```
package fr.fboschet.voiture.ui;

import java.awt.BorderLayout;
import java.awt.Font;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JSpinner;

import fr.fboschet.voiture.SecondHandVehicle;
import fr.fboschet.voiture.builder.CardinalFactory;
import fr.fboschet.voiture.builder.ComponentFactory;

// NO MORE
public class SecondHandVehicleFrame{

	private static SecondHandVehicleFrame INSTANCE;

	private SecondHandVehicleFrame() {}

	public static SecondHandVehicleFrame getInstance() {
		if (INSTANCE == null) {
			INSTANCE = new SecondHandVehicleFrame();
		}
		return INSTANCE;
	}

	public void run(SecondHandVehicle vehicle) {
		ComponentFactory factory = ComponentFactory.getInstance();
		CardinalFactory cardinal = CardinalFactory.getInstance();

		JFrame main = factory.getJFrame("SECOND HAND VEHICLE OPTIONS");

		JPanel mainPanel = factory.getVerticalPanel();

		//Panel that display information (make, model, nbOfOwner(s))
		JPanel infoPanel = factory.getFlowPanel();

		JPanel makePanel = factory.getHorizontalPanel();
		//factory.addBorders.accept(makePanel);
		JLabel make = factory.getJLabel(vehicle.getMake()+" ", 60, Font.
BOLD);
		JLabel model = factory.getJLabel(vehicle.getModel(), 60, Font.PL
AIN);
		JLabel year = factory.getJLabel(vehicle.getYear()+"", 12, Font.I
TALIC);
		makePanel.add(make);
		makePanel.add(model);
		makePanel.add(year);
		infoPanel.add(makePanel);

		// Panel that display the number of owners this car had
		JPanel ownerPanel = factory.getPanel();
		int owners = vehicle.getNumberOfOwners();
		JLabel now = factory.getJlabel(cardinal.getCardinal(owners), 13)
;
		JLabel owner = factory.getJlabel("owner", 13);
		ownerPanel.add(now);
		ownerPanel.add(owner);
		infoPanel.add(ownerPanel);
		//

		// The panel that display the value of the vehicle and
		// display the edit mode when needed
```



```
		final JPanel valuePanel = factory.getBorderPanel();
		//factory.addBorders.accept(valuePanel);

		// A panel to display the current value and a button to switch t
o edit mode
		JPanel displayValue = factory.getBorderPanel();
		JLabel value = factory.getJLabel(vehicle.getValue()+"M-^@", 100,
 Font.BOLD);
		JButton editMode = factory.getJButton("decrease");

		displayValue.add(value);
		displayValue.add(editMode, BorderLayout.EAST);

		// A hidden panel which is shown when edit mode is active
		JPanel editValue = factory.getBorderPanel();

		// We use a JSpinner to decrease the value of the vehicle
		// And it also ensure that the value of the spinner is a double
		JSpinner editField = factory.getDoubleJSpiner(vehicle.getValue()
);
		JPanel buttons = factory.getGridPanel(2,  1);

		JButton update = factory.getJButton("update");
		JButton discard = factory.getJButton("discard");

		buttons.add(update);
		buttons.add(discard);

		editValue.add(editField, BorderLayout.CENTER);
		editValue.add(buttons, BorderLayout.EAST);

		// listeners
		// When we click on the decrease button
		editMode.addActionListener(new ActionListener() {
			@Override
			public void actionPerformed(ActionEvent e) {
				if (e.getActionCommand() == editMode.getActionCo
mmand()) {
					// We display the edit panel in place of
 the display panel
					valuePanel.removeAll();
					editField.setValue(vehicle.getValue());
					valuePanel.add(editValue);
					main.pack();
				}
			}
		});

		update.addActionListener(new ActionListener() {
			@Override
			public void actionPerformed(ActionEvent e) {
				if (e.getActionCommand() == update.getActionComm
and()) {
					double newValue = 0.0;
					newValue = (Double)editField.getValue();
					// This shouldn't be happening since we
set the SpinnerNumberModel with a
					// maximum of vehicle.value()
					if (newValue > vehicle.getValue()) {
						JOptionPane.showMessageDialog(ma
in, "You can only reduce the value of the vehicle,\n value : "+value.getText(),
"Warning", JOptionPane.WARNING_MESSAGE);
					}
					else {
						// we decrease the value of the
current vehicle
						vehicle.decreaseValue(newValue);
						value.setText(vehicle.getValue()
