CS 111, Programming Fundamentals II
Lab 6: Polymorphism and Abstract Classes

Computer Science

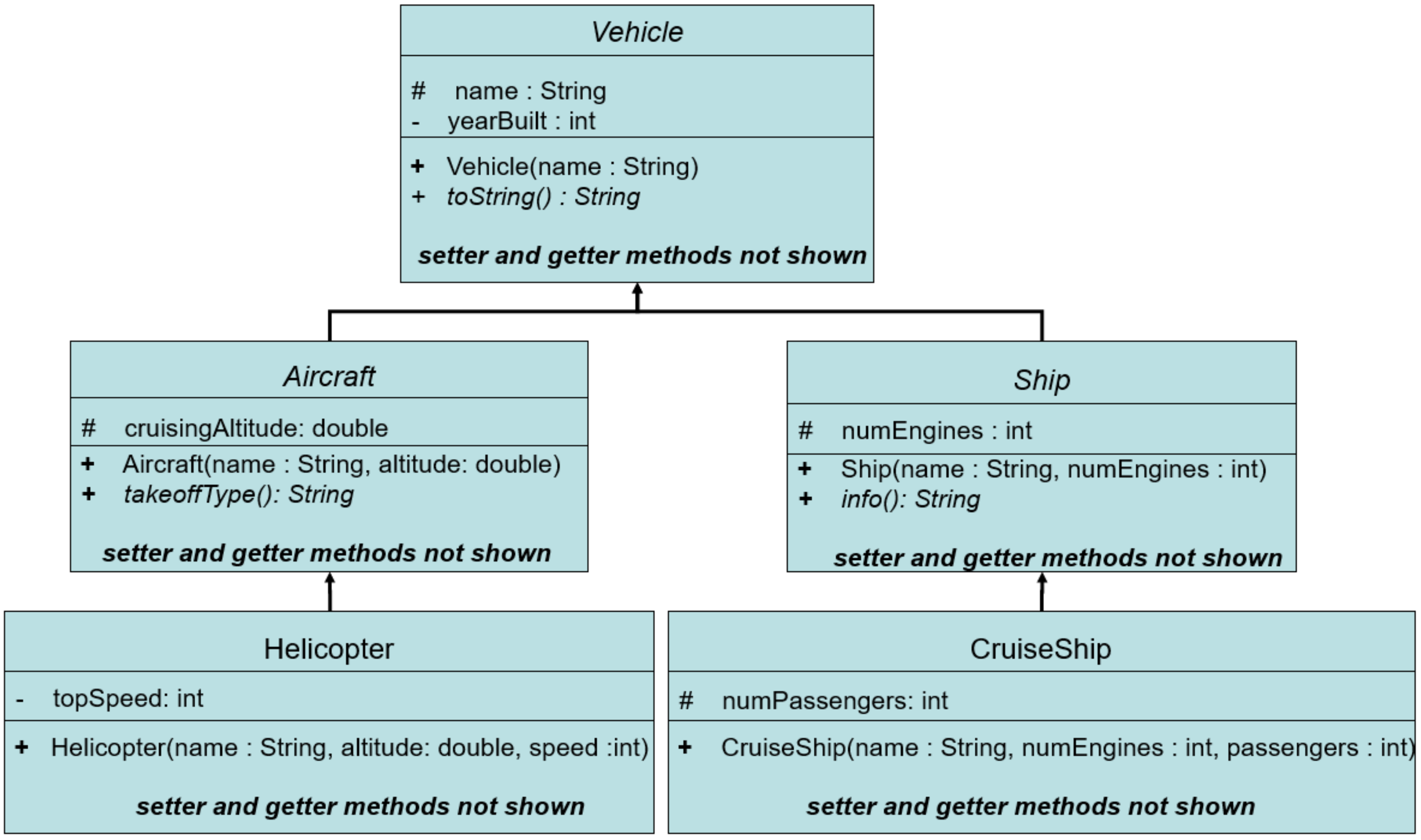
This lab is meant to further your understanding of inheritance and polymorphism, and give you hands-on experience with abstract classes.

