Use-case for adding a model

Actions performed by actor	Responses from the System
1. The company receives a new model to distribute in its store	
2. Store worker issues request to add new model to registry	
	3. The system asks for extra attributes for the model
4. Worker enters in extra attributes for model	
	5. System creates appliance id and stores attributes/id and asks user if they would like to enter another model
6. User answers in the affirmative or the negative	
	7. If answer is in the affirmative, goes to step 3, otherwise it exits

Use-case for adding to inventory

Actions performed by user	Responses from system
1. Company receives new stock and decides to add to inventory	
2. Worker issues a request to add to inventory	
	3. System asks user to input appliance id for appliance and amount added to inventory
4. User enters specific appliance id and amount being added to inventory	
	5. System updates inventory for appliance id
	6. System checks for back orders for this appliance and if there are, goes to step 7, otherwise goes to step 9
	7. System will check the earliest back order and if there is enough in inventory to process, goes to step 8, otherwise it goes to step 9

	8. Back order is fulfilled, total sales and inventory is updated. If there are more back orders, goes to step 7, otherwise goes to step 9
	9. System prompts user that items have been added to inventory and if back orders have been fulfilled, which ones have been completed and asks if more items need to be added
10. User reads prompt and answers yes or no to additional items added	
	11. If user response is yes, return to step 3, if no, the system exits

Use-case for purchases

Actions performed by user	Responses from system
1. Customer requests to purchase quantity of appliance in store	
2. Worker identifies appliance by appliance id and customer by customer id	
3. Worker requests purchase from the system	
	4. System asks for appliance id and quantity requested
5. Worker enters appliance id and quantity into system	
	6. System checks if there is enough stock in inventory for immediate purchase, if true, goes to step 7, if not, goes to step 10
	7. System updates inventory, calculates sale, updates total sales. It then asks user if they would like to enter another purchase
8. User answers in the affirmative or in the negative	
	9. If the answer is the affirmative, system goes to step 4, otherwise it exits

	10. System will check for other back orders with this appliance id and add this order behind them, then prompts user that order has been sent to back order, and asks user if they would like to add another purchase
11. User answers in the affirmative or in the negative	
	12. If the answer is the affirmative, system goes to step 4, otherwise it exits

Use-case for enrolling a repair plan

Actions performed by user	Responses from system
1. User is checking out appliance and enters user id and appliance id into system	
	2. System will check if the appliance id has a corresponding repair plan, if so, moves to step 3, otherwise it exits
	3. System has found repair plan for appliance id and prompts user if the customer would like to enroll in repair plan
4. User asks customer if they would like to enroll in repair plan	
5. If customer agrees to repair plan, the user inputs yes, if they don't, user inputs no	
	6. If answer is affirmative, system records customer id, user id and appliance id to generate a repair plan and provides user information on the new repair plan. If answer is negative, system exits

Use-case for charging repair plans

Actions performed by user	Responses from system
1. Company decides to charge for repair plan or has repair plan charges done on some interval	
	2. System will search for the customer id's who have repair plans and update their balances based on cost of repair plan. System will also update total sales from repair plans. System will then send a notification confirming completion of charges to company and customer and exits
3. Company receives notification that repair plan charges have been issued and customers received notification of new charges added to their balance	

