

Ch 13 Project (Ver 2)

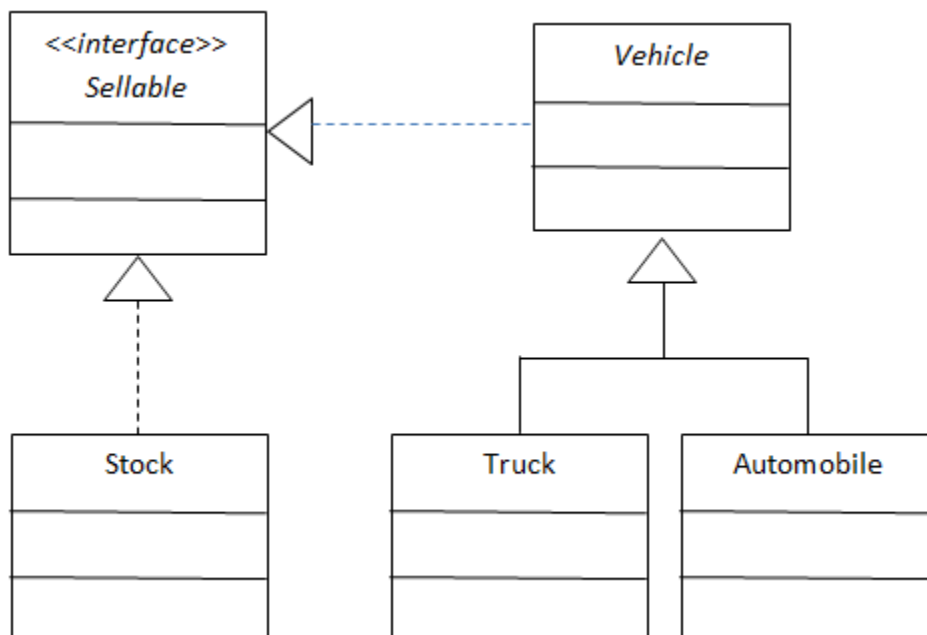
[Start Assignment](#)

Due Thursday by 11:59pm **Points** 100 **Submitting** a file upload
Available until Jul 23 at 11:59pm

Choose Version 1 or Version 2

THIS ASSIGNMENT IS COPYRIGHTED MATERIAL AND THE INTELLECTUAL PROPERTY OF THE AUTHOR. DO NOT POST ANY PART OF THE ASSIGNMENT TO THE INTERNET (OR ANYWHERE ELSE). DO NOT POST ANY PART OF A SOLUTION PROVIDED BY YOUR INSTRUCTOR OR YOUR OWN SOLUTION TO THE INTERNET (OR ANYWHERE ELSE). POSTING OF ANY PART OF THIS MATERIAL IS COPYRIGHT INFRINGEMENT AND THEFT OF INTELLECTUAL PROPERTY. ALL LEGAL AND COLLEGE REMEDIES WILL BE PURSUED TO PROSECUTE THOSE IN VIOLATION.


The purpose of this assignment is to use polymorphism with objects that are not related by inheritance. You will need to create an interface to describe things that are "sellable". That is, items that can be sold. To use this interface you will create classes for two types of things that are sold - stocks and vehicles (automobiles and trucks). Vehicles will be modeled using a hierarchy of classes for different types of vehicles. There will be a single class for stocks that are sold. An abbreviated version of the overall UML class diagram is shown below. Notice Sellable is an interface and Vehicle is an abstract class.



The UML class diagram for each service class and interface is provided below.

- [Class diagrams for Chapter 13 project](#)

The classes are all service classes. Each service class will have a custom exception class to indicate errors in parameters passed to the methods of that class. In addition you will need to create an application class named ManageSellables. Objects of the service classes will be processed polymorphically in the ManageSellables class. Observe how you can store any of the service class objects in a properly created ArrayList. The detail requirements for each class are provided below.

