COMP2232 Object-Oriented Programming Concepts

Design Document

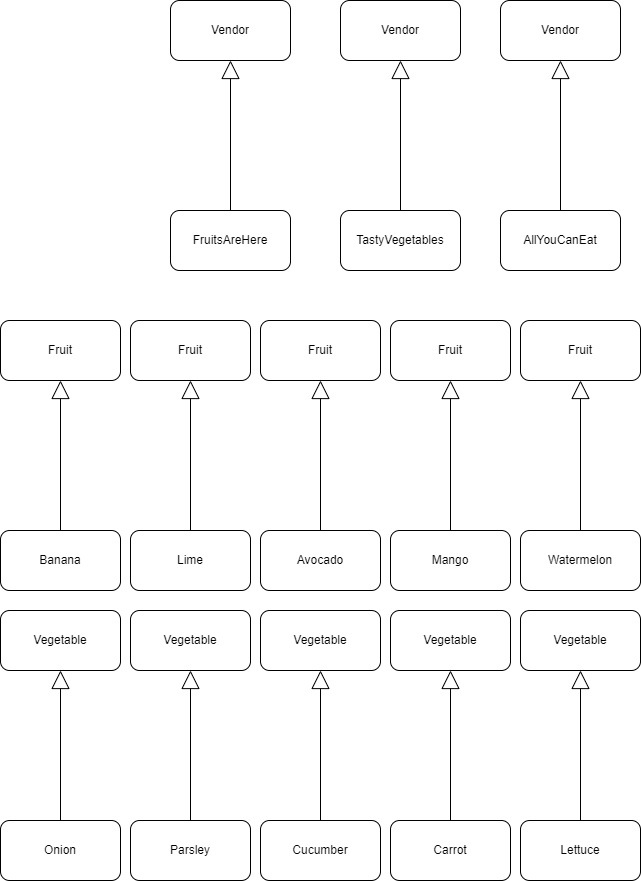
Group Members

The group members are Aquon Bovell, Kenez Horne and Dwayne Archer.

# Class Diagram

A diagram showing the relationships between the classes.

Inherited classes are shown in the below diagram.

****

**Shape

Description automatically generated with medium confidenceGraphical user interface

Description automatically generated with low confidenceGraphical user interface

Description automatically generated with medium confidence**

Graphical user interface

Description automatically generated

The UML diagram is shown above for the instances of the classes.

# General Class Descriptions

Include the purpose of the class and any other important information.

|  |  |
| --- | --- |
| **Name of Class** | **Description** |
| Fruit | The fruit class contains the spoilt value and the name for the individual fruits. |
| Banana | The Banana class contains a constructor as well as two constants for the selling price and cost price |
| Lime | The Lime class contains a constructor and two constants for the selling price and the cost price |
| Avocado | The Avocado class contains a constructor as well as two constants for the selling price and cost price |
| Mango | The Mango class contains a constructor as well as two constants for the selling price and cost price |
| Watermelon | The Watermelon class contains a constructor as well as two constants for the selling price and cost price |
| Vegetable | The vegetable class contains the spoilt value and the name of individual vegetables. |
| Onion | The Onion class contains a constructor as well as two constants for the selling price and cost price |
| Parsley | The Parsley class contains a constructor as well as two constants for the selling price and cost price |
| Cucumber | The Cucumber class contains a constructor as well as two constants for the selling price and cost price |
| Carrot | The Carrot class contains a constructor as well as two constants for the selling price and cost price |
| Lettuce | The Lettuce class contains a constructor as well as two constants for the selling price and cost price |

|  |  |
| --- | --- |
| **Name of Class** | **Description** |
| Vendor | This class provides a template for the other vendor classes. It describes the general state and functionality of a vendor object. |
| FruitsAreHere | This class provides the state and functionality of a vendor who sells fruits and how they operate daily as a provider of fruits to supermarkets. |
| TastyVegetables | This class provides the state and functionality of a vendor who sells vegetables and how they operate daily as a provider of vegetables to supermarkets. |
| Print | Display to the screen a specific format in verbose mode |
| Output | Writes to a log file in a specific format |
| Customer | Represents a person buying from the Supermarket |
| Supermarket | Simulate the management of a supermarket |
| AllYouCanEat | This class provides the state and functionality of a vendor who sells both fruits and vegetables and how they operate daily as a provider of both fruits and vegetables to supermarkets. |

# Individual Class Descriptions

Include descriptions for each field and method created for each class. Use the table below for each class described.

