Carvana Car Vending Machine Project Iteration 01

**Points:** 30 (see rubric in canvas) **Due Date:**  See Canvas.

Work will be accepted up to 24 hours after the due date with a 10% penalty. Meaning if you turn it in at 12:01 am of the next day you will be deducted 10% of the total points from your score. **If the assignment is more than 24 hours late, it will be a 0.**

Objectives

* Design modular programs by analyzing and decomposing user stories into actionable, testable tasks.
* Use test-driven development (TDD) to create and validate test cases prior to implementation.
* Employ testing and debugging techniques to maintain program quality.
* Implement solutions iteratively, ensuring correctness and functionality.
* Create quality code that is readable and appropriately documented (naming conventions, indentation, comments)
* Implement files, 2D arrays, sorting and object-oriented concepts
* Evaluate and identify sorting algorithms

**Effort: Collaborative and Individual following** [CS Academic Integrity and AI Policy - Harding](https://docs.google.com/document/d/1SY4-RMJ2B9GkEbTzmpRVSfKKsgQnYWYF/edit)

* You are encouraged to collaborate to discuss concepts but individually write **your own design, code and answer the questions in your own words** . When you ask questions and explain to others you get a deeper understanding. You can use AI tools but remember
  + ***Acceptable:*** Using ChatGPT to explain code or concepts
  + ***Unacceptable:*** Asking ChatGPT to generate answers to the learning and reflection questions

Deliverables: Upload each file separately and not as a zip file

* Well Organized Technical Design Document
* One java file containing the code solution.
* Learnings and reflection questions document

[Overview of Car Vending Machines](#_k2ibwtlu4tfn)

[Requirements](#_wopyt5so5ehg)

[Sample Output](#_2yuakxlox1nr)

[Specification for Implementation](#_uiu8d7c3qliz)

[Part 1: Software Development](#_xupto6yf6nep)

[Learnings and Reflection](#_etiiwgpookul)

[Rubric](#_fqmez9xrwpm6)

# Overview of Car Vending Machines

What is a Car Vending Machine?

A car vending machine is a real-world automated system that allows customers to purchase cars through a self-service mechanism. The vending machine consists of multiple levels, each containing various cars that customers can browse and purchase. Here is an example, [https://blog.carvana.com/2021/02/vegas-is-home-to-carvanas-newest-car-vending-machine/](https://www.carvana.com/vendingmachine)

In this project, we will model a Carvana Car Vending Machine as a grid structure (2D array in Iteration 01) where each position in the grid represents a parking space holding a car. In Iteration 02, we will improve our implementation by transitioning to a linked list, allowing for more flexibility in adding and removing cars dynamically.

The vending machine contains a tower of cars. To set up the vending machine’s tower the dealer will enter the number of floors and slots. Next, a file will be read to fill the vending machine’s tower with cars. For each car in the file, read its details, create the car, and place the car into the specified location in the vending machine’s tower. Note, not all locations in the vending machine’s tower will contain a car.

After all cars are placed in the Vending Machine, the employees can display the tower, print an inventory report by price or year and can retrieve a car.

## Requirements

**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.

Scenario:

* Given a vending machine with defined rows and columns entered by the dealer,
* When file is read containing car details,
* Then the cars are placed 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.

**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.

**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.

**User Story 4:** As a dealership employee, I want retrieve cars from a location for clients to test drive

Scenario:

* 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

**User Story 5:** As a dealership employee, I want a menu driven system to select an action.

Scenario:

* Given a menu list,
* When I input a number,
* That action is completed

=== 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

## Sample Output

File [cars1.txt](https://drive.google.com/file/d/1ScpIMUopFtDTLPcuHFeYHaVok1Eo7_aH/view?usp=drive_link)

|  |
| --- |
| 0 1 2018 24000.00 Toyota Corolla  1 2 2016 28000.00 Honda Accord  2 3 2019 26000.00 Ford Mustang  3 0 2021 30000.00 Hyundai Ioniq5  4 2 2017 20000.00 Nissan Altima |