+"M-^@");
```

```
                                        // and display the display panel
                                        valuePanel.removeAll();
                                        valuePanel.add(displayValue);
                                        main.pack();
                                }
                        }
                }
        });

        discard.addActionListener(new ActionListener() {
                @Override
                public void actionPerformed(ActionEvent e) {
                        if (e.getActionCommand() == discard.getActionCom
mand()) {
                                // discard the decrease of the value <=>
 do nothing but switch display/edit
                                valuePanel.removeAll();
                                valuePanel.add(displayValue);
                                main.pack();
                        }
                }
        });

        valuePanel.add(displayValue);
        //


        // Display tax and qualifiedForScrappage()
        JPanel taxPanel = factory.getPanel();

        // display the tax
        JPanel taxDisplay = factory.getFlowPanel();
        JLabel taxLabel = factory.getJlabel("Tax : ", 20);
        JButton updateTax = factory.getJButton("calculate");
        updateTax.addActionListener(new ActionListener() {
                @Override
                public void actionPerformed(ActionEvent e) {
                        if (e.getActionCommand() == updateTax.getActionC
ommand()) {
                                JSpinner spinner = factory.getIntJSpinne
r();
                                int option = JOptionPane.showOptionDialo
g(null,
                                                spinner,
                                                "Engine Size",
                                                JOptionPane.OK_CANCEL_OP
TION,
                                                JOptionPane.QUESTION_MES
SAGE,
                                                null, null, null);
                                if (option == JOptionPane.OK_OPTION) {
                                        double tax = vehicle.calculateTa
xPayable((Integer)spinner.getValue());
                                        taxLabel.setText("Tax : " + tax
+ "M-^@");
                                        taxDisplay.remove(updateTax);
                                }
                        }
                }
        });

        taxDisplay.add(taxLabel);
        taxDisplay.add(updateTax);

        // display scrappage
        JPanel scrappageDisplay = factory.getPanel();
        JLabel scrappage = factory.getJlabel("Good for scrappage? "+vehi
cle.qualifyForScrappage(), 20);
        scrappageDisplay.add(scrappage);
```



```
        taxPanel.add(taxDisplay);
        taxPanel.add(scrappageDisplay);
        //

        mainPanel.add(infoPanel);
        mainPanel.add(factory.getJSeparator());
        mainPanel.add(valuePanel);
        mainPanel.add(factory.getJSeparator());
        mainPanel.add(taxPanel);

        main.add(mainPanel);
        main.pack();
        main.setVisible(true);

    }
}
```

```
package fr.fboschet.voiture.test;

import java.util.ArrayList;
import java.util.List;

import javax.swing.JOptionPane;

import fr.fboschet.voiture.SecondHandVehicle;
import fr.fboschet.voiture.Vehicle;
import fr.fboschet.voiture.loader.JSONParser;
import fr.fboschet.voiture.loader.SeconHandVehicleLoader;
import fr.fboschet.voiture.ui.DialogFactory;
import fr.fboschet.voiture.ui.SecondHandVehicleFrame;

public class SecondHandVehicleTester{
        public static void main(String args[]){

                //              // Declare an Array of 5 SecondHandVehicles call
ed vehicles
                //              final int NUMBER_OF_VEHICLES = 5;
                //              SecondHandVehicle[] vehicles = new SecondHandVeh
icle[NUMBER_OF_VEHICLES];
                //
                //              // Create 5 new SecondHandVehicle objects with i
nitial values...