**Name of Class: Fruit**

|  |  |
| --- | --- |
| **Field or Method** | **Description** |
| private String nameOfFruit | The field will store the name of each fruit. |
| private int spoiltValue | The spoiltValue field will be a random number for each fruit. |
| getNameOfFruit() | The getNameOfFruit method returns the name of each fruit. |
| Fruit() | The Fruit constructor initializes the name field and the spoiltValue to a random value between 5 and 10. |
| getSpoiltValue() | The getSpoiltValue returns the spoilt value |
| decreaseSpoiltValue() | The decreaseSpoiltValue decrements the spoilt value. |

**Name of Class: Banana**

|  |  |
| --- | --- |
| **Field or Method** | **Description** |
| Banana() | The constructor calls the parent constructor via the super keyword and Banana is passed into super. |
|  |  |

**Name of Class: Lime**

|  |  |
| --- | --- |
| **Field or Method** | **Description** |
| Lime() | The constructor calls the parent constructor via the super keyword and Lime is passed into super. |
|  |  |

**Name of Class: Mango**

|  |  |
| --- | --- |
| **Field or Method** | **Description** |
| Mango() | The constructor calls the parent constructor via the super keyword and Mango is passed into super. |
|  |  |

**Name of Class: Avocado**

|  |  |
| --- | --- |
| **Field or Method** | **Description** |
| Avocado() | The constructor calls the parent constructor via the super keyword and Avocado is passed into super. |
|  |  |

**Name of Class: Watermelon**

|  |  |
| --- | --- |
| **Field or Method** | **Description** |
| Watermelon() | The constructor calls the parent constructor via the super keyword and Watermelon is passed into super. |
|  |  |

**Name of Class: Vegetable**

|  |  |
| --- | --- |
| **Field or Method** | **Description** |
| private String nameOfVegetable | The field will store the name of each vegetable |
| private int spoiltValue | The spoiltValue field will be a random number for each fruit |
| getNameOfVegetable() | The getNameOfVegetable returns the name of each vegetable |
| getSpoiltValue() | The getSpoiltValue returns the spoilt value of each vegetable |
| decreaseSpoiltValue() | The decreaseSpoiltValue decrements the spoilt value |
| Vegetable() | The Vegetable constructor initializes the name field and the SpoiltValue to a random value between 5 and 10 |

**Name of Class: Onion**

|  |  |
| --- | --- |
| **Field or Method** | **Description** |
| Onion() | The constructor calls the parent constructor via the super keyword and Onions is passed into super. |
|  |  |

**Name of Class: Cucumbers**

|  |  |
| --- | --- |
| **Field or Method** | **Description** |
| Cucumber() | The constructor calls the parent constructor via the super keyword and Cucumber is passed into super. |
|  |  |

**Name of Class: Lettuce**

|  |  |
| --- | --- |
| **Field or Method** | **Description** |
| Lettuce() | The constructor calls the parent constructor via the super keyword and Lettuce is passed into super. |
|  |  |

**Name of Class: Carrot**

|  |  |
| --- | --- |
| **Field or Method** | **Description** |
| Carrot() | The constructor calls the parent constructor via the super keyword and Carrot is passed into super. |
|  |  |

**Name of Class: Parsley**

|  |  |
| --- | --- |
| **Field or Method** | **Description** |
| Parsley() | The constructor calls the parent constructor via the super keyword and Parsley is passed into super. |
|  |  |

**Name of Class: Vendor**

|  |  |
| --- | --- |
| **Field or Method** | **Description** |
| String name | This field stores the name of any object created from the vendor class. |
| private boolean isAvailable | This field is of type Boolean, and it stores the availability of the vendor to provide goods to the supermarket. |
| private int inventory | This field stores the quantity of items to be sold by the vendor. |
| Vendor( ) | Is the constructor method which initializes the fields within this class to a starting value. |
| public void isNotAvailable( ) | This method sets the availability status of the vendor to false in the event of the inability to deliver goods to the supermarket. |
| public boolean getAvailability() | This method returns the value of the isAvailable field to say whether the vendor is available or not. |
| public sellStock( ) | This method returns the requested amount of goods to the supermarket as a sale of items. |
| Public void restock | This method regenerates new stock for the vendor at the end of operations for the next cycle. |

