COP3337
Inheritance- Mini Project

This is a mini project from you to get a better understanding of inheritance in Java. This mini project does not need to be turned in and will not count towards your grade, but I highly recommend that you do this project. Doing this project will reduce your studying time for the next exam and help you fully understand the important little details.

It is not a waste of your time….

All you need to do is setup your Netbeans Project such that it matches the setup in the package diagram below then type in all the code for the following files:

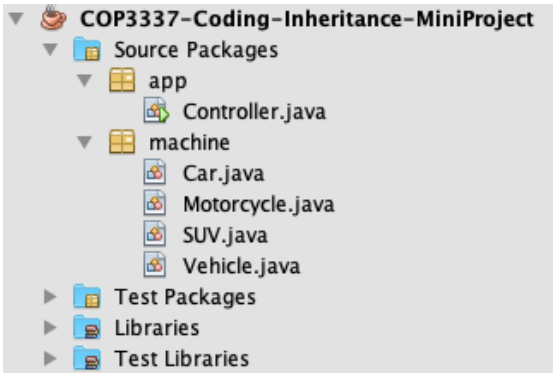
- Controller.java
- Car.java
- Motorcycle.java
- SUV.java
- Vehicle.java

The source code is provided at the end of this document.

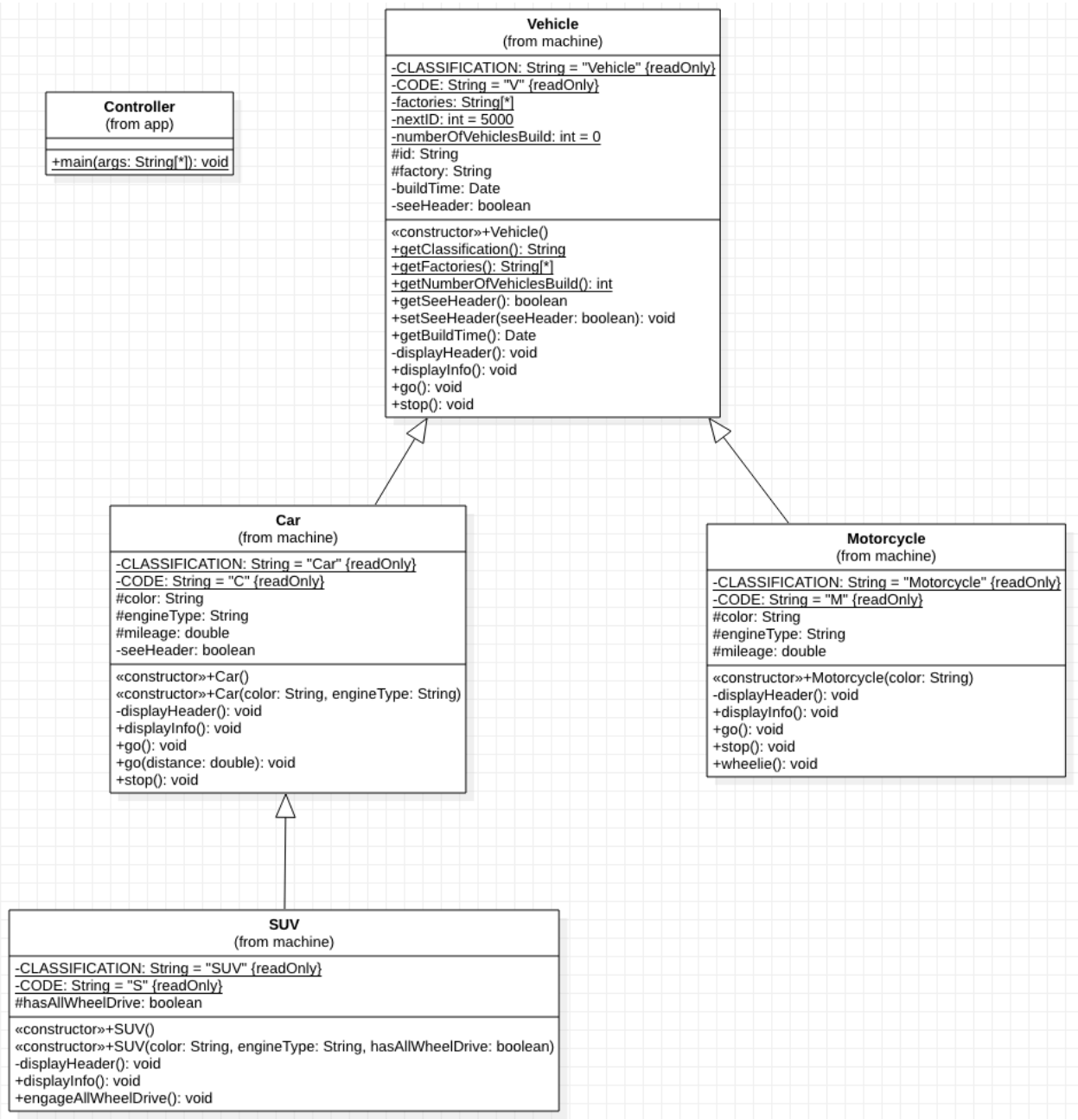
What to focus on:

- Understand UML Diagram in detail
    - Look at the UML diagram and identify methods that are refinements, overridden and overload
    - Be able to go from UML -> Code and Code -> UML
    - FYI, # means protected
- Understand the casting in the code and why some casts do not work
- Understand which code is running and where it is coming from (which class is the code being executed from)
- Understand why we created a subclass
- Understand the logic of the code

Package Diagram



UML Diagram


Vehicle
(from machine)
-CLASSIFICATION: String = "Vehicle" {readOnly}
-CODE: String = "V" {readOnly}
-factories: String[*]
-nextID: int = 5000
-numberOfVehiclesBuild: int = 0
#id: String
#factory: String
-buildTime: Date
-seeHeader: boolean
«constructor»+Vehicle()
+getClassification(): String
+getFactories(): String[*]
+getNumberOfVehiclesBuild(): int
+getSeeHeader(): boolean
+setSeeHeader(seeHeader: boolean): void
+getBuildTime(): Date
-displayHeader(): void
+displayInfo(): void
+go(): void
+stop(): void
Controller
(from app)
+main(args: String[*]): void
Car
(from machine)
-CLASSIFICATION: String = "Car" {readOnly}
-CODE: String = "C" {readOnly}
#color: String
#engineType: String
#mileage: double
-seeHeader: boolean
«constructor»+Car()
«constructor»+Car(color: String, engineType: String)
-displayHeader(): void
+displayInfo(): void
+go(): void
+go(distance: double): void
+stop(): void
Motorcycle
(from machine)
-CLASSIFICATION: String = "Motorcycle" {readOnly}
-CODE: String = "M" {readOnly}
#color: String
#engineType: String
#mileage: double
«constructor»+Motorcycle(color: String)
-displayHeader(): void
+displayInfo(): void
+go(): void
+stop(): void
+wheelie(): void
SUV
(from machine)
-CLASSIFICATION: String = "SUV" {readOnly}
-CODE: String = "S" {readOnly}
#hasAllWheelDrive: boolean
«constructor»+SUV()
«constructor»+SUV(color: String, engineType: String, hasAllWheelDrive: boolean)
-displayHeader(): void
+displayInfo(): void
+engageAllWheelDrive(): void

Code Section

Controller.java

```java
// COP3337 - Inheritance Mini-Project

package app;

import machine.Car;
import machine.Motorcycle;
import machine.SUV;
import machine.Vehicle;

public class Controller {


    public static void main(String[] args) {


        Vehicle vehicle = new Vehicle();

        vehicle.displayInfo();


        Car car1 = new Car("Blue", "Diesel");
        Car car2 = new Car();

        car1.displayInfo();
        car2.displayInfo();

        // look at the print out and understand what is going on and why
        ((Vehicle)car2).displayInfo();

        car2.go(172.3);
        car2.displayInfo();

        // why can I not do this
        // ((Vehicle)car2).go(293.2);

        SUV suv = new SUV();



        suv.displayInfo();
        ((Car)suv).displayInfo();
        ((Vehicle)suv).displayInfo();

        // why can I not do this
        // ((Motorcycle)suv).displayInfo();

        System.out.println("\n\nSUV calling go methods");
        System.out.println("==============================");
        suv.go();
        suv.go(2503.2);
        System.out.println("");



         Motorcycle motorcycle = new Motorcycle("Black");
         motorcycle.go();
         motorcycle.displayInfo();

        // LOOK AT THIS CAREFULLY
        Vehicle car3 = new Car("Red", "Electric");

        car3.go();

        // why can I not do this
        //car3.go(30.5);

        // why do this work
        ((Car)car3).go(30.3);

        car3.displayInfo();
    }

}//end class
```