1. Create an Eclipse project named ManageSellables. Use a package name of edu.seminolestate.managesellables. (You can substitute your domain name for edu.seminolestate if you wish.) Add one class to this package. The class name is ManageSellables. This class will have the main method. More requirements for this class are listed below.
2. Create a second package named edu.seminolestate.sellable. Create an interface in this package named Sellable. Create one public abstract method in this interface named getSaleAmount. This method should return a double and have no parameters.
-  3. Create a third package named edu.seminolestate.stock. Add a class named Stock to this package. This class represents a stock you could purchase and resell. The following requirements apply to the Stock class.
 - a. Create the private instance variables as shown on the UML class diagram.
 - b. The Stock class must implement the Sellable interface. The getSaleAmount method should return the number of shares times the price per share. You will need to import the edu.seminolestate.sellable package.
 - c. Create get and set methods for each private variable. Implement the following edits: the certificate number cannot be null and must have a length greater than zero. The price per share must be greater than zero. The number of shares must be greater than zero. The purchased date cannot be null. Throw an IllegalArgumentException if any of the values are invalid. Pass a specific, descriptive error message to the IllegalArgumentException constructor.
 - d. Create a single constructor that has parameters for the certificate number, price per share, number of shares, purchase date, and sell date. Call the set methods from the constructor using these parameters to update the class variables so the edits only have to be coded once.
 - e. Create a toString method that displays the class name and the values of each instance variable.
4. Create the IllegalArgumentException per the UML class diagram. This class must be a subclass of Exception (making it a "checked" exception). Place this class in the edu.seminolestate.stock package.
5. Create a fourth package named edu.seminolestate.vehicle. This package will contain the following classes: Vehicle, Truck, Automobile, and IllegalVehicleArgumentException. These are described below.

6. The following requirements apply to the Vehicle class.
- Create the private instance variables as shown on the UML class diagram.
 - The Vehicle class must implement the Sellable interface. Do NOT implement the `getSaleAmount` method. This will make the Vehicle class an abstract class. You will still need to import the `edu.seminolestate.sellable` package.
 - Create a single constructor that has parameters for the class variables as shown in the UML class diagram. Call the set methods (described below) passing the parameters to update the class variables.
 - Create get and set methods for each private variable. Implement the following edits: the VIN, make, and model cannot be null nor a length less than one; the purchase price, mileage any model year must be greater than zero. Throw an `IllegalVehicleArgumentException` if any of the values are invalid.
 - Create a `toString` method that displays the class name and the values of each of the instance variables.
7. The following requirements apply to the Truck class.
- Make the Truck class a subclass of Vehicle. This class doesn't need to implement the Sellable interface since its superclass does.
 - Create the private instance variables as shown on the UML class diagram.
 - Create a single constructor that has parameters for the class variables as shown in the UML class diagram. Call the set methods (described below) passing the parameters to update the class variables.
 - Create get and set methods for each private variable. Implement the following edits: the cargo capacity must be greater than zero; the horsepower must be greater than 0; the number of axles must be greater than zero. Throw an `IllegalVehicleArgumentException` if any of the values are invalid.
 - Override (implement) the `getSaleAmount`. Calculate the sale amount as 90% of the purchase price if the truck model year is the same as the current year. For each year after, reduce the original sales price by 4% to arrive at the sales price. For example, assume a 2014 Ford F100 truck was originally purchased for \$32,000. The sale amount if sold in 2014, 2015, 2016, or 2017 are shown below.
- | If sold in ... | Sale Amount |
|----------------|-------------------------------|
| 2014 | $32,000.00 * .9 = 28,800.00$ |
| 2015 | $28,800.00 * .96 = 27,648.00$ |
| 2016 | $27,648.00 * .96 = 26,542.08$ |
| 2017 | $26,542.08 * .96 = 25,480.40$ |
- Create a `toString` method that displays the data from the superclass and the class name and the values of each of the instance variables in the Truck class.
8. The following requirements apply to the Automobile class. This class doesn't need to implement the Sellable interface since its superclass does.