**Name of Class: FruitsAreHere**

|  |  |
| --- | --- |
| **Field or Method** | **Description** |
| private Watermelon[ ] watermelonInventory | This field is an array of type Watermelon that stores the total amount of watermelons that the vendor has at a given point. |
| private Mango[ ] mangoInventory | This field is an array of type Mango that stores the total mangoes that the vendor has available to sell at any given point. |
| private Banana[ ] bananaInventory | This field is an array of type Banana that stores the total bananas that the vendor has available to sell at any given point. |
| private Lime[ ] limeInventory | This field is an array of type Lime that stores the total limes that the vendor has available at any given point. |
| private Avocado[ ] avocadoInventory | This field is an array of type Avocado that stores the total number of avocado that the vendor has available. |
| private int totalWatermelons | An integer field that stores the total watermelons sold. |
| private int totalMangoes | An integer field that stores the total mangoes sold. |
| private int totalBananas | An integer field that stores the total bananas sold. |
| private int totalLimes | An integer field that stores the total limes sold. |
| private int totalAvocado | An integer field that stores the total avocados sold |
| private double watermelonProfit | A double field that stores the total revenue generated from watermelon sales. |
| private double mangoProfit | A double field that stores the total revenue generated from mango sales. |
| private double bananaProfit | A double field that stores the total revenue generated from banana sales. |
| private double limeProfit | A double field that stores the total revenue generated from lime sales. |
| private double avocadoProfit | A double field that stores the total revenue generated from avocado sales. |
| FruitsAreHere() | Class constructor. Sets all fields to a predetermined starting value. |
| private void initArray(Fruit[] inventory) | This method accepts a parameter array of type fruit which it uses to generate arrays objects of the respective fruit items inventory. This method returns no values. |
| public Watermelon[] sellWatermelons (int request) | This method accepts a request amount from the vendors’ customer of type integer. This value is checked against the watermelon inventory and if there is not enough inventory to complete the request then the customer is sold the total inventory. If there is sufficient then the customer is sold their request amount and the remainder is saved for the next transaction. The customer is returned the respective number of items of type Watermelon. |
| public Avocado[] sellAvocados(int request) | This method accepts a request amount from the vendors’ customer of type integer. This value is checked against the Avocado inventory and if there is not enough inventory to complete the request then the customer is sold the total inventory. If there is sufficient then the customer is sold their request amount and the remainder is saved for the next transaction. The customer is returned the respective number of items of type Avocado. |
| public Lime[] sellLimes(int request) | This method accepts a request amount from the vendors’ customer of type integer. This value is checked against the Lime inventory and if there is not enough inventory to complete the request then the customer is sold the total inventory. If there is sufficient then the customer is sold their request amount and the remainder is saved for the next transaction. The customer is returned the respective number of items of type Lime. |
| public Banana[] sellBananas(int request) | This method accepts a request amount from the vendors’ customer of type integer. This value is checked against the Banana inventory and if there is not enough inventory to complete the request then the customer is sold the total inventory. If there is sufficient then the customer is sold their request amount and the remainder is saved for the next transaction. The customer is returned the respective number of items of type Banana. |
| public Mango[] sellMangoes(int request) | This method accepts a request amount from the vendors’ customer of type integer. This value is checked against the Mango inventory and if there is not enough inventory to complete the request then the customer is sold the total inventory. If there is sufficient then the customer is sold their request amount and the remainder is saved for the next transaction. The customer is returned the respective number of items of type Mango. |
| public void restock() | This method restocks the inventory of the different fruit types given a limit and changes the availability status when it is complete. |
| public void calculateProfit() | This method calculates the total individual items profit as well as the overall profits of the vendor based on sales generated. |
| public double getProfit() | This method returns the overall profit of the vendor. |

