double type) and a *speed* (int type). It is assumed that both the name and the owner country are recorded as continuous strings with "_" to separate the different words if any (i.e. a name can be Royal_Fighter, and country can be United_Kingdom).

The file ***Initial_Warship_Info.txt***, which one of its versions is provided with this assignment, has the information of various warships that take part of an internationl mission. However, this file is always created manually by a 3rd party firm, which is histroically known to lack pricision and so errors in the serial numbers are expected to exist in this file. In specific, due to a cut-and-paste practice by employees of this firm, some serial numbers are mistakenly re-recorded later in the file as the serial numbers of other following warrships in that file. Consequently, a serial number can either appears once in the file, which is the correct case, or appears multiple times, in which case the second, and following, appearances are in error.

You must notice that the file *Initial_Warship_Info.txt* changes regularly, and it may have many records, or no records at all, depending on the current participants in the mission. The file provided with this assignment is only one version of the file, and must not be considered as the general case when writing your code.

For part II, you are required to:

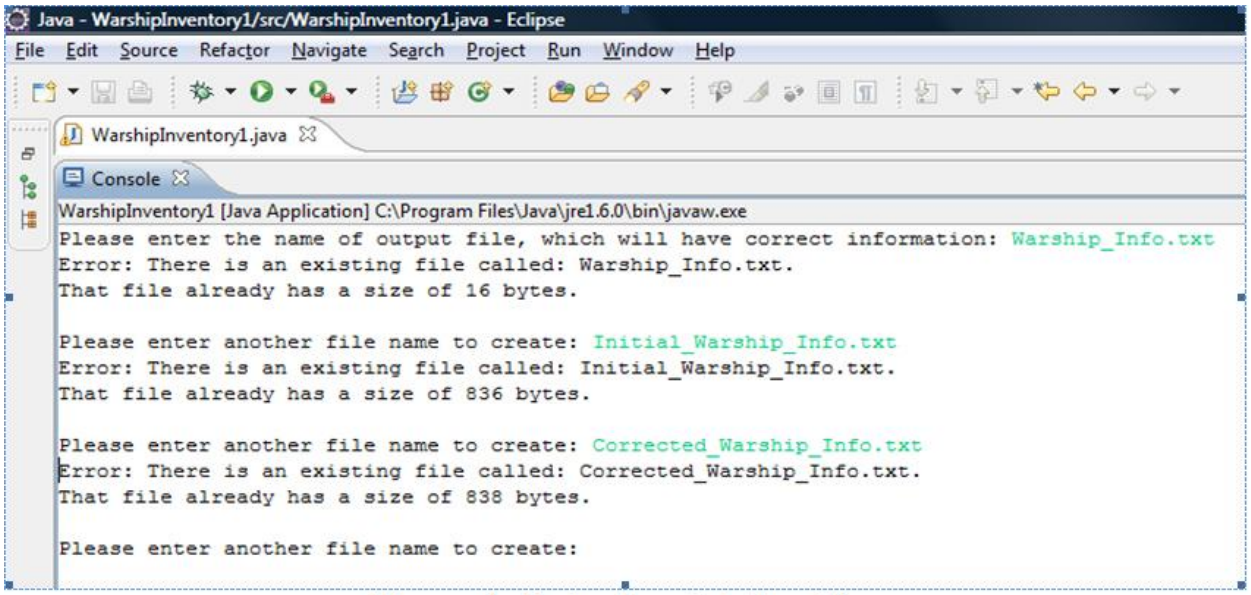
1. Write the implementation of the **Warship** class according to the above given specification.
2. Write the implementation of an exception class called **DuplicateSerialNumberException**, which extends the **Exception** class. Should a duplicate serial number is detected at any point, an object from this class will be thrown. More details will follow below.
3. Write the implementation of a public class, called **WarshipInventory1**, which will utilize the **Warship** class and the *Initial_Warship_Info.txt*, as explained below. The class has a static array of warships, called *wsArr[]*, and few methods. Beside the main() method, the class must have the following methods:
    a. A method called ***fixInventory()***, which accepts two parameters, an input file stream name and an output file stream name. The first parameter is the stream related to the *Initial_Warship_Info.txt*. The second parameter is related to an output file name, which will be entered by the user prior to calling this method (at the main() method). This output file will eventually store a correct version of the inventory. More information on that will follow below.
    b. A method called ***displayFileContents()***, which accepts an input file stream name, then displays the contents of this file to the standard output (the screen).
    c. You can add any other methods to that class if you wish to.

