Project Report:

Portfolio Milestone – Home Inventory Manager

Alejandro Ricciardi

Colorado State University Global

CSC320: Programming 1

Professor Herbert Pensado

May 26, 2024

Project Report:

Portfolio Milestone – Home Inventory Manager

This documentation is part of the Portfolio Milestone Assignment from CSC320: Programming 1 at Colorado State University Global. This Project Report is an overview of the program's functionality, pseudocode. The program is coded in Java JDK-21; and is named Portfolio Milestone (Home Inventory Manager). The program is composed of a Main class, Home class, and a HomeInventory class. The program is not final, it is an Alpha version of the Portfolio Assignment.

The Assignment Direction

Portfolio Milestone – Home Inventory:
Submit a document with methods for your home class (Portfolio Project choice), and pseudo code indicating functionality of each method.
Example:

```
public String RemoveHome(String homeModel, String homeAddress, int homeZipCode)
If
               values entered match values stored in private variables
               remove home information
else
                return message indicating mismatch
```

Portfolio Project – Home Inventory Manager:
Your Portfolio Project for CSC320 will consist of three components:

- Program corrections: Make the appropriate corrections to all the programming assignments submitted as Critical Thinking assignments from Modules 1-6. You will need to submit the programs along with the carefully outlined corrections needed in order for programs to run correctly.
- Lessons learned reflection: Create a 2-3-page summary that outlines the lessons learned in this Programming I course.
- Final program: Create a final program that meets the requirements outlined below.
  Final Program Requirements
  Create a home inventory class that will be used by a national builder to maintain inventory of available houses in the country. The following attributes should be present in your home class:
- private int square_feet
- private string address
- private string city
- private string state
- private int zip_code
- private string Model_name
- private string sale_status (sold, available, or under contract)

Your program should have appropriate methods such as:

- constructor
- add a new home
- remove a home
- update home attributes

All methods should include try..catch constructs. Except as noted, all methods should return a success or failure message (failure message defined in "catch").

1. Create an additional class to call your home class (e.g., Main or HomeInventory). Include a try..catch construct and print it to the console.
2. Call home class with parameterized constructor (e.g., "square_feet, address, city, state, zip_code, Model_name, sale_status").
    - Then, call the method to list the values. Loop through the array and print to the screen
3. Call the remove home method to clear the variables:
    - Print the return value.
4. Add a new home.
    - Print the return value.
    - Call the list method and print the new home information to the screen.
5. Update the home (change the sale status).
    - Print the return value.
    - Call the listing method and print the information to the screen.
6. Display a message asking if the user wants to print the information to a file (Y or N).
    - Use a scanner to capture the response. If "Y", print the file to a predefined location (e.g., C:\Temp\Home.txt). Note: you may want to create a method to print the information in the main class.
    - If "N", indicate that a file will not be printed.

Your final program submission materials must include your source code and screenshots of the application executing the application and the results.

Compile your Module 1-6 programs with corrections, lessons learned reflection, and final program course code and application screenshots.

⚠ **My notes:**

- This is an alpha version of the Portfolio Project
- I got permission from Professor Pensado for the program to manipulate a file.
- **For the source code please see Main.java, Home.java, and HomeInventory.java files.**

Git Repository

I use GitHub as my Distributed Version Control System (DVCS), the following is a link to my GitHub, Omegapy.

My GitHub repository that is used to store this assignment is named My-Academics-Portfolio and the link to this specific assignment is: https://github.com/Omegapy/My-Academics-Portfolio/tree/main/Programming-1-CSC320/Portfolio-Milestone

Classes Pseudocode

Home Class Pseudocode

Creates a Home object with various attributes such as id, square feet, address,
city, state, zip code, model name, and sale status.
The class has getters and setters' methods to access and manipulate the Home objects' data.

It is utilized by the HomeInventory class.

Constructor

```
public Home(Integer id, Integer squareFeet, String address, String city,
            String state, Integer zipCode, String modelName, String saleStatus) throws
                                                                                Exception
    If any of the input parameters (id, squareFeet, address, city,
                                    state, zipCode, modelName, saleStatus) is null:
        Throw an exception indicating which parameter is null.
    Else:
        Assign the input parameters to the corresponding class fields.
        Print a success message indicating the Home object was created successfully.
    End If
End Method
```

Getters

Get ID Method

```
public Integer getId() throws NullPointerException
    If id is null:
        Throw an exception indicating id is null.
    Else:
        Print a success message indicating id was found.
        Return id.
    End If
End Method
```

Get Square Feet Method

```
public Integer getSquareFeet() throws NullPointerException
    If squareFeet is null:
        Throw an exception indicating squareFeet is null.
    Else:
        Print a success message indicating squareFeet was found.
        Return squareFeet.
    End If
End Method
```