**Name of Class: TastyVegetables**

|  |  |
| --- | --- |
| **Field or Method** | **Description** |
| private Carrot[ ] carrotInventory | This field is an array of type Carrots that stores the total amount of carrot that the vendor has at a given point. |
| Private Lettuce[ ] lettuceInventory | This field is an array of type Lettuce that stores the total lettuces that the vendor has available to sell at any given point. |
| private Cucumber[ ] cucumberInventory | This field is an array of type Cucumber that stores the total cucumbers that the vendor has available to sell at any given point. |
| private Parsley[ ] parsleyInventory | This field is an array of type Parsley that stores the total parsleys that the vendor has available at any given point. |
| private Onion[ ] onionInventory | This field is an array of type Onion that stores the total number of onions that the vendor has available. |
| private int totalCarrots | An integer field that stores the total carrots sold. |
| private int totalLettuces | An integer field that stores the total lettuces sold. |
| private int totalCucumbes | An integer field that stores the total cucumbers sold. |
| private int totalParlseys | An integer field that stores the total parsleys sold. |
| private int totalOnion | An integer field that stores the total onions sold |
| private double carrotProfit | A double field that stores the total revenue generated from carrot sales. |
| private double lettuceProfit | A double field that stores the total revenue generated from lettuce sales. |
| private double cucumberProfit | A double field that stores the total revenue generated from cucumber sales. |
| private double parsleyProfit | A double field that stores the total revenue generated from parsley sales. |
| private double onionProfit | A double field that stores the total revenue generated from onion sales. |
| TastyVegetables() | Class constructor. Sets all fields to a predetermined starting value. |
| private void initArray(Vegetable[] inventory) | This method accepts a parameter array of type vegetable which it uses to generate arrays objects of the respective vegetable items inventory. This method returns no values. |
| public Carrot[] sellCarrots (int request) | This method accepts a request amount from the vendors’ customer of type integer. This value is checked against the carrot inventory and if there is not enough inventory to complete the request then the customer is sold the total inventory. If there is sufficient then the customer is sold their request amount and the remainder is saved for the next transaction. The customer is returned the respective number of items of type Carrot. |
| public Lettuce[] sellLettuces(int request) | This method accepts a request amount from the vendors’ customer of type integer. This value is checked against the lettuce inventory and if there is not enough inventory to complete the request then the customer is sold the total inventory. If there is sufficient then the customer is sold their request amount and the remainder is saved for the next transaction. The customer is returned the respective number of items of type Lettuce. |
| public Cucumber[] sellCucumbers(int request) | This method accepts a request amount from the vendors’ customer of type integer. This value is checked against the cucumber inventory and if there is not enough inventory to complete the request then the customer is sold the total inventory. If there is sufficient then the customer is sold their request amount and the remainder is saved for the next transaction. The customer is returned the respective number of items of type Cucumber. |
| public Parsley[] sellParsleys(int request) | This method accepts a request amount from the vendors’ customer of type integer. This value is checked against the parsley inventory and if there is not enough inventory to complete the request then the customer is sold the total inventory. If there is sufficient then the customer is sold their request amount and the remainder is saved for the next transaction. The customer is returned the respective number of items of type Parsley. |
| public Onion[] sellOnions(int request) | This method accepts a request amount from the vendors’ customer of type integer. This value is checked against the onion inventory and if there is not enough inventory to complete the request then the customer is sold the total inventory. If there is sufficient then the customer is sold their request amount and the remainder is saved for the next transaction. The customer is returned the respective number of items of type Onion. |
| public void restock() | This method restocks the inventory of the different vegetable types given a limit and changes the availability status when it is complete. |
| public void calculateProfit() | This method calculates the total individual items profit as well as the overall profits of the vendor based on sales generated. |
| public double getProfit() | This method returns the overall profit of the vendor. |