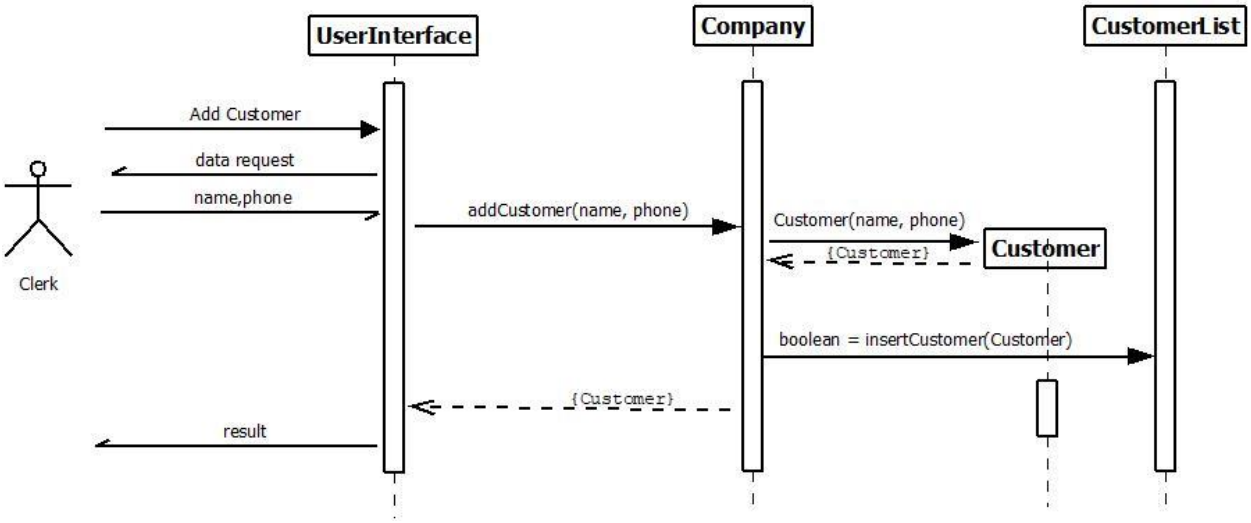
Use-case for printing revenue

Actions performed by user	Responses from system
1. User requests revenue from all sales and repair plans	
	2. System will prepare total sales and then total the amount of repair plans and format these numbers appropriately, then display these values to the user
3. User will view the total sales and total repair plan sales	
	4. System asks if user would like to recompute these values
5. User enters yes or no	
	6. If yes, goes to step 2. If no, system exits

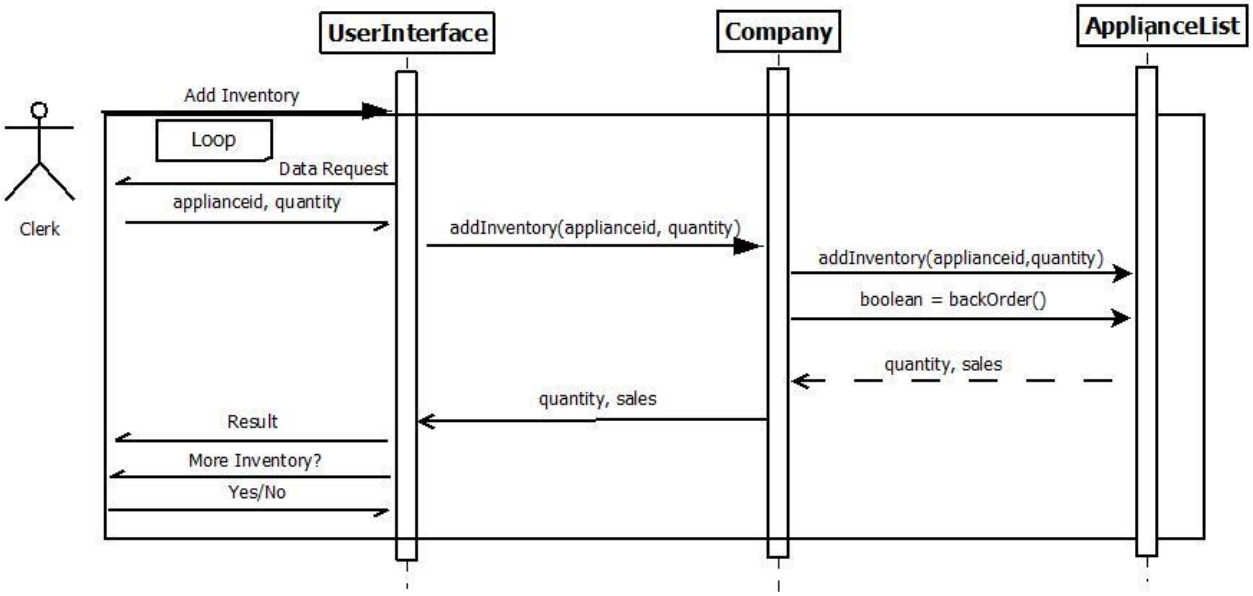
Use-case for listing all repair plans

Actions performed by user	Responses from system
1. User requests list of all repair plans	
	2. System searches all customer ids for an associated repair plan, if the customer has a repair plan associated with their id, the system will add the customer's name, phone, id, their account balances and brand/model of appliances for which they are enrolled to a list. After going through all the customers, this list will then be output to the User, the system will ask the user if they want to regenerate the list
3. User receives list of all repair plan information and enters yes or no	
	4. If yes, System will go to step 2, if no, system exits

