Warehouse Management System

CSCI 5448 Project: Part 6

Team: Max Hollingsworth
Jonathan Magiera
Dan Prendergast

Title: Warehouse Management System

1. Implemented/Not-Implemented Requirements

Because some of our requirements are written in a hierarchical fashion, we have combined the "Implemented" and "Not-Implemented" sections of this report to improve clarity. In the table below, any requirement highlighted **grey** was not implemented in the final state of the project.

ID	Requirement	Topic Area	Supports Use Case(s)...	Actor
U-01	The system shall depict the layout of the warehouse floor, including the locations of the loading docks, storage shelves, QA inspection area, and shipping center.	Pallet Management	1, 2, 8, 9	WO, LDS, QA, R, OS, SC
U-02	The system shall show the following information to the Warehouse Operator:	Pallet Management	(n/a)	(n/a)
U-03	- which product is currently on each shelf	Pallet Management	1, 2, 8, 9	WO, LDS, QA, R, OS, SC
U-04	- quantity of the product currently on each shelf	Pallet Management	1, 2, 8, 9	WO, LDS, QA, R, OS, SC
U-05	- the next pallet to be offloaded from the truck	Pallet Management	1, 2	WO, LDS, QA, R
U-06	- the next pallet to be moved from the QA Inspection Area	Pallet Management	1, 2	WO, LDS, QA, R
U-07	- which product is on each pallet	Pallet Management	1, 2	WO, LDS, QA, R
U-08	- quantity of the product on each pallet	Pallet Management	1, 2	WO, LDS, QA, R
U-09	The system shall allow the Warehouse Operator to select a pallet to move.	Pallet Management	1	WO, LDS, QA, R

U-10	The system shall allow the Warehouse Operator to select an empty shelf as the pallet's destination.	Pallet Management	1	WO, LDS, QA, R
U-11	The system shall allow the Warehouse Operator to select the QA Inspection Area as the pallet's destination.	Pallet Management	1	WO, LDS, QA, R
U-12	The system shall send a command to an autonomous forklift to move pallets based on the Warehouse Operator's input.	Pallet Management	1	WO, LDS, QA, R
U-13	The system shall indicate to the Warehouse Operator the following conditions:	Pallet Management	(n/a)	(n/a)
U-14	- when an autonomous forklift is “busy”	Pallet Management	1, 2	WO, LDS, QA, R
U-17	- when an autonomous forklift is malfunctioning	Pallet Management	1, 3	WO, LDS, QA, R
U-18	- when an autonomous forklift is operational	Pallet Management	1, 4	WO, LDS, QA, R
U-19	When a pallet is delivered to a shelf, the system shall update the shelf contents (product and quantity).	Pallet Management	1	WO, LDS, QA, R
U-21	The system shall allow the Warehouse Operator to cancel a "move pallet" task before the autonomous forklift has picked up the pallet.	Pallet Management	2	WO, R
U-22	The system shall allow the Warehouse Operator to take a robot (autonomous forklift or retrieval robot) out of service.	Pallet Management	3	WO, R
U-23	The system shall allow the Warehouse Operator to place a robot (autonomous forklift or retrieval robot) in service.	Pallet Management	4	WO, R
U-24	Upon receipt of an order, the system shall automatically send a command to a retrieval robot to retrieve the product from the shelf and move it to the warehouse shipping center.	Order Fulfillment	8	Online Order System
U-25	The system shall indicate the following conditions:	Order Fulfillment	(n/a)	(n/a)
U-26	- when a retrieval robot is "busy"	Order Fulfillment	8, 9	OS, SC, R, CS
U-29	- when a retrieval robot is malfunctioning	Order Fulfillment	8, 9, 3	OS, SC, R, CS, WO
U-30	- when a retrieval robot is operational	Order Fulfillment	8, 9, 4	OS, SC, R, CS, WO

