TMA01

Question 1.a.i  
private int getBodyXPos()  
{  
 int noseXPos;  
 noseXPos = this.nose.getXPos();  
 return noseXPos;  
}

Question 1.a.ii  
private int getBodyYPos()  
{  
 int noseYPos;  
 noseYPos = this.nose.getYPos()+20; //move body down 20units

return noseYPos;  
}

Question 1.a.iii  
private int getJetXPos()  
{  
 int bodyXPos;  
 bodyXPos = this.nose.getBodyXPos()+5; //calculations seem off, but this is what was needed for my display accessibility accommodations  
 return bodyXPos;  
}

Question 1.a.iv  
private int getJetYPos()  
{  
 int bodyYPos;  
 bodyYPos = this.nose.getBodyYPos()+20;  
 return bodyYPos;  
}

Question 1.b  
public void moveRocketBy(int anInt)  
{  
 int noseYPos = this.nose.getYPos(); //find nose position  
 noseYPos = noseYPos - anInt;   
 this.nose.setYPos(noseYPos); // move nose & follow suit for other components  
 this.body.setYPos(getBodyYPos());  
 this.jet.setYPos(getJetYPos());  
}

Question 1.c.i  
public void pulse1()  
{  
 this.jet.setDiameter(6);   
 this.jet.setColour(OUColour.RED);   
 this.jet.setXPos(this.getJetXPos()+2); *//calculations seem off, but this is what was needed for my display accessibility accommodations*  
 this.jet.setYPos(this.getJetYPos());  
}

Question 1.c.ii  
public void pulse2()  
{  
 this.jet.setDiameter(12);   
 this.jet.setColour(OUColour.ORANGE);   
 this.jet.setXPos(this.getJetXPos()-1); *//calculations seem off, but this is what was needed for my display accessibility accommodations*  
 this.jet.setYPos(this.getJetYPos());  
}  
Question 1.c.iii  
public void pulse3()  
{  
 this.jet.setDiameter(24);   
 this.jet.setColour(OUColour.RED);   
 this.jet.setXPos(this.getJetXPos()-7); *//calculations seem off, but this is what was needed for my display accessibility accommodations*  
 this.jet.setYPos(this.getJetYPos());  
}

Question 1.d  
public void ignition()  
{  
 //do this 5 times  
 for (int i=0; i<5; i++){  
 pulse1();  
 this.delay(200);   
 pulse2();  
 this.delay(200);   
 pulse3();  
 this.delay(200);   
 }  
}  
  
Question 1.e  
public void animateRocket(int speed)  
{  
 //to be written in Q1(e)   
 int c=0; //create counter for switch   
 for (int i=1; i<101; i++){  
 switch (c){  
 case 0:   
 pulse1();  
 this.delay(200);   
 c++;   
 case 1:   
 pulse2();  
 this.delay(200);   
 c++;  
 case 2:   
 pulse3();  
 this.delay(200);   
 c++;  
 case 3:   
 this.moveRocketBy(speed);   
 c=0; //last case. Reset counter  
 }  
 }  
}

Question 1.f  
public void launch()  
{  
 String userEntry = JOptionPane.showInputDialog("Enter speed:"); //get user entry  
 int input = Integer.parseInt(userEntry) //convert to int  
 int pos = this.nose.getYPos();  
 int limit = pos-this.nose.getHeight(); //how far is nose from edge of screen  
 if (input >= limit){   
 System.out.println("speed too high"); //will exceed limit   
 } else {   
 ignition();  
 animateRocket(input);   
 }  
}

Question 1.g

In the completed solution, the only method which needs to remain public is launch(). The other methods are utilised solely in the Rocket class, thus the private modifier would be suitable.

