

# Vending Machine Design Technical Document

<b>Pre Reflection:</b> .....	<b>1</b>
<b>Steps to approach this assignment:</b> .....	<b>2</b>
<b>User Stories and Acceptance Criteria</b> .....	<b>2</b>
User Story 1.....	2
Tasks.....	2
Add Car method.....	2
addCar() pseudocode.....	3
Read From File Setup method.....	3
Create Car Class.....	3
Create Vending Machine Class.....	4
User Story 2.....	4
Tasks.....	4
Display Tower method.....	4
displayTower() pseudocode.....	5
User Story 3.....	5
Tasks.....	6
Tower Converter method.....	6
Array Sorter method - by Price.....	6
Array Sorter Method - by Year.....	6
displayByAttribute() pseudocode.....	7
User Story 4.....	7
Tasks.....	7
Retrieve Car method.....	7
retrieveCar() pseudocode.....	8
User Story 5.....	8
Tasks.....	9
Display Menu method.....	9
Menu Actions Method.....	9
<b>UML Designs</b> .....	<b>10</b>
Vending Machine Class.....	10
Car Class.....	10
<b>Driver Class Design</b> .....	<b>11</b>
Main method.....	11
Static methods.....	11

## Pre Reflection:

*I have some ideas for how I would go about the different stories. It always feels a little daunting when I'm not sure about everything.*

## Steps to approach this assignment:

1. Read over everything
2. Break into multiple tasks/parts
3. Make test cases to drive development
4. In each part, design through pseudocode then implement (saves a lot of time personally)
5. Test code throughout the way

## User Stories and Acceptance Criteria

### User Story 1

As a dealership owner, I want to store cars in a vending machine at their designated positions based on reading a file so we can keep track of the cars in the tower.

- Given a vending machine with defined rows and columns entered by the dealer,
- When file is read containing car details,
- Then the cars are added into the vending machine at their designated positions.
  - Cars are only placed in valid positions within the vending machine.
  - A car cannot be placed in an already occupied slot, preventing overwrites.
  - Out-of-bounds positions are not allowed.

### Tasks

#### Add Car method

pre	post	notes
Add car at location (3,2): empty	Adds car to location (3,2)	If location is null, can store car reference in location
Add car at location (3,2): occupied	Print "cannot add car. Location already occupied"	
Add car at location (10,2) Tower.length = 5 Tower[0].length = 5	Print "cannot add car. Location out of bounds"	

## addCar() pseudocode

```

addCar()

addCar (int floor, int space, Car car)
try
    if (tower[floor][space] != null)
        print "Error! occupied at (floor+1), (space+1)"
        print "Car car cannot be added"
    else
        tower[floor][space] = car
catch (ArrayIndexOutOfBoundsException e)
    print "Error: Invalid location at floor(floor+1) space(space+1)"
    print "cannot place car car"

if floor

```

## Read From File Setup method

precondition	Postcondition	notes
Input correct filename: cars1.txt  File contents: 0 1 2018 24000.00 Toyota Corolla 1 2 2016 28000.00 Honda Accord	scanner.inputNext sequence collects data  Create car with data Try addcar() method	Normal scenario
Input INCORRECT file name: car1223.hi	Print "File does not exist. Please check file path"	FileNotFoundException

## Create Car Class

See [here](#) for Car class UML design

## Create Vending Machine Class

See [here](#) for Vending Machine class UML design

### User Story 2

As a dealership employee, I want to view the location of all the cars in the vending machine so I can show a customer.

- Given a vending machine containing cars,
- When I request an inventory location report,
- Then the cars are printed
  - Display if empty if no car found at location.

### Tasks

Display Tower method

Pre condition	Post condition	notes
Input tower array that has all spots filled	<p>Print: Print all cars and attributes in tower</p> <p>Example: Floor 1</p> <ul style="list-style-type: none"><li>- Space 1: Toyota Corolla 2018 - \$24000.0</li><li>- Space 2: Toyota Prius 2017 - \$28000.0</li></ul> <p>Floor 2</p> <ul style="list-style-type: none"><li>- Space 1: Honda Accord 2016 - \$28000.0</li><li>- Space 2: Ford Mustang 2019 - \$26000.0</li></ul>	Prints car attributes if the car is there
Input tower array with some elements pointing to null (not entirely filled tower)	<p>Elements pointing to null will print "EMPTY" at their spot</p> <p>Example: Floor 1</p> <ul style="list-style-type: none"><li>- Space 1: Toyota Corolla 2018 - \$24000.0</li></ul>	Prints EMPTY if there is no car at that spot in tower

	<ul style="list-style-type: none"> <li>- Space 2: EMPTY</li> </ul>	
	Floor2 <ul style="list-style-type: none"> <li>- Space 1: EMPTY</li> <li>- Space 2: EMPTY</li> </ul>	

displayTower() pseudocode

```

catch (InvalidLocationException)
    print "Invalid location! Does not exist!"

public void displayTower()
    print "Inventory location"
    → for (int i = 0, i < tower.length; i++)
        print "floor (i+1)"
        → for (int j = 0, j < tower[i].length; j++)
            print "Space (j+1):"
            → if tower[i][j] exist
                print car info.
            else
                print "empty"
  
```

### User Story 3

As a dealership employee, I want to view an inventory report sorted by car price or by year so that I can easily identify the cars.

- Given a vending machine containing cars,
- When I request an inventory report,
- Then the cars are sorted and displayed based on my selection of "price" or "year" from lowest to highest.