U-31	When a product is delivered to the Shipping Center, the system shall update the total number of that product shipped.	Order Fulfillment	8	OS, SC, R
U-33	The system shall show the following product information to the Inventory Manager:	Inventory Management	(n/a)	(n/a)
U-34	- product ID	Inventory Management	5, 6, 7	IM
U-35	- product type	Inventory Management	5, 6, 7	IM
U-38	- quantity on hand	Inventory Management	5, 6, 7	IM
U-39	- quantity on order	Inventory Management	5, 6, 7	IM
U-40	- total number sold	Inventory Management	5, 6, 7	IM
U-41	The system shall allow the Inventory Manager to place orders for new product shipments from the suppliers.	Inventory Management	5	IM
U-42	The system shall allow the Inventory Manager to define the following order specifications:	Inventory Management	(n/a)	(n/a)
U-43	- product ID	Inventory Management	5	IM
U-44	- quantity	Inventory Management	5	IM
U-45	- delivery date	Inventory Management	5	IM
U-46	The system shall allow the Inventory Manager to add a new product to the inventory database.	Inventory Management	6	IM
U-47	The system shall allow the Inventory Manager to remove a product from the inventory database.	Inventory Management	7	IM
U-49	The system shall simulate an interface to the Online Order System for test/demonstration purposes.	System Test	8, 13	SC, R
U-50	The system shall simulate loading dock operations (i.e., arrival of delivery trucks and scanning incoming pallets)	System Test	1	WO, LDS, R

	for test/demonstration purposes.			
U-51	The system shall simulate QA inspection operations (i.e., scanning inspected pallets) for test/demonstration purposes.	System Test	1	WO, QA, R

[Warehouse Operator = WO, Inventory Manager = IM, Loading Dock Sup = LDS, QA Inspector = QA, Order System = OS, Robot = R, Shipping Center = SC, Customer = C, Customer Service = CS]

3. Class Diagram

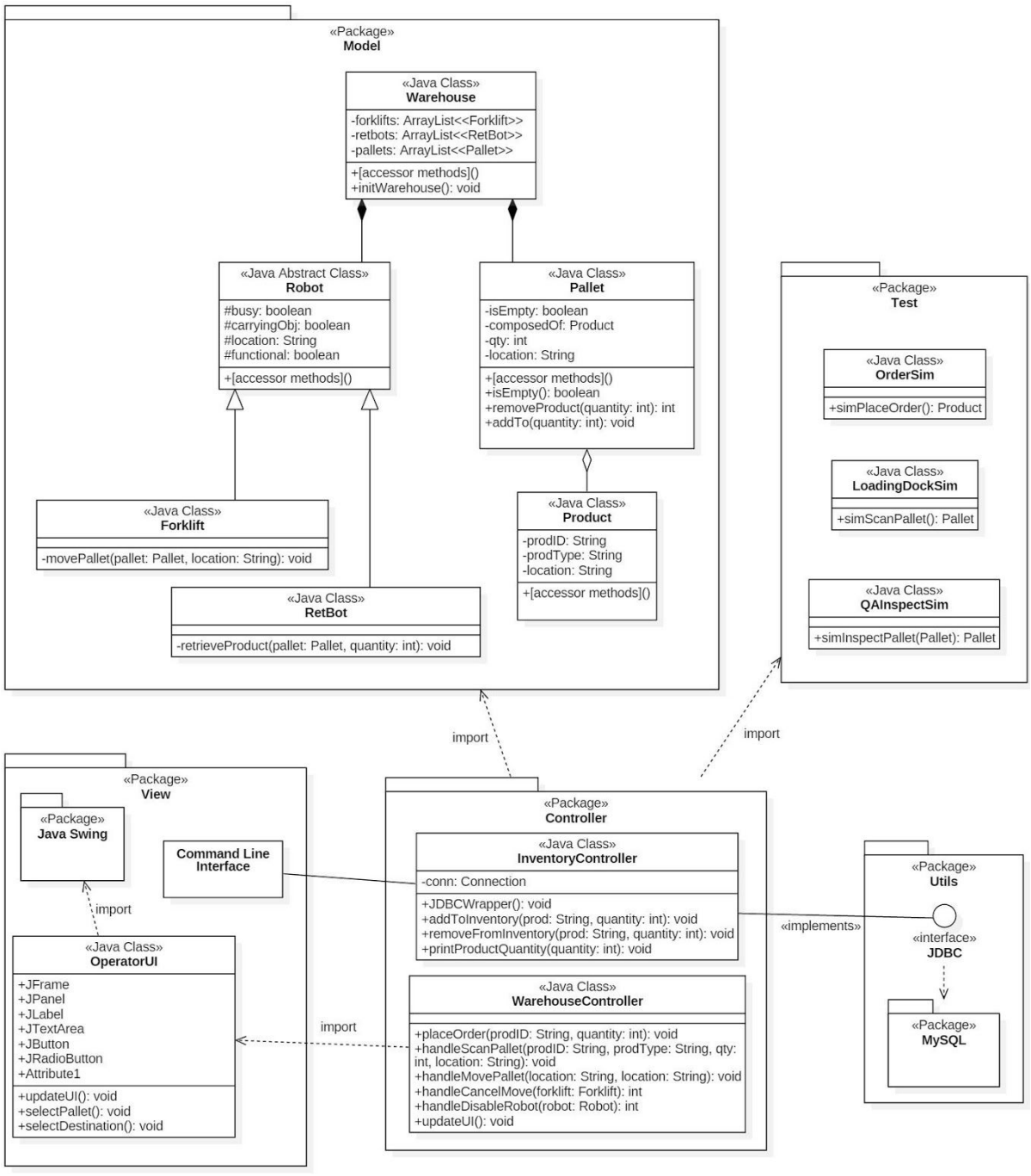
The following two figures illustrate a before and after of our class diagram. The final diagram has blue highlighting to show the new fields. Although it may look like a lot has changed from the part two diagram, all of the original classes are still intact and all of the relations remain more or less the same. The classes are larger in the final diagram, this is mostly because we decided to show all of the get and set methods which were not included in the part two diagram.

