Spring 2021

CS-240: Object-oriented Programming

Lab-12 Manual

Object Associations

GIFT School of Engineering and Applied Sciences

1.3
3/30/2020

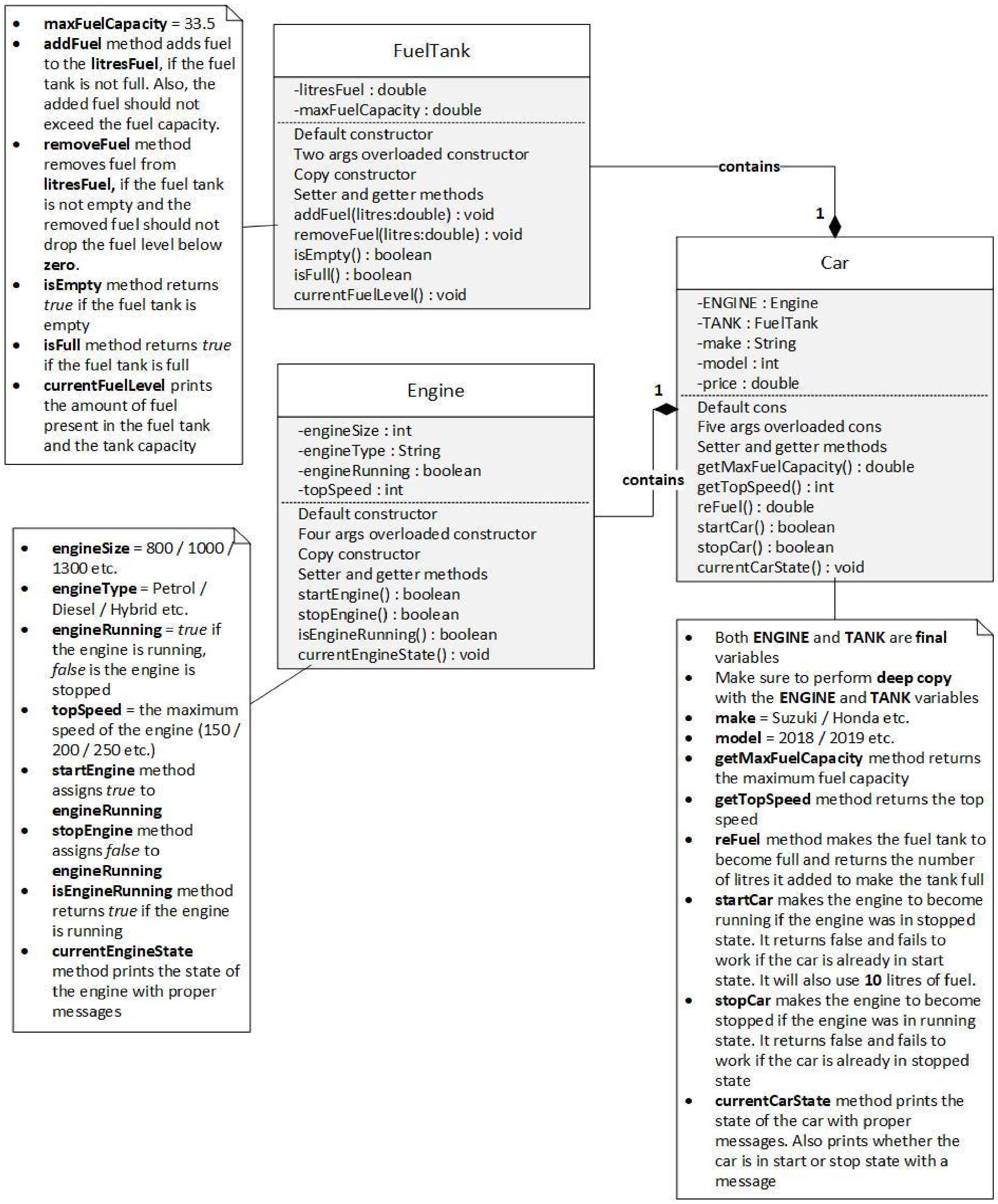
Task #1: Object Associations (Composition)

In this task, you are being asked to write Java code that demonstrates composition among objects.

