WAREHOUSE PROJECT REPORT

Developed By:

Mr.Krittin Ukarapattayakul 60340500052

Mr.Achita Chayasisrattanakorn 60340500048

Mr.Peeraput Chatniporn 60340500068

[School]

[Course title]

Warehouse Project Report

The software we created is a warehouse management software. It is created to support a warehouse with 5 different warehouses with a total of 9675 slots. There are 25 types of items each identified by the alphabet A to Y. Each type has 500 variations therefore the total of all possible product variations this warehouse is required handle is 12500. The solution is to store product types A, B and C in warehouse number 1, D, E and F in warehouse number 2, G, H and I in warehouse number 3. The rest will be stored in warehouse number 5. Warehouse number 4 will be reserved for products which cannot be stored in their intended position due to the intended slot being occupied.

The warehouse this software is required to handle has specifications as follows:

* A warehouse has many rows.
* Each row is a 2 dimensional grid.
* Each space in a grid is used to store a product
* Each warehouse has a robot to pick up and store items
* Warehouse 1 is connected to the conveyer belt. Warehouse 1 is connected to warehouses 2 and 3. Warehouse 2 is connected to warehouses 4 and 5.
* Warehouses 1, 2 and 3 each has 5 rows of 10x10 grid. (1500 slots)
* Warehouse 4 has 7 rows of 5x5 grid. (175 slots)
* Warehouse 5 has 20 rows of 20x20 grid. (8000 slots)
* The conveyer belt can hold up-to 10 items. Items are retrieved from the belt on a first-come-first-serve basis.

The Requirement for this software are as follows:

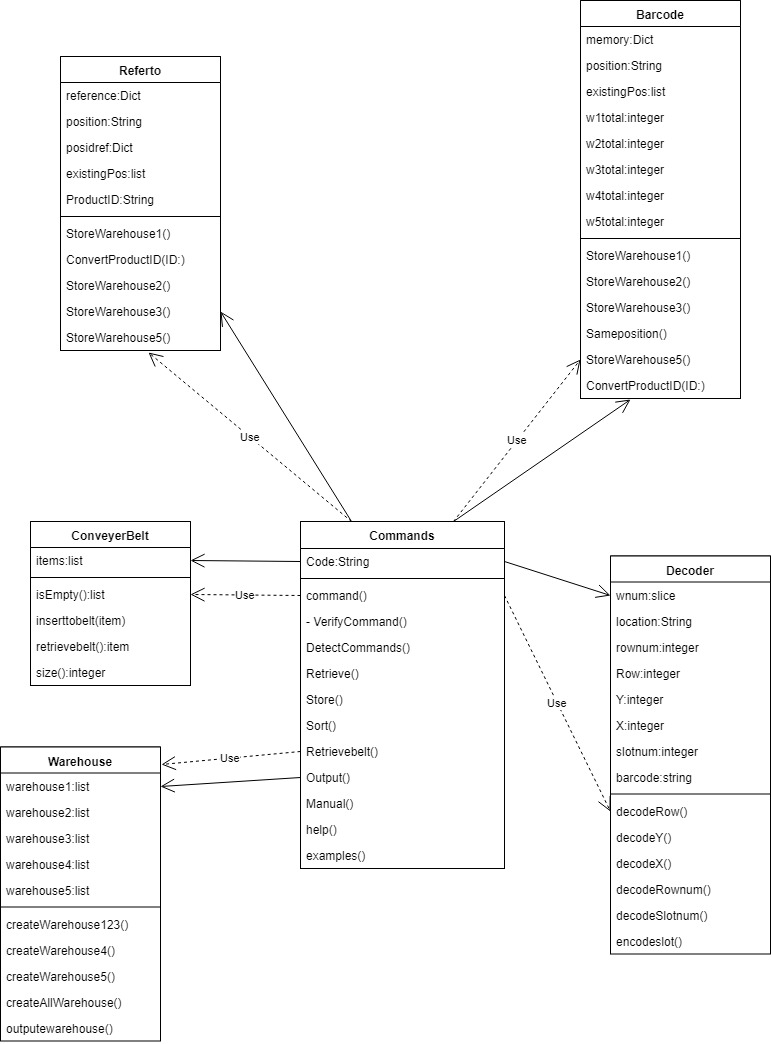
Functional Requirements:

* When an order is received, a robot will pick up an item from the slot in the warehouse and transfer it to the belt.
* When an order is received, the belt will output 1 item at a time.
* When a command is received, a robot will transfer and store an item at a specific location
* There are 7 commands the software must be able to perform
  + Retrieve product ID XXXX
  + Store product ID XXXX
  + Sort Warehouse X at row Y
  + Retrieve a product from the conveyer belt.
  + Output information of all warehouse.
  + Search for product ID XXXX.
  + Manually place product XXXX at position ABC.
* Command inputs must be an English alphabet excluding Z and numbers only.