**Name of Class: AllYouCanEat**

|  |  |
| --- | --- |
| **Field or Method** | **Description** |
| private Watermelon[ ] watermelonInventory | This field is an array of type Watermelon that stores the total amount of watermelons that the vendor has at a given point. |
| private Mango[ ] mangoInventory | This field is an array of type Mango that stores the total mangoes that the vendor has available to sell at any given point. |
| private Banana[ ] bananaInventory | This field is an array of type Banana that stores the total bananas that the vendor has available to sell at any given point. |
| private Lime[ ] limeInventory | This field is an array of type Lime that stores the total limes that the vendor has available at any given point. |
| private Avocado[ ] avocadoInventory | This field is an array of type Avocado that stores the total number of avocado that the vendor has available. |
| private int totalWatermelons | An integer field that stores the total watermelons sold. |
| private int totalMangoes | An integer field that stores the total mangoes sold. |
| private int totalBananas | An integer field that stores the total bananas sold. |
| private int totalLimes | An integer field that stores the total limes sold. |
| private int totalAvocado | An integer field that stores the total avocados sold |
| private double watermelonProfit | A double field that stores the total revenue generated from watermelon sales. |
| private double mangoProfit | A double field that stores the total revenue generated from mango sales. |
| private double bananaProfit | A double field that stores the total revenue generated from banana sales. |
| private double limeProfit | A double field that stores the total revenue generated from lime sales. |
| private double avocadoProfit | A double field that stores the total revenue generated from avocado sales. |
| private Carrot[ ] carrotInventory | This field is an array of type Carrots that stores the total amount of carrot that the vendor has at a given point. |
| Private Lettuce[ ] lettuceInventory | This field is an array of type Lettuce that stores the total lettuces that the vendor has available to sell at any given point. |
| private Cucumber[ ] cucumberInventory | This field is an array of type Cucumber that stores the total cucumbers that the vendor has available to sell at any given point. |
| private Parsley[ ] parsleyInventory | This field is an array of type Parsley that stores the total parsleys that the vendor has available at any given point. |
| private Onion[ ] onionInventory | This field is an array of type Onion that stores the total number of onions that the vendor has available. |
| private int totalCarrots | An integer field that stores the total carrots sold. |
| private int totalLettuces | An integer field that stores the total lettuces sold. |
| private int totalCucumbes | An integer field that stores the total cucumbers sold. |
| private int totalParlseys | An integer field that stores the total parsleys sold. |
| private int totalOnion | An integer field that stores the total onions sold |
| private double carrotProfit | A double field that stores the total revenue generated from carrot sales. |
| private double lettuceProfit | A double field that stores the total revenue generated from lettuce sales. |
| private double cucumberProfit | A double field that stores the total revenue generated from cucumber sales. |
| private double parsleyProfit | A double field that stores the total revenue generated from parsley sales. |
| private double onionProfit | A double field that stores the total revenue generated from onion sales. |
| AllYouCanEat() | Class constructor which sets all the fields to a starting value. |
| private void initArray(Vegetable[] inventory) | This method accepts a parameter array of type vegetable which it uses to generate arrays objects of the respective vegetable items inventory. This method returns no values. |
| public Carrot[] sellCarrots (int request) | This method accepts a request amount from the vendors’ customer of type integer. This value is checked against the carrot inventory and if there is not enough inventory to complete the request then the customer is sold the total inventory. If there is sufficient then the customer is sold their request amount and the remainder is saved for the next transaction. The customer is returned the respective number of items of type Carrot. |
| public Lettuce[] sellLettuces(int request) | This method accepts a request amount from the vendors’ customer of type integer. This value is checked against the lettuce inventory and if there is not enough inventory to complete the request then the customer is sold the total inventory. If there is sufficient then the customer is sold their request amount and the remainder is saved for the next transaction. The customer is returned the respective number of items of type Lettuce. |
| public Cucumber[] sellCucumbers(int request) | This method accepts a request amount from the vendors’ customer of type integer. This value is checked against the cucumber inventory and if there is not enough inventory to complete the request then the customer is sold the total inventory. If there is sufficient then the customer is sold their request amount and the remainder is saved for the next transaction. The customer is returned the respective number of items of type Cucumber. |
| public Parsley[] sellParsleys(int request) | This method accepts a request amount from the vendors’ customer of type integer. This value is checked against the parsley inventory and if there is not enough inventory to complete the request then the customer is sold the total inventory. If there is sufficient then the customer is sold their request amount and the remainder is saved for the next transaction. The customer is returned the respective number of items of type Parsley. |