NOTE: Write your classes and the *main* method in separate files.

Make sure to implement the composition through **`final`** variables and **perform deep copy** of those variables**.** You can see that in the **`Car`** class, both **`ENGINE`** and **`TANK`** are written in capitals, so they are both **`final`** variables. For details, see the notes attached to the **`Car`** class.

1. Create all classes having names as per the below UML diagram, and **`UsingCar.java`** having the **`main`** method**.**
2. Inside the **`main`** method, create instances of all part classes (**`Engine`** and **`FuelTank`**) using the overloaded constructors. Assign appropriate values to the state of all instances.
3. Then, create an instance of **`Car`** class using the overloaded constructor making sure to assign proper values to all arguments.
4. Finally, demonstrate the use of all methods as listed in the UML diagram for all created objects.



- **maxFuelCapacity** = 33.5
- **addFuel** method adds fuel to the **litresFuel**, if the fuel tank is not full. Also, the added fuel should not exceed the fuel capacity.
- **removeFuel** method removes fuel from **litresFuel,** if the fuel tank is not empty and the removed fuel should not drop the fuel level below **zero**.
- **isEmpty** method returns *true* if the fuel tank is empty
- **isFull** method returns *true* if the fuel tank is full
- **currentFuelLevel** prints the amount of fuel present in the fuel tank and the tank capacity

```
FuelTank
-litresFuel : double
-maxFuelCapacity : double
Default constructor
Two args overloaded constructor
Copy constructor
Setter and getter methods
addFuel(litres:double) : void
removeFuel(litres:double) : void
isEmpty() : boolean
isFull() : boolean
currentFuelLevel() : void
```

```
Engine
-engineSize : int
-engineType : String
-engineRunning : boolean
-topSpeed : int
Default constructor
Four args overloaded constructor
Copy constructor
Setter and getter methods
startEngine() : boolean
stopEngine() : boolean
isEngineRunning() : boolean
currentEngineState() : void
```

```
Car
-ENGINE : Engine
-TANK : FuelTank
-make : String
-model : int
-price : double
Default cons
Five args overloaded cons
Setter and getter methods
getMaxFuelCapacity() : double
getTopSpeed() : int
reFuel() : double
startCar() : boolean
stopCar() : boolean
currentCarState() : void
```

- **engineSize** = 800 / 1000 / 1300 etc.
- **engineType** = Petrol / Diesel / Hybrid etc.
- **engineRunning** = *true* if the engine is running, *false* is the engine is stopped
- **topSpeed** = the maximum speed of the engine (150 / 200 / 250 etc.)
- **startEngine** method assigns *true* to **engineRunning**
- **stopEngine** method assigns *false* to **engineRunning**
- **isEngineRunning** method returns *true* if the engine is running
- **currentEngineState** method prints the state of the engine with proper messages

- Both **ENGINE** and **TANK** are **final** variables
- Make sure to perform **deep copy** with the **ENGINE** and **TANK** variables
- **make** = Suzuki / Honda etc.
- **model** = 2018 / 2019 etc.
- **getMaxFuelCapacity** method returns the maximum fuel capacity
- **getTopSpeed** method returns the top speed
- **reFuel** method makes the fuel tank to become full and returns the number of litres it added to make the tank full
- **startCar** makes the engine to become running if the engine was in stopped state. It returns false and fails to work if the car is already in start state. It will also use **10** litres of fuel.
- **stopCar** makes the engine to become stopped if the engine was in running state. It returns false and fails to work if the car is already in stopped state
- **currentCarState** method prints the state of the car with proper messages. Also prints whether the car is in start or stop state with a message

Task #2: Object Associations (Composition)