Here are the details of how your program must behave:

- In the main() method, your program must prompt the user to enter the name of the output file that will be created to hold the modified/correct inventory. This output file is theoretically a copy of the *Initial_Warship_Info.txt*, but with all the duplicate serial numbers corrected.

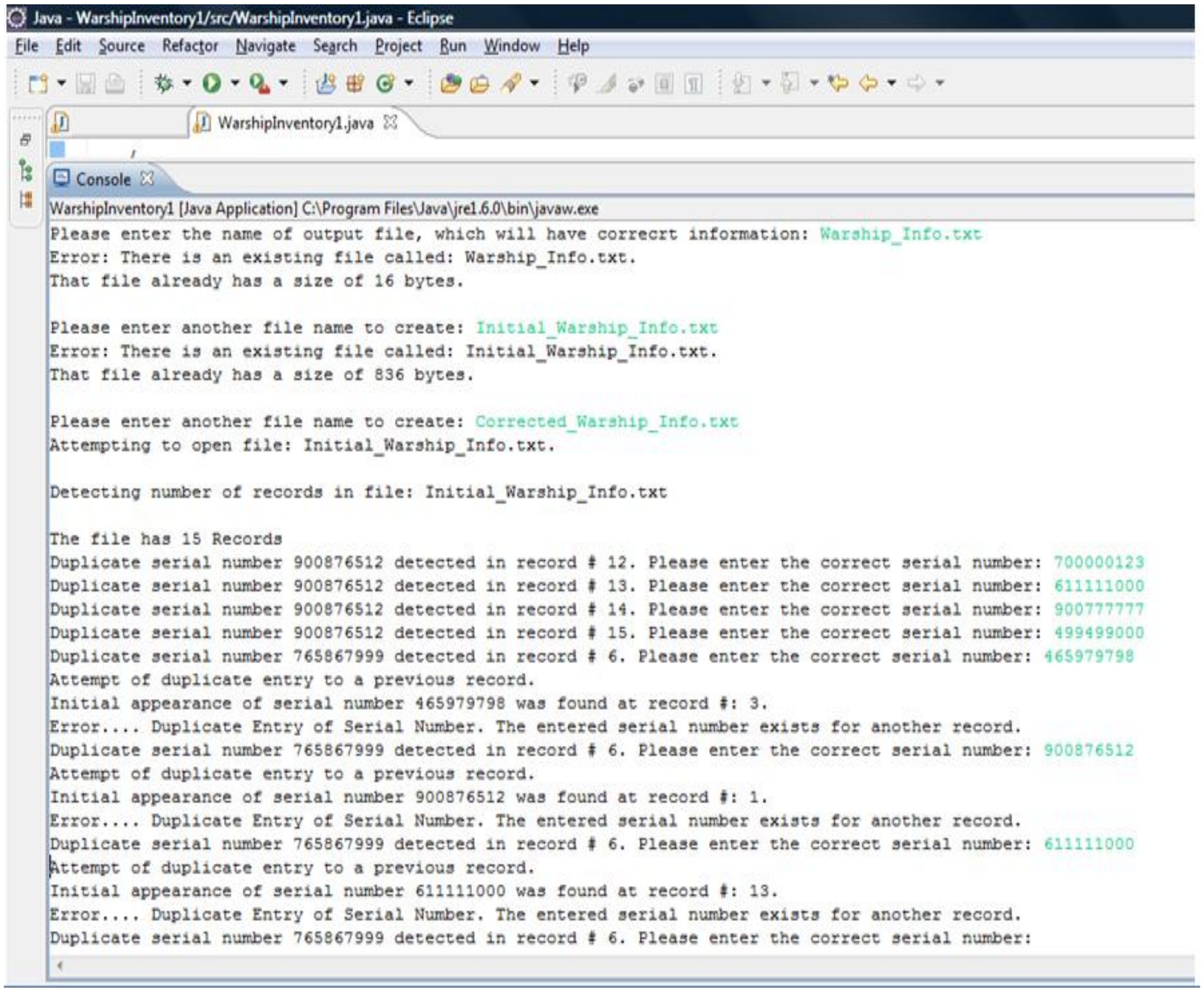
- If the entered file name by the user matches the name of an existing file, then the program must reject that name indicating to the user that a file with that name already exists, display the size of this existing file (in bytes) to the user, then prompt the user to enter a new name. This process would repeat indefinitely, until the user finally enters a name for a non-existing file. See Figure1 for illustration.
- Once the user finally enters a correct file name for the output file, the program will attempt to establish an input and output streams for the *Initial_Warship_Info.txt* and that output file accordingly. This process may surely throw specific exceptions, and your program must handle all these exceptions properly.
- The *fixInventory()* method (see details below) will utilize the *wsArr[]* array as follows: All warship objects recorded in the *Initial_Warship_Info.txt* file must first be copied into that array. All detections and corrections of the serial numbers will be conducted on that array. Once the array has finally all correct information, the objects will be recorded to the output file. However since it is always unknown at the time the program starts how many records are in the *Initial_Warship_Info.txt* file, you must first find this information. You may, and should, add a private helping method to the **WarshipInventory1** class to do so. Once the number of records is know, the array must be set to that size.
- If the number of records in the *Initial_Warship_Info.txt* was detected to be zero or one; the case when the file is empty or has only one record, then the program must display a message indicating that, performs any needed operations (such as closing files), then exits, since there is nothing to be fixed.