Question 2.a

|  |  |
| --- | --- |
| **Requirement** | **Your answer** |
| 1(a) Composite class | Engine |
| 1(b) Component class | Car |
| 1(c) Relationship between the two | A Car has an engine |
| 2(a) Composite class instance variables and examples of use | car, of type Car (to provide composition relationship) – perhaps could be allocated as reg/VIN number  mileage of type int, how many kilometers this engine has covered.  capacity of type int, how large is the engine capacity, in cubic centimetres  fuelType of type String. What type of fuel source does the engine need. E.g petrol, diesel, hybrid, electricity**.** |
| 2(b) Component class instance variables and examples of use | Manufacturer of type String, the company by whom the car was made  Model of type String, the model of the car to which this engine belongs. E.g. “340i”, “Focus”, “Vectra”  fuelLevel of type int (or double), how many litres of fuel is currently in the tank. E.g. 36  yearRegistered of type int, which year the car was registered |
| 3 Description of method (not toString()) that allows the composite class to communicate with the component class in order to compute some value | Engine has a method of drive() which takes an argument for distance, in kilometeres.  Using the distance, we then calculate if car.fuelLevel is enough to complete the journey in the car, assuming an average fuel economy of 50 KM/L. If there is enough fuel the engine.mileage and car.fuelLevel variables are adjusted to reflect this journey. Otherwise a message is printed advising that not enough fuel remains. This method is also used to decrease the fuelLevel of the car by an appropriate amount.  A future improvement we could make to this would be to change the method so that if not enough fuel is in the car, we enter a while loop in which we continually call the car.refuel() method until there is enough fuel to complete the journey. |
| 4 Other interesting features of the class | Engine and car provides getter and setter methods for it’s instance variables. **<<ENGINE = TO DO CAR = DONE>>**  Car also a method refuelNeeded() which returns a Boolean value based on the fuelLevel. **<<DONE>>**  Car has a toString() method which returns a simple statement about the current variable values. **<<DONE>>** |

Question 2.b

/\*\*

\* TMA02 Q2 Composite

\*

\* @author Michael Witts

\* @version 2021/02/02

\*/

public class Engine

{

// instance variables

private Car car; // an engine belongs to/has-a car of type Car

private int mileage; // the distance an engine has covered, in kilometers although i couldnt find the corret adjective

private int capacity; // how big is the engine

private String fuelType; //what powers the combustion/engine

/\*\*

\* Constructor for when we have no known values

\*/

public Engine() //zero argument constructor -- we dont know any values.

{

// initialise instance variables

this.car = null;

this.mileage=0;

this.capacity=0;

this.fuelType = null;

}

/\*\*

\* Constructor for when we know all values

\*/

public Engine(Car aCar, int aMileage, int aCapacity, String aFuelType) // all arguments initialised

{

// initialise instance variables

this.car = aCar;

this.mileage= aMileage;

this.capacity= aCapacity;

this.fuelType = aFuelType;

}

/\*\*

\* Getter method for car object

\*/

public Car getCar()

{

return this.car;

}

/\*\*

\* Getter method to return total distance engine has travelled

\*/

public int getMileage()

{

return this.mileage;

}

/\*\*

\* Getter method to find engine capicty.

\*/

public int getCapacity()

{

return this.capacity;

}

/\*\*

\* Getter method to return fuelType needed for the engine

\*/

public String getFuelType()

{

return this.fuelType;

}

/\*\*

\* Setter method to set manufacturer of component Car object

\*/

public void setCarManufacturer(String aManufacturer)

{

this.car.setManufacturer(aManufacturer);

}

/\*\*

\* Setter method to set model of component Car object

\*/

public void setCarModel(String aModel)

{

this.car.setModel(aModel);

}

/\*\*

\* Setter method to set mileage

\*/

public void setMileage(int newMileage)

{

this.mileage = newMileage;

}

/\*\*

\* Setter method to set capacity of engine

\*/

public void setCapacity(int newCapacity)

{

this.capacity = newCapacity;

}

/\*\*

\* Setter method to set fuelType

\*/

public void setFuelType(String newFuelType)

{

this.fuelType = newFuelType;

}

/\*\*

\* Method to drive the car. Takes distance parameter (in KM). Executes the following