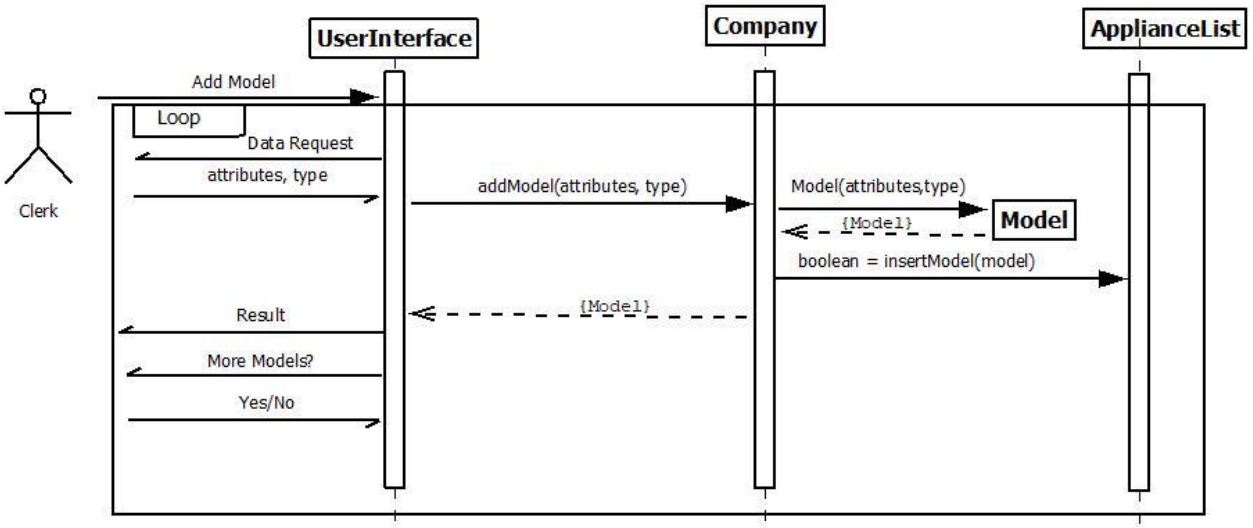
Sequence Diagram for Adding a Member



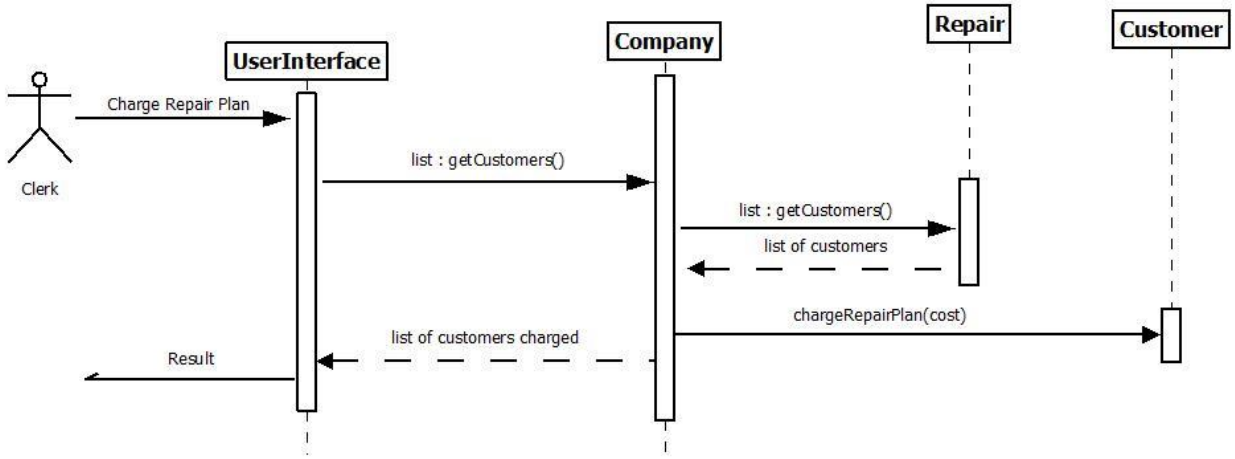
Sequnce Diagram for adding to Inventory



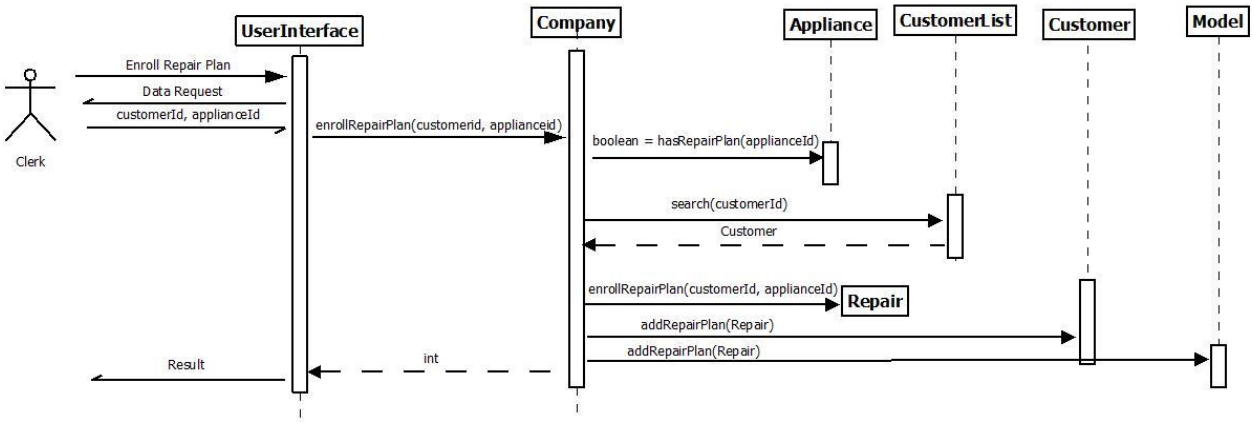
Sequence Diagram for adding a model



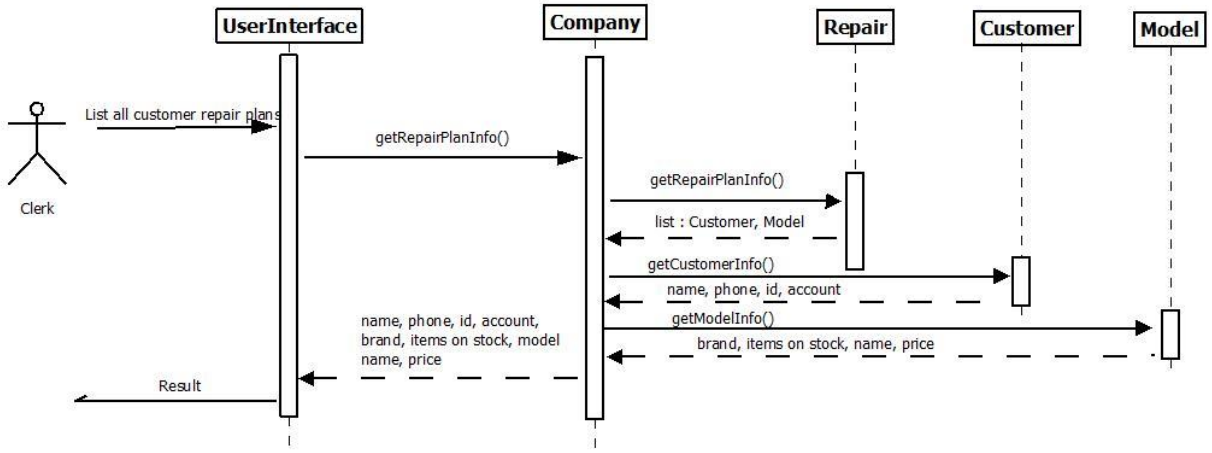
Sequence Diagram for charging repair plans



