Optional Feature Set	Changes required to support
Support all other legal denominations and coinage.	1. Main.java will have to ask user to input initialization amounts for all other denominations (as it does now for $20 and $50). This will need to be passed to CashDispenser constructor. CashDispenser will then store the state of # of numbers of notes for each denomination in same way it does now for $20 and $50 (ensure it is done in ascending order). 2. I believe the rest of the code will stay the same other than the `validateWithdrawRequest` function which will now need to ensure that any amount larger than 0 is a valid amount. 3. Will need to add unit tests to cover other scenarios.
The controller should dispense combinations of cash that leave options open. For example, if it could serve up either 5 $20 notes or 2 $50 notes to satisfy a request for $100, but it only has 5 $20 notes left, it should serve the 2 $50 notes.	1. `findAllPossibleCombinations` already returns a full list of valid possible combinations of cash that can be dispensed for a particular amount requested by user. We would have to update `dispenseMoneyToUser` method to accept the whole list provided by findAllPossibleCombinations and go through each combination to find the combination that gives us the least difference between note amounts. E.g. serving up 5 $20 notes or 2 $50 notes where we currently have 5 $20 notes in stock and 5 $50 notes in stock, we would serve 2 $50 notes since after serving this, the difference between the number of $20 notes and the number of $50 notes (5-3=2) is smaller than if we served 5 $20 notes (5-0= 5) 2. Will need to add unit tests.
The controller needs to be able to inform other interested objects of activity. Threshold notification in particular is desirable, so that the ATM can be re-supplied before it runs out of cash.	1. Every time we dispense money, we update the state of the dispenser. Perhaps we can send a message to any systems that has subscribed to stateUpdates via TCP. Or perhaps we only need to send a message once our note amount has fallen below a certain threshold. This other system can then supply our system with more bank notes where applicable. 2. Will need to add integration test between systems to ensure the request and responses are as expected.

Persistence of the controller is optional at this time. It can be kept in memory. However, it should go through a distinct initialisation period where it is told the current available amounts prior to being used.	1. Instead of asking user to initialize number of notes available in application, we can store this information in a database which we can query every time the application starts up. Before we close the application, we update the information in this database so that we have an accurate state for next time the application starts up.
Multi-currency support	1. Different currencies have different denominations. We would have to create a subclass of `CashDispenser` for each currency we decide to support. E.g. `CashDispenserEuro` or `CashDispenserUsd.` These classes will have their own implementation of populating `atmNotesStatusList.` 2. The rest of the functionality can stay in the base CashDispenser class since it can be shared between currencies. 3. When the application starts up, we will ask the user which currency they want to deal with and based on their answer we will call `withdrawMoney` on the corresponding CashDispenser. E.g. `cashDispenserEuro.withdrawMoney(amount)` if user wants to withdraw in EUROs. 4. Will need to add unit tests to cover other currencies.