HW 08: Residence Listings

Release date: Friday, October 28th, 2016
Due date: Friday, November 4th,, 2016 – 11:59 pm

Goals

- ★ To understand the fundamentals of inheritance in Java
- ★ To learn how to design and implement class hierarchies
- ★ To learn how to create and use custom, user-defined exceptions

Prerequisites

1. Thorough grasp on the concept of methods and classes.
2. Knowledge of the syntax and requirements of inheritance.
3. Understanding of throw, throws, and try-catch in terms of exceptions.

Description

In this homework assignment, you will write several classes to represent a residence listing system. This section will give you a high-level overview of the system and the requirements section will provide you with the necessary implementation details.

The residence listing system will be composed of a collection of residences, which can be either an apartment or a house, representing all the available listings, and a simple GUI through which the user can access and modify the data in the residence listing collection. Users will be able to perform one of three main actions: delete a listing, post a listing, or look through existing listings with some basic filters.

Requirements

You are required to write a program that matches the description given above. Exact details about the required methods are discussed next. You **must follow** the given naming for files, classes, data members, and methods.

1) Files to develop

You are required to create a total of 6 files as described in **Table 1**. One of these files, ResidenceListingsDriver.java, will be given to you but you must complete the missing code.

2) The Residence class

The Residence class is a public class. An Object of this class represents a Residence entity. This class is the superclass of our class hierarchy described in **Figure 1**. **Table 2** shows the Residence class fields. **Table 6** shows the Residence class constructor prototype and description. **Table 11** shows the Residence class method prototypes and descriptions.

3) The Apartment class

The Apartment class is a public class. An Object of this class represents an Apartment entity. This class is a subclass of the Residence class. **Table 3** shows the Apartment class fields. You should not define any extra fields in this class other than the ones shown in this table. **Table 7** shows the Apartment class constructor prototype and description. **Table 12** shows the Apartment class method prototypes and descriptions.

4) The House class

The House class is a public class. An Object of this class represents a House entity. This class is a subclass of the Residence class. **Table 4** shows the House class fields. You should not define any extra fields in this class other than the ones shown in this table. **Table 8** shows the House class constructor prototype and description. **Table 13** shows the House class method prototypes and descriptions.

5) The NoSuchResidenceException class

This class is a custom exception which is a subclass of Exception from the Java standard library. This custom exception will be thrown in the *ResidenceListing*s class when an attempt is made to remove a Residence that does not exist. The constructor prototype and description can be found in **Table 9**.

6) The ResidenceListings class

The ResidenceListings class is a public class, acting as a container to hold Residence objects in its Residence array field. **Table 5** shows the ResidenceListings class fields. **Table 10** shows the ResidenceListings class constructor prototype and description. **Table 14** shows the ResidenceListings class method prototypes and descriptions.

7) The ResidenceListingsDriver class

This class contains the main method for your program. You can find the skeleton code for this class on the course page homework section. All of the logic for this class has been implemented for you except for four sections which you will implement the logic for. Three of these missing sections will require you to generate the logic and UI for when a user wants to view all the listings. The user will have a choice of viewing only Apartments, only Houses, or both Apartments and Houses. Depending on the user's choice, you will display the correct Residences to the user in a JOptionPane Option Dialog window along with two buttons: 'Done' if the user does not want to view any more listings, or 'Next' if the user wants to view the next matching listing. An example of the information displayed in this window is given in the **Example Dialog** section near the end of this document.

Additionally, you need to complete the code to remove a residence and display a message dialog to indicate whether the removal was successful or not.