- If the *Initial_Warship_Info.txt* has more than one record, then finally the *fixInventory()* method will be called to create a the new output file with the correct information. The exact details of this method are as follows:
    - The method will accept two stream names for the input and output files,
    - The method must read each warship record from the input file and creates a warship object in the array *wsArr[]* based on that record,
    - Once the entire input file is read into the array (notice that the array has real warship objects, and not just information), the method starts to trace that array from start to end looking for any serial number duplicates,
    - If a serial number duplicate is detected, then the method displays a message to the user indicating that, and prompt the user to enter a new serial number,
    - You should notice that this new number must not be a duplicate of any other existing record, and your program must guarantee that. Should the user enters a serial number that is still a duplicate, the program must throw a **DuplicateSerialNumberException**, which must be catched to display a message indicating that, then the user must be prompted again to enter a new serial number. Further bad entries will result in the same action; that is throwing of the **DuplicateSerialNumberException** object, catching it to display the message, and the user is prompted again. Effectively, this process will repeat indefinitely until the user enters a correct serial number. Again, you may, and should, create a private helping method to find if a duplicate exists in the array. See Figure 2 for illustration.
    - Finally, once all the duplicate serial numbers are removed, the new information of the objects in the *wsArr[]* must be copied to the output file. This output file has now the correct information.
- Upon the return of the *fixInventory()* call in the main() method, the program must use the *displayFileContents()* method to display the information of both the *Initial_Warship_Info.txt* file, as well as the created output file. See Figure 3 for illustration. Hint: You must carefully keep track of the opening and closing of these files.

