**Programming Project 6 – Martian Houses**

*Note: When you turn in an assignment to be graded in this class, you are making the claim that you neither gave nor received assistance on the work you turned in (except, of course, assistance from the instructor or teaching assistants).*

As development on Mars grows, housing complexes are being created. The settlers need a program to help them organize their housing developments and houses. They would like a system to keep track of this. They need some abstraction to keep track of their developments and the houses that are in those developments. You will design an object using abstraction, that models the houses within the housing developments and another one that models the housing developments.

The name of the house Java class is **House.java**. The instance variables of a house object are as follows:

owner: String

lotNumber: int

squareFootage: double (ranging from 500.00 – 5000.00)

**Bedrooms**: ONE\_BEDROOM, TWO\_BEDROOM, THREE\_BEDROOM, STUDIO

**Baths**: ONE, TWO, THREE

**Color**:

The fields above in the right-hand column will be expressed as Java Enumeration Types to restrict the values that can be assigned to these instance variables. These types will be located in their own classes.

The parameterized constructor should have the following definition:

public House(String anOwner, int aLotNumber, double aSquareFootage, Bedrooms aBedrooms, Baths aBaths, Color aColor)

This constructor will pass in the owner, the lot number, the square footage, the number of bedrooms, the number of bathrooms, and the color of the house.

The default constructor will assign the default values to the instance variables of the house. The default values will be as follows: an empty String for owner, 1 for lotNumber, 500.00 for squareFootage, and the first value in each enumerated list.

The House class will have getters and setters for each instance variable as well as a toString() method that prints each instance variable of the House object indented by a tab on its own line. **You will start the printing with a new line and then a tab**. **All floating point variables should be formatted to two decimal points with String formatting.**

The other object that needs to be defined represents housing development that holds the houses. The name of the Java class is **Development.java**. The instance variables of a housing development are the following:

name: String

location: String

yearEst: int

numLots: int

houses: ArrayList<House>

The parameterized constructor should have the following definition:

public Development(String aName, String aLocation, int aYearEst, int aNumLots)

This constructor will initialize all instance variables making sure use the values passed in for associated instance variables. It will also create an empty array list in the houses attribute.

The default constructor will place an empty String into each of the String attribute fields and 0 into the int attribute fields and will also create an empty array list in the houses attribute.

You will have a getter and setter for the name, location, yearEst, and numLots instance variables. You will have an addHouse() method that will add a House object to the Development object’s houses instance variable. You will have a getNumHouses() that will let you know how many House objects are associated with a Development object. You will have a getHouses() method that will return the ArrayList of House objects stored in the Development object’s houses instance variable. The Development class will also have a toString() method to output the housing development information, separated by newlines, along with the houses associated with it.

Here are two examples of what a typical output would be from calling the toString() method for a housing development with two houses associated with it and with one house associated with it. Note that the Development’s toString() method must also call the toString() method **for each individual House object** the housing development has after it prints the String, “Houses:”.

Freeze Zone

North Polar Ice Cap

2023

20

Houses:

Caroline Budwell

15

2754.99

2\_BEDROOM

TWO

GREEN

Sam Zu

2

2500.00

3\_BEDROOM

THREE

WHITE

Crater Field

Korolev crater

100

Houses:

Zach Whitten

27

789.45

STUDIO

ONE

GRAY

***Make sure that you use the UML Class Diagrams below to build your classes***. You will test your classes using the JUnit tests provided. You can also test your code by building a HouseDevelopmentTest.java file, which will have a main method. This file is where you will build House and Development objects and perform the methods from the classes on them.

These and all program files in this course must include a comment block at the beginning (top) of the source code file that contains:

* the Java program name
* project description
* your name
* the version date
* the course number and section

The comment block for the header should look like this:

/\*

\* Java program name

\*

\* Project description

\*

\* Your name

\* The version date

\* The course number and section

\*/

In addition, each item created in each class file should be commented using **Javadocs** commenting format as show below:

/\*\*

The variable, constructor, or method is as specified.

\*/

**You will submit your code for your testing program, HouseDevelopmentTest.java, as your algorithm. It should create two Development objects and populate them with the House objects above. This code is due when your algorithm is due.**

You will test your code using the provided JUnit tests. You should use these tests within IntelliJ to make sure that your code is running correctly. You will take a screenshot of your code passing all JUnit tests. Make sure that the screenshot has your header comment block in it so that we know it is your code running. Once it is, then you will submit your code to Gradescope. You will only have 4 submissions to Gradescope, so please make sure your code is running correctly in IntelliJ before submitting to Gradescope.

You will submit your Java source code files (House.java, Development.java, and the Enum classes) to Gradescope. You will upload a screenshot of the JUnit tests by uploading the file to the Assignment link in Canvas. Please **do not** submit your files in a zipped folder.

Ask questions about any part of the programming project that is not clear!

**Grading Rubric**

|  |  |
| --- | --- |
| **Criteria** | **Points** |
| Algorithm submitted on time | 5 |
| All variables, methods and constructors are properly commented using **Javadocs** commenting | 10 |
| Enumerations are syntactically correct and used properly | 10 |
| All instance data members are correctly declared | 5 |
| House default and parameterized constructors correctly written | 10 |
| House specified getter and setter methods correctly constructed | 10 |
| House toString() method written correctly | 10 |
| Development default and parameterized constructors correctly written | 10 |
| Development specified getter and setter methods correctly constructed | 5 |
| Development specified methods correctly constructed | 10 |
| Development toString() method written correctly | 10 |
| Each file has comment at top with identifying information. Program layout and appearance (Coding style is clear and easily understood) | 5 |
| TOTAL | 100 |

**UML Diagrams**



