Description

The objectives of this assignment is to practice the concepts of inheritance, enums and abstract classes in Java. In this assignment, you will write a series of classes to construct an application for a real-estate firm. You will need to write 9 classes: Property, House, Land, MultiUnitBuilding, Cottage, Farm, Office, Apartment Building and a BadProperty exception class. The property class will be an **abstract class**. Something to think about as you read through the assignment description - should the MultiUnitBuilding class also be abstract?

Here is a visual depiction of the hierarchy:

Property -> House -> Cottage

Property -> Land -> Farm

Property -> MultiUnitBuilding -> Office

Property -> MultiUnitBuilding -> ApartmentBuilding

Structure of Program: Instead of describing methods class-by-class, I'm going to organize the descriptions by functionality. In some cases, you'll need to decide where the methods belong -- i.e. in which class(es).

Organization: You need to make decisions about where attributes and methods belong. Some must be in the concrete classes and some have to be shared in parent classes. Avoid duplicated code as much as you can. All your attributes must be private.

Information

The Property class is the root class for all properties. The Property class must have a constructor that takes the listing price of the property as a parameter. *The Property class will be an* ***abstract class*** *- this means that there must be abstract methods included in the Property class. You will need to implement these abstract methods in all of the classes that extend the Property class.*

Attributes:

- The property class has one attribute *listingPrice* which is a double value that represents a property's listing price. ***The listingPrice attribute should not be re-declared in any of the child classes.***
- All of the other classes that *extend* the property class have the following attributes in addition to *listingPrice*:

House: number of bathrooms, number of bedrooms, depth of the lot, width of the lot, floor-space (square meters).

Land: number of hectares of the property.

MultiUnitBuilding: number of units in the building, whether the building has elevator, floor-space (square meters)

- The Cottage class extends the House class. In addition to the attributes of the Property base class and the House class, the cottage class also has an attribute called *lakeFrontage.*
- The Farm class extends the Land class. In addition to the attributes of the Property base class and the Land class, the Farm class also has an attribute called *crop* which is a string attribute that represents the type of crop that is grown at the farm. For example, if it is a wheat farm, crop would be set to the string "wheat". You can assume that the value for *crop* can be any string value - please ensure that there is no attempt to set *crop* to a number (i.e. int, double, long or float).
- The Office class extends the MultiUnitBuilding class. In addition to the attributes of the Property base class and the MultiUnitBuilding class, the office class has an attribute called *classification*.
  The *classification* attribute is of type *OfficeType* which is an enum that is defined. The *OfficeType* enum has 3 "constant" values: SERVICE, SALES, INDUSTRIAL. Think about: why did we choose to use an enum? Is it necessary to check the values of *classification* for legality? Why or why not?

- The ApartmentBuilding class extends the MultiUnitBuilding class. In addition to the attributes of the Property base class and the MultiUnitBuilding class, the ApartmentBuilding class has an integer attribute representing the number of tenants in the building, an integer attribute representing the number of units in the building, as well as a double attribute representing the floor-space (in square meters) of each apartment.

Constructors:

The Property class must have a constructor that takes the listing price (double) of the property. The listing price *is in thousands of dollars*. Throw an exception object if the listing price is less than zero dollars.

The Land class must have a constructor that takes (in order) the listing price of the land and the number of hectares (double). Throw an exception object if the number of hectares of the property is less than or equal to 0.

The Farm class must have a constructor that takes (in order) the listing price, number of hectares of the property (double) and the crop that is grown on the farm (a string). Throw an exception object if the crop type is the *empty string* ("") or equal to *null*.

The MultiUnitBuilding class must have a constructor that takes (in order) the listing price, the floor-space in square meters (double), the number of units in the building (int) and whether the building has an elevator (boolean). Throw an exception object if the number of units is less than or equal to 0 and/or the floor-space is less than or equal to 0 square meters.

The Office class must have a constructor that takes (in order) the listing price (double), the floor space in square-meters (double), the number of units in the office (int), whether the office has an elevator (boolean) and the classification of the office. The classification of the office is represented by an enum called OfficeType with the values SERVICE, SALES, INDUSTRIAL. Throw an exception object if the number of units is less than or equal to 0 and/or the floor-space is less than or equal to 0 square meters.

The ApartmentBuilding class must have a constructor that takes (in order) the listing price, the floor space in square-meters (double), the number of apartment units in the building (int) and the number of tenants (int). Every apartment building has an elevator, so you can automatically set that value to *True.* Throw an exception object if the number of units is less than or equal to 0 and/or the floor-space is less than or equal to 0 square meters.