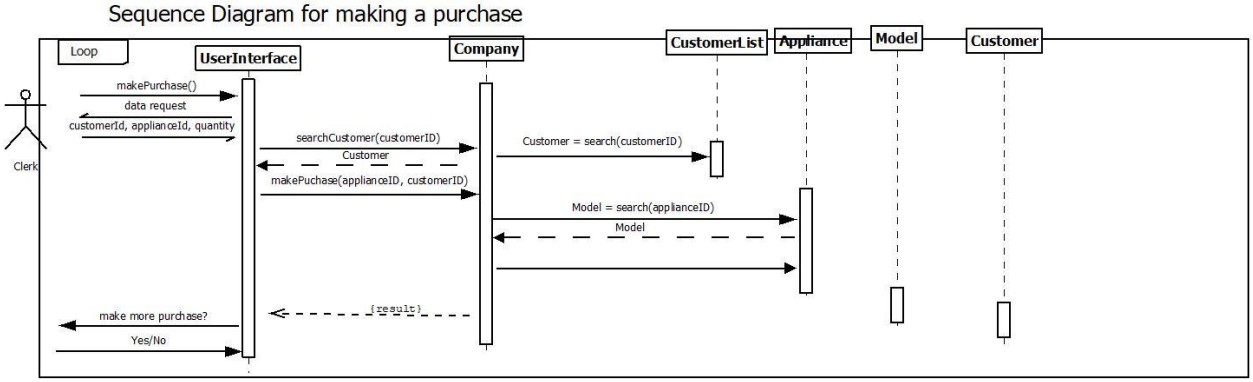
Sequence For Enrolling a Repair Plan

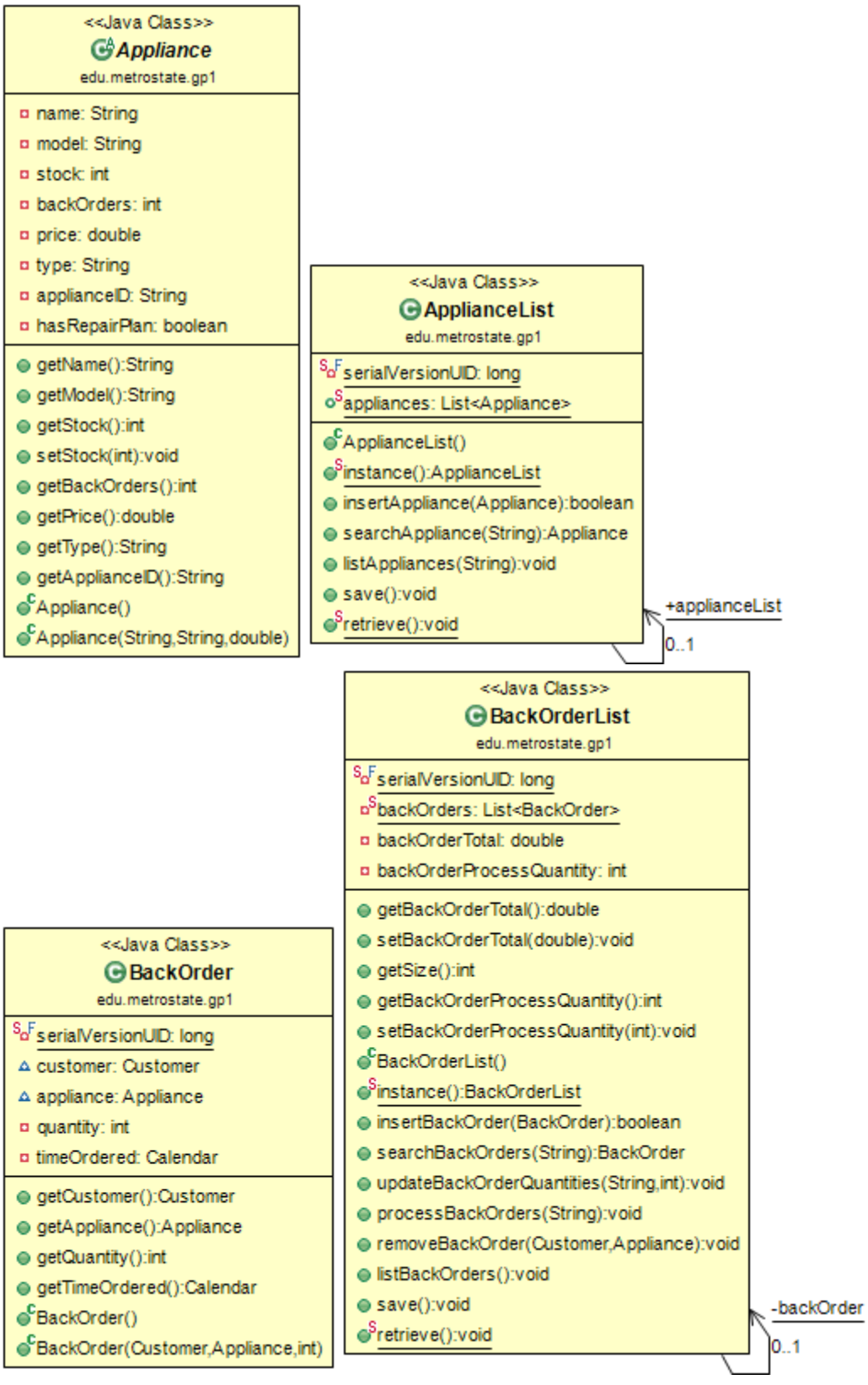


