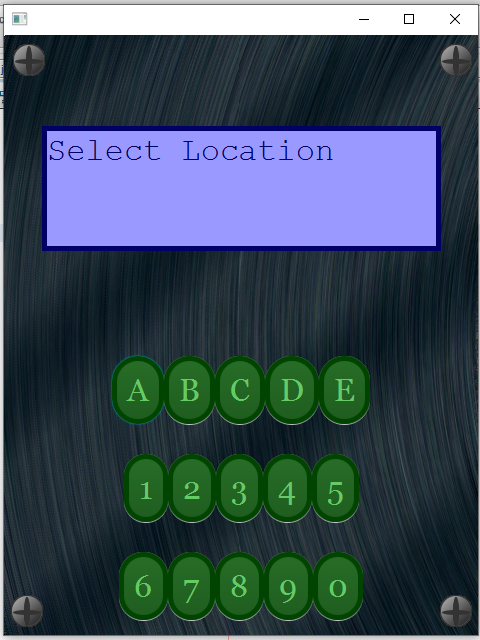
Vending Machine

Our project will simulate a vending machine interface. It will include a GUI that represents the buttons on a vending machine. The program will store inventory information and sales information in separate files. Our target audience will be vending machine owners and operators. Our application will allow people to interact with a vending machine without having to physically interact with one. The main features of this application will be as follows; Select items and accept payment, audit the machine by comparing physical stock to sales, and to alter displayed products.

# Sketches

# Documentation

|  |  |
| --- | --- |
| Machine | |
| Field Summary | |
| -inventories: [][] | A two-dimensional array representing inventory inventories on this machine |
| -transactions: ArrayList<Transactions> | A collection of transactions made on this machine |
| -vendor: Vendor | A representation of the vendor |
| Method Summary (for brevity, field setters and getters were omitted) | |
| +addTransaction(transaction: Transaction): void | Adds a transaction to collection |
| +voidTransaction(transaction: Transaction): boolean | Removes from collection and persistence |
| +addInventory(Inventory: Inventory): void | Adds an inventory to machine |
| +removeInventory(Inventory: Inventory): boolean | Removes an inventory from machine |
| +persistAll(): void | Persist all inventories and transactions to a file |
| +auditInventory(): void | Check inventory quantities against minimums |

|  |  |
| --- | --- |
| Vendor | |
| Field Summary | |
| -name: String | A name of the vendor to be displayed |
| -email: String | An email which would be displayed to customers for contact |
| -phone: String | A phone number which would be displayed to customers for contact |
| Method Summary | |
| +getName(): String | Returns the name. |
| +getEmail(): String | Returns the email. |
| +getPhone(): String | Returns the phone number. |

|  |  |
| --- | --- |
| *Unit* | |
| Field Summary | |
| +shortName | The short form of the unit name; for example, “ml” |
| +longName | The long form of the unit name; for example, “millilitre” |
| Constructors | |
| +Unit(shortName: String), longName: String) | Creates an instance using a short name and long name. |
| Literals | |
| ML(“ml”, “Milliliters”) | Milliliters |
| G(“g”, “Grams”) | Grams |

|  |  |
| --- | --- |
| Product | |
| Field Summary | |
| +SKU: String | The stock keeping unit; must be unique |
| -name: String | The display friendly name |
| -cost: float | The cost value of the product |
| -quantity: float | The amount of sold at a time |
| -unit: String | The unit of quantity |
| -shortDescription: String | The short description of the product |
| -longDescription: String | The long description of the product |
| -persisted: boolean | Records if the product has been persisted to file |
| Method Summary (for brevity, field setters and getters were omitted) | |
| +isPersisted(): Boolean | Returns true if product is persisted to file. |
| +persist(): boolean | Writes product to file by either update or appending, returns true if successful. |

|  |  |
| --- | --- |
| Entry | |
| Field Summary | |
| -product: Product | The product sold |
| -quantity: int | The number of this product sold |
| Method Summary (for brevity, field setters and getters were omitted) | |
| +setProduct(product: Product): void | Set the product. |
| +setQuantity(quantity: Quantity): void | Set the quantity. |
| +getProduct(): product | Get the product. |
| +getQuantity(): int | Get the quantity. |

|  |  |
| --- | --- |
| Transaction | |
| Field Summary | |
| -created: DateTime | The record date and time of when the transaction as made |
| -entries: ArrayList< TransactionEntry> | A collection of product and quantity sold |
| Method Summary (for brevity, field setters and getters were omitted) | |
| +setDateTime(datetime: DateTime): void | Sets the date and time. |
| +addEntry(entry: TransactionEntry): void | Add an entry to collection. |
| +voidEntry(entry: TransactionEntry): boolean | Remove an entry from collection and persistence. |
| +isPersisted(): Boolean | Returns true if persisted to file. |
| +persist(product: Transaction): boolean | Writes transaction to file by either update or appending. |

|  |  |
| --- | --- |
| Inventory | |
| Field Summary | |
| +ROW: Char | The row for this inventory |
| +COLUMN: Char | The column for this inventory |
| -product: Product | The product to record |
| -quantity | The current quantity of products on this inventory |
| -minimum | The minimum quantity at which this inventory may fall to before dispatching for a delivery of more products |
| -maximum | The maximum quantity of products this Inventory may hold |
| Method Summary (for brevity, field setters and getters were omitted) | |
| +isLow(): Boolean | Returns true if current quantity is below minimum quantity |
| +persist(product: Inventory): boolean | Writes the Inventory to a file by either updating or appending |

# Files

## products.db (Plaintext file)

#####

## SKU name cost quantity unit short\_description long\_description

#####

5GUMAPPLE "5 Gum – Apple" 2.50 10 pc "Sugar free gum" null

GUMMYWORMP "Whisky Dink" 5 1 pc "Whiskey flavour gummy worm" null

ASRTNUTSX" Sack o’ Nutz" 2 150 g "Extra salty assorted whole nuts" null

GUMBALLP "Chewing Balls" 2 1 pc "Pineapple flavour" nul

## inventories.db (Plaintext file)

# row column SKU quantity minimum maximum

A 4 5GUMAPPLE 20 10 40

# Research Opportunities

Inside the *Machine* class, *inventories* and *transactions* fields could benefit from a custom implementation of a *Collection* class; *persistAll()* could then make use of an instance method from the implementation that both iterates and persists data.

<https://docs.oracle.com/javase/tutorial/collections/interfaces/list.html>

<https://docs.oracle.com/javase/tutorial/collections/interfaces/collection.html>

<https://docs.oracle.com/javase/tutorial/collections/custom-implementations/index.html>