| public Onion[] sellOnions(int request) | This method accepts a request amount from the vendors’ customer of type integer. This value is checked against the onion inventory and if there is not enough inventory to complete the request then the customer is sold the total inventory. If there is sufficient then the customer is sold their request amount and the remainder is saved for the next transaction. The customer is returned the respective number of items of type Onion. |
| private void initArray(Fruit[] inventory) | This method accepts a parameter array of type fruit which it uses to generate arrays objects of the respective fruit items inventory. This method returns no values. |
| public Watermelon[] sellWatermelons (int request) | This method accepts a request amount from the vendors’ customer of type integer. This value is checked against the watermelon inventory and if there is not enough inventory to complete the request then the customer is sold the total inventory. If there is sufficient then the customer is sold their request amount and the remainder is saved for the next transaction. The customer is returned the respective number of items of type Watermelon. |
| public Avocado[] sellAvocados(int request) | This method accepts a request amount from the vendors’ customer of type integer. This value is checked against the Avocado inventory and if there is not enough inventory to complete the request then the customer is sold the total inventory. If there is sufficient then the customer is sold their request amount and the remainder is saved for the next transaction. The customer is returned the respective number of items of type Avocado. |
| public Lime[] sellLimes(int request) | This method accepts a request amount from the vendors’ customer of type integer. This value is checked against the Lime inventory and if there is not enough inventory to complete the request then the customer is sold the total inventory. If there is sufficient then the customer is sold their request amount and the remainder is saved for the next transaction. The customer is returned the respective number of items of type Lime. |
| public Banana[] sellBananas(int request) | This method accepts a request amount from the vendors’ customer of type integer. This value is checked against the Banana inventory and if there is not enough inventory to complete the request then the customer is sold the total inventory. If there is sufficient then the customer is sold their request amount and the remainder is saved for the next transaction. The customer is returned the respective number of items of type Banana. |
| public Mango[] sellMangoes(int request) | This method accepts a request amount from the vendors’ customer of type integer. This value is checked against the Mango inventory and if there is not enough inventory to complete the request then the customer is sold the total inventory. If there is sufficient then the customer is sold their request amount and the remainder is saved for the next transaction. The customer is returned the respective number of items of type Mango. |
| private void initArray(Vegetable[] inventory) | This method accepts a parameter array of type vegetable which it uses to generate arrays objects of the respective vegetable items inventory. This method returns no values. |
| public Carrot[] sellCarrots (int request) | This method accepts a request amount from the vendors’ customer of type integer. This value is checked against the carrot inventory and if there is not enough inventory to complete the request then the customer is sold the total inventory. If there is sufficient then the customer is sold their request amount and the remainder is saved for the next transaction. The customer is returned the respective number of items of type Carrot. |
| public Lettuce[] sellLettuces(int request) | This method accepts a request amount from the vendors’ customer of type integer. This value is checked against the lettuce inventory and if there is not enough inventory to complete the request then the customer is sold the total inventory. If there is sufficient then the customer is sold their request amount and the remainder is saved for the next transaction. The customer is returned the respective number of items of type Lettuce. |
| public Cucumber[] sellCucumbers(int request) | This method accepts a request amount from the vendors’ customer of type integer. This value is checked against the cucumber inventory and if there is not enough inventory to complete the request then the customer is sold the total inventory. If there is sufficient then the customer is sold their request amount and the remainder is saved for the next transaction. The customer is returned the respective number of items of type Cucumber. |
| public Parsley[] sellParsleys(int request) | This method accepts a request amount from the vendors’ customer of type integer. This value is checked against the parsley inventory and if there is not enough inventory to complete the request then the customer is sold the total inventory. If there is sufficient then the customer is sold their request amount and the remainder is saved for the next transaction. The customer is returned the respective number of items of type Parsley. |
| public Onion[] sellOnions(int request) | This method accepts a request amount from the vendors’ customer of type integer. This value is checked against the onion inventory and if there is not enough inventory to complete the request then the customer is sold the total inventory. If there is sufficient then the customer is sold their request amount and the remainder is saved for the next transaction. The customer is returned the respective number of items of type Onion. |
| public void restock() | This method restocks the inventory of the different vegetable types given a limit and changes the availability status when it is complete. |
| public void calculateProfit() | This method calculates the total individual items profit as well as the overall profits of the vendor based on sales generated. |
| public double getProfit() | This method returns the overall profit of the vendor. |