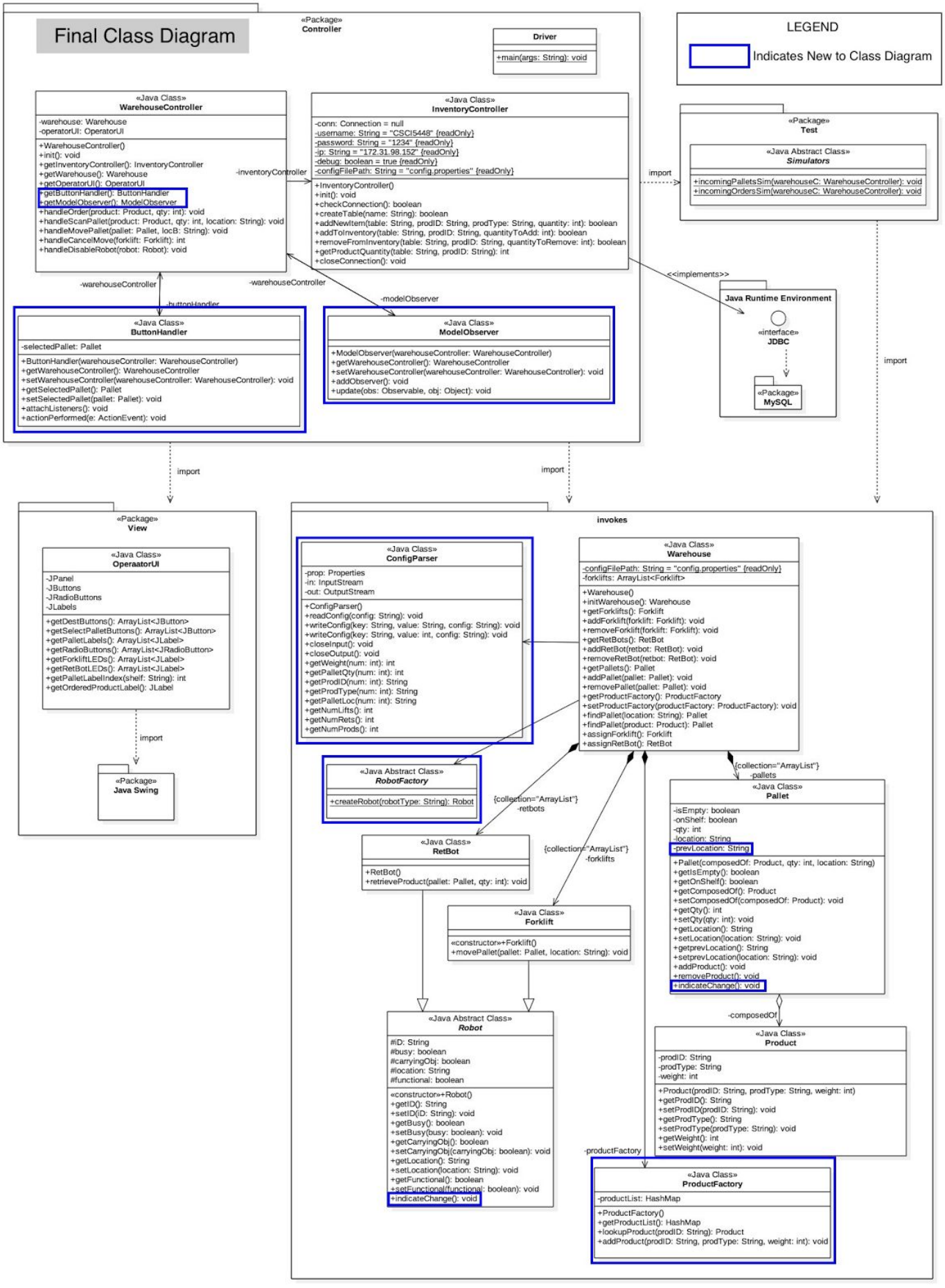
The biggest additions to the final diagram were the observers and the factory classes. These additions were not oversights, but came about because the of the refactoring phase during part 3. We decided to implement the Observer design pattern once we saw how it could update the view whenever changes were made to the model. Adding the Factory design pattern to our model also made sense because having multiples of the same product on the factory shelfs is commonplace.

