University of Glasgow | School of Computing Science

Assessed Coursework

Course Name	Object Oriented Software Engineering			
Coursework Number	1			
Deadline	Time:	4:30pm	Date:	12th February 2020
% Contribution to final course mark	10%			
Solo or Group ✓	Solo	solo	Group	
Anticipated Hours				
Submission Instructions	Submit via Moodle			
Please Note: This Coursework cannot be Re-Assessed				

You must complete an "Own Work" form via https://studentltc.dcs.gla.ac.uk/ for all coursework, UNLESS submitted via Moodle.

Object Oriented Software Engineering
Assessed Exercise 1
UML Class Diagrams and Software Metrics

Objectives

- To interpret, understand and generte UML Class Diagrams.
- To generate control flow graphs and understand how they are used to determine the complexity of program code.
- Understand how software metrics can be used to measure and improve the quality of software.

You will achieve these objectives by completing a set of tasks.

Setup 1: Identify, download and setup an UML editor of your choice to use for the exercise. Some options include:

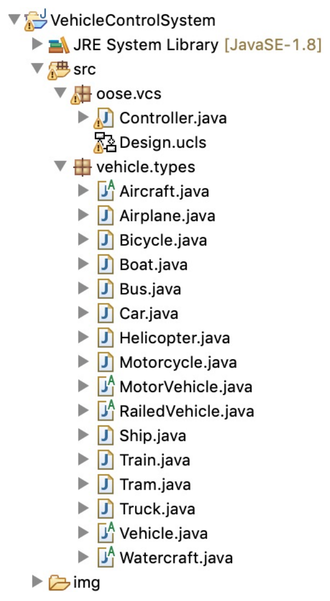
- Violet (free) http://horstmann.com/violet/
- ObjectAid UML Explorer. Works as an eclipse plugin http://www.objectaid.com/class-diagram

Setup 2: Create VehicleControlSystem Eclipse project.

1. Download vcs.zip from OOSE Moodle page and unzip it to a folder of your choice.
2. Create a Java eclipse project and name it `VehicleControlSystem`.
3. Copy the `img` folder from the unzipped content into the **project** directory of your Eclipse project.
4. Copy the `oose.vcs` and `vehicle.types` folders from the unzipped content into the `src` directory of your Eclipse project.

Your eclipse directory structure should be similar to Figure 1. A class diagram of the system design is as shown in Figure 2 (You can also access this diagram by clicking `Design.ucls` in the `oose.vcs` package if you have installed ObjectAid).

To run the project, right click `Controller.java` and run as Java application.



```
VehicleControlSystem
  JRE System Library [JavaSE-1.8]
  src
    oose.vcs
      Controller.java
      Design.ucls
    vehicle.types
      Aircraft.java
      Airplane.java
      Bicycle.java
      Boat.java
      Bus.java
      Car.java
      Helicopter.java
      Motorcycle.java
      MotorVehicle.java
      RailedVehicle.java
      Ship.java
      Train.java
      Tram.java
      Truck.java
      Vehicle.java
      Watercraft.java
  img
```

Figure 1: Project directory structure for Vehicle Control System on Eclipse

Setup 3: Running CKMetrics metrics suit.

- Download `CKMetrics.jar` from OOSE Moodle page to a folder of your choice.

`CKMetrics` is a tool for determining Chidamber and Kemerer (CK) metrics suite for java class files and Jar APIs. `CKMetrics` operates by using Apache Byte Code Engineering Library (BCEL) to analyse class files that are contained in a jar file. Assume you have `A.jar` located in your local drive ../`JarLocation`. To determine the quality of `A` using `CKMetrics`:

1. Select `Load File` button from the toolbar. You can either browse and select the folder containing `A` or select `A`. For the former, `CKMetrics` will load all the jar files contained in the folder.

2. Then select `Compute` button from the toolbar. `CKMetrics` will compute the mean CK metrics value for each selected jar file.

3. Optionally, you can include jdk class files in the analysis, or only analyse publicly accessible classes, fields and methods by checking `includeJdk` and `onlyPublic` checkboxes respectively.

Figure 3 is a screenshot showing `CKMetrics`'s output. You can also select a .class file following above steps to analyse java classes.

