Lab Inheritance CSIS-1410

Create the following 4 public classes: *Vehicle, Car, Plane,* and *Ship* from the ***Inheritance Hierarchy Class Diagram***, listed at the bottom. Create a main class InheritanceApp that instantiates (creates) an instance (object) of each of the *Vehicle* subclasses (*Car, Plane,* and *Ship*) and displays each instance to resemble the output example at the bottom.

- **Vehicle:**
  has two instance fields, two parameterized constructors, get and set methods (getters and setters) for the instance fields, and three public methods: start, stop, and an overridden toString method
  - start has no parameters and returns void
    it prints "starting . . . "
  - stop has no parameters and returns void
    it prints "stopping . . . "
  -
  - toString returns the String "`.. number of seats: `" and the field value indicating the number of seats.
    Eclipse Tip: in a new line type "t", then press Ctrl + SpaceBar (options are shown) select the toString method and press Enter ;
    This gives you a template for the toString method including the @Override annotation

- **Car:** derives from Vehicle
  has one constant field called `numberOfCylinders` of type int
  has one constructor that receives three arguments to initialize the `numberOfCylinders` and passes the other arguments to the super constructor to initialize the other fields.
  has one overridden method toString
  - toString returns the String "`Car with numberOfWheels wheels and numberOfCylinders cylinders`" - depending on the value of the fields, followed by the output of the super toString method
    suggestion: use the get methods for the values whenever possible

- **Plane:** derives from Vehicle
  has one constant field called `wingspan` of type int
  has one constructor that receives two arguments to initialize the `wingspan` and passes the other argument to the super constructor to initialize the other field.
  has one overridden method toString
  toString returns the String "`Plane with numberOfWheels wheels and wingspan wingspan ft`" - depending on the value of the field, followed by the output of the super toString method

- **Ship:** derives from Vehicle
  has one constant field called `hasSail` of type boolean
  has one constructor that receives two arguments to initialize the `hasSail` and passes the other argument to the super constructor to initialize the other field.
  has one overridden method toString
  - toString returns the String "`Ship with/without`" - depending on the value of the field, followed by the output of the super toString method

- **InheritanceApp:**
  includes the main method.
  In the main method create
  - an instance of ***Car***, then output the contents of the instance and call the ***start*** and ***stop*** methods.
  - an instance of ***Plane***, then output the contents of the instance and call the ***start*** and ***stop*** methods.
  - an instance of ***Ship***, then output the contents of the instance and call the ***start*** and ***stop*** methods.

Sample output:

```
Car with 4 wheels and 4 cylinders  .. number of seats: 5
starting ..
stopping ..

Plane with 3 wheels and wingspan 23ft  .. number of seats: 30
starting ..
stopping ..

Ship without sail  .. number of seats: 8000
starting ..
stopping ..

Car with 4 wheels and 6 cylinders  .. number of seats: 6
starting ..
stopping ..

Plane with 3 wheels and wingspan 13ft  .. number of seats: 10
starting ..
stopping ..

Ship with sail  .. number of seats: 8
starting ..
stopping ..
```

The table ***Inheritance Rules*** will help to understand the basic rules of object-oriented inheritance.

Inheritance Rules			
Superclass Members	**Inherited by subclass?**	**Directly Accessible by Subclass?**	**Directly Accessible by Client of Subclass?**
public fields	yes	yes, by using field name	yes
public methods	yes	yes, by calling method from subclass methods	yes
protected fields	yes	yes, by using field name	no, must call accessors and mutators
protected methods	yes	yes, by calling method from subclass methods	no
private fields	no	no, must call accessors and mutators	no, must call accessors and mutators
private methods	no	no	no
Superclass Members	**Inherited by subclass?**	**Directly Accessible by Subclass?**	**Directly Accessible by Client of Subclass Using a Subclass Reference?**
constructors	no	yes, using super(arg list) in a subclass constructor	no
public or *protected* inherited methods that have been overridden in the subclass	no	yes, using super.methodName(arg list)	no

Inheritance Hierarchy

Class Diagram