Another small addition was the configuration parser. We added this class because we needed a smooth way to initialize the warehouse already stocked with some inventory.

One blunder we made in the original diagram was showing that the Warehouse contains instances of the Robot class, when in reality the Warehouse has instances of the Forklift and Retbot classes which in turn inherit from the Robot class.

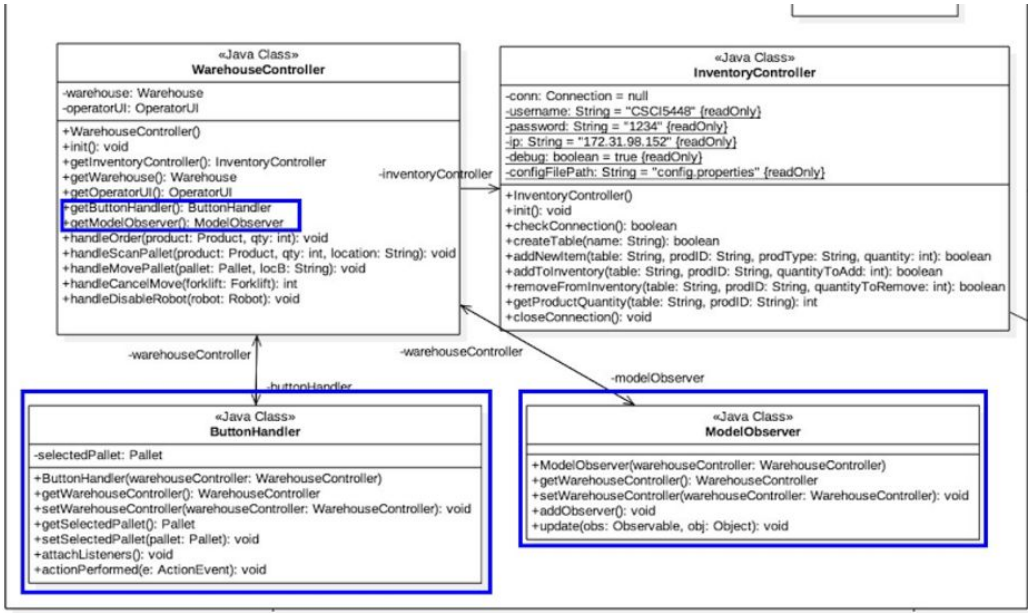
Part 2 Class Diagram



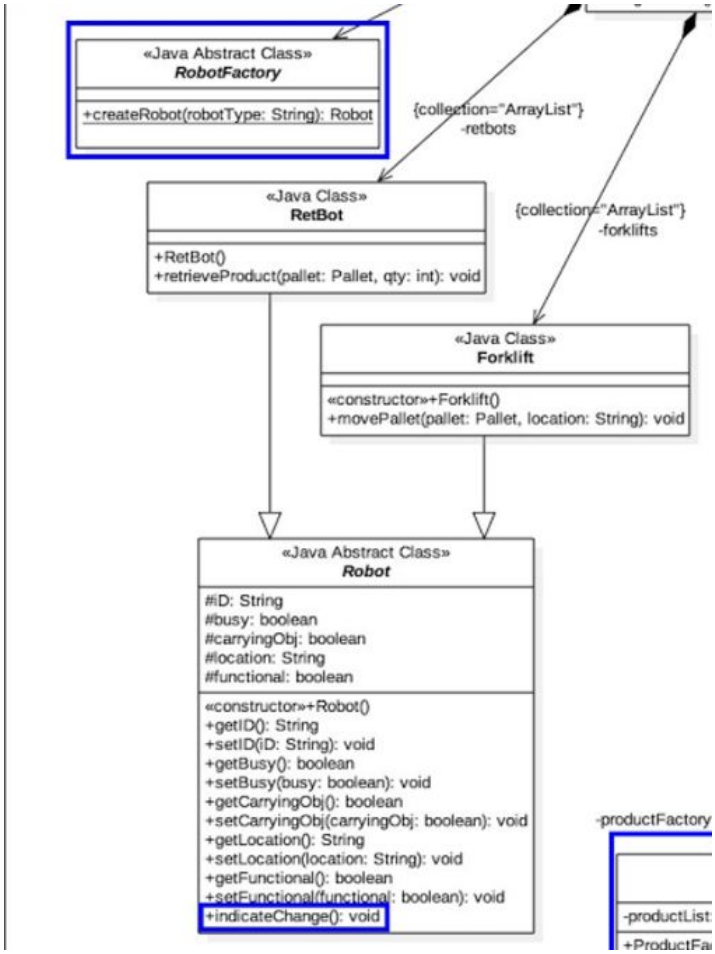

Final Class Diagram
«Package»
Controller
Driver
+main(args: String): void
LEGEND
Indicates New to Class Diagram
«Java Class»
WarehouseController
-warehouse: Warehouse
-operatorUI: OperatorUI
+WarehouseController()
+init(): void
+getInventoryController(): InventoryController
+getWarehouse(): Warehouse
+getOperatorUI(): OperatorUI
+getButtonHandler(): ButtonHandler
+getModelObserver(): ModelObserver
+handleOrder(product: Product, qty: int): void
+handleScanPallet(product: Product, qty: int, location: String): void
+handleMovePallet(pallet: Pallet, locB: String): void
+handleCancelMove(forklift: Forklift): int
+handleDisableRobot(robot: Robot): void
-inventoryController
«Java Class»
InventoryController
-conn: Connection = null
-username: String = "CSCI5448" {readOnly}
-password: String = "1234" {readOnly}
-ip: String = "172.31.98.152" {readOnly}
-debug: boolean = true {readOnly}
-configFilePath: String = "config.properties" {readOnly}
+InventoryController()
+init(): void
+checkConnection(): boolean
+createTable(name: String): boolean
+addNewItem(table: String, prodID: String, prodType: String, quantity: int): boolean
+addToInventory(table: String, prodID: String, quantityToAdd: int): boolean
+removeFromInventory(table: String, prodID: String, quantityToRemove: int): boolean
+getProductQuantity(table: String, prodID: String): int
+closeConnection(): void
«Package»
Test
«Java Abstract Class»
Simulators
+incomingPalletsSim(warehouseC: WarehouseController): void
+incomingOrdersSim(warehouseC: WarehouseController): void
import
-warehouseController
-warehouseController