Get Address Method

```
public String getAddress() throws NullPointerException
    If address is null:
        Throw an exception indicating address is null.
    Else:
        Print a success message indicating address was found.
        Return address.
    End If
End Method
```

Get City Method

```
public String getCity() throws NullPointerException
    If city is null:
        Throw an exception indicating city is null.
    Else:
        Print a success message indicating city was found.
        Return city.
    End If
End Method
```

Get State Method

```
public String getState()
    If state is null:
        Throw an exception indicating state is null.
    Else:
        Print a success message indicating state was found.
        Return state.
    End If
End Method
```

Get ZipCode Method

```
public Integer getZipCode() throws NullPointerException
    If zipCode is null:
        Throw an exception indicating zipCode is null.
    Else:
        Print a success message indicating zipCode was found.
        Return zipCode.
    End If
End Method
```

Get Model Name Method

```
public String getModelName() throws NullPointerException
    If modelName is null:
        Throw an exception indicating modelName is null.
    Else:
        Print a success message indicating modelName was found.
        Return modelName.
    End If
End Method
```

Get Sale Status Method

```
public String getSaleStatus() throws NullPointerException
    If saleStatus is null:
        Throw an exception indicating saleStatus is null.
    Else:
        Print a success message indicating saleStatus was found.
        Return saleStatus.
    End If
End Method
```

Setters

Set Square Feet Method

```
public void setSquareFeet(Integer squareFeet) throws NullPointerException
    If squareFeet is null:
        Throw an exception indicating squareFeet is null.
    Else:
        Assign squareFeet to the class field.
        Print a success message indicating squareFeet was set successfully.
    End If
End Method
```

Set Address Method

```
public void setAddress(String address) throws NullPointerException
    If address is null:
        Throw an exception indicating address is null.
    Else:
        Assign address to the class field.
        Print a success message indicating address was set successfully.
    End If
End Method
```

Set City Method

```
public void setCity(String city) throws NullPointerException
    If city is null:
        Throw an exception indicating city is null.
    Else:
        Assign city to the class field.
        Print a success message indicating city was set successfully.
    End If
End Method
```

Set State Method

```
public void setState(String state) throws NullPointerException
    If state is null:
        Throw an exception indicating state is null.
    Else:
        Assign state to the class field.
        Print a success message indicating state was set successfully.
    End If
End Method
```

Set ZipCode Method

```
public void setZipCode(Integer zipCode) throws NullPointerException
    If zipCode is null:
        Throw an exception indicating zipCode is null.
    Else:
        Assign zipCode to the class field.
        Print a success message indicating zipCode was set successfully.
    End If
End Method
```

Set Model Name Method

```
public void setModelName(String modelName) throws NullPointerException
    If modelName is null:
        Throw an exception indicating modelName is null.
    Else:
        Assign modelName to the class field.
        Print a success message indicating modelName was set successfully.
    End If
End Method
```

Set Sale Status Method

```
public void setSaleStatus(String saleStatus) throws NullPointerException
    If saleStatus is null:
        Throw an exception indicating saleStatus is null.
    Else:
        Assign saleStatus to the class field.
        Print a success message indicating saleStatus was set successfully.
    End If
End Method
```

toString Method

```
@Override
public String toString()
    Create and return a string that includes the class name (Home) and each field name followed
    by its value.
    The format should be:
    Home{id=<id>, squareFeet=<squareFeet>, address='<address>', city='<city>', state='<state>',
         zipCode=<zipCode>, modelName='<modelName>', saleStatus='<saleStatus>'}
    This string provides a readable representation of the Home object, displaying its current
    state.
End Method
```

HomeInventory Class Pseudocode

Creates an Inventory object that manages home data
allowing for adding, removing, updating, and saving home data.
The class can also create, read, write, and delete a home file text containing the home data.
The class has getters and setters' methods to access and manipulate the Home and Inventory objects' data.

It is utilized by the Main class.

Constructor

```
public HomeInventory(String filePath) throws Exception
    If filePath is null:
        Throw a NullPointerException indicating the file path is null.
    Else:
        Assign filePath to the class field.
        Initialize the homes ArrayList.
        Call loadHomes() to load homes data from the file.
        If homes list is not empty:
            Set numHome to the ID of the last home in the list.
        End If
        Print a success message indicating the HomeInventory object was created successfully.
    End If
End Method
```

Getters

Get Path File Method

```
public String getPathFile() throws NullPointerException
    If filePath is null:
        Throw a NullPointerException indicating the file path is null.
    Else:
        Print a success message indicating the file path was found.
        Return filePath.
    End If
End Method
```

Get Home by ID Method

```
public Home getHomeById(int id) throws Exception
    For each home in homes:
        If the home's ID matches the input id:
            Print a success message indicating the home was found.
            Return the home.
        End If
    Throw an exception indicating the provided id does not match a home.
End Method
```