- a. Make the Automobile class a subclass of Vehicle. This class doesn't need to implement the Sellable interface since its superclass does.
- b. Create the private instance variables as shown on the UML class diagram.
- c. Create a single constructor that has parameters for the class variables as shown in the UML class diagram. Call the set methods (described below) passing the parameters to update the class variables.
- d. Create get and set methods for each private variable. Implement the following edits: Throw an IllegalArgumentException if any of the values are invalid.
- e. Override (implement) the getSaleAmount. Calculate the sale amount as 85% of the purchase price if the automobile model year is the same as the current year. For each year after, reduce the original sales price by 9% to arrive at the sales price. For example, assume a 2013 Toyota Corolla automobile was originally purchased for \$22,000. The sale amount if sold in 2013, 2014, 2015, 2016, or 2017 are shown below.

If sold in ...	Sale Amount
2013	22,000.00 * .85 = 18,700.00
2014	18,700.00 * .91 = 17,017.00
2015	17,017.00 * .91 = 15,485.47
2016	15,485.47 * .91 = 14,091.78
2017	14,091.78 * .91 = 12,823.52



- f. Create a toString method that displays the data from the superclass and the class name and the values of each of the instance variables in the Automobile class.
9. Create the IllegalArgumentException per the UML class diagram. This class must be a subclass of Exception (making it a "checked" exception). Place this class in the edu.seminolestate.vehicle package.
 10. The following requirements apply to the ManageSellable application class. This class is in the edu.seminolestate.managesellables package.
 - a. Import all the classes from the following packages: edu.seminolestate.vehicle, edu.seminolestate.stock. Also import the edu.seminolestate.sellable package.
 - b. This class will have the main method. Create an ArrayList to store Sellable objects.
 - c. Display a menu that has choices for: 1 - Add stock; 2 - Add truck; 3 - Add automobile; 4 - Display all sales; 5 - Exit. Display an error message and repeat the menu if some other value is entered. This will require you to catch the appropriate exception if letters instead of numbers are entered. The application must continue to display the menu and process these options until the user enters a 5.
 - d. When the user chooses menu option 1, prompt for the user to enter the appropriate data. Read the data using a Scanner and System.in. Implement the following edits: the certificate number cannot be null and must have a length greater than zero. The price per share must be greater than zero. The number of shares must be greater than zero. The purchased date

cannot be null or an invalid date. The sold date cannot be null or an invalid date. If any one piece of data is invalid prompt the user to re-enter that data. Continue to prompt the user until a valid value is entered. If all the data is valid, create a Stock object and store it in the ArrayList. Catch the `IllegalStockArgumentException` when calling the constructor.

- e. When the user chooses menu option 2, prompt for the user to enter the appropriate data. Read the data using a Scanner and `System.in`. Implement the following edits: the VIN can't be blank, the purchase price must be greater than 0, the make can't be blank, the model can't be blank, the model year must be an integer and greater than 1900, the mileage must be an integer and greater than 0, the cargo capacity must be greater than zero; the horsepower must be greater than 0; the number of axles must be greater than zero. If any one piece of data is invalid, prompt the user to re-enter that data. If all the data is valid, create a Truck object and store it in the ArrayList. The Truck class can throw an `IllegalVehicleArgumentException` so write code to catch that exception.
- f. When the user chooses menu option 3, prompt for the user to enter the appropriate data. Read the data using a Scanner and `System.in`. Implement the following edits: the VIN can't be blank, the purchase price must be greater than 0, the make can't be blank, the model can't be blank, the model year must be an integer and greater than 1900, the mileage must be an integer and greater than 0, the number of seats must be greater than zero. If any piece of data is invalid prompt the user to re-enter that data. If all the data is valid, create an Automobile object and store it in the ArrayList. The Automobile class can throw an `IllegalVehicleArgumentException` so write code to catch that exception.
- g. When the user chooses menu option 4, list all the objects contained in the ArrayList by calling their `toString` method. In addition, display the sale amount by calling their `getSaleAmount` method. Format the sale amount as currency.
- h. Display a closing message and end the application when the user chooses option 5.
- i. Do not add a throws clause to the main method. In other words, use try, catch for any exceptions that make arise in the `ManageSellables` application class.