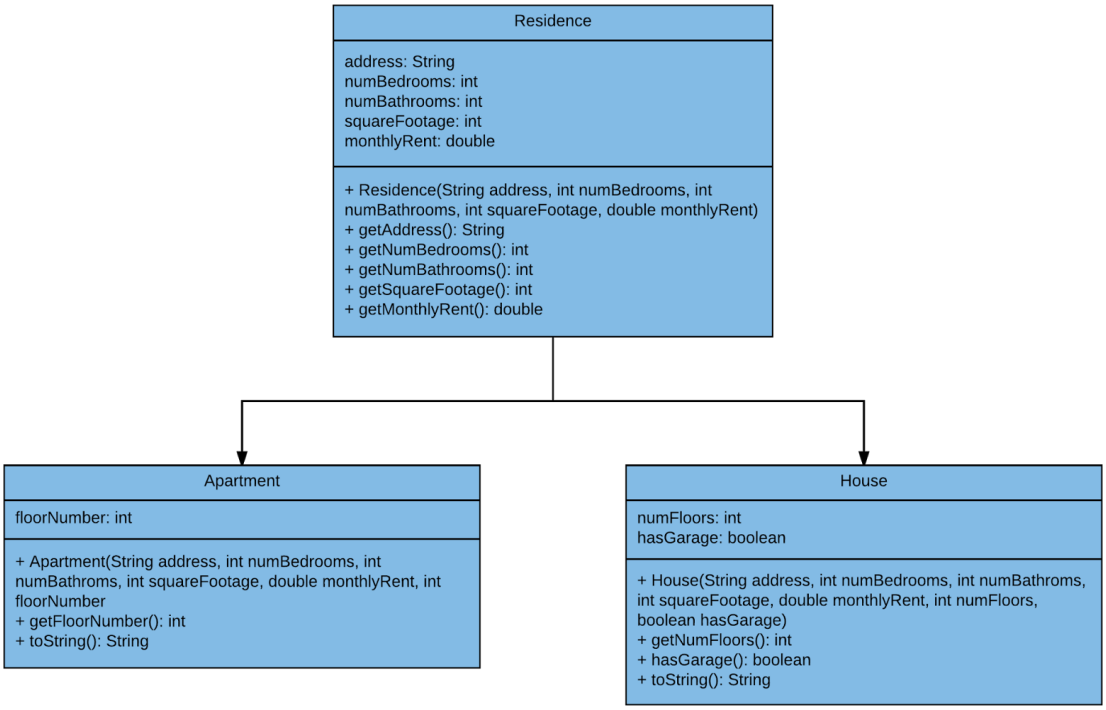

Figure 1: Residence class diagram

Filename	**Description**
Residence.java	Parent class of the class hierarchy for residence listings. This class represents all common properties that all residence listings have in common.
Apartment.java	Subclass of Residence representing a residence classified as an apartment. This type of residence differs by an additional field, the floor number of the apartment.
House.java	Subclass of Residence representing a residence classified as a house. This type of residence differs by two additional fields: the total number of floors the house has, and a boolean value indicating if the house has a garage or not.
NoSuchResidenceException.java	Custom exception thrown in the ResidenceListings class when an attempt is made to remove a Residence that does not exist.
ResidenceListings.java	Class representing a collection of Residences, providing methods for maintaining various properties of the collection and modifying the collection.
ResidenceListingsDriver.java	Contains the main method which provides the user interface

	for interacting with the residence listings.

Table 1: File requirements

Field Name	Type	Access Modifier	Description
`address`	String	private	The address of the residence
`numBedrooms`	int	private	The number of bedrooms in the residence
`numBathrooms`	int	private	The number of bathrooms in the residence
`squareFootage`	int	private	The square footage of the residence
`monthlyRent`	double	private	The monthly rent of the residence

Table 2: Residence class fields

Field Name	Type	Access Modifier	Description
`floorNumber`	int	private	The floor number of the apartment

Table 3: Apartment class fields

Field Name	Type	Access Modifier	Description
`numFloors`	int	private	The number of floors the house has
`hasGarage`	boolean	private	If the house has a garage or not

Table 4: House class fields

Field Name	Type	Access Modifier	Description
`numResidences`	int	private	The total number of non-null elements in the `residences` array
`maxResidences`	int	private	The maximum number of elements that the `residences` array can hold
`residences`	Residence[]	private	Array containing all the Residences to be stored for the ResidenceListings object

Table 5: ResidenceListings class fields

Constructor Prototype	Description
`public Residence(String address, int numBedrooms, int numBathrooms, int squareFootage, double monthlyRent)`	Sets each of the fields of this class to their corresponding arguments.

Table 6: Residence class constructors

Constructor Prototype	Description
`public Apartment(String address, int numBedrooms, int numBathrooms, int squareFootage, double monthlyRent, int floorNumber)`	Sets the *floorNumber* field of this class to the *floorNumber* argument and passes the corresponding to the superclass constructor in the correct order.