\* get current fuel level

\* calculate amount of fuel needed for journey, assuming economy of 50KM/l

\* if current fuel is less than amount needed, prints message adivising of this

\* if sufficient fuel available mileage increases by distance and remaining fuel is reduced by one litre per 50 kilometers travelled

\*/

public void drive(int distance)

{

int fuelLevel = this.car.getFuelLevel();

int fuelNeededForJourney = distance/50;

if(fuelLevel > fuelNeededForJourney)

{

int remainingFuel = fuelLevel - fuelNeededForJourney;

this.mileage =+ distance;

this.car.setFuelLevel(remainingFuel);

}

else

{

System.out.println("Not enough fuel refuel first");

}

}

/\*\*

\* returns a summary describing the engine and its variables. if a valid car object exists then we include the manufacturer in the returned statement

\*/

public String toString()

{

String message = null;

if(car != null)

{

String manufacturer = this.car.getManufacturer();

message = String.format("The %s engine from this %s has a capacity of %s and driven %s kilometers.", this.fuelType, manufacturer, this.capacity, this.mileage);

} else

{

message = String.format("This %s engine has a capacity of %s and driven %s kilometers. It is not currently installed in any car", this.fuelType, this.capacity, this.mileage);

}

return message;

}

}

/\*\*

\* TMA02 Q2 Component class

\*

\* @author Michael Witts

\* @version 2021/02/02

\*/

public class Car

{

// instance variables - replace the example below with your own

//private int x;

private String manufacturer; // who makes the car e.g. "BMW"

private String model; // which model car it is e.g. "340i"

private int fuelLevel; // how much fuel is in the car, in litres e.g. 54

private int yearRegistered; // year the car was registered -- this can differ from the engine e.g. 2017

/\*\*

\* Constructor for objects of class Car with unknown values

\*/

public Car()

{

// initialise instance variables

this.manufacturer = null; // indicates that we dont know which car this is

this.model = null; // indicates that we dont know which car this is

this.fuelLevel = 0; // indicates we dont know the amount of fuel in the car

this.yearRegistered = 0; // we dont know when this car has been registered, perhaps not yet

}

/\*\*

\* A constructor for when we know the make and model of the car

\*/

public Car(String aManufacturer, String aModel)

{

this.manufacturer = aManufacturer;

this.model = aModel;

}

/\*\*

\* A constructor for when we know all details of the car

\*/

public Car(String aManufacturer, String aModel, int fuelAmount, int year)

{

this.manufacturer = aManufacturer;

this.model = aModel;

this.fuelLevel = fuelAmount;

this.yearRegistered = year;

}

/\*\*

\* getter method to find manufacturer of car

\*/

public String getManufacturer()

{

return this.manufacturer;

}

/\*\*

\* getter method to find model of car

\*/

public String getModel()

{

return this.model;

}

/\*\*

\* getter method to find year car was made

\*/

public int getAge()

{

return this.yearRegistered;

}

/\*\*

\* getter method to find remaining fuel in the car

\*/

public int getFuelLevel()

{

return this.fuelLevel;

}

/\*\*

\* setter method to change manufacturer of car

\*/

public void setManufacturer(String aManufacturer)

{

this.manufacturer = aManufacturer;

}

/\*\*

\* setter method to change model of car

\*/

public void setModel(String aModel)

{

this.model = aModel;

}

/\*\*

\* setter method to change the year the car was made

\*/

public void setAge(int age)

{

this.yearRegistered = age;

}

/\*\*

\* setter method to change the absolute amount of remaining of car

\*/

public void setFuelLevel(int volume)

{

this.fuelLevel = volume;

}

/\*\*

\* method to return all infromation known of car

\*/

public String toString()

{

String message = String.format("This %s %s was built in %s and has %s fuel", this.manufacturer, this.model, this.yearRegistered, this.fuelLevel);

return message;

}

/\*\*

\* method to check if the car needs refuelling

\*/

public boolean refuelNeeded()

