xModz Java program.

CT2106 – 2022/23
Assignment 1
Maxwell Maia

Table of Contents

Code

TestCar class

```
/**
 * This class will be used to create objects and call functions to test the functionality of the xModz program.
 *
 * @author Maxwell Maia
 * @version 2.1
 */
public class TestCar
{
    public static void main(String[] args)
    {
        //Create a car object.
        //Also pass the attributes for the engine and wheel as arguments.
        Car nissan = new Car("R34", "Brrooooommm", 43, "Toyo15", 15);
        Car toyota = new Car ("C-HR", "Hybrid Electric", 25, "Bridgetone13", 13);
        Car subaru = new Car("Impreza", "Brrrr", 64, "OneOfEach12", 12);



        //Set the fuel. This is passed into car,
        //then from the car into the engine.
        nissan.setFuel(100);
        toyota.setFuel(100);
        subaru.setFuel(100);




        System.out.println("Nissan");
```

```
//Print the fuel status using a getter.
System.out.printf("Current fuel: %.2f\n", nissan.getFuel());
//Method to make the car drive, using up all its fuel.
nissan.drive();
//Method that prints the fields of the car to the terminal.
nissan.printState();

System.out.println("Toyota");
System.out.printf("Current fuel: %.2f\n", toyota.getFuel());
toyota.drive();
toyota.printState();



System.out.println("Subaru");
System.out.printf("Current fuel: %.2f\n", subaru.getFuel());
subaru.drive();
subaru.printState();



//Refilling fuel... going on another trip...
subaru.setFuel(80);
System.out.println("Subaru 2nd Trip");
//Print the fuel status using a getter.
System.out.printf("Current fuel: %.2f\n", subaru.getFuel());

//The car drives a second time, using up all its fuel.
subaru.drive();

//Method that prints the fields of the car to the terminal.
subaru.printState();
```

```
    }
}
```

Car class

```
/**
 * This is the car class. The car is composed of an engine.
 *
 * @author Maxwell Maia
 * @version 2.1
 */
public class Car
{
    //Declared fields to store information about a car.
    private String name;
    private double distance;
    private double totalKm;

    //Variable to store the engine object.
    //The car needs an engine object to function because a car is composed of an engine.
    //The car relies on the service of the engine.
    private Engine carEngine;

    /**
     * Constructor for objects of class Car
     */
    public Car(String carName, String engineName, double tpl, String wheelName, double radius)
    {
        //Instantiate the Car object.
        this.name = carName;

        //Create an engine object.
```

```
        //Also pass the attributes for wheel as arguments.
        carEngine = new Engine(engineName, tpl, wheelName, radius);
    }



    public void setFuel(double fuel) //Set the fuel in the engine object.
    {
        carEngine.setFuel(fuel);
    }

    public double getFuel()
    {
        //Get the fuel from the engine object.
        double tempFuel = carEngine.getFuel();

        //Return the fuel to the TestCar class.
        return tempFuel;
    }

    //Method to make the car drive until it runs out of fuel.
    public void drive()
    {
        //We are ready to drive.
        //Let's calculate how many times we can turn
        //the wheel with the fuel that we have. The result is stored in the "turns"
variable.
        double turns = carEngine.getFuel() * carEngine.getTpl();

        //The car used all of it's fuel to turn the wheel.
        //Set the fuel to 0.
        carEngine.setFuel(0);
```

```
        //We will now ask the engine to turn the wheel
        //this many times.
        //The "distance" variable will recieve a value for the distance
        //travelled by the wheel due to this request.
        distance = carEngine.turnWheelCalcDistance(turns);

        //Add the "distance of this trip" to the "total distance travelled."
        totalKm += distance;
    }

    public void printState()
    {
        //Print out statements showing the value of the fields that describe the car.
        //Getters are used to retrieve fields that are not stored in this class.
        System.out.println("Configuration: Car Body " + name);

        System.out.println("Engine name: "+ carEngine.getName());

        //.2f is used to show the value to 2 decimal places after the decimal point.
        System.out.printf("Engine turns per litre: %.2f\n", carEngine.getTpl());

        System.out.println("Engine's total turn count: " +
carEngine.getTotalNumTurns());

        System.out.println("Wheel name: " + carEngine.getWheelName());

        System.out.println("Wheel radius: " + carEngine.getWheelRadius());

        System.out.printf("Wheel circumference (distance per turn): %.2f\n",
carEngine.getWheelCircumference());
```

```
        System.out.printf("Distance this trip: %.2f\n", distance);

        System.out.printf("Total distance Travelled: %.2f\n", totalKm);

        System.out.printf("Current fuel: %.2f\n\n", carEngine.getFuel());
    }
}
```