**Supermarket**

|  |  |
| --- | --- |
| **Field or Method** | **Description** |
| DEFAULT\_TOTAL\_ITERATIONS | defines the amount of total iterations that the Supermarket must undergo if no iteration are passed at the command line and is the default amount |
| MAXIMUM\_NUMBER\_OF\_FRUITS | defines the maximum of amount of fruits that the Supermarket maintains |
| MAXIMUM\_NUMBER\_OF\_VEGETABLES | defines the maximum of amount of vegtables that the Supermarket maintains |
| NUMBER\_OF\_TYPES\_OF\_ITEMS | defines the number of different items that the Supermarket has |
| NUMBER\_OF\_CUSTOMERS | defines the number of customers that the Supermarket has |
| verbose | defines the modes in which to run the Supermarket |
| totalIterations | defines the total amount of iterations that the Supermarket must undergoes |
| totalPerformedIterations | holds the number of iterations that were carried out |
| profit | holds the profit that Supermarket made from the sale and purchasing of items, loss of spoilt items and random events |
| lossesCausedbyRandomEvent | holds the losses that the random event caused |
| cashOnHand | defines the starting amount that the Supermarket initially has on hand |
| fruitsSold | holds the total of each fruit the customer buys per cycle |
| vegetablesSold | holds the total of each fruit the customer buys per cycle |
| itemsPurchased | holds the total of each item the Supermarket buys per cycle |
| numberOfSpoiltItems | holds the total of each item that has spoilt per cycle |
| customers | holds the customers that the Supermarket has |
| tastyVegetables | Vegetable vendor |
| allYouCanEat | Fruits and vegetable vendor |
| fruitsAreHere | Fruits vendor |
| avocadoInventory | hold all the avocados the Supermarket has at any time |
| bananaInventory | hold all the bananas the Supermarket has at any time |
| carrotInventory | hold all the carrots the Supermarket has at any time |
| cucumberInventory | hold all the cucumbers the Supermarket has at any time |
| lettuceInventory | hold all the lettuces the Supermarket has at any time |
| limeInventory | hold all the limes the Supermarket has at any time |
| mangoInventory | hold all the mangoes the Supermarket has at any time |
| onionInventory | hold all the onions the Supermarket has at any time |
| parsleyInventory | hold all the parsleys the Supermarket has at any time |
| watermelonInventory | hold all the watermelons the Supermarket has at any time |
| run() | The run method the entry to the start of the actual Supermarket Simulation and runs the total amount of iterations specified by the totalIterations field the display the total amount of cycles that were carried out, the profit the Supermarket made and the profit the vendors made in total. The function takes no parameters and returns no vaule. It is public so the SimulationMain class can call it. |
| turnOnVerbose | This function turns on the verbose mode for the run method. It take no parameters and have to return values. |
| initCustomers | This function initialises each index in the customers array to a new instance  \* of Customer. There are no parameters and return no values. |
| decreaseInventorySpoiltValue | Decreases all the spoilt values of the items in the supermarket |
| randomEvent | Determines an execute a random event |
| spoilFruits | Spoil the fruits for the random event |
| spoilVegetables | Spoil the vegetables for the random event |
| spoilFruitsFaster | Spoil the fruits for the random event faster |
| spoilVegetablesFaster | Spoil the vegetables for the random event faster |
| sellAvocados | Sell the items to customers |
| sellBananas | Sell the items to customers |
| sellCarrots | Sell the items to customers |
| sellCucumbers | Sell the items to customers |
| sellLettuces | Sell the items to customers |
| sellLimes | Sell the items to customers |
| sellMangoes | Sell the items to customers |
| sellWatermelons | Sell the items to customers |
| buyLimes | Buy items from vendors |
| calculateProfit | Calculate the profit for each cycle |
| setTotalIterations | Sets the totalIterations to the amount passed in a the command line |
| reset | This function reset the counter fields of the Supermarket. |
| sellOnions | Sell the items to customers |
| sellParsleys | Sell the items to customers |
| buyCarrots | Buy items from vendors |
| buyCucumbers | Buy items from vendors |
| buyLettuces | Buy items from vendors |
| buyWatermelons | Buy items from vendors |
| buyParsleys | Buy items from vendors |
| buyOnions | Buy items from vendors |
| buyMangoes | Buy items from vendors |
| removeSpoiltAvocados | Remove the spoilt items from inventory |
| removeSpoiltBananas | Remove the spoilt items from inventory |
| removeSpoiltCarrots | Remove the spoilt items from inventory |
| removeSpoiltCucumbers | Remove the spoilt items from inventory |
| removeSpoiltLettuces | Remove the spoilt items from inventory |
| removeSpoiltLimes | Remove the spoilt items from inventory |
| removeSpoiltMangoes | Remove the spoilt items from inventory |
| removeSpoiltOnions | Remove the spoilt items from inventory |
| removeSpoiltParsleys | Remove the spoilt items from inventory |
| removeSpoiltWatermelons | Remove the spoilt items from inventory |
| buyAvocados | Buy items from vendors |
| buyBananas | Buy items from vendors |