The House class must have a constructor that takes (in order) the listing price, the floor space in square-meters (double), number of bedrooms (int), number of bathrooms (int), width of the property in meters (double) and the depth of the property in meters (double) and the classification of the house. The classification of the house is represented by an enum called HouseType with the values TOWNHOUSE, DETACHED, DUPLEX, WATERFRONT_HOME. Throw an exception object if the number of bedrooms or the number of bathrooms is equal to 0.

The Cottage class must have a constructor that takes (in order) the listing price, the floor space in square-meters, number of bedrooms, number of bathrooms, width of the property in meters, the depth of the property in meters and the lake frontage of the property in meters. The classification of the cottage (home) should be set by default to: WATERFRONT_HOME. Throw an exception object if the lake frontage amount is equal to or less than 0.

toString: All the classes (except for the Property class) have `toString` methods. The testing program below uses these `toString` methods in its display. Please make your `toString` result looks as much as possible like what is shown in the display; use the screen-prints for reference. Make sure that each `toString` includes the property type along with the values of all of the instance variables. Here's an example of the output for the output of the toString method in the Home and Office classes.

```
Home, 22 m width by 33 m deep lot, 2 bedrooms, 3000 square meter
building, asking price $110,000.
Office, Industrial purpose, 3 units, 1500 square meter building,
no elevator, asking price $120,000.
```

compareTo: Your program must include a compareTo method that will compare two properties (of any property type) based on their listing prices.

```
          ArrayList<Property> propertyDB = new
ArrayList<Property>();

           Property prop3 = new Office(120, 3, 1500, true,
OfficeType.SERVICE);
           propertyDB.add(prop3);
           Property prop4 = new Farm(250, 3400, "vegetables");
           propertyDB.add(prop4);
           System.out.println(prop3.compareTo(prop4));
```

Invoking the compareTo method that you wrote to compare *prop3* and *prop4* would return -130 (250-120) and

print out the following message: `The Farm is more expensive than the Office. Difference is $130.`

tax: You must write a method that is used to calculate the tax for each of the property types – House, Cottage, Land, Farm, Office and Apartment Building. The *calculateTax* method should be an **abstract method** because the tax is calculated differently for each of the various property types.
The *calculateTax* method can be declared in the Property class but should be implemented within each of the derived classes based on how the tax is calculated for that specific property type.

HERE IS THE FORMULA FOR CALCULATING THE ANNUAL TAX FOR EACH PROPERTY TYPE:

House: The tax starts at $1000 plus $50 per bedroom plus $10 per square-meter of floor-space.

Cottage: The tax for a cottage starts at $2000 dollars. $2 is added to the tax for every meter of lake frontage.

Land: The tax for a vacant land property is $100 dollars per hectare.

Farm: The tax for a farm property is $50 dollars per hectare of property.

Office: The tax for an office is $10 per square-meter of floor-space with an additional fee of $20 per unit and a $50 fee if the office has an elevator. There is a 15% discount if the Office is used for an INDUSTRIAL purpose and a 5% discount if the office is used for a SERVICE purpose.

Apartment Building: The tax for an apartment building is $35 dollars per square foot of floor-space in each apartment unit.

TESTING YOUR PROGRAM:

You will need to write a testing class to test the functionality of your classes. Here's a sample of what your testing class may look like. You can create an ArrayList of type *Property* to store your *Property* objects. Be sure that you try passing in both legal and illegal values to your constructor to test the exception class. You should also test the accessors/mutators and the toString and compareTo methods for each of your classes. The testing class that you write should demonstrate that all your classes meet the specifications and perform all of the tasks correctly. The TAs will run your testing class as part of the evaluation of your program.

```
          ArrayList<Property> propertyDB = new
ArrayList<Property>();
            Property prop3 = new Office(120, 3, 1500, true,
OfficeType.SERVICE);
            propertyDB.add(prop3);
            Property prop4 = new Farm(250, 3400, "vegetables");
            propertyDB.add(prop4);
            System.out.println(prop3.compareTo(prop4));
            for (Property element : db) {
                 System.out.println(element);
                 System.out.println("Listing Price = " +
String.format("%.3f", element.getListingPrice());
```

Object Oriented Programming

- Your code should carefully follow the principles of encapsulation, information hiding and inheritance.
  Attributes should be private - not public.
- Each of your classes should provide accessor ("getter") and mutator ("setter") methods for each of the instance variables of the class.
- Throw a BadProperty exception object if an attempt is made to set or alter any of the attributes to illegal values.
- The BadProperty exception class only needs the one constructor that accepts a String type message
- Try to avoid duplicated code as much as you can.
- Some classes should be concrete while others should, naturally, be abstract.
- Using constants makes your code safer and cleaner.