{

return(this.fuelLevel < 10 ? true : false ); // ternary used for simplicity of code

}

/\*\*

\* setter method to increase remaining fuel in the car by adding a certain volume

\*/

public void refuel(int volume)

{

this.fuelLevel =+ volume;

}

}

Question 2.c.i

Car c = new Car("BMW", "340i", 54, 2007);

Engine e = new Engine (c, 25060, 3000, "Petrol");

e.toString();

Question 2.c.ii

Car c = new Car("BMW", "340i", 54, 2007);

Engine e = new Engine (c, 25060, 3000, "Petrol");

e.drive(1000); // uses 20L fuel

e.drive(51); // uses 1L fuel

e.drive(49); // uses 0l fuel

e.drive(65536); // not enough fuel refuel first

e.drive(-65536); // adds 1310 fuel. *Reason: no check for negative int in drive()method so distance/50 == -1310. which is used in calculations and "subtracted from fuel level". Solutions could be to check > 0 before fuel level check.*

Question 3.a

/\*\*  
 \* TMA02 Q3  
 \*   
 \* @author Michael Witts   
 \* @version 2021/01/25  
 \*/

public class test extends Pet  
{  
}

Question 3.b

/\*\*  
 \* TMA02 Q3  
 \*   
 \* @author Michael Witts   
 \* @version 2021/01/25  
 \*/

public class test extends Pet  
{  
 private int happiness;  
 private int energyLevel;

public int getHappiness()  
 {   
 return happiness;  
 }  
 private int getEnergyLevel()  
 {  
 return energyLevel;  
 }  
}

Question 3.c

public Dog(String aName, String aDescription)  
{  
 // initialise instance variables  
 // x = 0;  
 super(aName, aDescription);  
 happiness=2;  
}

Question 3.d

private int decrementHappiness()  
{  
 if(happiness >= 0){  
 happiness--;  
 }  
 return happiness;  
}  
private int decrementEnergyLevel()  
{  
 if (energyLevel >= 0)  
 {  
 energyLevel--;  
 }  
 return energyLevel;  
}  
private int incrementHappiness()  
{  
 happiness++;  
 return happiness;  
}  
private int incrementEnergyLevel()  
{  
 energyLevel++;  
 return energyLevel;  
}

Question 3.e.i

public void walkies()  
{  
 if (energyLevel > 0)  
 {  
 System.out.println("I'm going for a walk!");  
 happiness++;  
 System.out.println("I'm getting hungry");  
 energyLevel--;  
 }  
}

Question 3.e.ii

public boolean sleep()  
{  
 if (happiness <= 0)   
 {   
 System.out.println("Not happy, can't sleep");  
 return false;  
 }  
 else if(energyLevel <= 0)  
 {  
 System.out.println("Hungry, can't sleep");  
 return false;  
 } else  
 {  
 return true;  
 }  
}

Question 3.e.iii

public void noWalkies()  
{  
 System.out.println("No walkies :-(");  
 if (happiness >= 0)  
 {  
 happiness--;  
 }  
}