Non-functional Requirements:

* Each command should have the following outputs:
  + Retrieving command: 0XXXX
    - If the system can operate successfully, the system will output the following statements in this order:
      * Moving from Belt to Warehouse 1.
      * Moving from Warehouse 1 to Warehouse 3.
      * Getting a product id XXXX from warehouse 3: row y slot z.
      * Moving from Warehouse 3 to Warehouse 1.
      * Moving from Warehouse 1 to Start.
      * Place product id XXXX on the belt.
      * Retrieving Successfully!
    - If any error occurs the system will output one of the following statements:
      * Belt is full. Cannot retrieve product.
      * Format invalid.
      * Product does not exist. Cannot retrieve the product.
      * Product is on the belt.
      * Oops something went wrong! (Followed by a list of available commands)
    - Values entered at XXXX must represent the product ID. E.g.A125
  + Storing command: 1XXXX
    - If the system can operate successfully, the system will output the following statements in this order:
      * Moving from Belt to Warehouse 1.
      * Moving from Warehouse 1 to Warehouse 3.
      * Storing a product id XXXX from warehouse 3: row y slot z.
      * Moving from Warehouse 3 to Warehouse 1.
      * Moving from Warehouse 1 to Start.
      * Place product id XXXX on the belt.
      * Storing Successfully!
    - If any error occurs the system will output one of the following statements:
      * Product already exists. Cannot store the product.
      * Slot is being occupied. Cannot store the product.
      * Format invalid.
    - Values entered at XXXX must represent the product ID. E.g.A125
  + Sort command: 2XY00
    - The system will output the following:
      * Sorting process for warehouse X is complete.
    - If any error occurs the system will output the following:
      * Oops something went wrong.
    - Values entered in this command must be numbers. X has a value of 1 to 5 and Y has a value of 01 to 20.
      * For Y, values less than 10 must be entered with a 0 before it e.g. 1 will be entered as 01, 2 will be entered as = 02.
  + Retrieve from belt command: 30000
    - If the system can operate successfully, the system should output the following:
      * Retrieve a product with ID XXXX from the belt.
      * The belt now has Y products on the line.
        + XXXX represents the product ID being retrieved
        + Y represents the amount of products left on the line.
    - If the belt is empty, the system will output the following:
      * The belt is empty. Cannot retrieve product from the belt.
    - If any error occurs the system will output one of the following:
      * Command is not recognized. Please refer to command help.
    - This command has no other variations.
  + Output command: 40000
    - The system will output the following:
      * Warehouse 1
      * Number of rows: 5
      * Number of total products: X
      * Products in row 1: id -
      * Products in row 2: id -
      * Products in row 3: id -
      * Products in row 4: id -
      * Products in row 5 :id –
    - This will repeat for all warehouses.
      * Row amount will correspond to amount of rows in the given warehouse.
    - This command has no other variations.
  + Search command: 5XXXX
    - If the system operates successfully, the system should output the following:
      * Found product XXXX at warehouse Y Row: A Slot: C
        + Y represents warehouse number. Value 1 to 5
        + A represents row number. Value 1 to 20
        + C represents slot number. Value 1 to 400
    - If any error occurs the system will output:
      * Product not found.
    - Values entered at XXXX must represent the product ID E.g.A125
  + Manually move a product command: 9XXXXABC
    - If the system operates successfully, the system should output the following:
      * Move product XXXX to Warehouse A Row: B Slot: C
        + A represents warehouse number. Value 1 to 5
        + B represents row number. Value 1 to 20
        + C represents slot number. Value 1 to 400
    - If the slot is occupied the system will output the following:
      * Slot is occupied. Failed to move.
    - If any error occurs the system will output the following:
      * Oops something went wrong!
    - Values entered at XXXX must represent the product ID. E.g.A125
* Each item will have a product ID in a form of 4 characters. ABCD
  + A represents the type of an item. It has a value of A to Y
  + B represents the row number. It has a value of 1 to 5
  + C represents the Y coordinate of the grid. It has a value of 0 to 9
  + D represents the X coordinate of the grid. It has a value of 0 to 9

Running the software:

* The software is required to run on the Python 3.6 Environment.
* Libraries Pandas and string.
* Try runnning Warehouse.bat"
* If failed. Run BestCommand.py
* If failed. Open up Visual Studio Code. Run BestCommand.py
* All files in the repository must be in the same folder.
* File checklist:
  + 'BestCommand.py'
  + 'Conveyerbelt2.py'
  + 'Decoder.py'
  + 'Mainwarehouse.py'
  + 'NewWarehouse.py'
  + 'Reference.py'
  + 'robotarm.py'
  + 'Warehouse.bat'

Class Command: Controls the whole warehouse via commands

Functions:

Command(): Receives input using built-in function input(). Converts all characters into uppercase using built-in .upper(). Detects if command help is used. If used call method help().Detects command examples. If detected call method examples().If neither,call private method VerifyCommand(). This function outputs none.

