# BTP400: Assignment 1

**March 15th, 2021**

## Overview

The requirement that must be fulfilled by this assignment is that a system which can model an assembly line must be built in order for a car to go through a set of stations. Each station has its own component which relates to the assembly process and once a car has passed all stations, it will be built.

## Documentation

### UML Diagram

- **Task** is a Parent class that holds all other tasks (i.e Axles, Brakes, Gearbox, Transmission, Doors, CollisionSensors and Paint). They have a generalisation relationship. Task has member variables: taskType and TaskName and a boolean to check if the tasks were completed or not. It also has getter methods to make it possible for these private properties to be accessed by other classes.
- **Job** has one to many relationships with the Task class. One job is composed of many Tasks while one task comes from one jobID. Job has three boolean member variables that check if a job is valid and is optional or not. Then checks if the particular job with the given JobID is complete. It also has an ArrayList of tasks and a *splitStr[]* to read from a file all the tasks.In addition to these it has seven methods, one to validate task, getters to get task and jobID and get the answer if job is valid or not and a constructor.

- Job has one to one relationship with all station:
    - **InitialStation**: Job has one on one relationship with Job. every job has to pass to an initial station one and one Job is done on one InitialStation. Until it's

    done, the InitialStation will pass the Job to the IntermediateStation. InitialStation has 5 member variables that. Two booleans, one to check if the task is optional and the other one to check if the station is offline. Has a string variable file data that will write to file read. The Jobcount is to see how many valid jobs were done. It also has three methods. One getter to get the Jobs and a constructor and one to initialize inventory. If inventory is not enough, tasks will fail.
    - **IntermediateStation** also has a one on one relationship with Job. Once the job is passed to it by the InitialStation, it will complete all the intermediate tasks and then pass the job to the finalStation. Has three boolean member variables. One to check if the task is optional, the other to check if the station is offline and last one to check if the job is done. It also has an array of type Task. This class has 5 methods. A constructor, two getters. One to get the station, the other to get the task type. One setter to set the state and doTask() to show what tasks are being done.
    - **FinalStation** just like the other stations has a one on one relationship with Job. one Job will pass once at the final station and that marks the completion of our Car object. It has 2 member variables vin that will randomly give completed car's vinNumber and a boolean to check if the station is online or not. It has 4 methods. One getter to get the state and one setter to set the state if it is complete or not. randomVin() method to randomly give vin number to car and a constructor.
- **Car** class has one on one relationship with Final station but has one to many relationship with Material class and Task class. One car will pass to the FinalStation once but one car has many materials(bolts, screws, doors etc.. These tasks have 4 member variables, taskName to store the task name, boolean to check if the task is optional, taskDuration to see the time and int number to see the numbers of tasks) but one material can be installed in one car only. Also one car was completed by many Tasks, but one task can be done on one car only. Car has one string member variable, the Vin number. It also has two methods. constructor and a getter to get the Vin Number.
- **Material** has one to many relationship with Tasks. One task uses a lot of material but one material will be used to complete one task. Material has two member

variables. A String to store the name of the material, and an integer to store quantity. It has three methods, two getters, one to get the name on the material, the other one to get the quantity of material. One setter to set the quantity.

- **InventoryManager** has one to many relationships with Material. An inventory manager manages many materials but one material will be managed by one inventory manager. It has one member variable that is an ArrayList of Materials. This class also has 4 methods. One method to update the inventory, the second to get inventory, the third to get the count of inventory( total count of each inventory) and the last one to get the index of each inventory for easy distribution when passing the appropriate materials to complete Tasks in the Stations.

## JavaFX

- *JavaFX* is a software platform for creating and delivering desktop applications by using a standard GUI library that can run across a wide variety of devices.
- An additional add on that works alongside *JavaFX* was used in order to create and design the interface, called scene builder.
- The first view is the login.
    - It incorporates images, labels, textboxes and buttons in order to simulate a login page that will only login with a specific set of credentials (username and password.
    - In order to login, the username is “Car” and password is “123”.
- Once a user has logged in, they will be redirected to the main screen.
    - This page only consists of 2 elements, a label and multiple buttons.
        - HBox’ and VBox’ have been used to ensure specific dimensions for where the buttons can sit on the interface.
        - At the top, cars, inventory, and buttons have been used, once clicked they will display the information in the white box below.
- Once a user clicks the *Logout* button on the right hand side, the program will terminate and the UI will be closed.
    - For the sake of user usability, we decided to not have a pop up to ensure a smooth transition to logging out of the system.

- Once a user clicks the *Jobs* button on the left hand side, they will be taken to another view
    - In the other view, there will be 2 more buttons.
        - *Back* Button
            - This button will take the user back to the main view.
        - *Jobs* Button
            - Once a user clicks this button, the jobs will be done and outputted in the white box below.
            - The program will automatically provide vertical and horizontal scroll bars to be able to view the information.
- Once a user clicks the *Inventory Count* button, they will be taken to another view
    - In this view, there will be buttons and text boxes
        - *Back* Button
            - This button will take the user back to the main view.
        - *Check* Button
            - Once a user types a material name in its specified textbox and clicks check it will find the number of bolts, screws, doors, etc..
            - If a user types in an incorrect name then the system will output a message indicating the user to type a valid name in the text box below.
            - If a user types a correct name then the system will output a message indicating the user the name and quantity of the material.
        - *Clear* Button
            - Once a user clicks the clear button than the 2nd text box will be cleared and blank
- Once the user clicks the *Inventory* button, the white box below will get filled in with the information about all the materials that are in stock. It will provide a simpler method of looking at all the materials in comparison to the *Inventory Count* button.
- Once the user clicks the *Cars* button, it will take them to another view which has 2 buttons.
    - *Back* Button
        - This button will take the user back to the main view.

