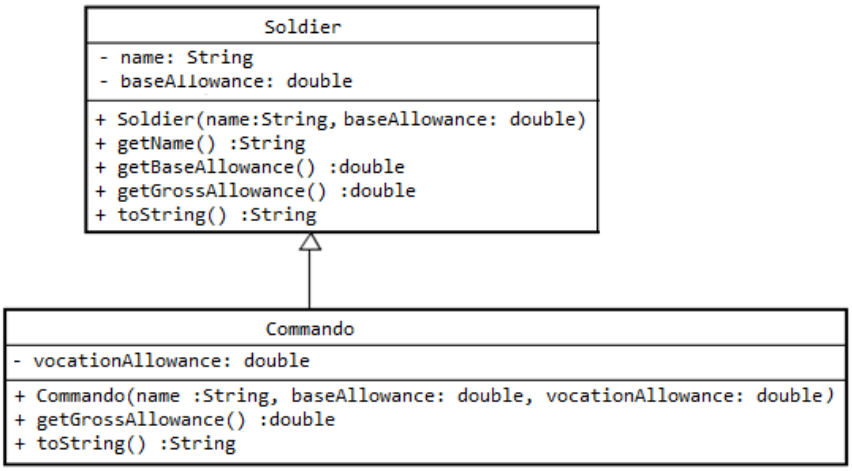
# Inheritance 1



1. **[ Difficulty: * ]** Most men who have completed their National Service with the military know that certain vocations (e.g. commandos) will be given an additional "vocation allowance" as a reward for their higher risk. Study the class diagram below.

(a) Code the Soldier and Commando classes. The getGrossAllowance() methods should calculate salary using this formula:

- Soldier's gross allowance = baseAllowance
- Commando's gross allowance = baseAllowance + vocationAllowance



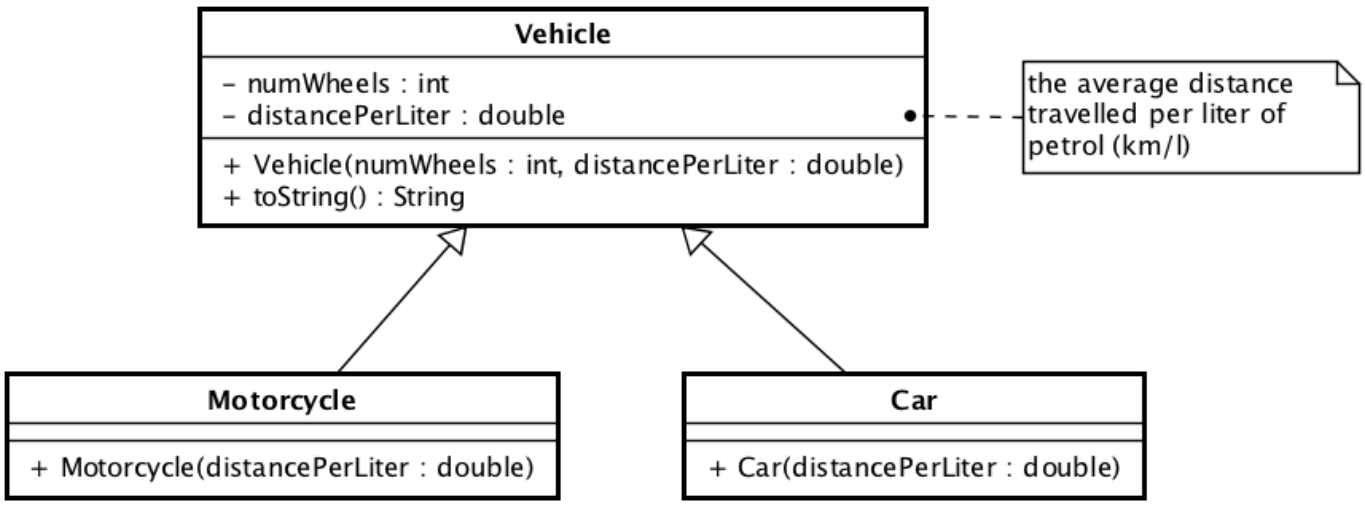

(b) Study the following **SoldierTest**.java file. Write down the output without running your code (and peeping at the next page).

```
public class SoldierTest {
  public static void main(String[] args) {
    Commando c = new Commando("Peter", 5000, 200);
    Soldier s = new Soldier("John", 3000);

    System.out.println(c);
    System.out.println(s);

    System.out.println("Commando's base allowance : " + c.getBaseAllowance());
    System.out.println("Commando's gross allowance : " + c.getGrossAllowance());
    System.out.println("Soldier's base allowance : " + s.getBaseAllowance ());
    System.out.println("Soldier's gross allowance : " + s.getGrossAllowance ());
  }
}
```