I. Inheritance and Abstract Classes

Explore the UML diagram below. In lecture we went over how the names of abstract classes are italicized in UML diagrams. The class **Vehicle** is an abstract class. The **Ship** and **Aircraft** classes inherit from the **Vehicle** class, and they are themselves are abstract as well. Abstract methods are also italicized in an UML diagram. The class **Vehicle** has an abstract method ***toString()***. Couple of things to keep in mind as you are working on this lab :

- An abstract class cannot be instantiated explicitly. You have to extend an abstract class and then you are able to instantiate objects of the subclass.

- If the parent class does not have a no-argument constructor, then the child class has to call the parent's class constructor with the keyword super.



Vehicle
# name : String
- yearBuilt : int
+ Vehicle(name : String)
+ *toString() : String*
setter and getter methods not shown
Aircraft
# cruisingAltitude: double
+ Aircraft(name : String, altitude: double)
+ *takeoffType(): String*
setter and getter methods not shown
Ship
# numEngines : int
+ Ship(name : String, numEngines : int)
+ *info(): String*
setter and getter methods not shown
Helicopter
- topSpeed: int
+ Helicopter(name : String, altitude: double, speed :int)
setter and getter methods not shown
CruiseShip
# numPassengers: int
+ CruiseShip(name : String, numEngines : int, passengers : int)
setter and getter methods not shown

Implement all five classes as described in the UML. You don't need to write the setter and getter methods for each field, only include them as you need to. You will need to include a setter and getter method for one of the fields. Later you will notice that you don't have a way to set/retrieve a value for one of the fields. Most of the fields are set through the constructors. Keep in mind that **Vehicle**, **Aircraft**, and **Ship** are all abstract classes. Make sure you give all of the fields/methods correct access modifiers.

All of these classes are really short and can be written between 15- 25 lines of code, including spaces and brief comments. Below are some hints and notes for each class in the UML diagram.

1. **Vehicle class**
    - The class **Vehicle** is italicized which means it is abstract. You have to use the keyword abstract to indicate that the class is abstract.

    - The constructor only takes in one parameter, a value of the ***name*** of the **Vehicle**. You will need to figure out how to set the other field named ***yearBuilt***. There are multiple ways to accomplish this, the UML is vague on purpose, giving to flexibility with the implementation.

    - The method ***toString()*** is also abstract, which means you have use the keyword abstract to indicate that it is abstract. Abstract methods end with a semicolon and do not a body. Each subclass will have to override it.

    - This class is the superclass, and it can be compiled as soon as you write. Compile the class and fix any syntax errors if needed before moving on.

2. **Aircraft class**
    - The class **Aircraft** inherits from **Vehicle** and it is also abstract. Make sure you include both keywords extends and abstract in the class declaration.

    - Even though this class extends an abstract class, it does not need to override the abstract methods of the parent class, because this class is also abstract. Rather that class which extends **Aircraft**, will have to override all of the abstract methods in the hierarchy.

    - You will have to invoke the superclass constructor and pass a value of type String for the ***name***. This can be accomplished with the super keyword. This needs to be done because **Vehicle** (parent class) does not have a default constructor.

3. **Helicopter class**
    - The class **Helicopter** is the first "concrete" class, it inherits from **Aircraft**.

    - Just like in the **Aircraft** class you passed some of the parameters to the parent class, do the same here by using the keyword super.

    - Inside of this class you have to override the methods ***toString()*** and t***akeoffType***(). Have a look at the sample output, for a better understating of how to implement these methods. You might need to add a getter and setter method for one of the fields in the parent class.

4. **Ship class**
   - The class **Ship** is very similar in structure to the **Aircraft** class. This class is abstract and inherits from **Vehicle**, just like the **Aircraft** class. Check the UML and compare **Aircraft** and **Ship** classes, refer to the **Aircraft** class for hints.

5. **CruiseShip class**
   - The class **CruiseShip** is very similar in structure to the **Helicopter** class. Check the UML and compare **Helicopter** and **CruiseShip** classes, refer to the **Helicopter** class for hints.

   - This class has to override the methods ***toString()*** and ***info()***. It is the first "concrete" class in its hierarchy. Have a look at the sample output for a better understanding of what to include in the ***toString()*** and ***info()*** methods.

II. Polymorphism

Recall that polymorphism, which means "many forms" refers to a superclass reference variable being able to reference objects of a subclass. Refer to the lecture slides for examples and for more information if needed.