11. Use a Scanner object for all user input. Use `System.out` for all user output.

12. Add a comment to each source code file for your name and the date.

13. Here is a summary of the packages required in this assignment and what classes or interfaces should be in each package.

- edu.seminolestate.managesellables: `ManageSellables`
- edu.seminolestate.sellable: `Sellable`
- edu.seminolestate.stock: `Stock`, `IllegalStockArgumentException`
- edu.seminolestate.vehicle: `Vehicle`, `Truck`, `Automobile`, `IllegalVehicleArgumentException`

14. This assignment is designed based on the material covered to this point in the text book and online notes. Use only features and code covered to this point in the book, online notes, or Discussions. Failure to follow this rule may result in a zero for your assignment. [Why?](#)


15. You can resubmit your assignment any time up to the deadline. Projects cannot be submitted late. Please zip ALL your project files and attach the .zip file to the assignment drop box.




Ch 13 Rev 2 Rubric



Criteria	Ratings		Pts
Create an Eclipse project named ManageSellables. Use a package name of edu.seminolestate.managesellables. Add one class to this package. The class name is ManageSellables.	2 pts Full Marks	0 pts No Marks	2 pts
Create a second package named edu.seminolestate.sellable. Create an interface in this package named Sellable. Create one public abstract method in this interface named getSaleAmount. This method should return a double and have no parameters.	4 pts Full Marks	0 pts No Marks	4 pts
Create a third package named edu.seminolestate.stock. Add a class named Stock to this package. Create the private instance variables as shown on the UML class diagram. The Stock class must implement the Sellable interface. The getSaleAmount method should return the number of shares times the price per share.	5 pts Full Marks	0 pts No Marks	5 pts
In the Stock class create get and set methods for each private variable. Implement the required edits. Throw the InvalidStockArgumentException for errors.	5 pts Full Marks	0 pts No Marks	5 pts
In the Stock class create a single constructor that has parameters for the certificate number, price per share, number of shares, purchase date, and sell date. Call the set methods from the constructor using these parameters to update the class variables so the edits only have to be coded once. Specify the IllegalStockArgumentException is thrown.	3 pts Full Marks	0 pts No Marks	3 pts
In the Stock class create a toString method that displays the class name and the values of each instance variable.	2 pts Full Marks	0 pts No Marks	2 pts
Create the Vehicle class in the edu.seminolestate.vehicle package with the private instance variables as shown on the UML class diagram. The Vehicle class must implement the Sellable interface. Do NOT implement the getSaleAmount method. This will make the Vehicle class an abstract class.	5 pts Full Marks	0 pts No Marks	5 pts
In the Vehicle class create a single constructor that has parameters for the class variables as shown in the UML class diagram. Call the set methods passing the parameters to update the class variables. Specify the IllegalVehicleArgumentException is thrown.	3 pts Full Marks	0 pts No Marks	3 pts
In the Vehicle class create get and set methods for each private variable. Implement the required edits: Throw an IllegalVehicleArgumentException if any of the values are invalid.	3 pts Full Marks	0 pts No Marks	3 pts

Criteria	Ratings		Pts
In the Vehicle class create a toString method that displays the class name and the values of each of the instance variables.	2 pts Full Marks	0 pts No Marks	2 pts
Create the Truck class in the edu.seminolestate.vehicle package. Make the Truck class a subclass of Vehicle. This class doesn't need to implement the Sellable interface since its superclass does. Create the private instance variables as shown on the UML class diagram.	5 pts Full Marks	0 pts No Marks	5 pts
In the Truck class create a single constructor that has parameters for the class variables as shown in the UML class diagram. Call the set methods (described below) passing the parameters to update the class variables. Specify the IllegalArgumentException is thrown.	3 pts Full Marks	0 pts No Marks	3 pts
 In the Truck class create get and set methods for each private variable. Implement required edits. Throw an IllegalArgumentException if any of the values are invalid.	5 pts Full Marks	0 pts No Marks	5 pts
In the Truck class override (implement) the getSaleAmount. Calculate the sale amount as 90% of the original sales price if the truck model year is the same or one year older than the current year. For each year after, reduce the original sales price by 4% to arrive at the sales price.description of criterion	5 pts Full Marks	0 pts No Marks	5 pts
In the Truck class create a toString method that displays the data from the superclass and the class name and the values of each of the instance variables in the Truck class.	2 pts Full Marks	0 pts No Marks	2 pts
Create the Automobile class in the edu.seminolestate.vehicle package. This class doesn't need to implement the Sellable interface since its superclass does. Make the Automobile class a subclass of Vehicle. Create the private instance variables as shown on the UML class diagram.	5 pts Full Marks	0 pts No Marks	5 pts
In the Automobile class create a single constructor that has parameters for the class variables as shown in the UML class diagram. Call the set methods passing the parameters to update the class variables. Specify the	3 pts Full Marks	0 pts No Marks	3 pts