    - *Car Button*
        - This button will display the completed tasks and the Vin number of the produced car(s).
- Once the user clicks the *Clear* button, the white form on the right hand side will be cleared of any text.
- Once the user clicks the *Authors* button, our names will appear in the white form on the right hand side.

## JUNIT Test Cases

- testCar.java - Tests the object creation for the Car object, tests assertNotNull
- testTask.java -  Tests the object creation for the Task object, tests assertNotNull
- testJob - Tests the object creation for the Job object, tests ValidateTask() functions
- testInitialStation - Tests the object creation for the InitialStation object, tests assertNotNull
- testMaterial - Tests the object creation for the Material object, tests assertNotNull
- testInvManagerCreate - Tests the object creation for the Car object, tests assertNotNull. Tests addition and subtraction of inventory, assertTrue and assertFalse respectively.
- testMidStations - Tests the object creation for the IntermediateStation object, tests assertNotNull
- testFinalStation - Tests the object creation for the FinalStation object, tests assertNotNull

## Main Program

### Car.java

- This is the class which declares the VIN number for the car as a string.
- It also has the setters and getters for the VIN.

### FinalStation.java

- This is the class for the final station which “engraves” or appends the VIN number to the car.

- It declares the array list which is the VIN number. It will be added into the string each time.
- The constructor has been set up to be able to write to a specified text file called "CarLog.txt". It contains the engraved VIN of the car. Once this is done it will close the file.
- The *public String random*VIN() is used to generate random VIN numbers for the car and then randomly select one to be used to engrave the VIN once the car assembly is completed.

**InitalStation.java**

- This is the class for the initial station which gets the chassis for the car.
- This is where we create the job object using the job id and tasks.
- Once the initialization and validation of both the job and tasks are completed, for each specific task the materials can be reserved.
- The inventory is initialized and deductions are applied to the materials on tasks that have valid jobs.
- *UpdateInventory*() receives the index of the material that is going to be used and then updates it.
    - For example, if there are 200 doors and we use 4 doors for one car then it will subtract 4 from 200 and update the inventory to be 196.
    - It will also take in a boolean which will indicate whether the update is addition or subtraction (FALSE = subtraction & TRUE = addition).

**IntermediateStation.java**

- This is the class for the Intermediate station which installs the brakes.
- *Public void doTask*() performs the task for the station and then logs it into the output file "JobLog.txt".

**InventoryManager.java**

- This is the class where the inventory is stored and manipulated based on the materials being used to add or remove.

- Each material object will be added to the ArrayList with their specified inventory number.
- *Public boolean UpdateInventory* will receive the index of the object for the material and then update the number which will be used to either add or remove.
    - The boolean on the other hand will be used to specify the type of operation (add or remove).
    - If *isAdding* is true then it will add to the current quantity from the *n*.
    - If *false* then subtract from the current quantity using the *n*.
- *Public String getInvCountOf* will be used to return the name and the inventory count of the material object based on the query being used.
- *Public int getMatIndexOf* will be used to return the index of the material in the list based on the query being used.

**Job.java**

- *Public Job* takes a *String[]* using the job id and task name
    - It will get the job ID from the start of the *String[]* at index 0 and the substring will return the number which is at the end.
    - A valid list of tasks will have either 6 or 7 tasks depending on the optional task (if we're following the given use case)
- *Public boolean ValidateTask* is used to receive a String which is the name of a task and then the string received will be tested using the *equals*() in order to determine if it's a mandatory task or an optional one.
    - We've planned for contingencies, if the name does not match then there will be a invalid task and will result in an invalid job.

**Main.java**

- This is the main file of the project and it actually launches off the app.
- This is where it will go through all the jobs and ensure they're valid. If they are then the system will shut down.
- The inventory is also stored in the "InvLog.txt" file because the *public static void writeInvLog(InventoryManager inv)* takes in the InventoryManager as an argument.

- The “InvLog.txt” contains the entire inventory count of all the materials used, the quantity of the materials and date it was logged into the system.
- *Public static void startSystem* is where the car assembly begins and starts off at the intermediate stations. For each job, all the tasks are completed and steps in order to complete the tasks are logged (name, status and date).

**Material.java**

- This class has been created for the materials that are going to be used in the tasks.
- It has the methods of getting the name of the materials, the quantity of materials, updating the material (setting quantity).

**Server.java**

- This is the server class which is being used to establish a socket connection.
  - It will be used to move from station to station.

**Task.java**

- This class has been created for the tasks.
  - It contains methods on getting the name of the task, getting the type of task, understanding if the task is completed or not.

**CarLog.txt**

- This file is for holding the information about the car if the job was completed for that specific car.
- The car text file also holds the VIN of the associated car and a list of tasks done on the car for assembly.

**InvLog.txt**

- This file contains the entire inventory count of all the materials used, the quantity of the materials and date it was logged into the system.
  - The materials consist of the following
    - Bolt

        - Screws
        - Axels
        - Doors
        - Brakes
        - WindShield
        - Gearbox
        - Sensors
        - Paint
        - Transmissions
    - The quantity of materials was created with the most realistic and educated assumptions possible for a total of 50 cars.

**JobLog.txt**

- This file contains the job id, the tasks which have been completed (ie. Brakes, Axels, Doors), the date of completion and the time which they were completed.

**jobs.txt**

- This file holds all the information about the jobs which are used by the initial station to validate the jobs and initialize the inventory.
    - The text file holds strings for job id and the tasks associated with the job in the following format (delimiter: ","): [Job ID], [Task 1], [Task 2], [Task 3], [Task 4], .... etc