Introduction

This controller builds off code written by Kevin Harrington for the RBE200x classes. The java files that are used by the application are:

- InventoryManager
- ItemCheckOutScreenController
- ItemSelectScreenController
- ListViewPart
- ListViewPartCell
- Main
- RobotActionScreenController
- RobotInterface
- RobotManagerScreenController
- WelcomeScreenController
- IWarehouseRobot
- WarehouseRobot
- WarehouseRobotStatus

The fxml files that the application uses are:

- ItemCheckOutScreen
- ItemSelectScreen
- partListCell
- RobotActionScreen
- RobotManagerScreen
- WelcomeScreen

All other files are either left over from the previous application and not used or were not modified in creating the Warehouse UI.

Overview Of Code

A brief overview of all files is given here to help aid in future development

Main

This is the main executable that the app starts from. The main class extends JavaFX Application so please refer to JavaFX documentation for more information of the inner workings of the class. The main class has a static variable for each Scene used in the app. There is one Scene per fxml file. The variable *theStage* is the application window object that displays the Scenes. There are also a few FXMLLoader objects. These are used when Main needs to access a function of a Scene controller. There are other static functions that allow Scene controllers to set *theStage* to a different scene which is how the application changes windows. Some Scenes also require items in the scene to be modified so there are other functions for this purpose that utilize the FXMLLoader objects, so that methods of the associated controller can be called from Main.

There is also data related to parts that needs to be shared between scenes.
partList -> Observable list that stores all ListViewPart objects that represent parts in the system's inventory
currentIDNum-> Stores the user's student ID number
currentPart-> stores the part selected by the user
numberRequested-> stores the number of parts the user wishes to borrow

InventoryManager

This class manages two JSON files, one with the current inventory and another with the borrowed parts list. On app start up, this class will check to see if the WareHouseRobot directory exists in the documents folder of the user and if not create it along with 2 empty JSON files. These files cannot be parsed so future work should add some JSON text to fix this but for now just copy the 2 JSON files in resources/json to use. There are separate load and write methods for each JSON file. The Inventory file contains a part list with fields:

- Name
- numberAvailable
- Row
- Col
- Height
- returnRequired

These fields are used to populate the ListViewPart Objects
The borrowed Inventory JSON file has a list of objects that contain fields:

- ID (student ID)
- Name (of part)
- numberBorrowed
- Class (part is needed for)

When the user borrows a part in the ItemCheckOutScreen, updateInventory() in Main is called. This calls addBorrowedParts() and removeItemsFromInventory() in the InventoryManager. addBorrowedParts() adds the part to the borrowed parts JSON file and removeItemsFromInventory() subtracts the number of parts borrowed from the associated part in the Inventory JSON file.

WelcomeScreenController

The welcome screen is the first Scene that *theStage* is set too. The associated FXML file is WelcomeScreen.fxml. There is a text entry and 2 buttons. Users enter their Student ID into the text entry and press the Enter Button. When the Enter button is pressed, enterCallback() is called. This verifies that the user ID is a 9 digit number and if so changes the active scene to the ItemSelectScene. If not then it makes an error label visible to alert the user they made a mistake. There is also a Maintenance button that when pressed brings the user to the RobotManagerScreen.

ItemSelectScreenController

This controller displays the list of inventory the user can select. The associated FXML file is ItemSelectScreen.fxml. The list is populated with partListCells using a factory. Change the **partListCell.fxml, ListViewPartCell** and **ListViewPart** files to modify what is shown in the table and how. When a list item is clicked on in the table, the ListViewPart object associated with it is saved to the currentPart of the main class to be used later. There is also a Finished button that changes the screen back to the Welcome Screen

ItemCheckOutScreenController

This Screen allows the user to specify the amount of a part they want and note what class it is for. When Main sets *theStage* to this Scene, it calls setScreenInfo() to populate all the relevant information about a part. If more data is added in the future, there is where the UI would be updated. When the Confirm button is pressed, the number of parts desired is saved to numberRequested in Main, the inventory file is updated and the Robot is commanded to retrieve this part. The Scene is then set to RobotActionScene phase 0. Read about the Robot Action Scene for more information on this. The user can also cancel this request with the cancel button and be brought back to the itemSelectScreen.

RobotActionScreenController

This screen contains all of the messages that are displayed when the robot is conducting a task or there is an error with the robot. The associated FXML file is RobotActionScreen.fxml.