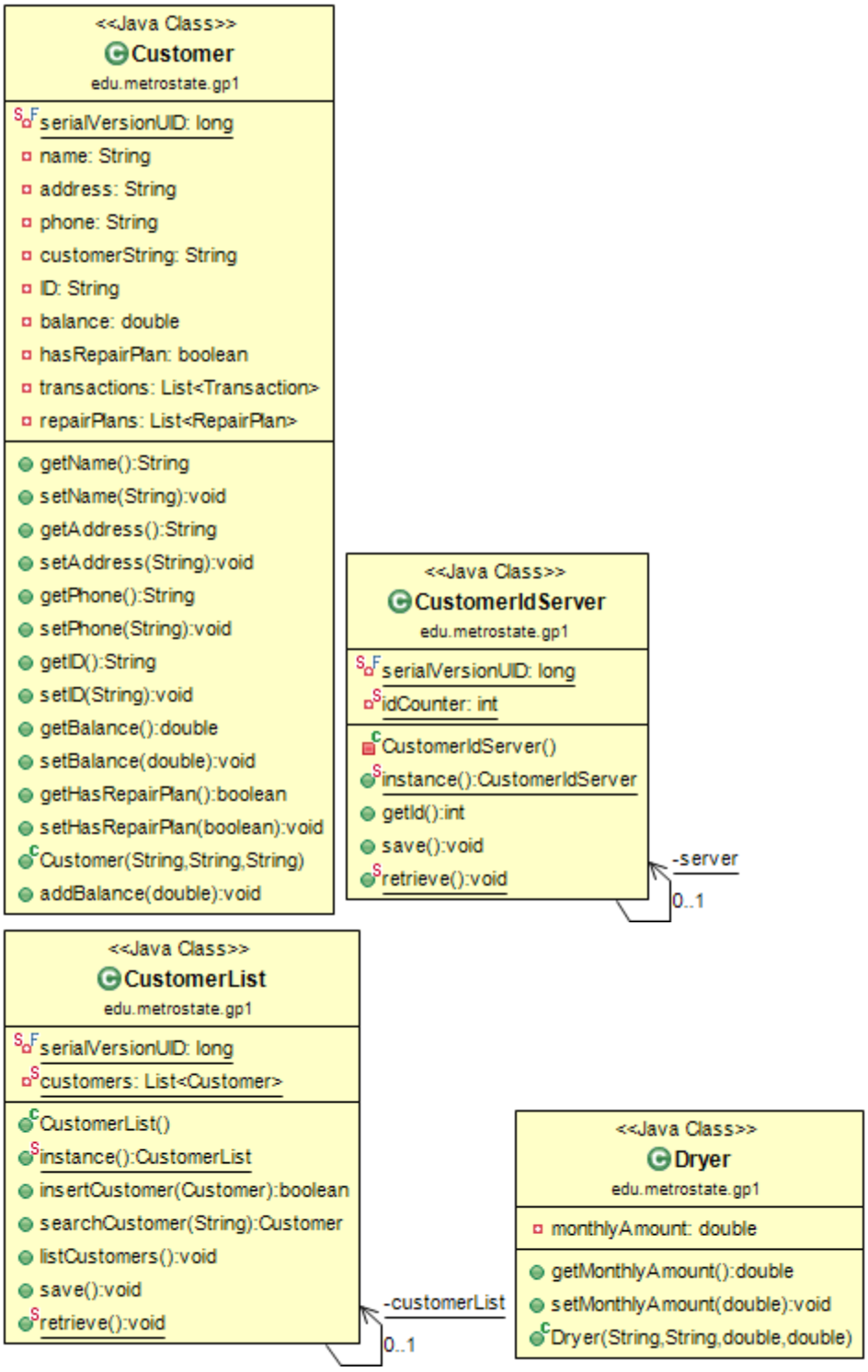
Sequence Diagram for listing all users in repair plans