Get Home by Address Method

```
public Home getHomeByAddress(String address) throws Exception
    For each home in homes:
        If the home's address matches the input address:
            Print a success message indicating the home address was found.
            Return the home.
        End If
    Throw an exception indicating the provided home address was not found.
End Method
```

Get Home List Method

```
public ArrayList<Home> getHomesList() throws Exception
    If homes is null:
        Throw an exception indicating the homes list is null.
    Else:
        Print a success message indicating the homes list was retrieved successfully.
        Return homes.
    End If
End Method
```

Setters

Add Home Method

```
public void addHome(Integer squareFeet, String address, String city, String state, Integer
zipCode, String modelName, String saleStatus) throws Exception
    Try:
        Create a new Home object with the provided parameters and increment numHome for the new
        home ID.
        Add the new home to the homes list.
        Print a success message indicating the home was added successfully.
    Catch an exception:
        Throw the exception with its message.
End Method
```

Remove Home by ID Method

```
public void removeHomeById(Integer id) throws Exception
    Try:
        Remove the home with the matching ID from the homes list.
        If a home was removed:
            Print a success message indicating the home was removed successfully.
        Else:
            Throw an exception indicating the home with the provided ID was not found.
        End If
    Catch an exception:
        Throw the exception with its message.
End Method
```

Remove Home by Address Method

```
public void removeHomeByAddress(String address) throws Exception
    Try:
        Remove the home with the matching address from the homes list.
        If a home was removed:
            Print a success message indicating the home was removed successfully.
        Else:
            Throw an exception indicating the home with the provided address was not found.
        End If
    Catch an exception:
        Throw the exception with its message.
End Method
```

Add Update Home by ID Method

```
public void updateHomeById(Integer id, Home updatedHome) throws Exception
    Try:
        For each home in homes:
            If the home's ID matches the input id:
                Update the home with the new data.
                Print a success message indicating the home was updated successfully.
                Return.
            End If
        Throw an exception indicating the home with the provided ID was not found.
    Catch an exception:
        Throw the exception with its message.
End Method
```

Add Update Home by Address Method

```
public void updateHomeByAddress(String address, Home updatedHome) throws Exception
    Try:
        For each home in homes:
            If the home's address matches the input address:
                Update the home with the new data.
                Print a success message indicating the home was updated successfully.
                Return.
             End If
        Throw an exception indicating the home with the provided address was not found.
    Catch an exception:
        Throw the exception with its message.
End Method
```

Methods for Saving and Loading Data

Save Homes Method

```
public void saveHomes() throws Exception
    Try:
        Create a BufferedWriter object to write to the file at filePath.
        For each home in homes:
            Write the home's data to the file in CSV format.
        Print a success message indicating the home inventory was saved successfully.
    Catch an IOException:
        Throw a new exception with the error message.
    Catch a general exception:
        Throw a new exception with the error message.
    Finally:
        Try:
            Close the BufferedWriter object.
        Catch an IOException:
            Throw a new exception with the error message.
End Method
```

Load Homes Method

```
private void loadHomes() throws IOException
    Try:
        Create a BufferedReader object to read from the file at filePath.
        While there are lines to read from the file:
            Split the line into parts.
            Try:
                Create a new Home object with the parts.
                Add the home to the homes list.
                Print a success message indicating the home was loaded successfully.
            Catch an exception:
                Print a failure message indicating the home could not be loaded.
    Catch an IOException:
        Throw a new IOException with the error message.
    Finally:
        Try:
            Close the BufferedWriter object.
        Catch an IOException:
            Throw a new exception with the error message.
End Method
```

Main Class Pseudocode

Main Method

```
public static void main(String[] args)
    Initialize a Scanner object for user input.
    Declare a HomeInventory object.
    Set quitProgram to false.
    Display the program banner.

    Call createInventory(scanner) to initialize the HomeInventory object.

    While quitProgram is false:
        Display the menu options.
        Prompt the user to enter their choice.
        Based on the user choice:
            Case 1:
                Call addHome(scanner, inventory).
            Case 2:
                Call removeHome(scanner, inventory).
            Case 3:
                Call updateHome(scanner, inventory).
            Case 4:
                Call listHomes(inventory).
            Case 5:
                Call displayHome(scanner, inventory).
            Case 6:
                Prompt the user to save changes.
                If the user chooses to save:
                    Try to call inventory.saveHomes().
                    Catch and print any exceptions.
                End If
                Prompt the user to delete the inventory file.
                If the user chooses to delete:
                    Call deleteFile(inventory.getPathFile()).
                End If
                Set quitProgram to true.
            Default:
                Print "Invalid choice. Please try again."

    Close the scanner.
    Print a thank you message.
End Method
```