This screen can show 1 of 5 screens.

1. setRetrieve() is shown when the robot is retrieving a part from the shelf
2. setWaitForDone is shown after the robot delivers a part and it displays the number of parts the user is to take and reveals the “Done” button. When the Done button is pressed, the robot is sent a message to return the bin
3. setPutBack() is shown while the robot is returning a bin.
4. setNoBinOnShelf() is shown when the robot detects no bin on shelf
5. setFailedProcurement() is shown when the robot is unable to lift a bin.

All of these functions are called from Main as part of setRobotActionScene and the input “phase” determines which screen is shown. setRobotActionScene is called by ItemcheckOutScreenController and RobotInterface.
There is another function called setErrorWarningVisible(). This gets called from Main from the function setRobotActionSceneWarning(). RobotInterface calls this function in Main when the robot detects that the bin was not placed on the cleat when the user pressed “Done” to ensure the robot does not try to return a bin it is not carrying.

RobotManagerScreenController

This controller is used and referred to as the Maintenance screen. The associated FXML file is RobotManagerScreen.fxml. This screen offers the ability to attempt to connect with a robot on the network and send commands to the robot.
The Nav, park, deliver and return commands are accessible from this screen. There are some text entry boxes that are for specifying a row, column and shelf height. When pressing a command button, all appropriate fields must have a valid integer value in order to be sent. The park and nav commands only require a row and column while deliver and return additionally require the shelf height. There are also buttons for homing the lift and setting the lift to a height. The lift height unit is mm and can be set to any decimal value. Pressing the Exit button brings the user back to the maintenance screen.

RobotInterface

The robot interface is the object used to communicate with the robot. It is created in the Main class and referenced by other Scene Controllers.

connectToDevice() is what is called to attempt connecting to the robot. The String name inside this function is where the robot name is set. Currently it is set to “*” so that all robots on the network respond. Upon connecting, the robot name in the RobotManagerScreen gets set, so if the name is populated in that screen then connection was successful. This method is called on startup and if it fails the application automatically changes to the RobotManagerScreen.

setFieldSim() is called upon connecting to the robot. It creates the IWarehouseRobot object and sets up a software interrupt that enables the software to receive the robot’s status messages. The software interrupt callback is contained within the setFieldSim() function. It

reads the status, changes a label in the RobotManagerScreen to display the current status and performs some actions for some status'.

- Returning: When this status is received, the setRobotActionScene() method is called in Main and the phase is set to 2.
- Delivery_Done: When this status is received, the setRobotActionScene() method is called in Main and the phase is set to 1 so that the waitForDone screen is shown.
- Bin_Not_On_Cleat: When this status is received then the setRobotActionSceneWarning method is called in Main.
- Bin_Not_On_Shelf: When this status is received, the setRobotActionScene() method is called in Main and the phase is set to 3 so that an error message is shown.
- Delivery_Failure: When this status is received, the setRobotActionScene() method is called in Main and the phase is set to 4 so that an error message is shown.

This class also contains methods for sending commands to the robot.
There are additional methods that set Booleans for *DeliverIsTest* and *ReturnIsTest.* When the RobotManagerScreen sends a deliver or return command, it calls these functions so that the appropriate variable is set to True. This is used in the status message callback so that when the robot reports it is done those tasks, the setRobotActionScene method is not called in Main. Those Booleans are set to false during the normal procurement process.

IWarehouseRobot

This interface contains the direct UDP packet transmission code. All robot commands are PacketType objects. Each PacketType must have a unique ID. If adding a new PacketType object, make sure to call its method waitToSendMode() in the addWarehouseRobot method of this class and add it into the for loop in the same method. This is essential for properly setting up the object and so that the packet gets sent only once per function call. To send a packet, use the writefloats() method and then call the PacketType object's oneShotMode() method, as shown in this class for the other commands. Many of the methods in this class are not used as they are left over from copying IRBE2001Robot.

WarehouseRobot

This class implements IWarehouseRobot and is created when the RobotInterface connects with a robot. It serves no other function and only has a toStringMethod to print the robot name.

WarehouseRobotStatus

This class contains all the warehouse robot status enumerations. If adding a new status, you may need to add it to the switch case of the RobotInterface that handles incoming robot status messages.