-buttonHandler
-modelObserver
«Java Class»
ButtonHandler
-selectedPallet: Pallet
+ButtonHandler(warehouseController: WarehouseController)
+getWarehouseController(): WarehouseController
+setWarehouseController(warehouseController: WarehouseController): void
+getSelectedPallet(): Pallet
+setSelectedPallet(pallet: Pallet): void
+attachListeners(): void
+actionPerformed(e: ActionEvent): void
«Java Class»
ModelObserver
+ModelObserver(warehouseController: WarehouseController)
+getWarehouseController(): WarehouseController
+setWarehouseController(warehouseController: WarehouseController): void
+addObserver(): void
+update(obs: Observable, obj: Object): void
<<implements>>
Java Runtime Environment
«interface»
JDBC
«Package»
MySQL
import
import
import
«Package»
View
«Java Class»
OperaatorUI
-JPanel
-JButtons
-JRadioButtons
-JLabels
+getDestButtons(): ArrayList<JButton>
+getSelectPalletButtons(): ArrayList<JButton>
+getPalletLabels(): ArrayList<JLabel>
+getRadioButtons(): ArrayList<JRadioButton>
+getForkliftLEDs(): ArrayList<JLabel>
+getRetBotLEDs(): ArrayList<JLabel>
+getPalletLabelIndex(shelf: String): int
+getOrderedProductLabel(): JLabel
import
«Package»
Java Swing
invokes
«Java Class»
ConfigParser
-prop: Properties
-in: InputStream
-out: OutputStream
+ConfigParser()
+readConfig(config: String): void
+writeConfig(key: String, value: String, config: String): void
+writeConfig(key: String, value: int, config: String): void
+closeInput(): void
+closeOutput(): void
+getWeight(num: int): int
+getPalletQty(num: int): int
+getProdID(num: int): String
+getProdType(num: int): String
+getPalletLoc(num: int): String
+getNumLifts(): int
+getNumRets(): int
+getNumProds(): int
«Java Class»
Warehouse
-configFilePath: String = "config.properties" {readOnly}
-forklifts: ArrayList<Forklift>
+Warehouse()
+initWarehouse(): Warehouse
+getForklifts(): Forklift
+addForklift(forklift: Forklift): void
+removeForklift(forklift: Forklift): void
+getRetBots(): RetBot
+addRetBot(retbot: RetBot): void
+removeRetBot(retbot: RetBot): void
+getPallets(): Pallet
+addPallet(pallet: Pallet): void