Below are sample snapshots of the program behavior for further illustrations:


Java - WarshipInventory1/src/WarshipInventory1.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
WarshipInventory1.java
Console

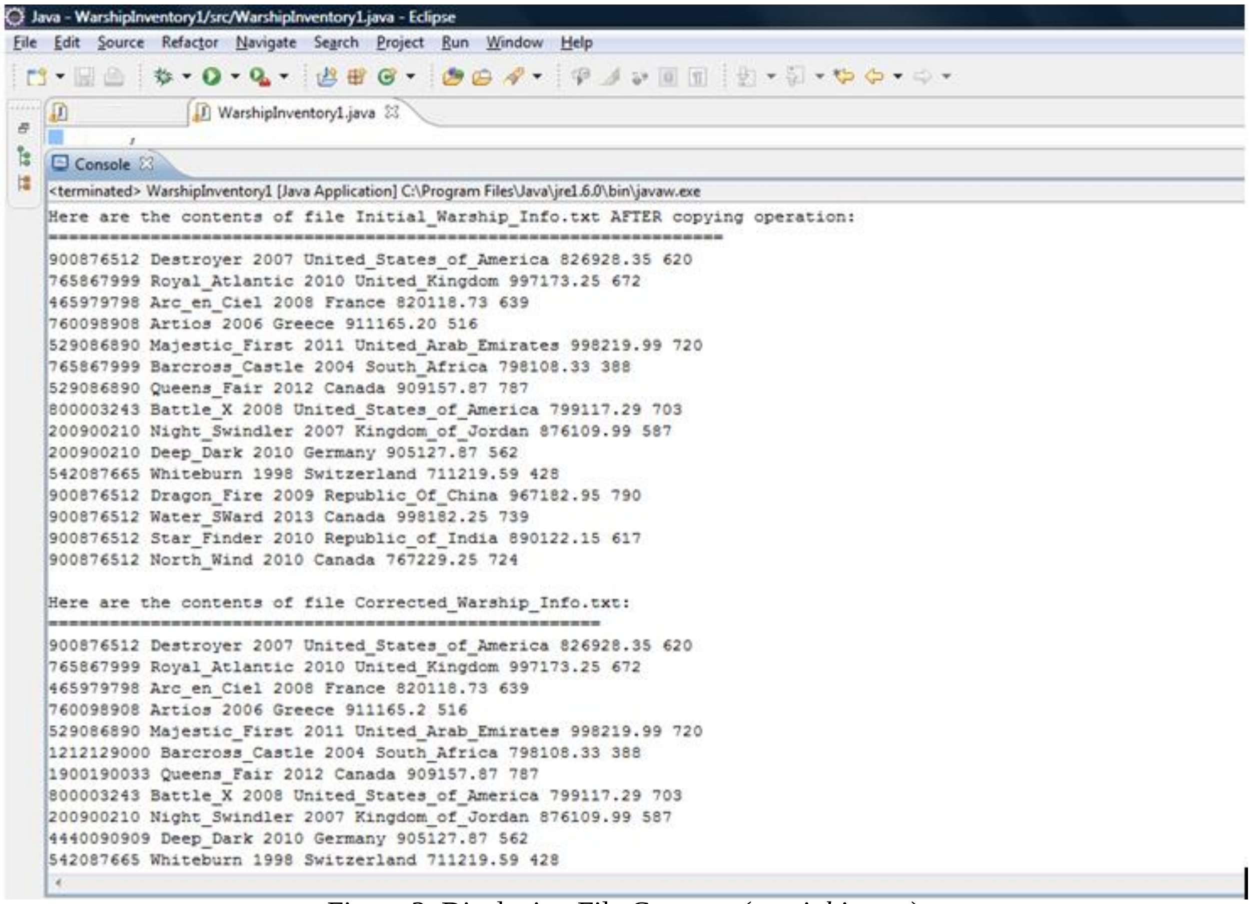
```
WarshipInventory1 [Java Application] C:\Program Files\Java\jre1.6.0\bin\javaw.exe
Please enter the name of output file, which will have correct information: Warship_Info.txt
Error: There is an existing file called: Warship_Info.txt.
That file already has a size of 16 bytes.

Please enter another file name to create: Initial_Warship_Info.txt
Error: There is an existing file called: Initial_Warship_Info.txt.
That file already has a size of 836 bytes.

Please enter another file name to create: Corrected_Warship_Info.txt
Error: There is an existing file called: Corrected_Warship_Info.txt.
That file already has a size of 838 bytes.

Please enter another file name to create:
```

Figure 1: Detecting Existing Files


Java - WarshipInventory1/src/WarshipInventory1.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
WarshipInventory1.java
Console

```
WarshipInventory1 [Java Application] C:\Program Files\Java\jre1.6.0\bin\javaw.exe
Please enter the name of output file, which will have correcrt information: Warship_Info.txt
Error: There is an existing file called: Warship_Info.txt.
That file already has a size of 16 bytes.

Please enter another file name to create: Initial_Warship_Info.txt
Error: There is an existing file called: Initial_Warship_Info.txt.
That file already has a size of 836 bytes.

Please enter another file name to create: Corrected_Warship_Info.txt
Attempting to open file: Initial_Warship_Info.txt.

Detecting number of records in file: Initial_Warship_Info.txt

The file has 15 Records
Duplicate serial number 900876512 detected in record # 12. Please enter the correct serial number: 700000123
Duplicate serial number 900876512 detected in record # 13. Please enter the correct serial number: 611111000
Duplicate serial number 900876512 detected in record # 14. Please enter the correct serial number: 900777777
Duplicate serial number 900876512 detected in record # 15. Please enter the correct serial number: 499499000
Duplicate serial number 765867999 detected in record # 6. Please enter the correct serial number: 465979798
Attempt of duplicate entry to a previous record.
Initial appearance of serial number 465979798 was found at record #: 3.
Error.... Duplicate Entry of Serial Number. The entered serial number exists for another record.
Duplicate serial number 765867999 detected in record # 6. Please enter the correct serial number: 900876512
Attempt of duplicate entry to a previous record.
Initial appearance of serial number 900876512 was found at record #: 1.
Error.... Duplicate Entry of Serial Number. The entered serial number exists for another record.
Duplicate serial number 765867999 detected in record # 6. Please enter the correct serial number: 611111000
Attempt of duplicate entry to a previous record.
Initial appearance of serial number 611111000 was found at record #: 13.
Error.... Duplicate Entry of Serial Number. The entered serial number exists for another record.
Duplicate serial number 765867999 detected in record # 6. Please enter the correct serial number:
```