                //          SecondHandVehicle v1 = new SecondHandVehicle("Opel",
"Astra", 1999, 800.00, 2);
                //          SecondHandVehicle v2 = new SecondHandVehicle("Toyota"
, "RAV4", 2006, 15000.00, 2);
                //          SecondHandVehicle v3 = new SecondHandVehicle("Mazda",
 "323F", 1998, 1000.00, 1);
                //          SecondHandVehicle v4 = new SecondHandVehicle("Ford",
"Fiesta", 2009, 8000.00, 3);
                //          SecondHandVehicle v5 = new SecondHandVehicle("Alfa",
"Romeo", 2005, 7500.00, 1);
                //
                //              //...and add them to the array called vehicles
                //              vehicles[0]=v1;
                //              vehicles[1]=v2;
                //              vehicles[2]=v3;
                //              vehicles[3]=v4;
                //              vehicles[4]=v5;


                // It’s very boring to add vehicles in the main mathod. So I mad
e a lil’ JSON parser and a VehicleBuilder
                JSONParser jp = SeconHandVehicleLoader.getInstance("res/car1.jso
n");

                List<Vehicle> vehicles = new ArrayList<>();
                // popultate our list with vehicles from the json file
                jp.populate(vehicles);

                SecondHandVehicleFrame frameFactory = SecondHandVehicleFrame.get
Instance();
                DialogFactory dialogFactory = DialogFactory.getInstance();

                // showInputDialog() passing in the vehicles array
                // returnedValue will get the selected vehicle
                SecondHandVehicle currVehicle = dialogFactory.choiceDialog(vehic
les);

                // I.E. if the user selected one vehicle
                if (currVehicle != null) {
                        int hash = 0;
                        int attempt = 0;
                        // 3 attempt to input the correct password
                        while(hash != currVehicle.getPwd() && attempt < 3 && has
h != -1) {
```



```
                                hash = dialogFactory.passwordDialog();
                                attempt++;
                        }
                        if (hash == currVehicle.getPwd() && attempt < 4) {
                                frameFactory.run(currVehicle);
                        }
                        else if (hash != -1) {JOptionPane.showMessageDialog(null
, "Too many failed attempts", "Failure", JOptionPane.ERROR_MESSAGE);}
                }
        }
}
```

```
package fr.fboschet.voiture.loader;

import java.io.File;
import java.io.IOException;
import java.nio.file.Files;
import java.util.List;

import org.json.JSONArray;
import org.json.JSONException;
import org.json.JSONObject;

import fr.fboschet.voiture.Vehicle;
import fr.fboschet.voiture.builder.SecondHandVehicleBuilder;
import fr.fboschet.voiture.builder.VehicleBuilder;

public class SeconHandVehicleLoader implements JSONParser {

	// TODO: I should use a List of file and make the populate method thread
−safe
	// in this case this is not relevant since we call the parser only once.
	private static File toParse;

	private static SeconHandVehicleLoader INSTANCE;

	private SeconHandVehicleLoader() {}

	public static SeconHandVehicleLoader getInstance(String path) {
		if (INSTANCE == null) {
			INSTANCE = new SeconHandVehicleLoader();
		}
		toParse = new File(path);
		return INSTANCE;
	}

	// Populate the given list of Vehiclse with those from the File toParse
	@Override
	public void populate(List<Vehicle> lv) {
		StringBuilder sb = new StringBuilder();
		JSONObject json = new JSONObject();
		VehicleBuilder vb = SecondHandVehicleBuilder.getInstance();

		try {
			// Read a file (java−lambda style)
			Files.lines(toParse.toPath()).forEach((String s) −> sb.a
ppend(s));
		} catch (IOException e) {
			// in case the file don’t exist
			sb.append("{}");
		}
		// cerate a jsonobject from our file
		json = new JSONObject(sb.toString());


		// I know this isn’t the prettiest way to parse json files
		// TODO: a correct interpreter of our json file
		try {
			JSONArray vehicles = json.getJSONArray("vehicles");
			for(int i = 0; i < vehicles.length(); i++) {
				// call the vehicleBuilder for each vehicule if
the json file
				lv.add(vb.build(vehicles.getJSONObject(i)));
			}
		}catch (JSONException e) {
			// if the file isn’t well formated, return a blank vehic
le
			lv.add(vb.getBlankVehicule());
		}
	}
}
```

```
package fr.fboschet.voiture;

import java.time.Year;


// NOTE FOR THE PERSON WHO WRITE THIS FILE
// ======================================
//
// Why are you writing code like:
// ‘’
// if (boolean_expression)
//      return true
// else
//      return false
// ‘’
//
// Do you even know how boolean algebra works?
// Why aren’t you writing :
// ‘’
// return boolean_expression
// ‘’
//

//Class to model a Vehicle
//Used for inheritance only so its abstract
public abstract class Vehicle{
        // Instance Variables
        protected int number;
        protected String make;
        protected String model;
        protected int year;
        protected double value;
        // hash of the password
        protected int pwdHash;

        private static int nextUniqueNumber=1;
        // Next available unique Vehicle number
        // static − means nextUniqueNumber is SHARED
        // amongst all Vehicle objects, so if one
        // of them change it, it is changed for all.

