Assignment 7 – Ship Registry

CMSC203

Concepts tested by this program:

Reading from / Writing to a file

ArrayList

For each loop

Inheritance

Polymorphism

Abstract classes

Overriding methods

super()

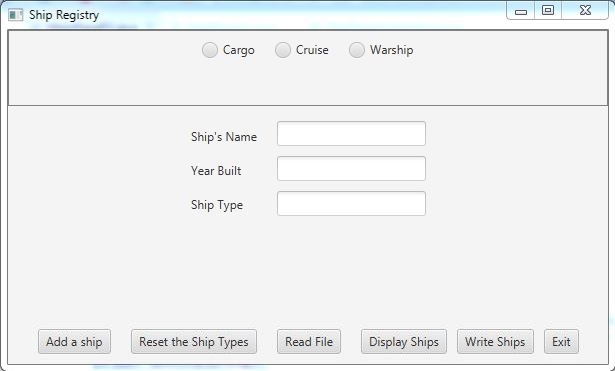
Sort ArrayList

Writing to a file

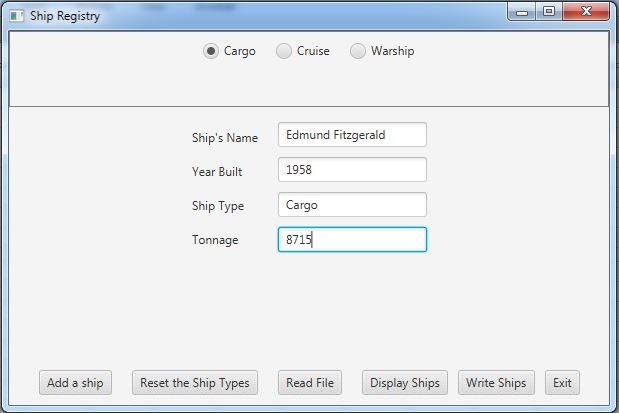
In this assignment, you will enter, store, and retrieve information about ships. You have been provided with a JavaFX GUI, a JUnit test file, and an interface file that will guide you in developing the application. (Note: the information about each ship is not exactly as the maritime experts would state it. Follow these requirements instead.)

**Operation:**

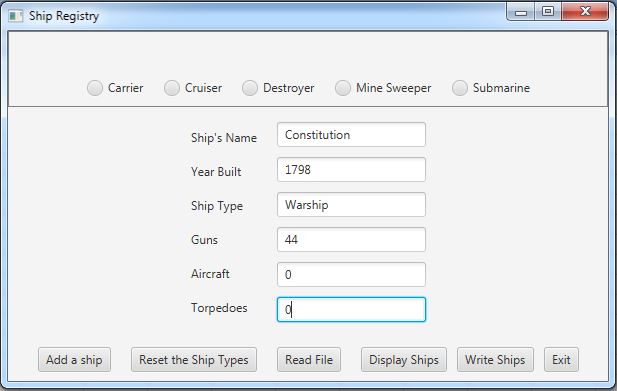
1. When the user starts the application, the following window will be presented (JavaFX GUI provided):



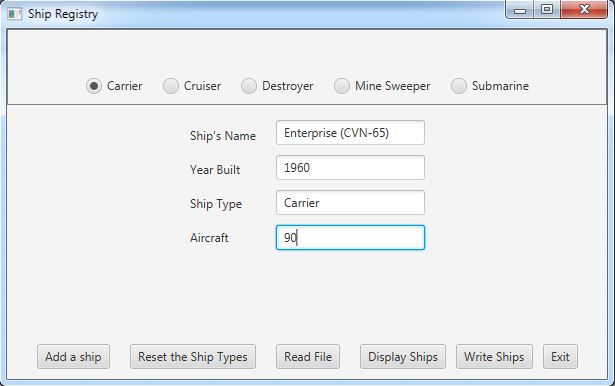
1. The user will select a radio button specifying which type of ship they wish to create. The ship type text field will be pre-populated, but the user will type in the Ship’s name and year built, and for a cargo ship, the tonnage, or for a cruise ship, the number of passengers.