**Output**

|  |  |
| --- | --- |
| **Field or Method** | **Description** |
| fileLocation | stores the filepath the is passed at the command line or keep the default setting |
| setFileLocation | set the filepath the is passed at the command line |
| getFileLocation | return the filepath in fileLocation |
| createLogFile | Create a log file before the simulation starts |
| appendCustomerPurchaseFruitsToLogFile | Adds customers fruits purchase to the log file |
| appendCustomerPurchaseVegetablesToLogFile | Adds customers vegetable purchase to the log file |
| appendToLogFileAmountOfFruitsPurchase | Adds amounts of fruits bought to the log file |
| appendToLogFile | Adds a string to the log file |
| appendToLogFileAmountOfVegetablesPurchase | Adds amounts of vegetables bought to the log file |

**Print**

|  |  |
| --- | --- |
| **Field or Method** | **Description** |
| verboseSpoiltFruit | Display a general fruit spoilt format for verbose |
| verboseSpoiltVegetable | Display a general fruit spoilt format for verbose |
| verboseTotalAmountCustomerPurchase | Display a general customer purchase format for verbose |
| verboseTotalItemsPurchased | Display a general supermarket purchase format for verbose |

**Customer**

|  |  |
| --- | --- |
| **Field or Method** | **Description** |
| willPurchase | The state of the customer for the random events |
| itemsToBePurchased | Determines the amount of items to be purchased |
| typeOfItem | Determines the type of item to be purchased |
| willPurchase() | Determines if the customer will purchase |
| willNotPurchase | Tell the customer to not purchase for the random event |
| reset | Set the willPurchase to true at the end of the day |

**SimulationMain**

|  |  |
| --- | --- |
| **Field or Method** | **Description** |
| main | The entry point to the supermarket simulation and process any command arguments |
| commandsPassed | Stores all the commands passed to then be displayed later |

# Constants

All constants that were created.

|  |  |
| --- | --- |
| **Name of Constant** | **Description** |
| SELLING\_PRICE | The SELLING\_PRICE is the price that the supermarket sells its items to the customers. |
| COST\_PRICE | The COST\_PRICE is the cost of buying items from the vendors. |
| DEFAULT\_TOTAL\_ITERATIONS | The constant is the number of total iterations that the supermarket must undergo |
| MAXIMUM\_NUMBER\_OF\_FRUITS | The constant is the maximum number of fruits the supermarket can maintain |
| MAXIMUM\_NUMBER\_OF\_VEGETABLES | The constant is the maximum number if vegetables the supermarket can maintain |
| NUMBER\_OF\_TYPES\_OF\_ITEMS | The constant is the number of customers the supermarket has |
| NUMBER\_OF\_CUSTOMERS | The constant represents the number customers the supermarket has |
|  |  |
|  |  |
|  |  |

# Assumptions

The customer has an infinite amount of money to spend.

The vendor’s profit is assumed to be its revenue.

The buyAvocados method in the Supermarket class has the same functionality as the over buy methods the type is different as so the comments for the buyAvocados is the same for each