        private final double TAX_A = 175.00;
        private final double TAX_B = 300.00;
        private final double TAX_C = 500.00;
        private final double TAX_D = 670.00;
        private final double TAX_E = 850.00;


        // Constructor 1
        public Vehicle(){
                // Set number to nextUniqueNumber, then increment it for next Ve
hicle
                this.number=nextUniqueNumber++;
                this.make=this.model=new String();
                this.year=0;
                this.value=0.0;
        }

        // Constructor 2
        public Vehicle(String make, String model, int year, double value){
                // Set number to nextUniqueNumber, then increment it for next Ve
hicle
                this.number=nextUniqueNumber++;
                this.make=make;
                this.model=model;
                this.year=year;
                this.value=value;
        }
```



```
        public Vehicle(String make, String model, int year, double value, int pw
d) {
                this.number=nextUniqueNumber++;
                this.make=make;
                this.model=model;
                this.year=year;
                this.value=value;
                this.pwdHash=pwd;
        }

        // getMake()
        public String getMake(){
                return make;
        }

        // getModel()
        public String getModel(){
                return model;
        }

        // getYear()
        public int getYear(){
                return year;
        }

        // getValue()
        public double getValue(){
                return value;
        }

        // toString()
        public String toString(){
                return("VEHICLE "+number+"==>"+make+", "+model+", "+year+", M−^@
"+value);
        }

        // equals() method
        // #boolean
        public boolean equals(Vehicle vehicleIn){
                //              if(this.number == vehicleIn.number)
                //                      return true;
                //              else
                //                      return false;
                return this.number == vehicleIn.number;
        }

        // decreaseValue() method
        public String decreaseValue(double newValue){
                if(newValue < value){
                        value = newValue;
                        // return message to indicate value changed
                        return ("Value changed to " + value + ".");
                }
                else
                        // return message to indicate newValue is not less than
value
                        return (newValue + " is not less than current value of "
 + value + ".");
        }

        // calculateTaxPayable() method
        public double calculateTaxPayable(double engineSize){
                if(engineSize > 2000)
                        return TAX_E;
                else if(engineSize > 1800)
                        return TAX_D;
                else if(engineSize > 1500)
                        return TAX_C;
                else if(engineSize > 1000)
```

```
                        return TAX_B;
                else
                        return TAX_A;
        }

        // calculateAge() - calculates the age
        public int calculateAge(){
                // in this case, the use of 'this' is a good way of disambiguati
on
                return Year.now().getValue() - this.year;
        }

        // qualifyForScrappage() method
        // AGAIN ?!?!?!?!?!?
        public boolean qualifyForScrappage(){
                //              if(calculateAge() >= 10)
                //                      return true;
                //              else
                //                      return false;
                return calculateAge() >= 10;
        }

        public int getPwd() {
                return this.pwdHash;
        }
}
```

```
package fr.fboschet.voiture.builder;

import org.json.JSONObject;

import fr.fboschet.voiture.Vehicle;

/**
 *
 * @author Francois Boschet
 *
 */
public interface VehicleBuilder {

        /**
         * Build a car given a JSONObject
         * @param car
         * @return
         */
        public Vehicle build(JSONObject car);

        /**
         *
         * @return
         */
        public Vehicle getBlankVehicule();
}
```

```
package fr.fboschet.voiture.builder;

public class CardinalFactory {

	// My factory is a lazy singleton.
	// It means there will only be one instance of it
	// @runtime, and it will be instantiate only when it
	// will be needed
	private static CardinalFactory INSTANCE;

	private CardinalFactory() {}

	public static CardinalFactory getInstance() {
		if (INSTANCE == null) {
			INSTANCE = new CardinalFactory();
		}
		return INSTANCE;
	}

	// Method that return the cardinal of a given (positive obviously)ordina
l
	public String getCardinal(int n) {
		String res = ""+n;
		if (n > 0) {
			int lastDigit = n%10;
			int last2Digits = n%100;
			if (lastDigit == 1 && last2Digits != 11)
				res += "st";
			else if (lastDigit == 2 && last2Digits != 12)
				res += "nd";
			else if (lastDigit == 3 && last2Digits != 13)
				res += "rd";
			else
				res +="th";
		}
		return res;
	}
}
```

```
package fr.fboschet.voiture.builder;

import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Component;
import java.awt.FlowLayout;
import java.awt.Font;
import java.awt.GridLayout;
import java.awt.KeyEventDispatcher;
import java.awt.KeyboardFocusManager;
import java.awt.event.KeyEvent;
import java.awt.event.WindowEvent;
import java.util.function.Consumer;

import javax.swing.BorderFactory;
import javax.swing.BoxLayout;
import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JFormattedTextField;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JSeparator;
import javax.swing.JSpinner;
import javax.swing.JTextField;
import javax.swing.SpinnerNumberModel;

public class ComponentFactory {

        // My factory is a lazy singleton.