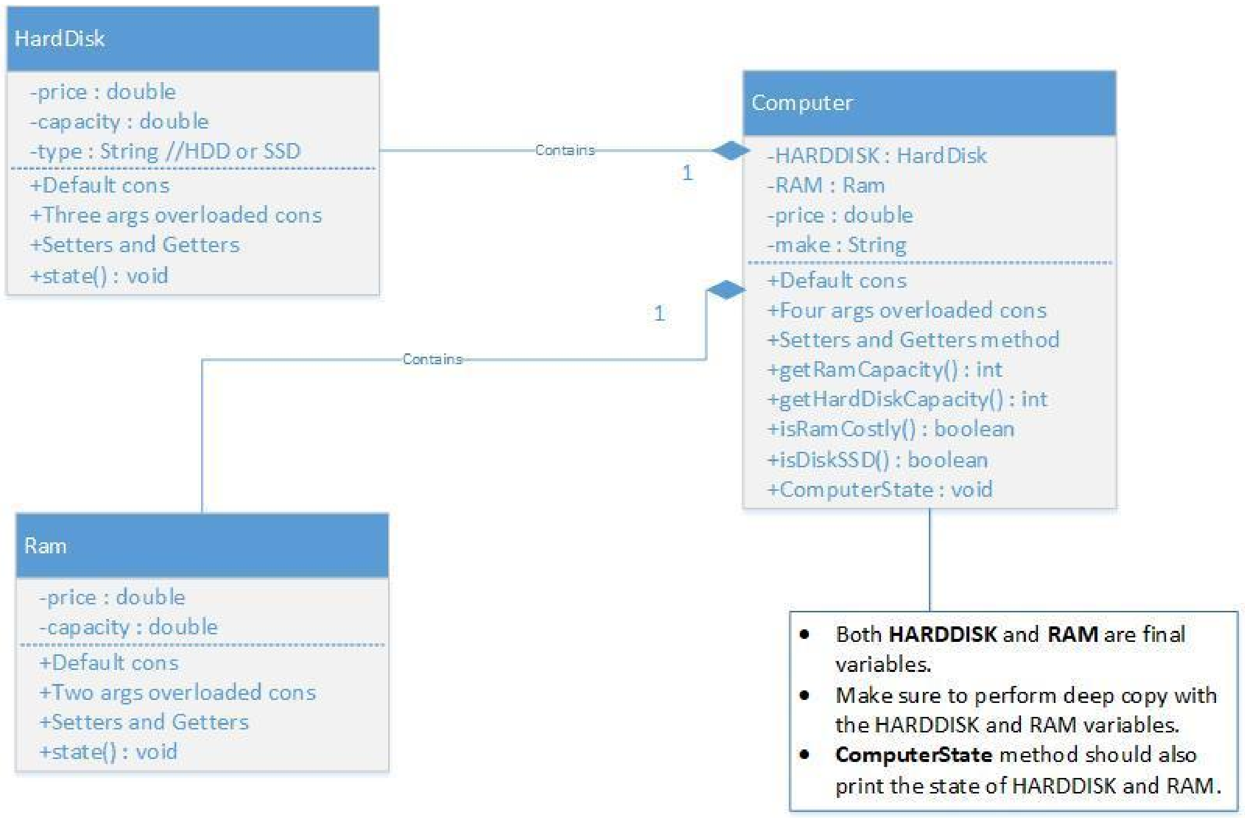
In this task, you are being asked to write Java code that demonstrates composition among objects.

NOTE: Write your classes and the *main* method in separate files.

Make sure to implement the composition through **`final`** variables and **perform deep copy** of those variables**.** You can see that in the **`Computer`** class, both **`HARDDISK`** and **`RAM`** are written in capitals, so they are both **`final`** variables. For details, see the notes attached to the **`Computer`** class.

1. Create all classes having names as per the below UML diagram, and **`UsingComputer.java`** having the **`main`** method**.**
2. Inside the **`main`** method, create instances of all part classes (**`HardDisk`** and **`Ram`**) using the overloaded constructors. Assign appropriate values to the state of all instances.
3. Then, create an instance of **`Computer`** class using the overloaded constructor making sure to assign proper values to all arguments.
4. Finally, demonstrate the use of all methods as listed in the UML diagram for all created objects.

`



- Both **HARDDISK** and **RAM** are final variables.
- Make sure to perform deep copy with the HARDDISK and RAM variables.
- **ComputerState** method should also print the state of HARDDISK and RAM.