Engine class

```
/**
 *
 * This is the engine class. The engine is composed of a wheel.
 *
 * @author Maxwell Maia
 * @version 2.1
 */
public class Engine
{
    //Declared fields to store information about the engine of a car.
    private double fuelLevel;
    private String name;
    private double tpl;
    private int totalNumTurns;

    //Variable to store the engine object.
    //The engine needs an wheel object to function because a engine is composed of a wheel.
    //The engine relies on the service of the wheel.
    private Wheel engineWheel;

    /**
     * Constructor for objects of class Engine
     */
    public Engine(String name, double tpl, String wheelName, double radius)
    {
        // initialise instance variables
        this.name = name;
```

```
        //Make sure that the tpl is a positive number using Math.abs() .
        this.tpl = Math.abs(tpl);

        //Create a wheel object.
        engineWheel = new Wheel(wheelName, radius);
    }

    public void setFuel(double fuel) //Set the fuel
    {
        //Protect against in valid in puts.
        //Only allow a range of 0 to 100 (inclusive).
        if(fuel >= 0 && fuel <= 100)
        {
            this.fuelLevel = fuel;
        }
        else
        {
            System.out.println("Fuel input is not in the correct range. Fuel input must be a
value from 0 - 100 (inclusive).");
        }
    }

    public double getFuel() //Return the fuel to the car.
    {
        return fuelLevel;
    }

    public String getName() //Return the name of the engine.
    {
        return name;
    }
```

```
public double getTpl() //Return the tpl.
{
    return tpl;
}

public int getTotalNumTurns() //Return the totalNumTurns.
{
    return totalNumTurns;
}



//The car has called on the engine to turn the wheel..
//Here the engine calls on the wheel to turn, and
//the distance travelled by the wheel is returned.
public double turnWheelCalcDistance(double turns)
{
    //Call for wheel to turn and store the returned value.
    double distanceDueToTurn = engineWheel.turn(turns);

    //Increment the engine's total turn count.
    /*
     * Note: In the sample output, the "Engine's total
     * turn count" shows an integer with no decimal places.
     * For this reason an integer is used here.
     * This is okay but when this function is called
     * while the engine has either a fuelLevel that is not
     * a whole number or a TPL that is not a whole number,
     * data will be lost when converting a double to an
     * integer. This happens in the statement below.
```

```
         * Solution:
         * a) use a double for totalNumTurns.
         * b) be okay with a number of turns
         * that is not decimal precise. AKA number of
         * completed turns.
         */
        totalNumTurns += turns;



        //Return the distance travelled to the Car object.
        return distanceDueToTurn;
    }

    public String getWheelName() //Return the name of the wheel that is stored in the wheel object.
    {
        return engineWheel.getName();
    }

    public double getWheelRadius() //Return the radius of the wheel that is stored in the wheel object.
    {
        return engineWheel.getRadius();
    }

    public double getWheelCircumference() //Return the circumference of the wheel that is stored in the wheel object.
    {
        return engineWheel.getCircumference();
    }
}
```