        // It means there will only be one instance of it
        // @runtime, and it will be instantiate only when it
        // will be needed
        private static ComponentFactory INSTANCE;

        private ComponentFactory() {}

        public static ComponentFactory getInstance() {
                if (INSTANCE == null) {
                        INSTANCE = new ComponentFactory();
                }
                return INSTANCE;
        }

        // consumer that add a 1px border to a JPanel
        // I replaced bordes by JSeparator
        public Consumer<JPanel> addBorders = (JPanel it) -> it.setBorder(BorderF
actory.createLineBorder(Color.BLACK));

        // We use the Arial font by default. Can't be override
        // We still can change style (default is PLAIN) and size
        public JLabel getJlabel(String s, int size) {
                JLabel j = new JLabel(s);
                j.setFont(new Font("Arial", Font.PLAIN, size));
                return j;
        }

        public JLabel getJLabel(String s, int size, int style) {
                JLabel j = new JLabel(s);
                j.setFont(new Font("Arial", style, size));
                j.setAlignmentY(Component.BOTTOM_ALIGNMENT);
                return j;
        }

        // standard button
        public JButton getJButton(String string) {
                return new JButton(string);
        }
```



```
        // standard textField (has been replaced by spinner)
        @Deprecated
        public JTextField getJtextField(String text) {
                JTextField jtf = new JTextField(text);
                jtf.setFont(new Font("Arial", Font.BOLD, 100));
                return jtf;
        }

        // Spinner that allow only Double. Used for decrease the value
        public JSpinner getDoubleJSpiner(double init) {
                // we need to attach a model to our spinner
                SpinnerNumberModel sModel = new SpinnerNumberModel(0.0, 0.0, ini
t, 1.0);
                JSpinner spinner = new JSpinner(sModel);
                spinner.setValue(init);
                spinner.setFont(new Font("Arial", Font.BOLD, 100));
                // get the inner editor inside the JSpinner
                // then set the alignment.
                // We need this since JSpinner is only a container
                ((JSpinner.DefaultEditor)spinner.getEditor()).getTextField().set
HorizontalAlignment(JFormattedTextField.LEFT);
                // otherwise its ugly
                ((JSpinner.DefaultEditor)spinner.getEditor()).getTextField().set
Columns(3);
                return spinner;
        }

        // Spinner of Integer . Used for setting the engine size
        public JSpinner getIntJSpinner() {
                SpinnerNumberModel sModel = new SpinnerNumberModel(0, 0, Integer
.MAX_VALUE, 1);
                JSpinner spinner = new JSpinner(sModel);
                spinner.setValue(0);
                return spinner;
        }

        public JSeparator getJSeparator() {
                return new JSeparator();
        }

        // standard panel
        public JPanel getPanel() {
                return new JPanel();
        }

        // panel with a X layout (X ::= getXPanel)

        // ...
        public JPanel getFlowPanel() {
                return new JPanel(new FlowLayout());
        }

        // ...
        public JPanel getVerticalPanel() {
                JPanel j = getPanel();
                j.setLayout( new BoxLayout(j, BoxLayout.Y_AXIS));
                return j;
        }

        // ...
        public JPanel getHorizontalPanel(){
                JPanel j = getPanel();
                j.setLayout( new BoxLayout(j, BoxLayout.X_AXIS));
                return j;
        }

        // ...
        public JPanel getBorderPanel() {
                return new JPanel(new BorderLayout());
```

```
        }

        // ...