(c) Then run **SoldierTest** to check if your code in **Soldier** and **Commando** are correct:

```
Commando{Soldier{name='Peter', baseAllowance=5000.0}, vocationAllowance=200.0}
Soldier{name='John', baseAllowance=3000.0}
Commando's base allowance : 5000.0
Commando's gross allowance : 5200.0
Soldier's base allowance : 3000.0
Soldier's gross allowance : 3000.0
```

2. **[ Difficulty: * ]** Given the following class diagram:

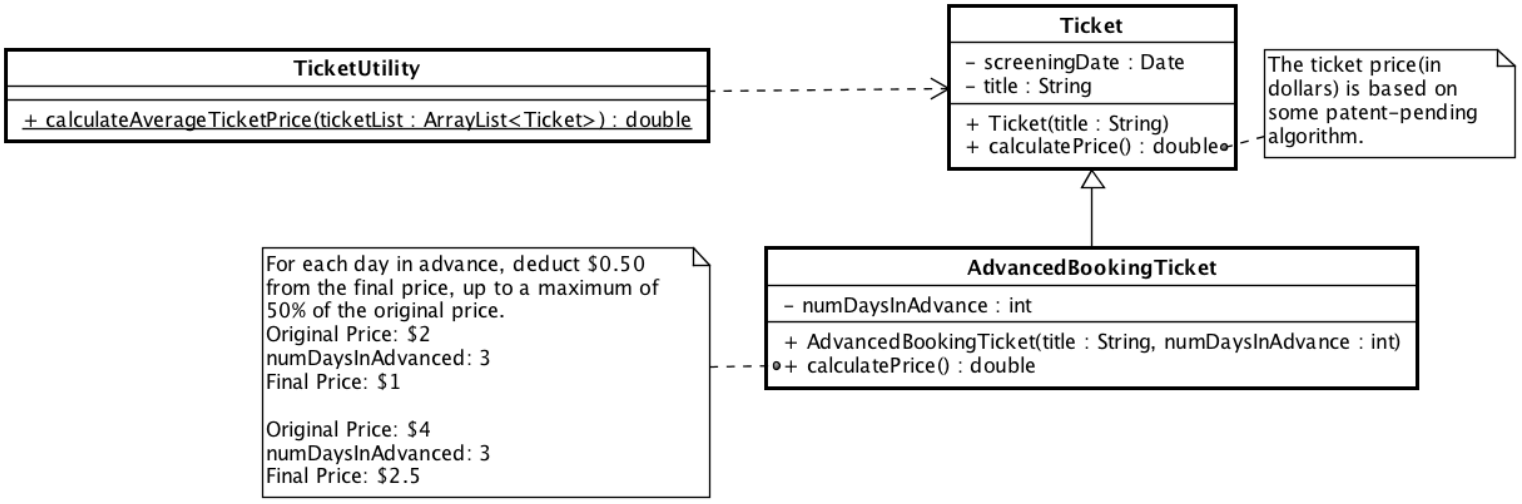



a. Implement the three classes shown. The numWheels variable should be initialized to 4 for Car, and 2 for Motorcycle making use of the superclass (Vehicle) constructor.

b. The `toString()` method returns a textual representation of a `Vehicle` in the following format:
`Vehicle[numWheels=<numWheels>, distancePerLiter=<distancePerLiter>]`

c. Write a `VehicleTest` class that contains a `main` method. In `main`, create an instance of a `Motorcycle`, and an instance of a `Car`. Then display the objects' values by calling their `toString()` methods.

3. **[ Difficulty: ** ]** Given the class diagram (below) and the **Ticket** class. Note that in UML an underlined method is to represent "static" method