VerifyCommand(): Detects if there is any input using len(). If there is nothing, print ‘No command. Use help.’ If there is something call method DetectCommands().

DetectCommands(): Detects the type of input. If the first character of the input is 0, call method Retrieve(). If the first character of the input is 1, call method Store(). If the first character of the input 2, call method Sort(). If the first character of the input is 3, call method RetrieveBelt(). If the first character of the input 4, call method Output(). If the first character of the input is 5, call method Search(). If the first character is 9, call method Manual(). If anything else is recognized, print ‘Command is not recognized.’ Then call method help() and call command().This function outputs none.

Retrieve(): Retrieves the given product and checks if the product exists, on the belt, if the product ID format is invalid or if the belt is full or not. If one the above is true, print ‘Product does not exist. Cannot retrieve the product’ , ‘Product is on the belt’ , ‘Format invalid’ or ‘Belt is full. Cannot retrieve product’ respectively. If it passes all of the conditions above, the given product ID will the act as a key to call the barcode from dictionary memory from class Barcode which determines where the product is. The barcode will then be put through the class Decoder which outputs the exact coordinates of the product. If the product is found, the system will output in this format:

* Moving from Belt to Warehouse 1.
* Moving from Warehouse 1 to Warehouse 3.
* Getting a product id XXXX from warehouse 3: row y slot z.
* Moving from Warehouse 3 to Warehouse 1.
* Moving from Warehouse 1 to Start.
* Place product id XXXX on the belt.
* Retrieving Successfully!

The product will then place the product on the belt. Calling the method inserttobelt() from the class conveyerbelt. Then call method command()

If the key is not found the system will call the method help() and command().

Store():Stores the given product and checks if the product already exists or the slot is occupied or not. If one of the above is true, print ‘Product already exists. Cannot store product’ or ‘Slot is occupied. Cannot store product’ respectively. If it passes all of the conditions above, the system will call method ConvertProductID() from class Barcode. If the product ID is valid the system will output in this format:

* Moving from Belt to Warehouse 1.
* Moving from Warehouse 1 to Warehouse 3.
* Storing a product id XXXX from warehouse 3: row y slot z.
* Moving from Warehouse 3 to Warehouse 1.
* Moving from Warehouse 1 to Start.
* Place product id XXXX on the belt.
* Storing Successfully!

The product will then be placed into the respective position. The method command() will then be called.

If the product ID is not valid the system will print ‘Format invalid’ then call the method command().

Sort(): Sorts the products at the declared row returning them to their respective slots. The system will collect all the products in the row. The products will then be crossed checked with the dictionary reference from the class Referto. The products will then be removed from its initial position and placed into the reference position. If done the system will print ‘Sorting process for warehouse X is complete’ where X is the respective warehouse. If the product cannot be sorted it will be left as is. If the row entered does not exist the system will print ‘That row does not exist’.

RetrieveBelt(): Retrieves a product from the belt. Checks if the input command is 30000.If not print ‘Command is not recognized. Please refer to command help.’ If it is, check if there are any products on the belt. If not print ‘Belt is empty. Cannot retrieve product from belt.’ If there is retrieve the first product which entered the belt calling the method retreivebelt() from class conveyerbelt. The system should print the following:

* Retrieve a product with ID XXXX from the belt.
* The belt now has Y products on the line.

The product ID will then be removed from the dictionary memory.

Output(): Prints the information of all warehouses in the following format,

The total number of products will be called from the class Barcode for each warehouse. The product IDs will be retrieved from each physical warehouse from class Warehouse via for loops.

* + - * Warehouse 1
      * Number of rows: 5
      * Number of total products: X
      * Products in row 1: id -
      * Products in row 2: id -
      * Products in row 3: id -
      * Products in row 4: id -
      * Products in row 5 :id –

This will be repeated for all 5 warehouses.

Search(): The system will use the product ID as the key. It will then call the value from the memory dictionary from class Barcode. If the product is not found the system will print ‘Product not found’ or ‘Oops something went wrong!’. If found the system should print ‘Found product at warehouse: X row: Y slot: Z’.