Figure 2: Detecting Existing Serial Numbers


Java - WarshipInventory1/src/WarshipInventory1.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
WarshipInventory1.java
Console

```
<terminated> WarshipInventory1 [Java Application] C:\Program Files\Java\jre1.6.0\bin\javaw.exe
Here are the contents of file Initial_Warship_Info.txt AFTER copying operation:
==================================================================

900876512 Destroyer 2007 United_States_of_America 826928.35 620
765867999 Royal_Atlantic 2010 United_Kingdom 997173.25 672
465979798 Arc_en_Ciel 2008 France 820118.73 639
760098908 Artios 2006 Greece 911165.20 516
529086890 Majestic_First 2011 United_Arab_Emirates 998219.99 720
765867999 Barcross_Castle 2004 South_Africa 798108.33 388
529086890 Queens_Fair 2012 Canada 909157.87 787
800003243 Battle_X 2008 United_States_of_America 799117.29 703
200900210 Night_Swindler 2007 Kingdom_of_Jordan 876109.99 587
200900210 Deep_Dark 2010 Germany 905127.87 562
542087665 Whiteburn 1998 Switzerland 711219.59 428
900876512 Dragon_Fire 2009 Republic_Of_China 967182.95 790
900876512 Water_SWard 2013 Canada 998182.25 739
900876512 Star_Finder 2010 Republic_of_India 890122.15 617
900876512 North_Wind 2010 Canada 767229.25 724

Here are the contents of file Corrected_Warship_Info.txt:
======================================================

900876512 Destroyer 2007 United_States_of_America 826928.35 620
765867999 Royal_Atlantic 2010 United_Kingdom 997173.25 672
465979798 Arc_en_Ciel 2008 France 820118.73 639
760098908 Artios 2006 Greece 911165.2 516
529086890 Majestic_First 2011 United_Arab_Emirates 998219.99 720
1212129000 Barcross_Castle 2004 South_Africa 798108.33 388
1900190033 Queens_Fair 2012 Canada 909157.87 787
800003243 Battle_X 2008 United_States_of_America 799117.29 703
200900210 Night_Swindler 2007 Kingdom_of_Jordan 876109.99 587
4440090909 Deep_Dark 2010 Germany 905127.87 562
542087665 Whiteburn 1998 Switzerland 711219.59 428
```

Figure 3: Displaying File Contents (*partial image*)

Assignment Submission

– Create **one** zip file, containing the two parts: I, and II separately, where each part contains the files (.java and .html and test cases).
– Naming convention for the zip file:
    - o If the work is done by 1 student: Your file should be called *a#_studentID_Part#*, where a# is a3 for assignment3, *studentID* is your student ID number, and *Part#* PartI, PartII or PartIII.
    - o If the work is done by 2 students: The zip file should be called *a#_studentID1_studentID2*, where a# is a3 for assignment3, *studentID1* and *studentID2* are the student ID numbers of each student, and *Part#* PartI, PartII or PartIII.
– Assignments must be submitted in the right folder of the assignments. Upload your zip file at the URL: https://fis.encs.concordia.ca/eas/ as *Programming Assignment* ***3.*** **Assignments uploaded to an incorrect folder will not be marked and result in a zero mark. No resubmissions will be allowed.**

Evaluation Criteria for Assignment 3 (10 points)

Part I (5 points)	**5**
UML class diagram describing the classes and the class relationships, based on problem specification.	0.5
Programming style, Comments and Javadoc.	0.5
Implementation of class hierarchy, abstract class, handling exception, methods (constructor, accessor, mutator, helping, static), method overriding, access rights, etc.	2.5
CargoTest code (main class).	0.5
Output test cases (correctness and completeness).	**1**
Part II (5 points)	5
Warship & DuplicateSerialNumberException classes	0.5 pt
Correct implementation to fix the inventory	4 pt
displayFileContents() method	0.5 pt