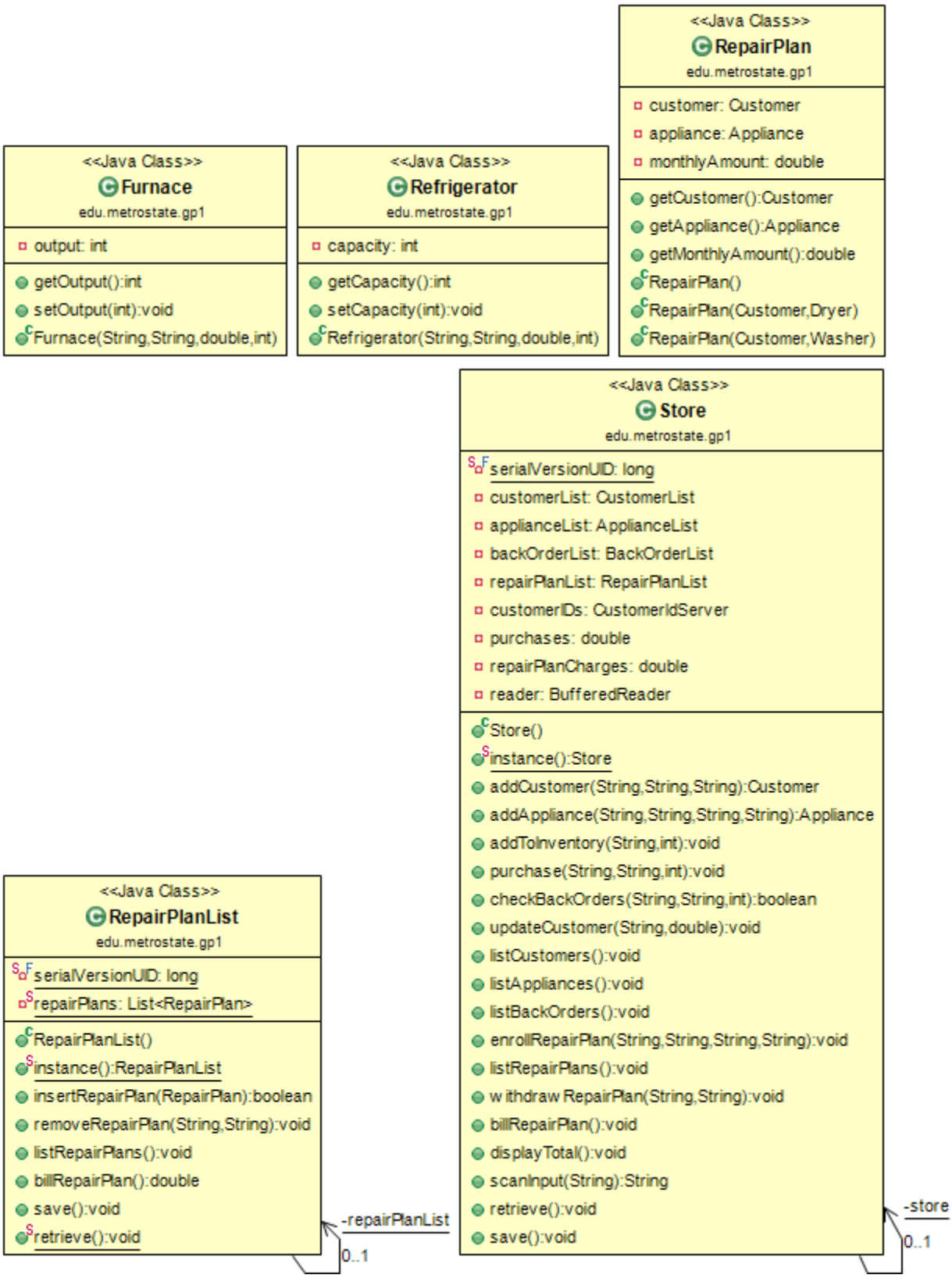


Sequence Diagram for making a purchase



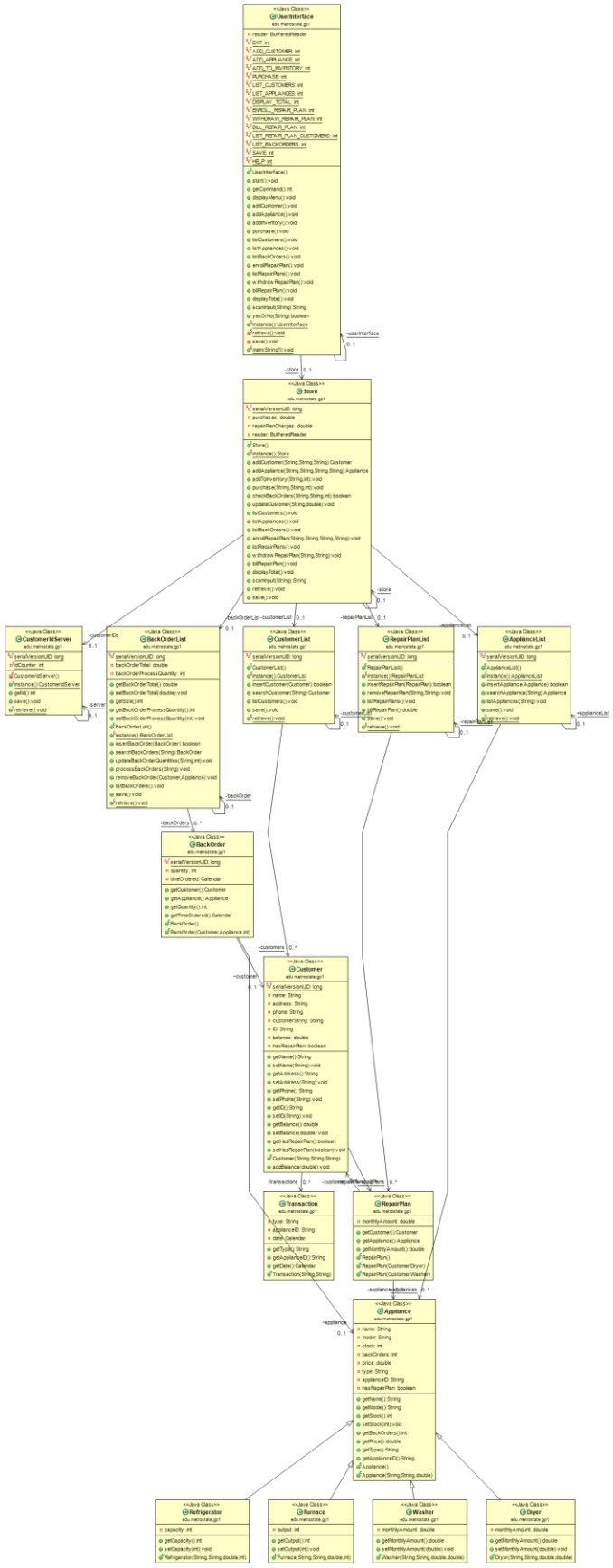

<<Java Class>>
Appliance
edu.metrostate.gp1
name: String
model: String
stock: int
backOrders: int
price: double
type: String
applianceID: String
hasRepairPlan: boolean
getName():String
getModel():String
getStock():int
setStock(int):void
getBackOrders():int
getPrice():double
getType():String
getApplianceID():String
Appliance()
Appliance(String,String,double)
<<Java Class>>
ApplianceList
edu.metrostate.gp1
serialVersionUID: long
appliances: List<Appliance>
ApplianceList()
instance():ApplianceList
insertAppliance(Appliance):boolean
searchAppliance(String):Appliance
listAppliances(String):void
save():void
retrieve():void
+applianceList
0..1
<<Java Class>>
BackOrderList
edu.metrostate.gp1
serialVersionUID: long
backOrders: List<BackOrder>
backOrderTotal: double
backOrderProcessQuantity: int
getBackOrderTotal():double
setBackOrderTotal(double):void
getSize():int
getBackOrderProcessQuantity():int
setBackOrderProcessQuantity(int):void
BackOrderList()
instance():BackOrderList
insertBackOrder(BackOrder):boolean
searchBackOrders(String):BackOrder
updateBackOrderQuantities(String,int):void
processBackOrders(String):void
removeBackOrder(Customer,Appliance):void
listBackOrders():void
save():void
retrieve():void
-backOrder
0..1
<<Java Class>>
BackOrder
edu.metrostate.gp1
serialVersionUID: long
customer: Customer
appliance: Appliance
quantity: int
timeOrdered: Calendar
getCustomer():Customer
getAppliance():Appliance
getQuantity():int
getTimeOrdered():Calendar
BackOrder()
BackOrder(Customer,Appliance,int)