## Tasks

### Tower Converter method

precondition	Postcondition	notes
Input 2D tower array with 4 rows, 3 columns	New array initialized with length: 12	Multiply row and column for the length of new array

### Array Sorter method - by Price

precondition	Postcondition	notes
{20 000, 19 000, 18 000}	{18 000, 19 000, 20 000}	Regular array can use normal selection sort
{19 000, 21 000, null}	{19 000, 21 000, null}	Already sorted + null
{null, 20 000, 18 000}	{null, 18 000, 20 000}	Reverse sorted + null
{null, null, 20 000, 18 000}	{null, null, 18 000, 20 000}	Duplicate nulls

### Array Sorter Method - by Year

precondition	Postcondition	notes
{1990, 2000, 2001, 2009}	{1990, 2000, 2001, 2009}	Regular array
{2001, null, 2000, 1990}	{1990, null, 2001, 2009}	Reverse sorted + null
{null, 1990, 2001, 2009}	{null, 2001, 2001, 2009}	Already sorted + null
{null, null, 2000, 1990}	{null, null, 1990, 2000}	Duplicate nulls



displayByAttribute() pseudocode

```
public void displayByAttribute(String attribute)
    car[] list = towerConverter(tower)
    car[] sortedList = InventorySorter(list, attribute)
    car[][] sortedTower = towerConverter(sortedList, row column)
    displayInventory(sortedTower)
```

## User Story 4

As a dealership employee, I want retrieve cars from a location for clients to test drive

- Given a vending machine containing cars,
- When I request a car by floor and space,
- Then the car is retrieved
  - Display if no car found at location.
  - Display car retrieved and the details if found

## Tasks

Retrieve Car method

precondition	Postcondition	notes
--------------	---------------	-------

Input floor and space for spot in array car exists at that spot	Print "Car retrieved from floor (floor+1) space (space+1)" "Car ____ (car attributes) ____"	
Spot points to null	Print "No car located at floor (floor+1) space (space+1)"	
Spot is outside of array	Print "Invalid location! floor (floor+1) space (space+1) does not exist!"	ArrayOutOfBoundsException

retrieveCar() pseudocode

Sort

~~retrieve~~

```

public void retrieveCar (int floor, int location)
try
    Car possibleCar = getCar (tower[floor] location)
    if (possibleCar != null)
        print "Car retrieved from floor (floor) location (location)"
    else
        print "no car located at this location"
catch ArrayIndexOutOfBoundsException
    print "Invalid location! Does not exist!"
  
```

## User Story 5

As a dealership employee, I want a menu driven system to select an action.

- Given a menu list,



- When I input a number,
- That action is completed

## Tasks

### Display Menu method

=== Car Vending Machine Menu ===

1. Load Car Data
2. Display Vending Machine
3. Retrieve a Car
4. Print Sorted Inventory (Price)
5. Print Sorted Inventory (Year)
6. Exit

### Menu Actions Method

precondition	Postcondition	notes
User input choice 1	Call readFromFileSetup method	Load Car Data
User input choice 2	Prints all slots in vending machine <ul style="list-style-type: none"> <li>- If car exists at slot, prints car attributes</li> <li>- If no car exists at slot, prints "EMPTY"</li> </ul>	Display Vending Machine
User input choice 3	Calls retrieveCar method	Retrieve car
User input choice 4	Calls towerConverter method Calls arraySorter method  Prints sorted array <ul style="list-style-type: none"> <li>- If car exists at index, print car attributes</li> <li>- If no car exists at index, do not print</li> </ul>	Print Sorted Inventory (Price)
User input choice 5	Calls towerConverter method Calls arraySorter method  Prints sorted array	Print Sorted Inventory (Year)

	<ul style="list-style-type: none"> <li>- If car exists at index, print car attributes</li> <li>- If no car exists at index, do not print</li> </ul>	
User input choice 6	Exit. stop Menu loop  print "Exiting Program. Goodbye!"	Exit program
User input choice 7	Print "Invalid choice! Please select a valid choice"	Handles invalid user input

## UML Designs

### Vending Machine Class

VM Class
<ul style="list-style-type: none"> <li>- tower[][]: Car</li> </ul>
<ul style="list-style-type: none"> <li>+ VM(int floors, int spaces)</li> <li>+ getTower(): Car[]</li> <li>+ getCar(): Car</li> <li>+ addCar(): void</li> <li>+ displayByAttribute(String): void</li> <li>+ retrieveCar(int, int): Car</li> <li>+ inventorySorter(Car[], String): array[]</li> <li>- towerConverter(Car[][]): Car[]</li> <li>- towerConverter(Car[], int, int): Car[][]</li> <li>+ displayInventory(Car[][]): void</li> </ul>

### Car Class

Car Class
<ul style="list-style-type: none"> <li>- Make: String</li> <li>- Model: String</li> <li>- Price: double</li> <li>- Year: int</li> </ul>
<ul style="list-style-type: none"> <li>+ car(String make, String model, double price, int year)</li> <li>+ getMake(): String</li> </ul>

<ul style="list-style-type: none"><li>+ getModel(): String</li><li>+ getYear(): int</li><li>+ getPrice(): double</li><li>+ toString(): String</li></ul>
---

## Driver Class Design

### Main method

- Ask user input for dimensions for 2D tower array
- Create car vending machine based on user input
- Start do-while loop to display menu options and perform menu actions depending on choice
  - Load car data
    - Reads cars from a file and inserts them in created tower array
  - Display Vending Machine
  - Retrieve a Car
  - Print Sorted Inventory (Price)
  - Print Sorted Inventory (Year)
  - Exit
- Continues looping menu options until Exit is chosen
- If Exit is chosen, menu program stops looping and program ends

### Static methods

- displayMenu: Displays all menu options
- readFromFileSetup: used to read cars in from a file and add to an existing tower array