Vending Machine: Build, Execution and Design Decisions

Introduction

This document outlines the build process, execution procedures and design decisions.

Note to reader:

I understand that the requirement was to only produce the change tracking API, implementation and command line interface test harness, but as I was enjoying the project so much, I've ended up extending the system to be a more realistic and hopefully an more exciting user experience for you to play with.

Build and Execute

Pre-requisites

- Windows OS
- Maven 3.8.5 installed

Build Procedure

1. Open a command prompt the windows machine
    a. If downloading the repository from GIT navigate to the location you wish to store the repository.
    b. Enter 'git clone https://github.com/neilkrupa/vending-machine.git'
    c. The repository will be downloaded from GIT
2. Navigate to the 'vending-machine' source directory
3. Run 'mvn clean install'
4. The source code should build successfully

Execution Procedure

1. Either via file explorer or via console navigate to 'vending-machine-parent\vendm-core-impl'
2. Run the 'runVendingMachine.bat' script

Design Decisions

API and Implementation

The change tracker implementation has been separated into two modules, an API (vendm-change-api) and an implementation (vendm-change-impl). This allows the main functionality of the implementation to be abstracted from the rest of the system, providing code security by limiting the available actions that can be performed by classes interacting with the implementation. Further to this, defining a separate API module enables the implementation of the API to be interchanged without dependencies on the implementation affecting the rest of the system. The API provides a window to external modules with regard to the functionality of the API implementation.

The API has been grouped into 3 main packages:

1. Enums – Enumerations define common objects used by the implementation and throughout the rest of the system.
2. Interfaces – Interfaces define the available methods that can be externally executed on the implementation as well as defining the methods that must be implemented when creating an implementation of the defined API.
3. Utils – Utility classes provide useful common functionality used to convert, generate and process objects.

The implementation has been grouped into 3 main packages:

1. Core – The core class that implements the defined API interface.
2. Exceptions – Implementation-specific exceptions created to handle specific events.
3. Processing – Classes to abstract processing functionality away from the main core class.

Build Process

I have selected Maven to build the project because it is the tool I have had the most exposure to and the fact it allows multi-module java projects to be built with ease alongside enabling unit tests to be run as part of the build process.

Unit Testing

I have selected the Junit-5 framework to run unit tests for the change tracker API implementation as it integrates well with eclipse, can be used in conjunction with maven during the build process and enables visibility of code coverage.

I have written unit tests for the change tracker API implementation to ensure validity of any future code changes within the implementation. The coverage per package is as follows:

- Processing: 98.7% - This is not 100% due to the fact that all methods in the processing class are static and there is no constructor to test.
- Core (Implementation of API) – 100%
- Exceptions – 100%

Extension to Requirements

As stated in the introduction to this document I have extended the system to be a more realistic simulation of a vending machine in an attempt to showcase my software abilities. This includes an inventory in which items can be defined and inventory handled, a core vending machine manager to handle each module and an extended CLI to allow for simple future expansion.

If you have any questions or queries regarding anything in the source code or this document, please don't hesitate to contact me.