Manual(): The system will identify the product id and where it needs to go. Find where it is located using the value from dictionary memory and using the class Decoder determine where it needs to be placed. If the slot is occupied the system should output the following ‘Slot is occupied. Failed to move’. If the method is successful the system will print out the following ‘Move Product XXXX to warehouse: X Row : Y slot: Z’. If any error occurs the system should print out ‘Oops something went wrong!’.

Help():Prints out how each command should be used.

Examples():prints examples of each command.

Class Referto: Creates the references for each product ID

Functions:

ConvertProductId(): Checks which warehouse the item is supposed to be stored in. Warehouse 1:’ABC’ Warehouse 2:’DEF’ Warehouse 3:’GHI’ Warehouse 5:’J to Y’. Warehouse 4 is reserved for products which cannot be placed in their intended position.

If the product needs to be stored in warehouse X, method StoreWarehouseX() will be called. X is the warehouse number 1 2 3 and 5.

StoreWarehouseX(): Methods StoreWarehouse1() StoreWarehouse2() and StoreWarehouse3() has the same algorithm. This method converts the product ID into a 7 digit barcode indicating the Warehouse number, row number, and coordinate x and y. This is done by adding a 0 to the front of each digit in the product ID. E.g.A125 will become 1010205.For warehouse 5 when the digits for the row x and y exceed 10, the barcode will remove the 0 in front and insert the actual value. If the barcode already exists the X will shift by 10 if it exists again Y will shift by 10 if again the value will shift by X 10 Y 10 then shift rows by 5 and repeat then by 10 repeat and then 15 and repeat.The product ID will then be stored as the key with the barcode as the value into dictionary reference. The barcode will become the key and the product ID will become the value for dictionary posidref. The barcode will be appended into list existingPos.

Class Barcode: Encodes the product ID’s

Functions:

ConvertProductId(): Checks which warehouse the item is supposed to be stored in. Warehouse 1:’ABC’ Warehouse 2:’DEF’ Warehouse 3:’GHI’ Warehouse 5:’J to Y’. Warehouse 4 is reserved for products which cannot be placed in their intended position.

If the product needs to be stored in warehouse X, method StoreWarehouseX() will be called. X is the warehouse number 1 2 3 and 5.

StoreWarehouseX(): Methods StoreWarehouse1() StoreWarehouse2() and StoreWarehouse3() has the same algorithm. This method converts the product ID into a 7 digit barcode indicating the Warehouse number, row number, and coordinate x and y. This is done by adding a 0 to the front of each digit in the product ID. E.g.A125 will become 1010205.If the value exists in the list existingPos, call the method Sameposition(). For warehouse 5 when the digits for the row x and y exceed 10, the barcode will remove the 0 in front and insert the actual value. If the barcode already exists the X will shift by 10 if it exists again Y will shift by 10 if again the value will shift by X 10 Y 10 then shift rows by 5 and repeat then by 10 repeat and then 15 and repeat. The product ID will then be stored as the key with the barcode as the value into dictionary memory. The barcode will be appended to the list existingPos.

Sameposition(): This method will store items into warehouse 4 by dividing the x and y value by 2. If the product still cannot be placed, repeat the process and add the row by 1. If again repeat the process adding the row by 2 from its initial value. If again divide X y by 4 and add the row by 1 from its initial value. If still the product cannot be place, the product will not be placed in the warehouse.

Class conveyerbelt: Creates the Conveyer belt

Functions:

isEmpty():Deletes the whole belt by replacing it with a new one.

Inserttobelt(item): Places an item on the belt by appending it to list items.

Retrievebelt():Retrieves the first item which entered it via .pop().

Size():returns the length of the items list.

Class Mainwarehouse: Creates the physical warehouse.

Functions:

createWarehouseX: Methods createWarehouse1(), createWarehouse2(), createWarehouse3(), createWarehouse4(), createWarehouse5() have the same algorithm with different values. They create the physical warehouse as a 3d array.

createAllWarehouse():Calls method createWarehouse1(), createWarehouse2(), createWarehouse3(), createWarehouse4(), createWarehouse5()

outputwarehouse():prints all warehouses in a nice grid via pandas.DataFrame()

How the products are stored:

* Product types A B and C are stored in warehouse 1.
* Product types D E and F are stored in warehouse 2.
* Product types G H and I are stored in warehouse 3.
* The rest are stored in warehouse 5.
* Warehouse 4 is reserved for Products which cannot fit into warehouse 1 2 and 3.