Vehicle.java

```
1
2   package machine;
3
4   import java.text.SimpleDateFormat;
5   import java.util.Date;
6   import java.util.Random;
7
8
9   public class Vehicle {
10
11      //---------------------------------
12      // Class variables and constants
13      //---------------------------------
14      private static final String CLASSIFICATION = "Vehicle";
15      private static final String CODE = "V";
16      private static String[] factories = {"MIAMI", "AUSTIN", "LOGAN", "CHARLOTTE"};
17      private static int nextID = 5000;
18      private static int numberOfVehiclesBuild = 0;
19
20      //---------------------------------
21      // Instance variables
22      //---------------------------------
23      protected String id;
24      protected String factory;
25      private Date buildTime;
26      private boolean seeHeader;
27
28      //---------------------------------
29      // Constructor
30      //---------------------------------
31      public Vehicle(){
32
33          Random rndGen = new Random();
34
35          int index = rndGen.nextInt(4);
36
37          String letters = factories[index].substring(0, 3);
38          id = CODE + "-" + letters + "-" + nextID;
39
40          factory = factories[index];
41          buildTime = new Date();
42
43          seeHeader = true;
44
45          nextID++;
46          numberOfVehiclesBuild++;
47
48      }
49
50      //---------------------------------
51      // Class methods
52      //---------------------------------
53      public static String getClassification() {
54          return CLASSIFICATION;
55      }
56
57      public static String[] getFactories() {
58          return factories;
59      }
60
```

```
61      public static int getNumberOfVehiclesBuild() {
62          return numberOfVehiclesBuild;
63      }
64
65      //-----------------------------------
66      // Instance methods setter and getters
67      //-----------------------------------
68
69      public boolean getSeeHeader() {
70          return seeHeader;
71      }
72
73      public void setSeeHeader(boolean seeHeader) {
74          this.seeHeader = seeHeader;
75      }
76
77
78      // LOOK CARFULLY AT THIS METHOD
79      // SEE SOMETHING DIFFERENT
80      // WHAT IS THE FINAL DOING
81      public final Date getBuildTime() {
82          return buildTime;
83      }
84
85
86
87      //-----------------------------------------------
88      // Instance methods that may be overridden or overload
89      //-----------------------------------------------
90      private void displayHeader(){
91          System.out.println("");
92          System.out.println("=================================");
93          System.out.println("Vehicle Info");
94          System.out.println("=================================");
95      }
96
97
98
100     public void displayInfo(){
101
102         SimpleDateFormat dateFormat = new SimpleDateFormat("MM-dd-yyyy hh:mm:ss");
103
104         if(seeHeader){
105             displayHeader();
106             System.out.println("Classification:\t\t" + CLASSIFICATION);
107         }
108
109         System.out.println("ID:\t\t\t" + id);
110         System.out.println("Factory:\t\t" + factory);
111         System.out.println("Build Time\t\t" + dateFormat.format(buildTime));
112     }
114     public void go(){
115         System.out.println("");
116         System.out.println("Vehicle's go() executed");
117     }
118
120     public void stop(){
121         System.out.println("");
122         System.out.println("Vehicle's stop() executed");
123     }
124 }//end class
```

Car.java

```
1
2   package machine;
3
4   import java.text.SimpleDateFormat;
5
6
7   public class Car extends Vehicle {
8
9       //---------------------------------
10      // Class variables and constants
11      //---------------------------------
12      private static final String CLASSIFICATION  = "Car";
13      private static final String CODE = "C";
14
15      //---------------------------------
16      // Instance variables
17      //---------------------------------
18      // refinement instance variables
19      protected String color;
20      protected String engineType;
21      protected double mileage;
22      private boolean seeHeader;
23
24      //---------------------------------
25      // Constructor
26      //---------------------------------
27      public Car(){
28          this("Black", "Gasoline");
29      }
30
31      public Car (String color, String engineType){
32          super();
33
34          // look this carefully... where is the id coming from
35          // where is it declared
36          id = CODE + "-" + id;
37
38          this.color = color;
39          this.engineType = engineType;
40          mileage = 0.0;
41          seeHeader = true;
42
43      }
44
45
46      //-----------------------------------------------
47      // Instance methods that may be overridden or overload
48      //-----------------------------------------------
49      // this is not overridden because it is not inheritance from Vehicle
50      private void displayHeader(){
51          System.out.println("");
52          System.out.println("=================================");
53          System.out.println("Car Info");
54          System.out.println("=================================");
55      }
56
```

```
57      // overriden methed
58      @Override
59      public void displayInfo(){
60
61          SimpleDateFormat dateFormat = new SimpleDateFormat("MM-dd-yyyy hh:mm:ss");
62
63          if(seeHeader){
64              displayHeader();
65              System.out.println("Classification:\t\t" + CLASSIFICATION);
66
67              // what am I doing here and why
68              super.setSeeHeader(false);
69          }
70
71          super.displayInfo();
72          System.out.println("Color:\t\t\t" + color);
73          System.out.println("Engine Type:\t\t" + engineType);
74
75          // why am I using a printf here
76          System.out.printf("Mileage:\t\t%-10.1f\n", mileage);
77
78          // why I am doing this
79          if(!(super.getSeeHeader() ) ){
80              super.setSeeHeader(true);
81          }//end if
82      }
83
84      // overriden methed
85      public void go(){
86          System.out.println("");
87          System.out.println("Car's go() executed");
88      }
89
90      // overload method
91      public void go(double distance){
92
93          mileage += distance;
94
95          System.out.println("");
96          System.out.println("Car's go(double) executed");
97          System.out.printf("The car has " + mileage + " mile on it now.");
98      }
99
100     // overriden methed
101      public void stop(){
102         System.out.println("");
103         System.out.println("Car's stop() executed");
104     }
105
106 }//end class
```