Wheel class

```
/**
 * This is the wheel class. The wheel is not composed of any objects.
 *
 * @author Maxwell Maia
 * @version 2.1
 */
public class Wheel
{
    //Declared fields to store information about the wheel that is connected to the engine of a car.
    private double radius;
    /*
     * A wheel's radius may include a decimal,
     * so a double is used.
     */

    private String name;

    private double distanceTravelled;

    //1 turn of a wheel is 1 circumference of distance.
    //calculate circumference
    private double circumference;

    /**
     * Constructor for objects of class Wheel
     */
    public Wheel(String name, double radius)
    {
```

```
    // initialise instance variables
    this.name = name;

    //Make sure that the radius is a positive number using Math.abs() .
    this.radius = Math.abs(radius);
    circumference = 2 * Math.PI * Math.abs(radius);
}

//Function to "turn the wheel" the amount of times
//that it has been requested to.
//The function returns the distance it has travelled
//due to this call to turn the wheel.
public double turn(double turns)
{
    //1 turn of a wheel is 1 circumference of distance.
    distanceTravelled = circumference * turns;
    return distanceTravelled;
}


//Getters to return the fields of the wheel.

public String getName()
{
    return name;
}

public double getRadius()
{
    return radius;
```

```
    }

    public double getCircumference()
    {
        return circumference;
    }
}
```

An outline of how the code works

In the test class, a call to create a new engine object is made. Attributes that describe the car, engine and wheel are passed in as parameters. The Car class constructor is called.

In the Car class constructor the car object's only field (the car name) is initialized. Also in the constructor of the Car class, a call to create a new Engine object is made. Attributes that describe the engine and the wheel are passed into this new Engine object. These came from the constructor of the Car object. The Engine class constructor is called.

In the Engine class constructor the name of the engine and turns per litre fields are initialized. Also in the Engine class constructor, a call to create a new wheel object is made. Attributes that describe the wheel are passed in as parameters. The Wheel class constructor is called.

In the Wheel class constructor the name and radius fields are initialized and the circumference is calculated and initialized.

We now have 3 objects. Car, Engine and Wheel. The car object is composed of an engine object. The engine object is composed of a wheel object.

