Manufacturing Part Assembly Manager

# Introduction

Your boss yells from the other room, “Hey! Remember that Dungeons and Dragons app you were working on?”

You hesitantly walk into her office, shaking your head side to side, and she asks, “We took on a new contract… do you think you could turn that code into a manufacturing app in the next week?”

# Goal

As an object-oriented programmer, your goal is to write code that is reusable and flexible. Over time, as you model more objects, you will become comfortable with determining their structure and begin to see patterns in their implementation and interaction with other objects.

The goal of this assignment is to demonstrate the similarities between the Encounters and Assemblies, as well as, Monster and Parts, by quickly refactoring your code. Also, this will also give you an opportunity to fine tune your driver from the Dungeons and Dragons app and repurpose it for this application. With this project, you will also be required to follow the coding standards guidelines for documentation and naming conventions. You will not be required to create a UML document for this assignment, but you must follow the UML class diagram specification to receive full credit.

# Description

You have been tasked with building an application that will allow the job scheduling manager to build assemblies containing an assembly name that will be composed of any number of parts. The assemblies will be passed on to the production manager to be added to the production schedule.

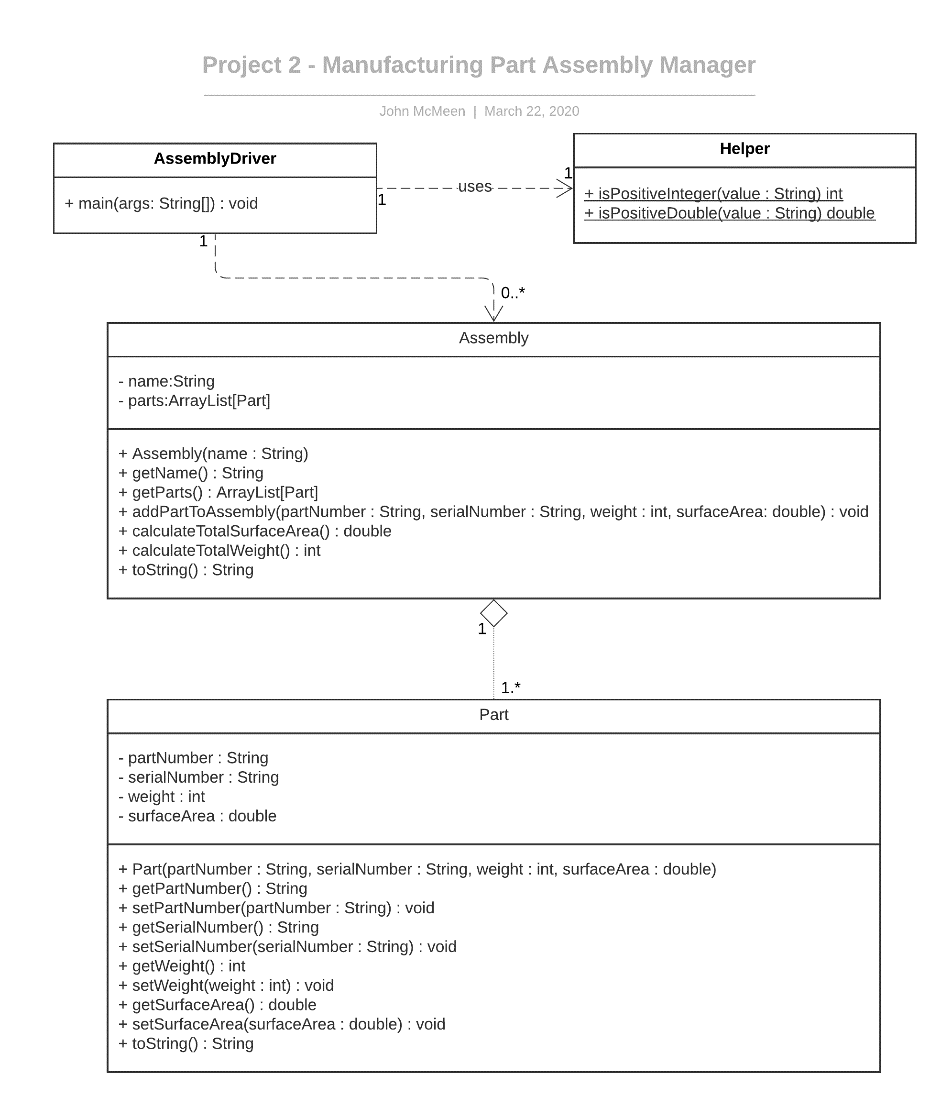
The job scheduling manager application will first prompt the user to add an assembly number (it can be any combination of characters and numbers including dashes), then it will ask for parts to be added to that assembly. The user should be able to add any number of assemblies.

Each part will consist of a part number, serial number, weight, and surface area. The part number will be used for inventory, the serial number will be used to track a single part, the weight will be used for shipping cost, and surface area will be used to calculate the amount of paint required for top-coating.

