## **Development Tools**

Apache Maven 3.3.0  
Java Version 1.8.0\_111  
IntelliJ IDEA 2016.3 Community Edition

## **Minimum requirements to run the fitbits application**

Java Version 1.8.0\_111.

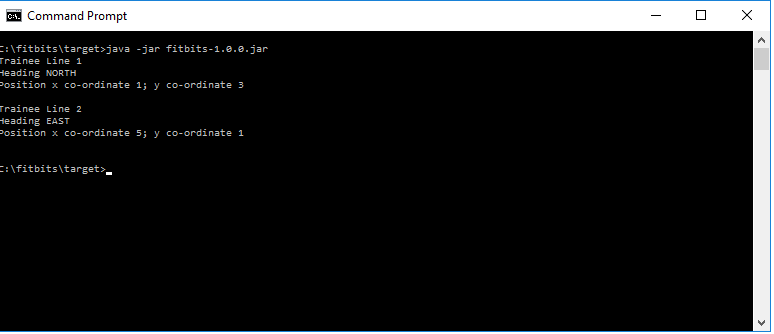
## **Minimum requirements to build the fitbits application**

Apache Maven 3.3.0  
Java Version 1.8.0\_111.

## **Deployment**

Extract the *fitbits.zip* file to any folder 🡪 <FITBITS\_ROOT>.

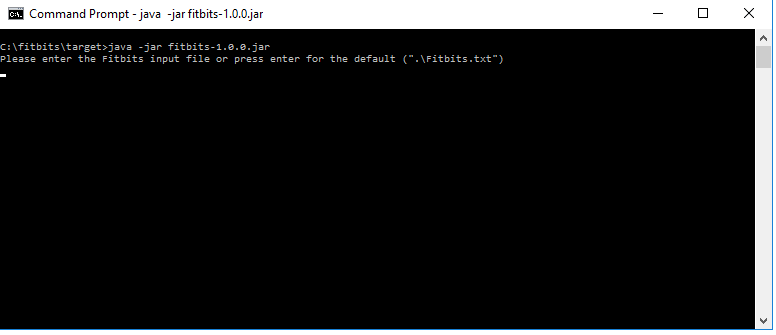
## **Quick View**

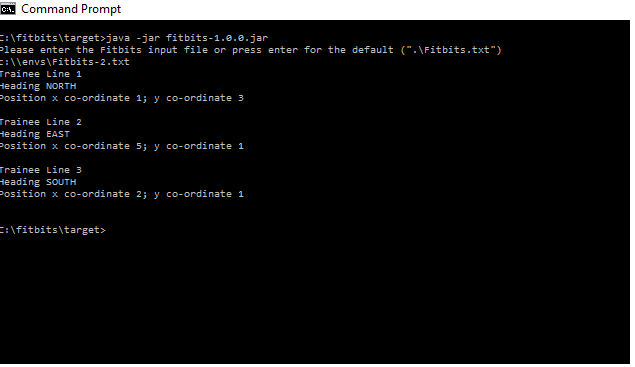
Navigate to *<FITBITS\_ROOT>\target* and run the following command  
  
***java –jar fitbits-1.0.0.jar  
  
***

You will see the screen above. This simple run takes as input the file *<FITBITS\_ROOT>\target\Fitbits.txt* **which has the same contents as provided in the *Code Problem Fitbits-v1.docx* document, and will show the actual results of the moves match those expected**.

## **Advanced View**

Open *<FITBITS\_ROOT>\target\application.properties* and remove the # from the start of the following line and save the file

***#retrieve.file.name.class=au.com.medibank.dao.ConsoleFitbitRetrieveFileName***Re-enter the command line  
  
***java –jar fitbits-1.0.0.jar***  
  


Now you will be invited to enter an input file of your own, or just press enter for the same default file that you submitted in the *Quick View* section. Enter a file of your choosing  
  


## **Application Startup**

This is a stand-alone java application. The *Main* method optionally takes a string class name as an input argument which it will use to instantiate an instance of ***au.com.medibank.startup.context.ApplicationContext***.   
  
If a string class name is **not** supplied (as in the examples above), the start up process (see ***au.com.medibank.starting.context.Configuration*)** will default to creating an instance of ***au.com.medibank.startup.context.PropertyApplicationContext  
  
PropertyApplicationContext*** takes its configuration details from an internal file called *default.properties* but, as demonstrated above, the default settings can be changed by updating properties in a file called *application.properties* if such a file is anywhere on the classpath.

## **Design Overview**

The design has involved creating a loose-coupling between the mechanism by which the data is retrieved and how that data is then processed (i.e. how the *Trainees* are moved around the *Pitch*).

This loose coupling is achieved by allowing the *controller* in this application (see *au.com.medibank.controller.Controller*) to know only about an interface (*au.com.medibank.dao.FitbitsEntitiesDAO)* which implements 2 methods *getPitch()* and *getNextTrainee().*

Currently*, t*he only implementation of this interface is *au.com.medibank.dao.FitbitsEntitiesFileDAO* which, as shown earlier, gets the data from files on a network. Other implementations of this data retrieval could be integrated by changing the *entitities.dao* property in the *application.properties* without changing any of the existing code.  
  
The principle of loose coupling has been applied throughout the application as demonstrated earlier with the mechanism to either load the file automatically or, via an external property change, allow the user to manually enter the location of the input file. The loose coupling has allowed testing via Mocks and stubs to be a straight-forward process

All the entities (i.e. *Trainee, Pitch Position, Heading*)have been implemented and test cases are available to support their function.   
  
Likewise the 3 instructions (*Move, TurnLeft* and *TurnRight*) have been implemented and tested. If new instructions were to be required, they could easily to the *au.com.medibank.intsruction* package and instantiated via the *InstructionFactory* class without any changes required to other parts of the software

## **Assumptions**

The following assumptions have been made

1. A *Trainee* cannot leave the boundary as defined by the *Pitch*. If they do the application will throw an error to the console and terminate. A new instruction could be introduced easily that just ignored the instruction in this case.
2. Two *Trainees* are allowed in the same position (e.g. x=3, y=4) on a *Pitch*. No validation is in place to prevent that happening.
3. I have followed the request of the ***Code Problem Fitbits-v1.docx* document,** to not use any external libraries. In particular that has involved electing not to use Spring framework to instantiate some of the core classes in the application. For the same reason Log4j has also not been integrated.

## **(re)Building the application**

The build is a standard maven approach. Maven and java 8 must be installed

The testing framework uses *Mockito 2.2.29* and *jacoco 0.7.8* for testing and testing coverage respectively.

To perform a standard build enter the following from the *<FITBITS\_ROOT>****mvn clean install***  
  
(WARNING …be careful to not lose any changes to *<FITBITS\_ROOT>\target\application.properties*, the above build will clean up the target folder*)*

To view code coverage you can run

***mvn clean jacoco:prepare-agent install jacoco:report***

## **Viewing the application via IntelliJ**

The software can be imported into IntelliJ via the *pom.xml* in the *<FITBITS\_ROOT>*. It is structured as a standard maven project.