**Problem Statement**

To design/develop a cash dispensing application for use in an ATM or similar device. There is no need to request authorisation or availability of funds. The application should assume that all requests are legitimate; there will be other components of the system that will do things such as communicating with bank accounts and authorising withdrawals.

**Technologies Used**

JAVA 8, SpringBoot, MAVEN, JUnit

**Functionalities achieved**

The application is capable of following tasks

* Depositing money
* Checking available balance in the machine
* Withdraw Amount
* Exit confirmation
* Exception handling
* Dynamic feature to add other denominations as well

NOTE- FOR FURTHER DETAIL REFER USAGE DOCUMENT.

**How to run?**

* Open command prompt
* Point to the “Automated Build folder”
* Run “java -jar atm.spring-boot-0.0.1-SNAPSHOT.jar”
* Follow the usage document provided with source code

**What is Spring Boot?**

Spring Boot aims to make it easy to create Spring-powered, production-grade applications and services with minimum fuss. It takes an opinionated view of the Spring platform so that new and existing users can quickly get to the bits they need. You can use it to create stand-alone Java applications that can be started using ‘java -jar’ or more traditional WAR deployments.

**What is Maven and Why Maven?**

Maven’s primary goal is to allow a developer to comprehend the complete state of a development effort in the shortest period of time. In order to attain this goal there are several areas of concern that Maven attempts to deal with:

* Making the build process easy
* Providing a uniform build system
* Providing quality project information
* Providing guidelines for best practices development
* Allowing transparent migration to new features

**Test cases involved**

The test cases are automated within the build written in JUint. The test cases are written for an ATM-style of machine (with $20 and $50 notes), the following dispensed amounts are of particular interest:

* $20
* $40
* $50
* $70
* $80
* $100
* $150
* $60
* $110
* $200, when there is only 3x$50 notes and 8x$20 notes available.

To ensure the proper execution of JUnit so as to check all scenarios against combination of only 3x$50 notes and 8x$20 notes, after every successful withdrawal, test automation adds the withdrawn currency again to the available stack

**The test cases can be written to cover any other required scenarios.**