Question 3.f.i

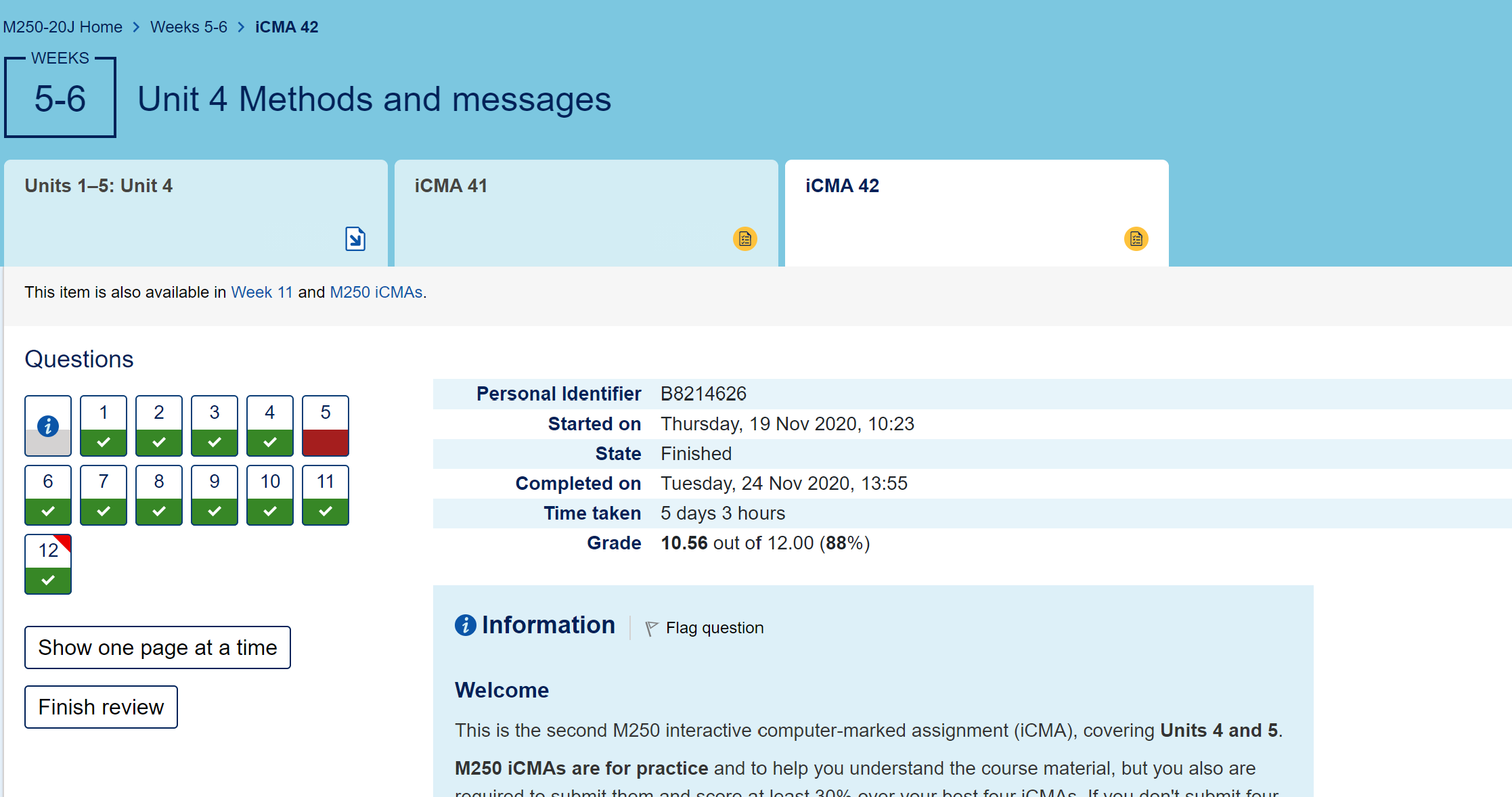
/\*\*  
 \* Write a description of interface Lovable here.  
 \*   
 \* @author Michael Witts  
 \* @version 2021/01/25  
 \*/  
public interface Lovable  
{  
 public void stroke();  
 public boolean canStroke();  
 public void feed();  
}

Question 3.f.ii  
/\*\*  
 \* TMA02 Q3  
 \*   
 \* @author Michael Witts   
 \* @version 2021/01/25  
 \*/  
public abstract class Dog extends Pet implements Lovable  
{...}

Question 3.f.iii

PondFish extends the Fish superclass. As the Fish class defines the methods from the Lovable interface the PondFish there is no need to redefine within the subclass unless the implementation is different.

Question 4.a



Question 4.b

For me, the questions which helped develop my understanding were eleven and twelve. The reason for this, is that I find that I learn quicker, and more permanently, by using and implementing new ways of doing things.