File [cars2.txt](https://drive.google.com/file/d/1q6MuV_uK_G4skW2T5TNtYtZz6628vX8u/view?usp=drive_link)

|  |
| --- |
| 0 2 2017 28000.00 Toyota Prius  1 2 2005 8000.00 Toyota Camry  3 4 2022 53000.00 Chevrolet Traverse  2 4 2022 26000.00 Ford F150 |

|  |  |
| --- | --- |
| Sample Output | Notes |
| Enter the number of floors for the car vending machine: 4  Enter the number of spaces for the car vending machine: 5  === 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  Enter your choice: 1  Enter the file name: cars1.txt  Error: Invalid position at Floor: 5 Space: 3  Can not place Car Nissan Altima 2017 - $20000.0  === 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  Enter your choice: 2  Inventory Location  Floor 1:  Space 1 EMPTY  Space 2: Toyota Corolla 2018 - $24000.0  Space 3 EMPTY  Space 4 EMPTY  Space 5 EMPTY  Floor 2:  Space 1 EMPTY  Space 2 EMPTY  Space 3: Honda Accord 2016 - $28000.0  Space 4 EMPTY  Space 5 EMPTY  Floor 3:  Space 1 EMPTY  Space 2 EMPTY  Space 3 EMPTY  Space 4: Ford Mustang 2019 - $26000.0  Space 5 EMPTY  Floor 4:  Space 1: Hyundai Ioniq5 2021 - $30000.0  Space 2 EMPTY  Space 3 EMPTY  Space 4 EMPTY  Space 5 EMPTY  === 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  Enter your choice: 4  Sorted Inventory by Price:  Toyota Corolla 2018 - $24000.0  Ford Mustang 2019 - $26000.0  Honda Accord 2016 - $28000.0  Hyundai Ioniq5 2021 - $30000.0  === 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  Enter your choice: 1  Enter the file name: cars2.txt  Error: Slot at Floor: 2 Space: 3 is already occupied.  Car Toyota Camry 2005 - $8000.0 cannot be placed.  === 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  Enter your choice: 2  Inventory Location  Floor 1:  Space 1 EMPTY  Space 2: Toyota Corolla 2018 - $24000.0  Space 3: Toyota Prius 2017 - $28000.0  Space 4 EMPTY  Space 5 EMPTY  Floor 2:  Space 1 EMPTY  Space 2 EMPTY  Space 3: Honda Accord 2016 - $28000.0  Space 4 EMPTY  Space 5 EMPTY  Floor 3:  Space 1 EMPTY  Space 2 EMPTY  Space 3 EMPTY  Space 4: Ford Mustang 2019 - $26000.0  Space 5: Ford F150 2022 - $26000.0  Floor 4:  Space 1: Hyundai Ioniq5 2021 - $30000.0  Space 2 EMPTY  Space 3 EMPTY  Space 4 EMPTY  Space 5: Chevrolet Traverse 2022 - $53000.0  === 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  Enter your choice: 3  Enter floor to retrieve car: 3  Enter location to retrieve car: 4  Car retrieved from Floor 3 Location 4: Ford Mustang 2019 - $26000.0  === 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  Enter your choice: 3  Enter floor to retrieve car: 1  Enter location to retrieve car: 1  No car located at Floor 1 Location 1  === 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  Enter your choice: 5  Sorted Inventory by Year:  Honda Accord 2016 - $28000.0  Toyota Prius 2017 - $28000.0  Toyota Corolla 2018 - $24000.0  Ford Mustang 2019 - $26000.0  Hyundai Ioniq5 2021 - $30000.0  Ford F150 2022 - $26000.0  Chevrolet Traverse 2022 - $53000.0  === 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  Enter your choice: 6  Exiting program. Goodbye! |  |

## Specification for Implementation

* Read car data from a file and place cars into the correct floor and space.
  + Manufacturer and Model will contain only one word so you can use next() to read from the file.
* Implement a Car class to store the year, price, manufacturer and model.
* Implement a VendingMachine class that:
  + Stores cars in a 2D array.
  + Provides methods to add cars to tower, display cars sorted by year and price, retrieve a car on a certain floor and at a certain space, and display location of cars in tower.
  + Implements an inventory sorting feature.
* Only need to handle file exceptions. Assume all data entered is valid type.

**Tip 1:** Convert the 2D Array into a 1D Array

Sorting directly within a 2D array is complex because:

* Sorting by rows vs. columns can lead to different behaviors.
* The easiest approach is to flatten the 2D array into a 1D array, sort it, and then place the values back into the 2D structure.

Example Process:

* Extract all cars from the 2D array into a 1D list.
* Sort the 1D array using the selected sorting algorithm.
* Repopulate the sorted data back into the 2D vending machine.
* Choose between Selection Sort, Insertion Sort, or Bubble Sort for sorting the inventory and provide justification.

# Part 1: Software Development

Follow [software development processes](https://docs.google.com/presentation/d/1GNVTjZu4YdGPs6kaXF_DOECYMwzhDGxYufdeTUN9XYk/edit#slide=id.p1) and create a separate technical design document containing parts 1 to 6 below.