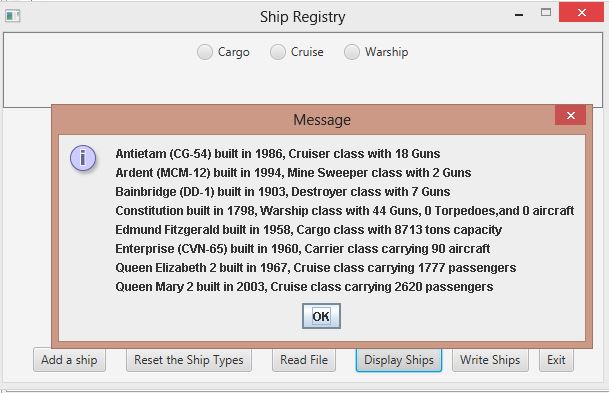
1. If the user selects the Warship radio button, another set of radio buttons will be displayed corresponding to six categories of warship. If the user decides to enter a general warship (i.e., not one of the six sub-types), they are presented with appropriate textboxes.



1. If the user decides to enter one of the six sub-categories of warships by selecting a radio button, they will be provided with a pre-populated ship type and appropriate text fields to fill in.



1. In any of the above cases, the user will then select the “Add a Ship” button to create an instance of the specified ship and add it to the ShipRegistry arraylist. If the user wishes to return to the radio-buttons for the three class of ships, they can select “Reset the Ship Types”.
2. To present a window showing the current ships, the user should select “Display Ships”. The ships should be displayed in sorted order, by the alphabetic order of the ship name.



1. To read from a file, select the “Read File” button. The user will be presented with a file explorer from which they can select an input file. If the file is correctly formatted, each line will become a ship added to the ShipRegistry. Similarly, to write the current ShipRegistry to a file, the user selects “Write Ships”.

## **Specifications**

* General - Write a Java program to allow the user to enter information about a ship, add it to a “registry”, write the ship information to a file, and read the information into the registry.
* Create an abstract class called Ship with private fields ‘name’ and ‘year’ of type String, and ‘type’ of ShipType, and public getters and setters for each. The field ‘name’ represents the name of the ship, ‘year’ is the year commissioned, and ‘type’ is a string that represents one of the enumeration values below.
* Data Elements: Create three classes that inherit from Ship: CargoShip, CruiseShip, and WarShip. Instances of WarShip will be further refined into five types, but will all be WarShip objects.
* Create an enumeration class named ShipType.java that has the following values: CARGO, CRUISE, WARSHIP, CARRIER, CRUISER, DESTROYER, MINE\_SWEEPER, and SUBMARINE. These categories are meant to represent the following:

CARGO – a merchant ship carrying cargo

CRUISE – a passenger ship

WARSHIP – a general warship, which is not further refined by the following categories (we will record guns, torpedoes, and aircraft, although older warships only had guns)

CARRIER – an aircraft carrier

CRUISER – a cruiser (while cruisers have sophisticated electronic and sonar equipment, missile launchers, etc, which define them, we will simplify them by just recording a number of guns)

DESTROYER – a destroyer (while destroyer have torpedo launchers, missile launchers, depth charges, and sophisticated electronic and sonar equipment, etc, we will simplify them by just recording a number of guns)

MINE\_SWEEPER – a counter-mine ship (while mine-sweepers have sophisticated electronics, sonar, other counter-mine equipment, we will simplify them by just recording a number of guns)

SUBMARINE – a submarine (while modern submarines have missile launch capability, we will simplify them by just recording a number of torpedo launchers)

* Create a Data Manager class called ShipRegistry that implements ShipRegistryInterface. It will contain an Arraylist which holds Ship objects (Cargo, Cruise, and Warship all extending Ship). It will have methods required by the ShipRegistryInterface, i.e., addShip, getShips, getShipDescriptions, getWarshipDescriptions, readFile, and writeFile.
* Create a toString method for each ship class that will return a String describing the ship’s name, year commissioned, and type, and the following field or fields depending on ship type. For each ship type, the fourth and subsequent fields will be:

CARGO – number of tons of cargo

CRUISE – number of passengers

WARSHIP – number of guns, number of torpedo launchers, and number of aircraft

CARRIER – number of aircraft

CRUISER – number of guns

DESTROYER – number of guns

MINE\_SWEEPER – number of guns

SUBMARINE – number of torpedo launchers

* Create a toString() method for the ShipRegistry class that sorts the current ships according to the alphabetic order of the ship’s name, and then iterates through the current ships and appends each ship’s toString method to the string returned. This will require the Ship class to implement the Comparable interface, which requires Ship to implement the compareTo method. This method should get the ships’ names and compare the names using the library compareTo method for Strings.
* Create a writeString method for each ship class and category that returns an exact string that will be written to an output file, and then perhaps read back in. The writeString method for each type of ship will have the following form. Note that all fields are separated by commas, so that the output file that is written using this method becomes a comma-delimited file (e.g., a csv file). These methods do need to write in exactly the following format. The first three fields are the string representation, and the subsequent field(s) is a string representation of an integer.

CARGO – “name,shipType,year,tonnage”

CRUISE – “name,shipType,year,passengers “

WARSHIP – “name,shipType,year,guns,torpedos,aircraft”

CARRIER – “name,shipType,year,aircraft”

CRUISER – “name,shipType,year,guns”

DESTROYER – “name,shipType,year,guns”

MINE\_SWEEPER – “name,shipType,year,guns”

SUBMARINE – – “name,shipType,year,torpedoes”

* Your ShipRegistry methods must pass the JUnit Test class (provided). Do not modify this file. You also are provided a JUnit STUDENT Test file, which you should modify to further test your program. Your program will also be tested by private JUnit tests run by the instructor.
* Create Javadoc for your classes.
* Turn in all your java code, including ShipRegistryInterface.java, ShipRegistryTest.java, and ShipRegistrySTUDENTTest.java.
* Create and turn in a final UML Class Diagram and a Design Reflection Word document, explaining how your initial design was modified, and how your final design differs from or matches the design solution.

**Grading Rubric**

**CMSC 203 Project #7**

Name \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Overview:**

There are two parts to the rubric. First, the project is graded on whether it compiles, whether it runs without errors, and whether it satisfies the specifications. These points add up to 100. Second, the score is decremented if various requirements are not met, e.g., no Javadoc, no UML diagram, uses constructs that are not allowed, etc.

**PROGRAMMING** (100 pts)

Compiles 40 pts \_\_\_\_\_

Accuracy

Passes JUnit tests (provided) 10 pts \_\_\_\_\_

Passes JUnit student tests 10 pts \_\_\_\_\_

Passes private instructor tests 10 pts \_\_\_\_\_

Execution: runs without errors (either run-time or logic errors) 30 pts \_\_\_\_\_

Possible Sub-total 100 pts \_\_\_\_\_

**REQUIREMENTS** (Subtracts from Programming total)

**Documentation:**

Javadoc was not submitted - 10 pts \_\_\_\_\_

STUDENT JUnit tests were not submitted - 5 pts \_\_\_\_\_

Documentation within source code was missing or incorrect - 5 pts \_\_\_\_\_

Description of what class does was missing

Author’s Name, @author, was missing

Methods not commented properly using Javadoc @param, @return

**Programming Style:**

Incorrect use of indentation, statements, structures - 4 pts \_\_\_\_\_

**Design:**

Implementation does not match final design - 8 pts \_\_\_\_\_

Does not create an abstract Ship class - 6 pts \_\_\_\_\_

Inherited Classes

Does not create CargoShip,CruiseShip, and WarShip subclasses - 6 pts \_\_\_\_\_

Does not use super within the constructors as appropriate - 4 pts \_\_\_\_\_

Enumerated Type – does not use enum ShipType for WarShip categories - 5 pts \_\_\_\_\_

ShipRegistry Class

Does not contain methods addShip, getShips, getShipDescriptions,

getWarshipDescriptions, readFile, writeFile and toString. - 7 pts \_\_\_\_\_

Possible decrements: -60 pts \_\_\_\_\_

Possible total grade: 100 pts \_\_\_\_\_