<<Java Class>>
Customer
edu.metrostate.gp1
serialVersionUID: long
name: String
address: String
phone: String
customerString: String
ID: String
balance: double
hasRepairPlan: boolean
transactions: List<Transaction>
repairPlans: List<RepairPlan>
getName():String
setName(String):void
getAddress():String
setAddress(String):void
getPhone():String
setPhone(String):void
getID():String
setID(String):void
getBalance():double
setBalance(double):void
getHasRepairPlan():boolean
setHasRepairPlan(boolean):void
Customer(String,String,String)
addBalance(double):void
<<Java Class>>
CustomerIdServer
edu.metrostate.gp1
serialVersionUID: long
idCounter: int
CustomerIdServer()
instance():CustomerIdServer
getId():int
save():void
retrieve():void
-server
0..1
<<Java Class>>
CustomerList
edu.metrostate.gp1
serialVersionUID: long
customers: List<Customer>
CustomerList()
instance():CustomerList
insertCustomer(Customer):boolean
searchCustomer(String):Customer
listCustomers():void
save():void
retrieve():void
-customerList
0..1
<<Java Class>>
Dryer
edu.metrostate.gp1
monthlyAmount: double
getMonthlyAmount():double
setMonthlyAmount(double):void
Dryer(String,String,double,double)


<<Java Class>>
Furnace
edu.metrostate.gp1
output: int
getOutput():int
setOutput(int):void
Furnace(String,String,double,int)
<<Java Class>>
Refrigerator
edu.metrostate.gp1
capacity: int
getCapacity():int
setCapacity(int):void
Refrigerator(String,String,double,int)
<<Java Class>>
RepairPlan
edu.metrostate.gp1
customer: Customer
appliance: Appliance
monthlyAmount: double
getCustomer():Customer
getAppliance():Appliance
getMonthlyAmount():double
RepairPlan()
RepairPlan(Customer,Dryer)
RepairPlan(Customer,Washer)
<<Java Class>>
Store
edu.metrostate.gp1
serialVersionUID: long
customerList: CustomerList
applianceList: ApplianceList
backOrderList: BackOrderList
repairPlanList: RepairPlanList
customerIDs: CustomerIdServer
purchases: double
repairPlanCharges: double
reader: BufferedReader
Store()
instance():Store
addCustomer(String,String,String):Customer
addAppliance(String,String,String,String):Appliance
addToInventory(String,int):void
purchase(String,String,int):void
checkBackOrders(String,String,int):boolean
updateCustomer(String,double):void
listCustomers():void
listAppliances():void
listBackOrders():void
enrollRepairPlan(String,String,String,String):void
listRepairPlans():void
withdrawRepairPlan(String,String):void
billRepairPlan():void
displayTotal():void
scanInput(String):String
retrieve():void
save():void
<<Java Class>>
RepairPlanList
edu.metrostate.gp1
serialVersionUID: long
repairPlans: List<RepairPlan>
RepairPlanList()
instance():RepairPlanList
insertRepairPlan(RepairPlan):boolean
removeRepairPlan(String,String):void
listRepairPlans():void
billRepairPlan():double
save():void
retrieve():void
-repairPlanList
0..1
-store
0..1


<<Java Class>>
UserInterface
edu.metrostate.gp1
store: Store
reader: BufferedReader
EXIT: int
ADD_CUSTOMER: int
ADD_APPLIANCE: int
ADD_TO_INVENTORY: int
PURCHASE: int
LIST_CUSTOMERS: int
LIST_APPLIANCES: int
DISPLAY_TOTAL: int
ENROLL_REPAIR_PLAN: int
WITHDRAW_REPAIR_PLAN: int
BILL_REPAIR_PLAN: int
LIST_REPAIR_PLAN_CUSTOMERS: int
LIST_BACKORDERS: int
SAVE: int
HELP: int
UserInterface()
start():void
getCommand():int
displayMenu():void
addCustomer():void
addAppliance():void
addInventory():void
purchase():void
listCustomers():void
listAppliances():void
listBackOrders():void
enrollRepairPlan():void
listRepairPlans():void
withdrawRepairPlan():void
billRepairPlan():void
displayTotal():void
scanInput(String):String
yesOrNo(String):boolean
instance():UserInterface
retrieve():void
save():void
main(String[]):void
-userInterface
0..1
<<Java Class>>
Washer
edu.metrostate.gp1
monthlyAmount: double
getMonthlyAmount():double
setMonthlyAmount(double):void
Washer(String,String,double,double)

Physical class diagram:


UserInterface
edu.metrostate.gz1
Store
edu.metrostate.gz1
CustomerIdServer
edu.metrostate.gz1
BackOrderList
edu.metrostate.gz1
CustomerList
edu.metrostate.gz1
RepairPlanList
edu.metrostate.gz1
ApplianceList
edu.metrostate.gz1
BackOrder
edu.metrostate.gz1
Customer
edu.metrostate.gz1
Transaction
edu.metrostate.gz1
RepairPlan
edu.metrostate.gz1
Appliance
edu.metrostate.gz1
Refrigerator
edu.metrostate.gz1
Furnace
edu.metrostate.gz1
Washer
edu.metrostate.gz1
Dryer
edu.metrostate.gz1