1. Analyze the user stories and acceptance criteria.
2. Break the problem into classes and methods
3. UML Design for the classes attributes, constructors and methods
4. Design driver class containing main method and method other needed static methods
5. Test Driven Development where you develop varied unit tests with edge cases and what can go wrong.
   1. Preconditions: Initial state before the method runs.
   2. Postconditions: Expected output after execution.
6. Develop pseudocode for algorithms on paper, tablet or whiteboard for the following methods
   1. Adding car to tower
   2. Retrieving car from tower
   3. Displaying car tower
   4. Print Sorted cars in ascending order by price or by year
7. Implement a little bit of code and unit test until all parts are implemented.
   1. Implement Quality Code such as [Generate JavaDoc Comments](https://codingtechroom.com/question/how-to-generate-javadoc-comments-for-existing-code-in-eclipse) for methods and classes and then edit
   2. Version and back up your code

Example Unit Testing for Adding a car

|  |  |  |
| --- | --- | --- |
| **Preconditions** | **Expected Output** | **Notes** |
| Empty VendingMachine  Add Car at (1,2) | Car stored at (1,2) | If no car exists, location is null then can add car reference to this location |
| VendingMachine contains a car at (2,2)  Try adding another car at (2,2) | "Error: Slot at (2,2) is already occupied." |  |

Suggestion for approach:

1. UML design for car and vending machine class
2. Implement car class and unit test
3. Implement method in main driver class to read from a file
4. Implement vending machine class and unit test
5. Do final testing
6. Go back and highlight all parts of the assignment to make sure you completed everything. Then review [Agile Software Development](https://docs.google.com/presentation/d/1GNVTjZu4YdGPs6kaXF_DOECYMwzhDGxYufdeTUN9XYk/edit#slide=id.p1) to see if you missed anything such as commenting code. Review rubric.

# Learnings and Reflection

Create a separate learnings and reflection document. Include code snippets and examples from your design document to answer the following.

* 1. Explain what sorting algorithm you implemented and why.

I chose Insert Sort because it is the best efficient algorithm among Selection, Bubble and Insert sort.

Explain Insert Sort algorithm: Inserts each element into the correct position within the sorted part of the array

Step-by-step:

* 1. Start with the first element as sorted.
  2. Take the next element and shift it left until it’s in the correct position.
  3. Repeat for all elements.

*Example code*:

A computer code with text

AI-generated content may be incorrect.

* Compare Selection Sort, Insertion Sort, and Bubble Sort.

|  |  |  |
| --- | --- | --- |
| Insert Sort | Selection Sort | Buble Sort |
| Small inputs, all three were "about the same | | |
| Sorting algorithm in which each value, one at a time, is inserted into a sorted subset of the entire list | Sorting algorithm in which each value, one at a time, is placed into its final sorted position in the list | Sorting algorithm in which values are repeatedly compared to neighboring elements in the list and their positions are swapped if they are not in the correct order relative |
| Best performance, especially for small/nearly sorted arrays. | Middle performance, but does unnecessary swaps, making it slower than Insertion Sort. | Worst performance, due to excessive swaps |
| Efficiency:  Best Case (): If the array is already sorted, only one comparison per element.  Worst Case (): If the array is sorted in reverse order, each element moves across the entire sorted part. | Efficiency: Always runs in due to repeated comparisons, making it inefficient for large datasets | Efficiency:  Best Case (): If already sorted, only one pass is needed.  Worst Case (): Requires multiple swaps per element.  Rarely used due to inefficiency |
| A screenshot of a computer  AI-generated content may be incorrect. | A screenshot of a computer  AI-generated content may be incorrect. | A screenshot of a phone  AI-generated content may be incorrect. |

* 1. Give an example of how you used test-driven development to create varied unit tests, design, and implement your solution to add a car to the tower.
* What were the key preconditions and postconditions in your test cases?

They were Valid position, Occupied position and Out of given position

|  |  |  |
| --- | --- | --- |
| Precondition | Postcondition | Comment |
| Vending machine has valid position:  Emty at (1,1). Add car at (1,1) | Car adds to valid position  Car store at (1,1) | *Valid position* |
| Vending machine don’t have valid position:  Has car at (1,2), try to add another car at (1,2) | Error: position (1,2) is already occupied. Can’t not add car | *Occupied position* |
| Vending machine don’t have valid position:  There are no (5,1) position. Add car A to (5,1) | Error: Invalid position at (5,1). Car A can not be added | *Out of given position* |

* Describe how you handled an edge case, such as attempting to add a car to an already occupied space.