Criteria	Ratings		Pts
IllegalVehicleArgumentException is thrown.			
In the Automobile class create get and set methods for each private variable. Implement the required edits. Throw an IllegalVehicleArgumentException if any of the values are invalid.	3 pts Full Marks	0 pts No Marks	3 pts
In the Automobile class override (implement) the getSaleAmount. Calculate the sale amount as 85% of the original sales price if the automobile model year is the same or one year older than the current year. For each year after, reduce the original sales price by 9% to arrive at the sales price.	3 pts Full Marks	0 pts No Marks	3 pts
In the Automobile create a toString method that displays the data from the superclass and the class name and the values of each of the instance variables in the Automobile class.	2 pts Full Marks	0 pts No Marks	2 pts
 Create the IllegalVehicleArgumentException in the edu.seminolestate.vehicle package per the UML class diagram. Create the IllegalStockArgumentException in the edu.seminolestate.stock package per the UML class diagram.	4 pts Full Marks	0 pts No Marks	4 pts
Create the ManageSellable application class. This class is in the edu.seminolestate.managesellables package. This class will have the main method. Create an ArrayList to store Sellable objects.	2 pts Full Marks	0 pts No Marks	2 pts
Display a menu. Catch the appropriate exception if letters instead of numbers are entered. The application must continue to display the menu and process these options until the user enters a 5. Values outside 1 through 5 must generate an error message.	3 pts Full Marks	0 pts No Marks	3 pts
When the user chooses menu option 1, prompt for the user to enter the appropriate data to create a Stock object. Read and edit the data per the required edits. If any one piece of data is invalid prompt the user to re-enter that data. Continue to prompt the user until a valid value is entered. If all the data is valid, create a Stock object and store it in the ArrayList. Catch the IllegalStockArgumentException when calling the constructor.	5 pts Full Marks	0 pts No Marks	5 pts
When the user chooses menu option 2, prompt for the user to enter the appropriate data. Read and edit the data per the required edits. If any one piece of data is invalid prompt the user to re-enter that data. Continue to prompt the user until a valid value is entered. When all the data is valid, create a Truck object and store it in the ArrayList. Catch the IllegalVehicleArgumentException when calling the constructor.	5 pts Full Marks	0 pts No Marks	5 pts

Criteria	Ratings		Pts
When the user chooses menu option 3, prompt for the user to enter the appropriate data to create an Automobile object. Read and edit the data per the required edits. If any one piece of data is invalid prompt the user to re-enter that data. Continue to prompt the user until a valid value is entered. When all the data is valid, create an Automobile object and store it in the ArrayList. Catch the <code>IllegalArgumentException</code> when calling the constructor.	5 pts Full Marks	0 pts No Marks	5 pts
When the user chooses menu option 4, list all the objects contained in the ArrayList by calling their <code>toString</code> method. In addition, display the sale amount by calling their <code>getSaleAmount</code> method. Format the sale amount as currency.	3 pts Full Marks	0 pts No Marks	3 pts
Display a closing message and end the application when the user chooses option 5.	1 pts Full Marks	0 pts No Marks	1 pts
 Add a comment to each source code file for your name and the date.	2 pts Full Marks	0 pts No Marks	2 pts
Total Points: 100			