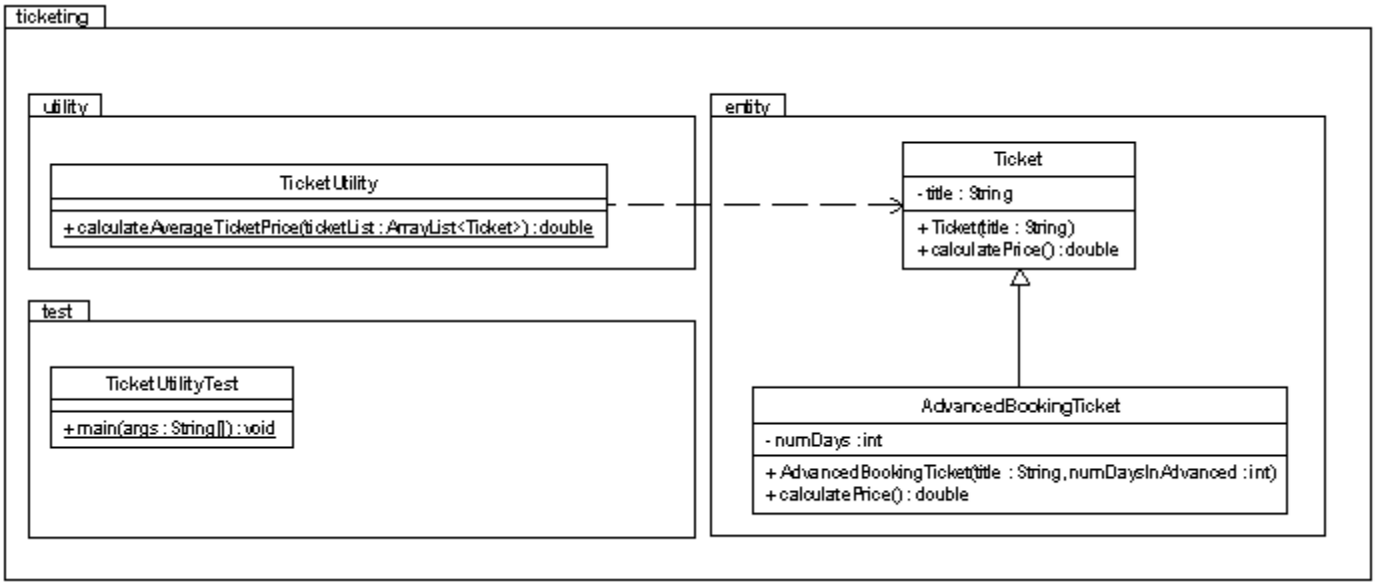

a. Write the **AdvancedBookingTicket** class.

b. Write the **TicketUtility** class.

The output of **TicketUtilityTest** should be as follows:

```
Ticket 1: 12.42
Ticket 2: 5.13
Ticket 3: 2.52
Ticket 4: 8.36
Ticket 5: 2.925

Test 1:
Expected: 0.0
Actual  : 0.0

Test 2:
Expected: 0.0
Actual  : 0.0

Test 3:
Expected: 6.271
Actual  : 6.271
```

4. **[ Difficulty: *** ]** Given the following class diagram:




**IMPORTANT:** Use Ticket.class from the resource Q4. **DO NOT REUSE** Ticket.class from Q3.

a. Update AdvancedBookingTicket, TicketUtility and TicketUtilityTest with the correct package and import statements (you could make use of your solution from Q3 and update).
b. In your question directory, create the following directory structure , and place the Java files in the correct sub-directory under src, and the Ticket.class in the correct sub-directory under classes.

```
<Q4>
 |--- src
 |     |--- ...
 |--- library
 |     |--- Ticketing.jar // contains Ticket.class
 |--- classes
 |     |--- ...
 |--- compile.bat
 |--- run.bat
```

c. Write a one-liner command for both the batch files (compile.bat and run.bat). The compile.bat batch file will compile all the Java files and place the class files in the classes folder. The run.bat will run the TicketUtilityTest.

5. **[ Difficulty: *** ]** Read the Python to Java Guide (https://smu.sg/python2java).
   a. Properties: https://www.journaldev.com/14893/python-property-decorator.
   b. Private Variables: https://docs.python.org/3.3/tutorial/classes.html#private-variables

**Task**: Recode Q1 in Python.

6. **[ Difficulty: ** ]** Read about Java sockets
   ( https://docs.oracle.com/javase/tutorial/networking/sockets/definition.html ). Download the following files:
   a. EchoClient.java (https://docs.oracle.com/javase/tutorial/networking/sockets/examples/EchoClient.java)
   b. EchoServer.java
      (https://docs.oracle.com/javase/tutorial/networking/sockets/examples/EchoServer.java)

   a. Run the server with the following command.
      **Note:**
      i. 1234 is the port number.
      ii. In other words, you will not get back the prompt. The program waits for an incoming client connection.

```
c:\IS442>java EchoServer 1234
```

   b. Run the client with the following command.

```
c:\IS442>java EchoClient localhost 1234
happy
echo: happy
sad
echo: sad
```

   c. Quit the application by closing the command prompt window or using Control-C.

   d. **Task:** Pair up with your neighbour. You run the server program, and he/she runs the client program.

   e. **Task:** How do you modify the code so that the program will
      i. continue to accept client connections one after another? Can you add in a loop so that it will process requests an infinite number of times.

   f. **Task:** How do you modify the code so that the client sends over a Java object (Person class with name and age attributes), and the server prints the name and age values to the screen. It then returns a string "<name> is old" if the age of the person is more than 18. Otherwise, it returns a string "<name> is young".

**Reference:** https://docs.oracle.com/javase/tutorial/networking/

- END -