SUV.java

```java

package machine;

import java.text.SimpleDateFormat;


public class SUV extends Car{

    //---------------------------------
    // Class variables and constants
    //---------------------------------
    private static final String CLASSIFICATION  = "SUV";
    private static final String CODE = "S";

    //---------------------------------
    // Instance variables
    //---------------------------------
    // refinement
    protected boolean hasAllWheelDrive;

    //---------------------------------
    // Constructor
    //---------------------------------

    public SUV(){
        super();
        id = CODE + "-" + id;
        hasAllWheelDrive = false;
    }

    public SUV(String color, String engineType, boolean hasAllWheelDrive){

        super(color, engineType);
        this.hasAllWheelDrive = hasAllWheelDrive;
    }

    // uses Car's stop and go methods

    //-----------------------------------------------
    // Instance methods that may be overridden or overload
    //-----------------------------------------------
    // this is not overridden because it is not inheritance from Vehicle
    private void displayHeader(){
        System.out.println("");
        System.out.println("==================================");
        System.out.println("SUV Info");
        System.out.println("==================================");
    }

```

```
50          // overriden methed
51          @Override
52          public void displayInfo(){
53
54              SimpleDateFormat dateFormat = new SimpleDateFormat("MM-dd-yyyy hh:mm:ss");
55
56              displayHeader();
57
58               // what am I doing here and why
59               super.setSeeHeader(false);
60
61               // what is running here
62               super.displayInfo();
63
64               System.out.println("All-Wheel-Drive:\t" + hasAllWheelDrive);
65
66               // why I am doing this
67               if(!(super.getSeeHeader() ) ){
68                  super.setSeeHeader(true);
69               }//end if
70          }
71
72
73          //-----------------------------------------------
74          // Instance methods refinment in SUV
75          //-----------------------------------------------
76          public void engageAllWheelDrive(){
77              if(hasAllWheelDrive){
78                  System.out.println("Engaging All-Wheel-Drive in the  SUV");
79              }else{
80                  System.out.println("Sorry All-Wheel-Drive cannot be engaged in the SUV ");
81              }//end if-else
82          }
83
84      }//end class
85
```

Motorcycle.java

```java

package machine;

import java.text.SimpleDateFormat;

public class Motorcycle extends Vehicle{

    //-------------------------------
    // Class variables and constants
    //-------------------------------
    private static final String CLASSIFICATION  = "Motorcycle";
    private static final String CODE = "M";

    //-------------------------------
    // Instance variables
    //-------------------------------
    // refinement
    protected String color;
    protected String engineType;
    protected double mileage;

    //-------------------------------
    // Constructor
    //-------------------------------

    public Motorcycle(String color){
        super();

        // look this carefully... where is the id coming from
        // where is it declared
        id = CODE + "-" + id;

        this.color = color;
        this.engineType = "Gasiline";
        mileage = 0.0;

    }


    //-----------------------------------------------
    // Instance methods that may be overridden or overload
    //-----------------------------------------------
    private void displayHeader(){
        System.out.println("");
        System.out.println("=====================================");
        System.out.println("Motorcycle Info");
        System.out.println("=====================================");
    }


```

```
51
52      public void displayInfo(){
53
54          SimpleDateFormat dateFormat = new SimpleDateFormat("MM-dd-yyyy hh:mm:ss");
55
56           displayHeader();
57           System.out.println("Classification:\t\t" + CLASSIFICATION);
58
59
60          System.out.println("ID:\t\t\t" + id);
61          System.out.println("Factory:\t\t" + factory);
62          System.out.println("Build Time\t\t" + dateFormat.format(super.getBuildTime()) );
63      }
64
65
66      public void go(){
67          System.out.println("");
68          System.out.println("Motorcycle's go() executed");
69      }
70
71
72       public void stop(){
73          System.out.println("");
74          System.out.println("Motorcyle's stop() executed");
75      }
76
77      //-----------------------------------------------
78      // Instance methods refinment in Motorcycle
79      //-----------------------------------------------
80      public void wheelie(){
81          System.out.println("Motorcyle's wheelie() executed");
82      }
83
84  }//end class
85
```