        public JPanel getGridPanel(int i, int j) {
                return new JPanel(new GridLayout(i, j));
        }

        // Custom JFrame
        public JFrame getJFrame(String title) {
                JFrame frame = new JFrame(title);
                frame.setLocationRelativeTo(null);
                frame.setResizable(false);
                frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                // default icon image
                frame.setIconImage((new ImageIcon("res/images/icon.png")).getIma
ge());
                // we bind the ESCAPE key to the WINDOW_CLOSING event
                KeyboardFocusManager fm = KeyboardFocusManager.getCurrentKeyboar
dFocusManager();
                fm.addKeyEventDispatcher(new KeyEventDispatcher() {
                        @Override
                        public boolean dispatchKeyEvent(KeyEvent arg0) {
                                if (arg0.getKeyCode() == KeyEvent.VK_ESCAPE)
                                        frame.dispatchEvent(new WindowEvent(fram
e, WindowEvent.WINDOW_CLOSING));;
                                        return arg0.getKeyCode() == KeyEvent.VK_
ESCAPE;
                        }
                });
                return frame;
        }
}
```

```
package fr.fboschet.voiture.ui;

import java.util.List;

import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JPasswordField;

import fr.fboschet.voiture.SecondHandVehicle;
import fr.fboschet.voiture.Vehicle;

public class DialogFactory {

        private static DialogFactory INSTANCE;

        private DialogFactory() {
        }

        public static DialogFactory getInstance() {
                if (INSTANCE == null) {
                        INSTANCE = new DialogFactory();
                }
                return INSTANCE;
        }

        public int passwordDialog() {
                JPanel panel = new JPanel();
                JLabel label = new JLabel("Enter a password:");
                JPasswordField pass = new JPasswordField(10);
                panel.add(label);
                panel.add(pass);
                String[] options = new String[]{"OK", "Cancel"};
                int cancel = JOptionPane.showOptionDialog(null, panel, "The titl
e",
                                                JOptionPane.NO_OPTION, JOptionPane.PLAI
N_MESSAGE,
                                                null, options, options[1]);
                if (cancel == 1) {
                        return -1;
                }
                return new String(pass.getPassword()).hashCode();
        }

        public SecondHandVehicle choiceDialog(List<Vehicle> vehicles) {
                return (SecondHandVehicle)JOptionPane.showInputDialog(null, "Cho
ose a SecondHandVehicle",
                                "VEHICLES AVAILABLE", JOptionPane.INFORMATION_ME
SSAGE, null, vehicles.toArray(), vehicles.get(0));
        }

}
```

```
package fr.fboschet.voiture.loader;

import java.util.List;

import fr.fboschet.voiture.Vehicle;

public interface JSONParser {

        public void populate(List<Vehicle> lv);
}
```

```
package fr.fboschet.voiture;

//Class to model SecondHandVehicle objects
//SecondHandVehicle IS-A Vehicle ==> Inherit from Vehicle
public class SecondHandVehicle extends Vehicle{
        // Instance Variables
        private int numberOfOwners;

        // Constructor 1
        public SecondHandVehicle(){
                this.numberOfOwners=0;
        }

        // Constructor 2
        public SecondHandVehicle(String make, String model, int year, double val
ue, int numberOfOwners){
                super(make,model,year,value);
                this.numberOfOwners = numberOfOwners;
        }

        // Constructor 3
                public SecondHandVehicle(String make, String model, int year, do
uble value, int numberOfOwners, int pwd){
                        super(make,model,year,value, pwd);
                        this.numberOfOwners = numberOfOwners;
                }

        // toString() - called when a SecondHandVehicle object is displayed
        public String toString(){
                return super.toString() + ", " + numberOfOwners + " owner(s).";
        }

        public int getNumberOfOwners() {
                return this.numberOfOwners;
        }

        // equals() method
        // AND AGAIN, AND AGAIN, ...
        public boolean equals(SecondHandVehicle secondHandVehicleIn){
                //         if(super.equals(secondHandVehicleIn))
                //                         return true;
                //              else
                //                 return false;
                return super.equals(secondHandVehicleIn);
        }
}
```

```
package fr.fboschet.voiture.builder;

import org.json.JSONException;
import org.json.JSONObject;

import fr.fboschet.voiture.SecondHandVehicle;
import fr.fboschet.voiture.Vehicle;

public class SecondHandVehicleBuilder implements VehicleBuilder {

        // This Builder is a lazy singleton.
        // It means there will only be one instance of it
        // @runtime, and it will be instantiate only when it
        // will be needed
        private static SecondHandVehicleBuilder INSTANCE;

        private SecondHandVehicleBuilder() {}

        public static SecondHandVehicleBuilder getInstance(){
                if (INSTANCE == null) {
                        INSTANCE = new SecondHandVehicleBuilder();
                }
                return INSTANCE;
        }

        // TODO: finish a beautiful builder with custom classloaders
        //       and extra infos to put in the json and display
        @Override
        public Vehicle build(JSONObject car) {
                try {
                        String make = car.getString("make");
                        String model = car.getString("model");
                        int year = car.getInt("year");
                        double value = car.getDouble("value");
                        int noo = car.getInt("noo");
                        int hash = car.getInt("pwd");
                        return new SecondHandVehicle(make, model, year, value, n
oo, hash);
                }catch(JSONException e){
                        e.printStackTrace();
                }
                        return new SecondHandVehicle("error", "error", -1, -1, -
1);
        }

        @Override
        public Vehicle getBlankVehicule() {
                return new SecondHandVehicle();
        }

}
```