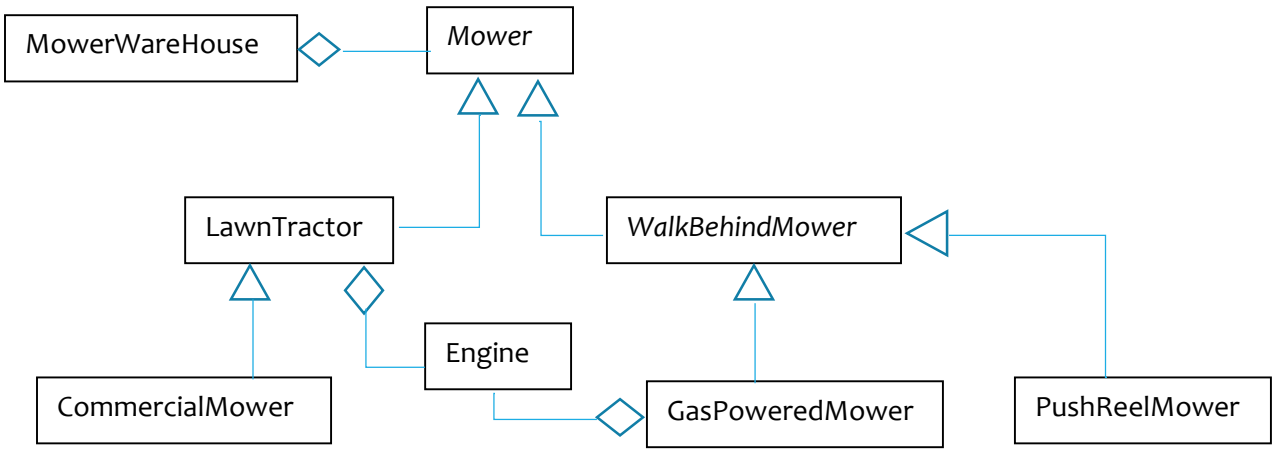
GEEN165 - Major Programming Assignment 2 – Fall 2015

1 Introduction

This assignment requires you to create a Java application that will be used to inventory mowers in a mower store. In addition to the classes that store the mower data, you will create a GUI to manage the inventory.

2 UML Domain/Structure Diagram



3 Classes

You are required to implement the following classes at a minimum. You may add other classes (and methods) if you need them.

3.1 Engine Class

Engine	
-manufacturer : String -horsePower : double -cylinders : integer	Engine manufacturer Horse power of engine Number of cylinders
+Engine() +Engine(//all Engine properties) +//Accessor and Mutators +toString():String	 Put each property on a separate line.

3.2 Mower Class (Abstract)

Mower	
-manufacturer : String -year : integer -serialNumber : String	Mower manufacturer Year of manufacture Serial number of mower
+Mower() +Mower(//All properties) +//Getters and setters +toString()	 Put each property on a separate line.

3.3 LawnTractor Class

This class will define each grade level that can be obtained in the course (D to A).

LawnTractor	
-engine : Engine -model : String -deckWidth : double	Mower engine Model of Lawn Tractor Width of mower deck
+LawnTractor() +//Accessor and Mutators +toString():String	 Put each property on a separate line.

3.4 CommercialMower Class

This class will track the points received and the maximum points for each class assignment.

CommercialMower	
-operatingHours : double -zeroTurnRadius : boolean	Points earned on the assignment Maximum points (e.g. Most labs have maxPoints equal to 20)
+Commercial() +//Accessor and Mutators +toString():String	 Put each property on a separate line.

3.5 WalkBehindMower Class (Abstract)

WalkBehindMower	
-cutWidth : double -wheelDiameter : double	Blade width of mower Diameter of the mower wheels
+WalkBehind() +//Accessor and Mutators +toString():String	 Put each property on a separate line.

3.6 GasPoweredMower

GasPoweredMower	
-engine : Engine -selfPropelled : boolean	Mower engine Is the mower self-propelled
GasPowered() +//Accessor and Mutators +toString():String	 Put each property on a separate line.

3.7 PushReelMower

PushReelMower	
-numWheels : integer	Number of wheels on the mower
+PushReelMower() +//Accessor and Mutators +toString():String	 Put each property on a separate line.

3.8 MowerWareHouse

MowerWareHouse	
-storeName : String -mowers : ArrayList<Mower>	Name of the mower store
+MowerWareHouse() +//Accessor and Mutators +readMowerData(inputFileName : String) : void +saveMowerData(outputFileName :String):void +toString():String	 Put each property on a separate line.

Handling ArrayLists

Each ArrayList should have five associated methods to perform: getNum, add, remove, get and set. So if you have an ArrayList named *widgets* that stored items of type Widget, then the associated UML behaviors would be:

+getNumWidget**s**() : int //Return the number of items in the ArrayList widgets.
+getWidget(index:int) : Widget //get the Widget at location index in ArrayList widgets
+setWidget(index:int, item:Widget):void //store item at location index in the ArrayList widgets.
+addWidget(item:Widget):void //Append the Widget to the ArrayList.
+removeWidget(index:int) : Widget //remove and return the Widget at location index

4 Input File

The name of the input file will be supplied using command-line arguments. If no command-line argument is supplied, then your program should prompt the user for the input file using the JFileChooser class. Here is the format of the input file:

Store name

Mower Class Properties

Mower Subclass Type (L, C, G, P)

Subclass Properties

Note: Each properties will be on a separate line in the same order listed in the UML diagrams.

Figure 1: Input File Format

5 Output File

The format for the output file should be identical to that of the input file. In other words, after writing your output file, you should be able to read it back in as an input file. The toString() methods of your classes are designed to make file output simple.

6 Graphical User Interface

If you would like to add a GUI to your application, look for the GUI addendum. It will be posted later in the same folder as this assignment. You should not attempt this portion of the assignment if you have not completed the other classes.

7 Grading

If your project does not compile, it receives a grade of zero. If you do not document your program according to the documentation guidelines, the graders have been instructed to deduct **up to 25%.**

Level 1 (40%): Implement all the classes except the MowerWareHouse class.

Level 2 (60%): Implement the MowerWareHouse class except the readMowerData() and saveMowerData() methods. Use the main method to create one of each type of concrete class and

add them to the ArrayList. Output the contents of the MowerWareHouse object to a JOptionPane using the toString() method of the MowerWareHouse class.

Level 3 (85%): Modify your main so that it uses command-line arguments to provide the inputfile name. Add the logic to obtain the input file name from a JFilechooser if no command-line argument is provided. Implement the readMowerData and saveMowerData methods. To prove your code works, read the input file and add additional mowers by creating objects in the main method and adding them to the MowerWareHouse object. Save the updated MowerWareHouse object using the saveMowerData() method.

Level 4 (100%): Implement the GUI. See the GUI addendum for details

Extra Credit: Talk to me after you finish Level 4 for extra credit ideas.

8 Submission

Create a file named Readme in the root directory of your project that describes what grade level you fill you completed. If there is any other information the grader should know about your assignment, you should include it in this file. Archive your entire NetBeans project with a zip tool and uploaded it using the assignment upload link.