Table 7: Apartment class constructors

Constructor Prototype	Description
`public House(String address, int numBedrooms, int numBathrooms, int squareFootage, double monthlyRent, int numFloors, boolean hasGarage)`	Sets the *numFloors* and *hasGarage* fields of this class to the *numFloors* and *hasGarage* arguments and passes the corresponding arguments to the superclass constructor in the correct order.

Table 8: House class constructors

Constructor Prototype	Description
`public NoSuchResidenceException(String mesage)`	Calls the superclass constructor, passing to it the *message* argument.

Table 9: NoSuchResidenceException class constructors

Constructor Prototype	Description
`public ResidenceListings()`	Initializes all the fields of this ResidenceListings, with a capacity to hold 10 Residences and no Residences in the listing.

Table 10: ResidenceListings class constructors

Method Prototype	Description
`public String getAddress()`	Accessor for the field *address*
`public int getNumBedrooms()`	Accessor for the field *numBedrooms*
`public int getNumBathrooms()`	Accessor for the field *numBathrooms*
`public int getSquareFootage()`	Accessor for the field *squareFootage*
`public double getMonthlyRent()`	Accessor for the field *monthlyRent*

Table 11: Residence class methods

Method Prototype	Description
`public int getFloorNumber()`	Accessor for the field *floorNumber*
`public String toString()`	*Overrides* the toString() method of the *Object* class. Returns a formatted String containing all the fields of the Apartment class, each on a new line. The format for each field is as follows: **Field-name: field-value**

Table 12: Apartment class methods

Method Prototype	Description
`public int getNumFloors()`	Accessor for the field *numFloors*
`public boolean hasGarage()`	Accessor for the field *hasGarage*
`public String toString()`	*Overrides* the toString() method of the *Object* class. Returns a formatted String containing all the fields of the Apartment class, each on a new line. The format for each field is as follows: **Field-name: field-value**

Table 13: House class methods

Method Prototype	Description
`public void addResidence(Residence residence)`	Adds the specified Residence argument to the array field *residences*. The Residence to be added is inserted in the first null position from the left within the array. If *residences* is full, then its size should be doubled before adding the Residence

	argument. The fields *numResidences* and *maxResidences* should be updated accordingly.
`public void removeResidence(Residence residence)`	Searches through the array *residences*, looking for the Residence passed as argument to this method. If the Residence is found, then it is removed from the array and all non-null elements to the right of the removed Residence are shifted to the left one position in the array. The field *numResidences* should be updated accordingly. If the Residence is not found, then the exception NoSuchResidenceException is thrown with a meaningful error message.
`public Residence findResidenceByAddress(String address)`	Searches through the array *residences*, looking for the first Residence whose address is equal to that of the address passed as argument to this method. If a match is found, the matched Residence object in the *residences* array is returned, else return null
`public int getNumResidences()`	Accessor for the field *numResidences*
`public int getMaxResidences()`	Accessor for the field *maxResidences*
`public Residence[] getResidences()`	Accessor for the field *residences*

Table 14: ResidenceListings class methods

Example Dialog

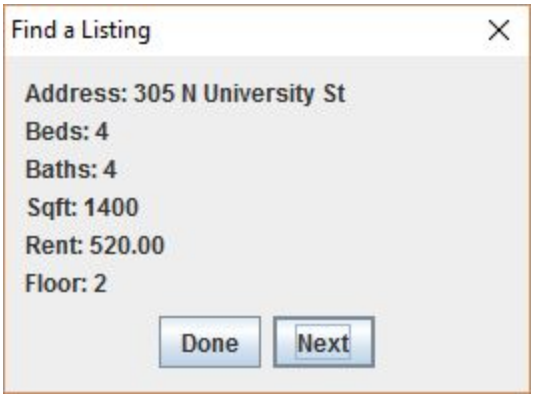

Figure 4: Optional dialog displayed to the user when viewing listings

Submission Instructions

Submit **all your files** to Vocareum by following these instructions. Keep in mind that only your last submission will be considered.

Rubric

- Residence.java: 12%
- Apartment.java: 12%
- House.java: 12%
- NoSuchResidenceException.java: 4%
- ResidenceListings.java: 30%
    - Constructor 4
    - addResidence 6
    - findResidence 5
    - removeResidence 9
    - Accessors (3) 6
- Complete missing code in main(): 30%
    - Apartment searching (iterate & display) 7
    - House searching (iterate & display) 7
    - Apartment+House searching (iterate & display) 7
    - Remove a residence and display a message 9

Good Luck