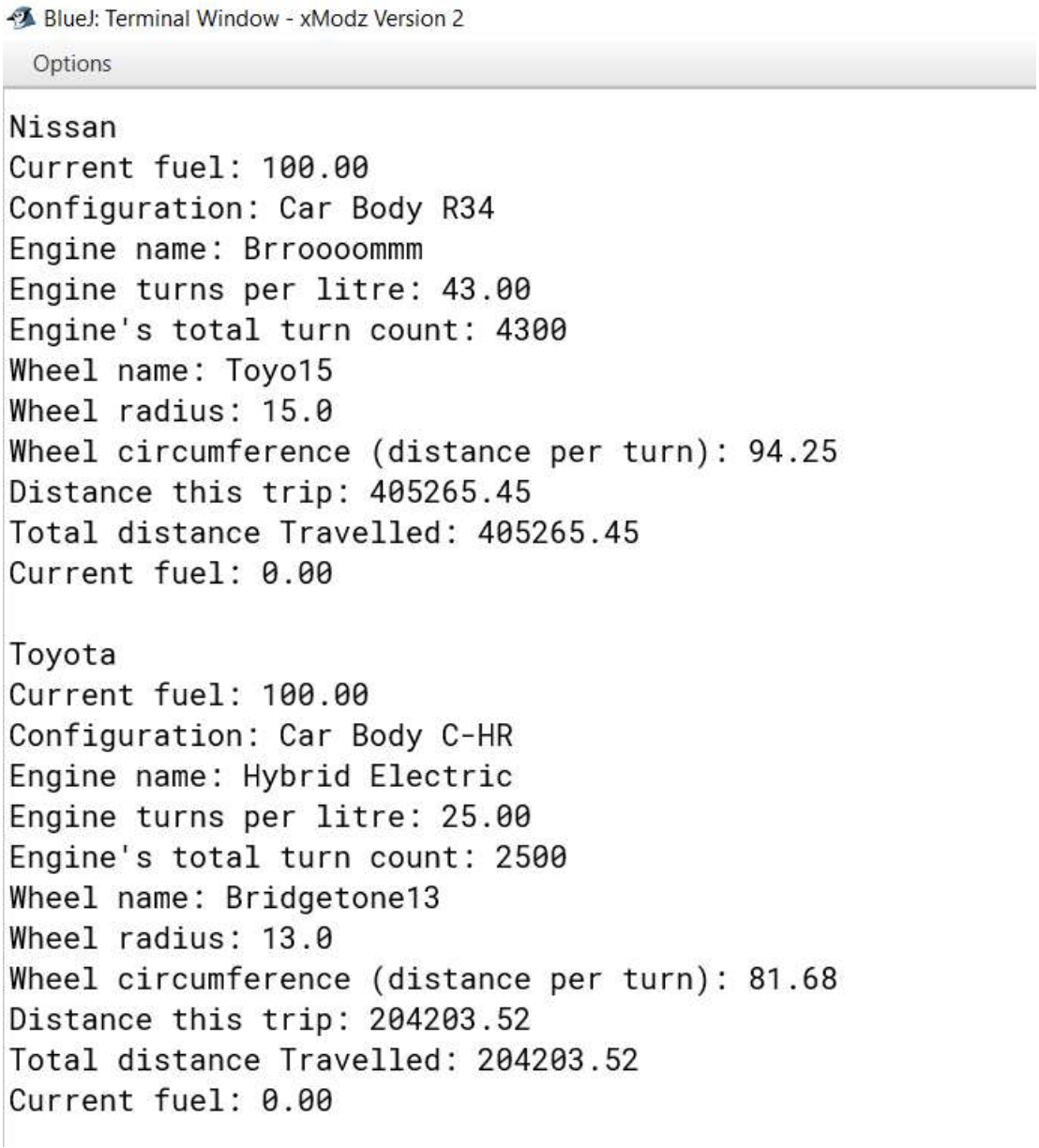
In the test class, we can ask the Car object to set the fuel of the car to a number (of the type double) using a method. The car object asks the engine object to set it's fuel field to that number (double) using a setter method.

In the test class, we can ask the car object to get the fuel. The car object asks the engine object to get that value using a method. The engine object returns that value using a getter method. This value can now be printed from the test class.

In the test class, the car calls a method to make the car drive. The car object calculates the number of turns it can make the wheel do with the fuel that it has (information retrieved using getter methods). The distance that the car will move is calculated based off of the number of turns of the wheel the car can do (previously calculated) and the circumference of the wheel. A call to calculate this is made in the car object, performed in the engine object and then returned to the car object. This distance is then added to the "total distance travelled", which is a variable stored in the car object. The car asks the engine to set its fuel to 0 because it has just used all of its fuel.

In the test class, the we call a function of the car object to print out a statement showing the value of many fields. These statements can be seen in the sample output below. The values of these fields are retrieved using getter methods of the engine object and methods of the engine object that the return the result of the getter methods in the wheel object.

Sample Output



```
Subaru
Current fuel: 100.00
Configuration: Car Body Impreza
Engine name: Brrrr
Engine turns per litre: 64.00
Engine's total turn count: 6400
Wheel name: OneOfEach12
Wheel radius: 12.0
Wheel circumference (distance per turn): 75.40
Distance this trip: 482548.63
Total distance Travelled: 482548.63
Current fuel: 0.00

Subaru 2nd Trip
Current fuel: 80.00
Configuration: Car Body Impreza
Engine name: Brrrr
Engine turns per litre: 64.00
Engine's total turn count: 11520
Wheel name: OneOfEach12
Wheel radius: 12.0
Wheel circumference (distance per turn): 75.40
Distance this trip: 386038.91
Total distance Travelled: 868587.54
Current fuel: 0.00
```

Can only enter input while your program is running

Testing

The setFuel() method.

The setFuel() method allows an input of any number without causing a compiler error. However, a negative fuel input does not make sense in this context. Furthermore, I have decided that the biggest amount of fuel that a car can be set to is 100.