When all assemblies have been entered, the application will generate a simple report showing all assemblies with all parts for each assembly. The report will also show totals for each assembly: Total weight and total surface area per assembly.

# Design

Following is the UML that was generated by the design team. The implementation should not vary from the design unless specifically approved by the project manager. You will not need to reproduce the UML diagram for this project. Focus on perfecting the implementation outline below.



## Design Notes

* You may reuse code from your Project 1, but please remove all references to Project 1, Encounter, Monster, etc.
* The design for the Helper class has changed. An updated version of the Helper class is available on class Github; you must use this version. Please note, the updated method returns either a -1 if an invalid number is passed in or the validated positive number (int or double depending on which Helper method is being used). Update your driver/input logic as necessary.
* The ArrayList attribute in Assembly (parts) should be of type Part.
* The ArrayList reference used in AssemblyDriver should be of type Assembly.
* The toString method must use the StringBuilder class.
* The AssemblyDriver class will contain only the main() method; no attributes or other methods.
* The AssemblyDriver should handle all input errors for any numeric values (all integers and doubles). See below for example input and output for the AssemblyDriver:
* The output should match mine as close as possible.
* Please follow the commenting guidelines provided in the course resources.

# Input Loop Example Ouput

The following is example output for the application. The output should match mine as close as possible.

**==================== Example 1 ==========================**

Enter the assembly number you would like to build (D if done): PACCAR-12345

Add parts:

Part number: 123

Serial number: 432

Part weight: 100

Part surface area: 1200.25

Add another part to this assembly? (Y or N): Y

Part number: ABC-9

Serial number: 123456789

Part weight: 200

Part surface area: 2000.75

Add another part to this assembly? (Y or N): N

Enter the assembly number you would like to build (D if done): VOLVO-999

Add parts:

Part number: P-A22

Serial number: 987654321

Part weight: 25

Part surface area: 199.5

Add another part to this assembly? (Y or N): N

Enter the assembly number you would like to build (D if done): D

Assembly Report

============================

Assembly Name: PACCAR-12345

Total Weight: 300

Total Surface Area: 3201.0

Part [partNumber=123, serialNumber=432, weight=100, surfaceArea=1200.25]

Part [partNumber=ABC-9, serialNumber=123456789, weight=200, surfaceArea=2000.75]

Assembly Name: VOLVO-999

Total Weight: 25

Total Surface Area: 199.5

Part [partNumber=P-A22, serialNumber=987654321, weight=25, surfaceArea=199.5]

**==================== Example 2 ==========================**

Enter the assembly number you would like to build (D if done): VOLVE HOOD 1121-111

Add parts:

Part number: ZX-999

Serial number: 987AAA-12

Part weight: ASDF

Invalid weight, must be a positive integer. Try again.

Part weight: 123

Part surface area: FFFF

Invalid surface area, must be a positive floating point value. Try again.

Part surface area: 1000.5

Add another part to this assembly? (Y or N): N

Enter the assembly number you would like to build (D if done): D

Assembly Report

============================

Assembly Name: VOLVE HOOD 1121-111

Total Weight: 123

Total Surface Area: 1000.5

Part [partNumber=ZX-999, serialNumber=987AAA-12, weight=123, surfaceArea=1000.5]

# Coding Standards Examples

Class Header:

/\*\*

\* -------------------------------------------------

\* File name: Player.java

\* Project name: BlackJack Game

\* -------------------------------------------------

\* Creator's name: Joe Programmer

\* Email: jprogrammer@northeaststate.edu

\* Course and section: CISP 1020 A01

\* Creation date: Jan 01, 2016

\* -------------------------------------------------

\*/

Class Header example:

/\*\*

\* <b>

\* Purpose: Player class controls all aspects of the player object:

\* - Default constructor and parameterized constructor

\* - Getter methods: getHand, getName, getIsDealer

\* - Setter methods: none

\* - Public methods: updateBank, addACard, toString

\* </b>

\* <hr>

\* Date created: Jan 01, 2016

\* <hr>

\* @author Joe Programmer

\*/

Method Header example:

/\*\*

\* Method description: Calculates the hand value replacing ace’s with

\* one’s or 11’s depending on what gives the hand the best value.

\* Date: Jan 01, 2016

\* Parameters: ArrayList<Hand> hand

\* Return type: int – The hand value, 21 for example

\*/

PascalCase:

FileNames, Classes, Interfaces, Enumerations

camelCase:

attributeName, variableNames, parameterNames, argumentNames

# Deliverables

You will turn in all java files, zipped up into a single .zip file (zip file format only!), and dropped into the D2L drop box (assignment box) before the due date. No late work accepted. Please follow the additional guidelines listed below.

Project Submission Checklist

* Make sure you have included all project files requested.
* Make sure you have followed all instructions in this project specification and your program performs requested functionality, matches UML class diagram design, and follows the design notes provided.
* Make sure you have followed the coding standards examples for commenting and variable naming.
* Please remove all unnecessary comments, commented code, questions to the instructor, and general ramblings. Questions are for office hours.