+removePallet(pallet: Pallet): void
+getProductFactory(): ProductFactory
+setProductFactory(productFactory: ProductFactory): void
+findPallet(location: String): Pallet
+findPallet(product: Product): Pallet
+assignForklift(): Forklift
+assignRetBot(): RetBot
«Java Abstract Class»
RobotFactory
+createRobot(robotType: String): Robot
{collection="ArrayList"}
-retbots
{collection="ArrayList"}
-forklifts
{collection="ArrayList"}
-pallets
«Java Class»
RetBot
+RetBot()
+retrieveProduct(pallet: Pallet, qty: int): void
«Java Class»
Forklift
«constructor»+Forklift()
+movePallet(pallet: Pallet, location: String): void
«Java Class»
Pallet
-isEmpty: boolean
-onShelf: boolean
-qty: int
-location: String
-prevLocation: String
+Pallet(composedOf: Product, qty: int, location: String)
+getIsEmpty(): boolean
+getOnShelf(): boolean
+getComposedOf(): Product
+setComposedOf(composedOf: Product): void
+getQty(): int
+setQty(qty: int): void
+getLocation(): String
+setLocation(location: String): void
+getprevLocation(): String
+setprevLocation(location: String): void
+addProduct(): void
+removeProduct(): void
+indicateChange(): void
-composedOf
«Java Class»
Product
-prodID: String
-prodType: String
-weight: int
+Product(prodID: String, prodType: String, weight: int)
+getProdID(): String
+setProdID(prodID: String): void
+getProdType(): String
+setProdType(prodType: String): void
+getWeight(): int
+setWeight(weight: int): void
«Java Abstract Class»
Robot
#iD: String
#busy: boolean
#carryingObj: boolean
#location: String
#functional: boolean
«constructor»+Robot()
+getID(): String
+setID(iD: String): void
+getBusy(): boolean
+setBusy(busy: boolean): void
+getCarryingObj(): boolean
+setCarryingObj(carryingObj: boolean): void
+getLocation(): String
+setLocation(location: String): void
+getFunctional(): boolean
+setFunctional(functional: boolean): void
+indicateChange(): void
-productFactory
«Java Class»
ProductFactory
-productList: HashMap
+ProductFactory()
+getProductList(): HashMap
+lookupProduct(prodID: String): Product
+addProduct(prodID: String, prodType: String, weight: int): void