The mutator in the Engine class is, therefore, protected. It is protected because the mutator will only change the value of the fuelLevel field if the input variable (fuel) is within the valid range.

If the input outside of the valid range, the field will not be assigned a value. This is okay because in this case the value of fuelLevel will always be 0. The fuelLevel field will either be assigned to 0 from the object's instantiation earlier in the code or the field will be set to 0 after executing the drive() function earlier in the code. This will not cause any errors or alter any other fields in other classes because when the drive() function is run again, the distance will be 0 and the "total distance travelled" will not change. I want this to happen in the case of an input of invalid range because the value of the "total distance travelled" shouldn't change when an invalid input is made.

```
public void setFuel(double fuel) //Set the fuel
{
    if(fuel >= 0 && fuel <= 100)
    {
        this.fuelLevel = fuel;
    }
    else
    {
        System.out.println("Fuel input is not in the correct range. Fuel input must be a value from 0 - 100 (inclusive).");
    }
}
```

Test

The value for fuel is too large.

```
car.setFuel(1000);
```

Result

The fuel level is not changed. The program gives the user an error message and gracefully finishes executing.

```
Fuel input is not in the correct range. Fuel input must be a value from 0 - 100 (inclusive).
Current fuel: 0.00
Configuration: Car Body X7
Engine name: DR9
Engine turns per litre: 43.00
Engine's total turn count: 0
Wheel name: Wichelin15
Wheel radius: 15.0
Wheel circumference (distance per turn): 94.25
Distance this trip: 0.00
Total distance Travelled: 0.00
Current fuel: 0.00
```

Test

The fuel level is negative.

```
car.setFuel(-50);
```

Result

The fuel level is not changed. The program gives the user an error message and gracefully finishes executing.

```
Fuel input is not in the correct range. Fuel input must be a value from 0 - 100 (inclusive).
Current fuel: 0.00
Configuration: Car Body X7
Engine name: DR9
Engine turns per litre: 43.00
Engine's total turn count: 0
Wheel name: Wichelin15
Wheel radius: 15.0
Wheel circumference (distance per turn): 94.25
Distance this trip: 0.00
Total distance Travelled: 0.00
Current fuel: 0.00
```

Protecting constructors.

The engine constructor

The value for the variable: “tpl”, which is a double may not be negative because it does not make sense for a car’s engine to have a negative turns per litre. Therefore, I have used the Math.abs() function to change any negative number input into a positive number.

```
public Engine(String name, double tpl)
{
    // initialise instance variables
    this.name = name;
    this.tpl = Math.abs(tpl);
}
```

Test

```
//Create an engine object.
Engine engine = new Engine("DR9", -43);
```

Result

```
Current fuel: 100.00
Configuration: Car Body X7
Engine name: DR9
Engine turns per litre: 43.00
Engine's total turn count: 4300
Wheel name: Wichelin15
Wheel radius: 15.0
Wheel circumference (distance per turn): 94.25
Distance this trip: 405265.45
Total distance Travelled: 405265.45
Current fuel: 0.00
```

Everything works fine.

The wheel constructor.

It does not make sense to have a negative wheel radius so I have used the Math.abs() function in the constructor to change any negative number input into a positive number. I also included this in the calculation of the circumference of the wheel in the constructor as well.

```
public Wheel(String name, double radius)
{
    // initialise instance variables
    this.name = name;
    this.radius = Math.abs(radius);
    circumference = 2 * Math.PI * Math.abs(radius);
}
```

Test

```
//Create a wheel object.
Wheel wheel = new Wheel("Wichelin15", -15);
```

Result

```
Current fuel: 100.00
Configuration: Car Body X7
Engine name: DR9
Engine turns per litre: 43.00
Engine's total turn count: 4300
Wheel name: Wichelin15
Wheel radius: 15.0
Wheel circumference (distance per turn): 94.25
Distance this trip: 405265.45
Total distance Travelled: 405265.45
Current fuel: 0.00
```

Everything works fine.