Add Home Method

```
private static void addHome(Scanner scanner, HomeInventory inventory)
    Try:
        Prompt the user to enter the square feet.
        Prompt the user to enter the address.
        Prompt the user to enter the city.
        Prompt the user to enter the state.
        Prompt the user to enter the zip code.
        Prompt the user to enter the model name.
        Prompt the user to enter the sale status.
```

```
        Call inventory.addHome() with the provided details.
        Print "Home added successfully."

        Get the ID of the newly added home.
        Call displayHomeUsingId() with the new home ID and inventory.
    Catch any exceptions:
        Print the exception message.
End Method
```

Remove Home Method

```
private static void removeHome(Scanner scanner, HomeInventory inventory)
    Try:
        Prompt the user to enter the ID of the home to remove.
        Call inventory.removeHomeById() with the provided ID.
        Print "Home removed successfully."
    Catch any exceptions:
        Print the exception message.
End Method
```

Update Home Method

```
private static void updateHome(Scanner scanner, HomeInventory inventory)
    Try:
        Prompt the user to enter the ID of the home to update.
        Get the home object using inventory.getHomeById(id).

        If the home is found:
            Set continueUpdating to true.
            While continueUpdating is true:
                Display the update options.
                Prompt the user to enter their choice.
                Based on the user choice:
                    Case 1:
                        Prompt the user to enter the new square feet.
                        Call homeToUpdate.setSquareFeet() with the new value.
                        Print "Square feet updated successfully."
                    Case 2:
                        Prompt the user to enter the new address.
                        Call homeToUpdate.setAddress() with the new value.
                        Print "Address updated successfully."
                    Case 3:
                        Prompt the user to enter the new city.
                        Call homeToUpdate.setCity() with the new value.
                        Print "City updated successfully."
                    Case 4:
                        Prompt the user to enter the new state.
                        Call homeToUpdate.setState() with the new value.
                        Print "State updated successfully."
                    Case 5:
                        Prompt the user to enter the new zip code.
                        Call homeToUpdate.setZipCode() with the new value.
                        Print "Zip code updated successfully."
                    Case 6:
                        Prompt the user to enter the new model name.
                        Call homeToUpdate.setModelName() with the new value.
```

```
                        Print "Model name updated successfully."
                    Case 7:
                        Prompt the user to enter the new sale status.
                        Call homeToUpdate.setSaleStatus() with the new value.
                        Print "Sale status updated successfully."
                    Case 8:
                        Set continueUpdating to false.
                    Default:
                        Print "Invalid choice. Please try again."

            Call inventory.updateHomeById() with the home ID and updated home.
            Print "Home updated successfully."
            Call displayHomeUsingId() with the home ID and inventory.
        Else:
            Print "Home not found."
    Catch any exceptions:
        Print the exception message.
End Method
```

List Homes Method

```
private static void listHomes(HomeInventory inventory)
    Try:
        Get the list of homes from inventory.getHomesList().
        If the homes list is empty:
            Print "No homes in inventory."
        Else:
            For each home in the list:
                Print the home details.
    Catch any exceptions:
        Print the exception message.
```

Display Home Method

```
private static void displayHome(Scanner scanner, HomeInventory inventory)
    Try:
        Prompt the user to enter the ID of the home to display.
        Get the home object using inventory.getHomeById(id).
        If the home is found:
            Print the home details.
        Else:
            Print "Home not found."
    Catch any exceptions:
        Print the exception message.
End Method
```

Display Home Using ID Method

```
private static void displayHomeUsingId(Integer id, HomeInventory inventory)
    Try:
        Get the home object using inventory.getHomeById(id).
        If the home is found:
            Print the home details.
        Else:
            Print "Home not found."
    Catch any exceptions:
        Print the exception message.
End Method
```

Create Inventory Method

```
public static HomeInventory createInventory(Scanner scanner)
    While true:
        Try:
            Prompt the user to enter the file path of an existing or new inventory file.
            Create a File object with the provided file path.
            If the file is a new file:
                While true:
                    Prompt the user to create a file populated with fake data or an empty file.
                    If the user chooses to create a file with fake data:
                        Create a new HomeInventory object.
                        Add fake homes to the inventory.
                        Print a success message.
                        Return the inventory.
                    Else if the user chooses to create an empty file:
                        Create a new HomeInventory object.
                        Print a success message.
                        Return the inventory.
                    Else:
                        Print "Invalid Entry. Please try again."
                    End If
            Else:
                Create a new HomeInventory object with the existing file.
                Print a success message.
                Return the inventory.
            End If
        Catch any exceptions:
            Print the exception message.
            Print "Please try again."
End Method
```

Delete File Method

```
private static void deleteFile(String filePath)
    Create a File object with the provided file path.
    Try:
        Attempt to delete the file.
        Print a success or failure message based on the result.
    Catch a SecurityException:
        Print "Permission denied cannot delete the file."
    Catch any exceptions:
        Print the exception message.
End Method
```