4. Design Patterns

Given that we used the Model-View-Presenter architecture for our project, the observer played an important role to tie everything together. We used the WarehouseController class as the observer between the model and the GUI, or the view. Whenever a user clicks on the loading dock to move items within the warehouse, the action listeners notify the WarehouseController, which in turn sends the information to the backend model. In addition to this, we have an InventoryController observer which changes the MySQL database to reflect the product changes in the warehouse.



Along with the observer pattern, we also used the factory method pattern to instantiate new instances of our forklifts or autonomous robots. Instead of our Warehouse class using the “new” operator and instantiating objects itself, it passes a string parameter to the factory to create different types of the “Robot” subclasses.



5. Lessons Learned

Through the design, creation, and implementation of a system, we have learned just how difficult it is to build a system from scratch. In addition to this, we have realized the importance of the planning phase, as well as how much the project design will change between iterations. Without adequate drafting, like creating class, sequence, and activity diagrams, it would be almost impossible to create a cogent system that is not a jumbled mess of spaghetti code. Because we met about once a week to talk extensively about the project and make mock-ups of our design on whiteboard and paper, our finished product works in the way we intended, and the code is easily readable and understandable. We noticed that our design and implementation changed somewhat drastically from iteration to iteration, and that even with proper planning and forethought we could not conceive the potential problems that we would run into while coding our system. Perhaps the biggest change was our architecture; initially, we intended to build using a Model-View-Controller pattern. Instead, after more thorough research and discussion about what we needed, we settled on a Model-View-Presenter design. Although mostly semantic in difference, the minor changes that we implemented due to our architecture allowed us to code and organize in a more coherent fashion.

Not only did we change our design on a macro level, but we also had to add classes and amend methods for our system to transpire. We quickly realized that the way we had organized our classes in

our class diagram would result in highly-coupled code, and that if we wanted to change one thing, we would in turn have to closely look at the rest of the files to make sure our refactoring reflected that one change. Once we understood this, we we able to be mindful of how to organize our classes and dependencies properly, so as to follow good object oriented design principles.