Using if/ else if structure to check if this position was occupied or out of given position, if it was not, then add car to this valid position.

public static void **addCar** (Car[][] carTower, int floor, int space, Car carToAdd)

if floor or space < 0 or floor or space > carTower.length

println :

Error: Invalid position at floor and space

Can not place car detail

else if carTower at floor and space != null

println:

Error: Slot at floor and space already occupied

Car detai can’t not be placed

else carTower[floor][space] = carToAdd

* 1. How did you handle invalid user input and exceptions?
* What happens when a user tries to retrieve a car from an empty space?

When a user tries to retrieve a car from an empty space, program will show message that this position is empty.

I used an if statement to check whether this position is null, if so, show message. Otherwise program will retrieve car at this position.

*Example code:*

A screen shot of a computer code

AI-generated content may be incorrect.

* How did you handle file reading errors?

Using try when read data from file and catch FileNotFoundException

A screenshot of a computer program

AI-generated content may be incorrect.

* 1. What parts of the program could cause the program to crash? Pick one and explain how it could be fixed.

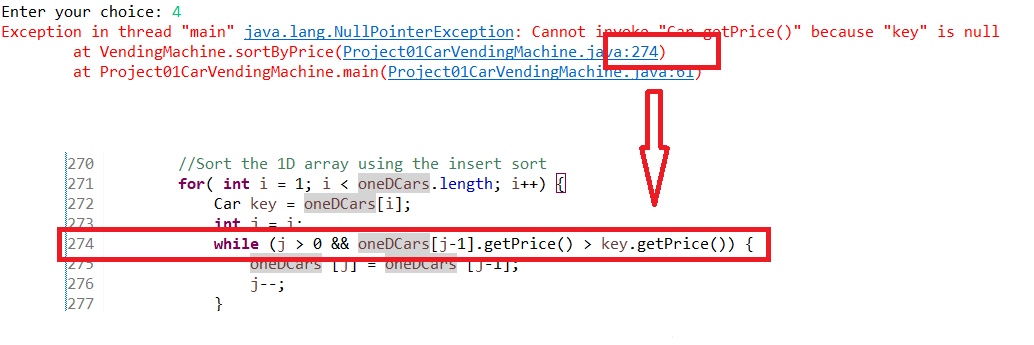
Task “Display cars sorted by year and price” could cause program to crash.

Explanation:

In this part, I extracted *all cars from the 2D array into a 1D list* then applied sorting

During sorting, I used getter method such as getYear() or getPrice() to compare cars. However, If a car object does not exit (null), calling the methods would result in a crash.

Original 2D array have some null values (empty car), and these null values were also added to 1D list, which caused program to crash.



Solution: Filter out all null value form 1D list and create a new 1D car array that only contain valid (non-null) car objects.

A screenshot of a computer code

AI-generated content may be incorrect.

* 1. Explain what you did to improve your problem-solving abilities?

I implemented my solution, and if it worked correctly, I thought about whether there was a shorter or better way to improve it. If there was an error, I reviewed and resolved it. Each time I fixed an error, I gained a better understanding of the problem

How did you use A.I. or other resources to help you without it solving the problem for you?

When an error occurred, I read the error message shown by the program, which indicated the type of error and where it happened. I then moved to that line of code and examined it. After that, I used AI to help explain the error and compare it to my own understanding. This helped me understand the error more clearly.

* 1. What was the most challenging part of this project?

Designing the solution was the most challenging part of this project. For example, my first design for the sort-by-price-and-year algorithm didn’t work. I had to revise it to fix the problem.

* 1. What are you most proud of?

I was able to complete this program on time, and the output matched the given sample output.

# Rubric

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Design 20 Points  Contains UML diagrams, varied unit tests,  Algorithms in pseudocode | 20 to >19 pts  Exemplary A+ | 19 to >15 pts  Proficient A-B | 15 to >11 pts  Developing C-D | 11 to >0 pts  Insufficient F |
| Solution (20 pts)  Implement class concepts to meet requirements and specification | 20 to >19 pts  Exemplary A+ | 19 to >15 pts  Proficient A-B | 15 to >11 pts  Developing C-D | 11 to >0 pts  Insufficient F |
| Quality Code (10pts) Readable, Maintainable, Documented Code | 10 to >9 pts  Exemplary A+ | 9 to >7 pts  Proficient A-B | 7 to >6 pts  Developing C-D | 5 to >0 pts  Insufficient F |
| Learning and Reflection (10pts)  Thoroughly explains learnings and reflects making clear the connection(s) to work and concepts from lectures. | 10 to >9 pts  Exemplary A+ | 9 to >7 pts  Proficient A-B | 7 to >6 pts  Developing C-D | 5 to >0 pts  Insufficient F |