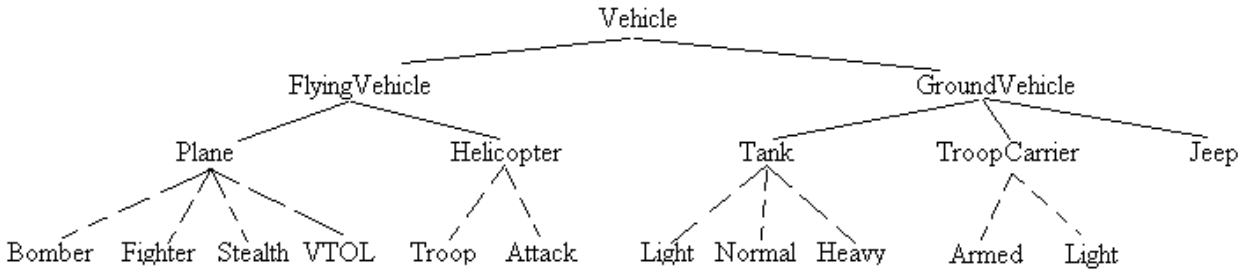
CSC 360 Programming Assignment #2
Due date: Monday, September 8

In this assignment, you will create a Vehicle class and then subclasses to demonstrate the use of inheritance. The vehicles defined will all be military vehicles for use in an armed conflict scenario to determine the best vehicle(s) for that scenario. Below is the hierarchy of classes to implement. The specifications for the classes are described below along with the UML on the next page.



The subclasses of Plane, Helicopter, Tank and TroopCarrier are annotated by dashed lines. The subclasses of these classes can be thought of as either subclasses or types. The difference being that you can either implement these as subclasses or by using logic within the class constructor to deal with the difference between the types. The reason that you do not have to create subclasses is that the differences between the type (e.g., Plane) and the subtypes (e.g., Bomber, Fighter, Stealth and VTOL) will only be in the specific values of the class' instance data but not in any new instance data or methods. It is left up to you whether to implement these subtypes as subclasses or through logic (explained later).

The Vehicle class defines the common instance data that will be shared among all vehicles and common methods which are accessor methods for each of the instance data, a toString method and a getBattleUtility method which returns an int value of the worth of the Vehicle given this war game scenario. The two subclasses of Vehicles are GroundVehicle and FlyingVehicle. GroundVehicle adds the instance datum roughTerrainCapable, FlyingVehicle adds instance data refuelInFlight and vtolCapable ("Vertical take-off and landing" meaning the vehicle does not require a runway to takeoff or land). For GroundVehicle, we have subclasses of Tank, TroopCarrier and Jeep. For FlyingVehicle we have subtypes of Plane and Helicopter. Although Helicopter does not add any instance data, Plane has an additional instance datum called stealth. Jeep does not add any instance data over GroundVehicle but Tank adds instance datum weight and TroopCarrier adds armed. For some of the classes beneath Vehicle, you will need to modify the constructor and toString methods because of added instance data (for the constructor, you have to initialize them, for the toString to report on the values of the added instance data). Whenever overriding, use @Override and then have the method start by calling its parent class method (e.g., super( ) for the constructor, super.toString( ) for the toString). For example, if a Vehicle can inherit from its parent the toString method but also needs to report on the variable roughTerrainCapable, you can accomplish this as follows.

```
@Override
public String toString( ) { return super.toString( ) +
      " and is rough terrain capable: " + roughTerrainCapable; }
```

As you will see below, Vehicle's getBattleUtility merely returns -100. Override this in GroundVehicle and FlyingVehicle and, aside from new accessor methods, implement getDistance( ) to compute the int distance that the Vehicle can travel. All subclasses will override getBattleUtility and most will override getDistance (but not all). Details for these methods are shown in a table on page 3-4 of this assignment.

The Plane, Helicopter, Tank and TroopCarrier classes can be of different subtypes (e.g., a Plane can be a Bomber, Fighter, Stealth or VTOL). You can implement these as subclasses if you wish. However, you will find that the subtypes do not introduce any new instance data or require changes to the getDistance and getBattleUtility methods. The only difference between the subtypes of one of these classes is in the values of its instance data. Therefore, you can also implement these types by using nested if-else logic in your class' constructor. For instance, a Helicopter can either be a normal Helicopter, a TroopHelicopter or an AttackHelicopter. You might implement these variations in Helicopter's constructor as follows

```
// assign the values that all Helicopters have
if(type.equals("Attack Helicopter") { /* assign specialized values for attack helicopter */}
else if(type.equals("Troop Helicopter") {/* assign specialized values for troop helicopter */}
else {/*assign values of a normal helicopter */}
```

Below is the UML specification for all of the classes.

Vehicle
-type: String
-heavilyArmored: boolean
-capacity: int
-speed: int
-armament: int

+Vehicle(type: String)
+getType( ): String
+getHeavilyArmored( ): boolean
+getCapacity( ): int
+getSpeed( ): int
+getArmament( ): int
+toString( ): String
+getBattleUtility( ): int

GroundVehicle extends Vehicle
-roughTerrainCapable: boolean

+GroundVehicle(type: String)
+getDistance( ): int
~~+getBattleUtility(…): int // details described later~~
@Override: toString, getBattleUtility

Jeep extends GroundVehicle
+Jeep(type: String)
@Override getBattleUtility

TroopCarrier extends GroundVehicle
-armed: boolean

+**TroopCarrier**(type: String)
@Override getBattleUtility, toString

Tank extends GroundVehicle
-weight: int

+Tank(type: String)
@Override getDistance, getBattleUtility, **toString**

FlyingVehicle extends Vehicle
-refuelInFlight: boolean
-vtolCapable: boolean

+FlyingVehicle(type: String)
+getVtol( ): boolean
+getRefuelInFlight( ): boolean
+getDistance( ): int
@Override getBattleUtility, toString

Helicopter extends FlyingVehicle
+Helicopter(type: String)
@Override getBattleUtility, toString

Plane extends FlyingVehicle
-stealth: boolean

+Plane(type: String)
@Override getDistance, getBattleUtility, toString

What follows is a table illustrating the initial values of the instance data for the various classes. The grey shaded boxes indicate that the particular instance datum does not exist in that class. Note that Tank is a parent class but can also be "Standard Tank" or "Normal Tank" and the same with Helicopter.