For this part of the lab write a class named **VehicleDemo**. In the main method of this class do the following:

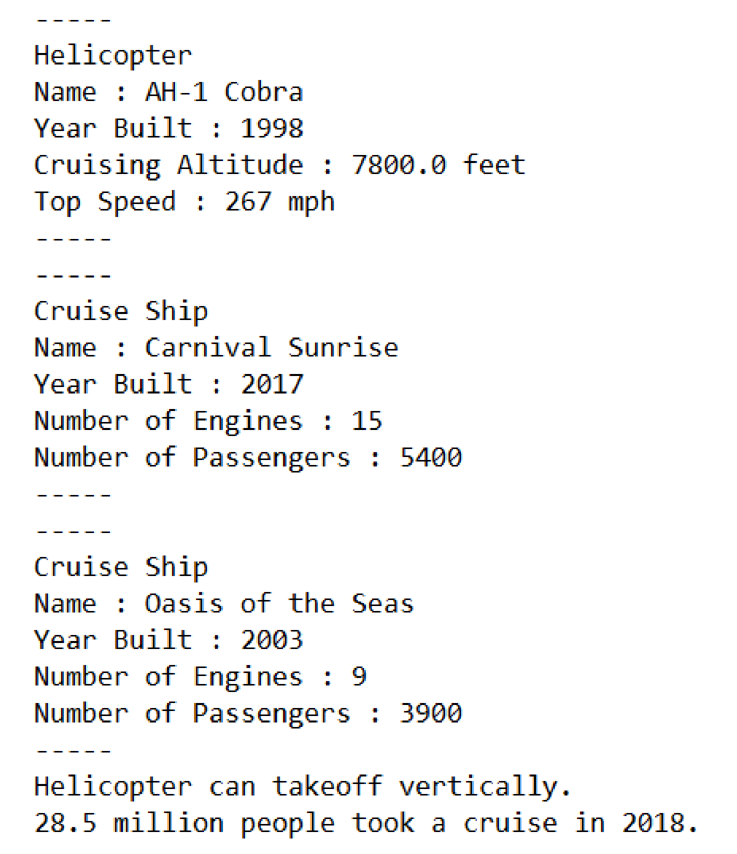
1. Create a few objects of type **CruiseShip** and **Helicopter**. Refer to the sample output for basic values which can be used to create these objects. Have at least **3 objects**. One of the objects has to be of type **Helicopter**, and another object has to be of type **CruiseShip**. It is up to you for the third object.

2. Create an array of type **Vehicle** and populate it with the newly created objects of type **Helicopter** and **CruiseShip**. This is allowed because both **CruiseShip** and **Helicopter** are subclasses for **Vehicle**.

3. Using a for loop print out each object in the **Vehicle** array. Keep in mind that the method ***toString()*** is invoked automatically when an objects reference variable is passed into a ***System.out.print*** or ***System.out.println***.

4. Call the method ***takeoffType()*** on an object of type **Helicopter**, and the method ***info()*** on an object of type **CruiseShip**.

5. Compile and run your code. Sample output can be seen in **Figure 1**.



```
-----
Helicopter
Name : AH-1 Cobra
Year Built : 1998
Cruising Altitude : 7800.0 feet
Top Speed : 267 mph
-----
-----
Cruise Ship
Name : Carnival Sunrise
Year Built : 2017
Number of Engines : 15
Number of Passengers : 5400
-----
-----
Cruise Ship
Name : Oasis of the Seas
Year Built : 2003
Number of Engines : 9
Number of Passengers : 3900
-----
Helicopter can takeoff vertically.
28.5 million people took a cruise in 2018.
```

Figure 1: Sample Output

III. Upload your work to Canvas

For this lab, make sure that you upload the following files to the Lab 6 assignment in your Canvas account:

> ***Vehicle.java***, ***Aircraft.java***, ***Ship.java***, ***Helicopter.java***, ***CruiseShip.java***, *and* ***VehicleDemo.java***
> *Screenshot image containing the output*

Rubric

File / Lab	**Points**
Class *Vehicle* is implemented correctly and as described in the UML	10
Class *Aircraft* is implemented correctly and as described in the UML	10
Class *Ship* is implemented correctly and as described in the UML	10
Class *Helicopter* is implemented correctly and as described in the UML	20
Class *CruiseShip* is implemented correctly and as described in the UML	20
Class *VehicleDemo* is implemented according to the 4 steps in part II	25
Code is indented, commented, compiles/runs and a screenshot of the output is provided	5
Total	100