Vehicle	Heavily armored	Capacity	Speed	Armament	Armed	Rough terrain capable	Refuel in flight	Vtol capable	Stealth	Weight
Vehicle	false	1	0	0						
Ground Vehicle	true	4	50	10		false				
Flying Vehicle	false	1	200	2			false	false		
Jeep	false	4	75	5		true				
Tank	true	4	40	20		false				2
Light Tank	true	2	60	10		true				1
Medium Tank	true	4	50	25		true				3
Heavy Tank	true	4	30	30		false				5
TroopCarrier	true	25	60	10	false	true				
Armed Troop Carrier	true	8	50	30	true	true				
Light Troop Carrier	true	20	70	10	false	true				
Helicopter	false	8	75	3			false	true	false	
Attack Helicopter	true	2	120	6			false	true	false	
Troop Helicopter	false	20	80	2			false	true	false	
Plane	false	8	200	2			true	false	false	
Bomber	true	5	120	40			False	false	false	
FighterPlane	true	2	300	5			True	false	false	
VTOLPlane	false	2	200	2			True	true	false	
Stealth	false	4	150	5			false	false	true	

The following table provides the information for the getDistance and getBattleUtility methods for each class. The getBattleUtility class receives 6 boolean variables and an int. These are in order: night (boolean), rough terrain (boolean), need ground support (boolean), need heavy arms (boolean), anti aircraft guns in vicinity (boolean), need equipment dropped (boolean) and distance (int).

Type	getDistance	getBattleUtility
Vehicle	Not implemented	Returns -100
Ground Vehicle	If heavily armored speed * 30 else speed * 45	Start with score = 0 If rough terrain and vehicle is not rough terrain capable then -5 else +8 If need ground support then +*capacity* - 1 (the value of capacity - 1) If distance > getDistance then -10
Flying Vehicle	If refuel in flight then 10000 else speed * 10	Start with score = 0 If distance > getDistance then -15 else +30 If rough terrain then +5 If ! need for ground support then +5 If need ground support and vtol capable and capacity > 2 then +*capacity* (the value of capacity)
Jeep	Inherited from Ground Vehicle	super.getBattleUtility(…) + each of the following

		If anti aircraft guns then +10 If need equipment dropped then +8 If night then -5
Tank	super( ).getDistance( ) – 1000*weight	super.getBattleUtility(…) + each of the following if anti aircraft guns then +12 if need heavy arms then +*armament**3 (value of armament multiplied by 3) if rough terrain and ! rough terrain capable then -25 if distance > getDistance then –(*distance* – *getDistance*) / 105
Light Tank	Same as Tank	Same as Tank
Medium Tank	Same as Tank	Same as Tank
Heavy Tank	Same as Tank	Same as Tank
TroopCarrier	Inherited from Ground Vehicle	super.getBattleUtility(…) + each of the following if need ground support +*capacity**2 if distance <= getDistance then +7 else -33 if need heavy arms and armed then +*armament* else +5 if need equipment then +10
Armed Troop Carrier	Same as Troop Carrier	Same as Troop Carrier
Light Troop Carrier	Same as Troop Carrier	Same as Troop Carrier
Helicopter	Inherited from FlyingVehicle	super.getBattleUtility(…) + each of the following if night +5 else -4 if need ground support and capacity < 5 then -4 else +20 if need heavy arms then +*armament* * 2 if anti aircraft guns -15 if need equipment then +8
Attack Helicopter	Same as Helicopter	Same as Helicopter
Troop Helicopter	Same as Helicopter	Same as Helicopter
Plane	If refuel in flight then 10000 Else if heavily armored then speed * 15 Else speed * 25	super.getBattleUtility(…) + each of the following if anti aircraft guns and not (heavily armored or stealth) then -10 if night +6 if need heavy arms then +*armament* if need equipment then +5
Bomber	Same as Plane	Same as Plane
FighterPlane	Same as Plane	Same as Plane
VTOLPlane	Same as Plane	Same as Plane
Stealth	Same as Plane	Same as Plane

Implement a WarGame class which creates instances of various classes (see the table below) and tests Objects of those classes against the three scenarios. Submit for this assignment all of your classes (copy all of the classes into one text file for easier output and to save paper) and the results of running the three scenarios. If you submit this assignment electronically, again please copy the text of all of your classes

into a single text file (or a word document). The three scenarios are given in this table. For each scenario, create an object of each type of Vehicle listed, obtain the Vehicle's toString output and the result of getBattleUtility when passed the parameters listed in the rightmost column. These parameters are in order: night, rough terrain, need ground support, need heavy arms, anti aircraft guns in area, distance (an int).

Scenario 1	attack helicopter, troop helicopter, light tank, medium tank, heavy tank, bomber, vtol	T, T, T, T, F, F, 2500
Scenario 2	Flying vehicle, helicopter, attack helicopter, troop helicopter, plane, bomber, fighter, vtol plane, stealth	F, T, F, T, T, F, 1000
Scenario 3	Ground vehicle, tank, light tank, medium tank, heavy tank, jeep, troop carrier, armed troop carrier, light troop carrier	F, F, T, T, F, F, 4000