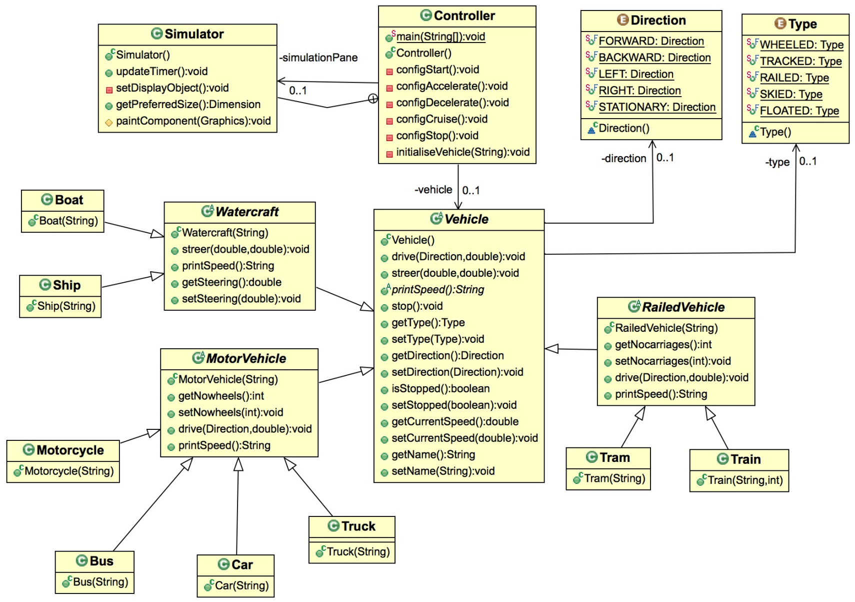


Figure 2: Design Diagram for a Vehicle Control System

Setup 4: Create VehicleControlSystem runnable jar file.

Finally, to build the jar file required to check the quality of `Vehicle Control System` based on above steps:

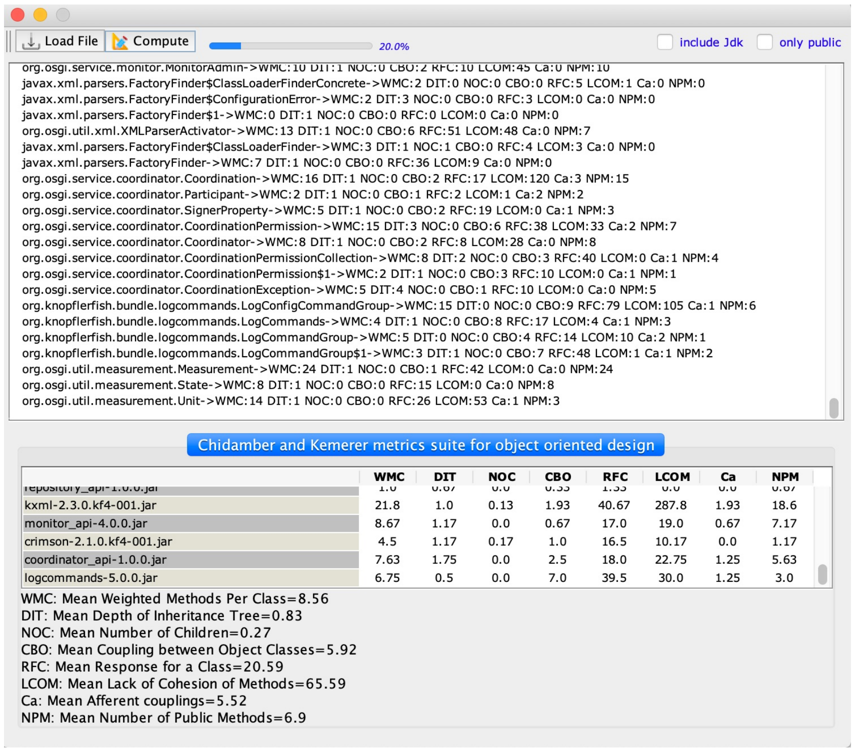
1. Right click on the project in Eclipse.
2. Select export.
3. Export as runnable jar file.

Tasks

1. Draw the control flow graph for the method `initialiseVehicle` in `Controller.java`. [**7 Marks**]

2. Assume the objective is to increase encapsulation, modularity and reduce the complexity of `VehicleControlSystem`. List 3 metrics that can be used to measure these quality factors in `VehicleControlSystem`. For each metric, provide a couple of sentences to justify your choice. [**3 Marks**]

3. Identify the most important class in `VehicleControlSystem` that should be refactored to enhance the quality factors in Task 2. Justify your choice in a maximum


Load File
Compute
20.0%
include Jdk
only public
Chidamber and Kemerer metrics suite for object oriented design

```
org.osgi.service.monitor.MonitorAdmin->WMC:10 DIT:1 NOC:0 CBO:2 RFC:10 LCOM:45 Ca:0 NPM:10
javax.xml.parsers.FactoryFinder$ClassLoaderFinderConcrete->WMC:2 DIT:0 NOC:0 CBO:0 RFC:5 LCOM:1 Ca:0 NPM:0
javax.xml.parsers.FactoryFinder$ConfigurationError->WMC:2 DIT:3 NOC:0 CBO:0 RFC:3 LCOM:0 Ca:0 NPM:0
javax.xml.parsers.FactoryFinder$1->WMC:0 DIT:1 NOC:0 CBO:0 RFC:0 LCOM:0 Ca:0 NPM:0
org.osgi.util.xml.XMLParserActivator->WMC:13 DIT:1 NOC:0 CBO:6 RFC:51 LCOM:48 Ca:0 NPM:7
javax.xml.parsers.FactoryFinder$ClassLoaderFinder->WMC:3 DIT:1 NOC:1 CBO:0 RFC:4 LCOM:3 Ca:0 NPM:0
javax.xml.parsers.FactoryFinder->WMC:7 DIT:1 NOC:0 CBO:0 RFC:36 LCOM:9 Ca:0 NPM:0
org.osgi.service.coordinator.Coordination->WMC:16 DIT:1 NOC:0 CBO:2 RFC:17 LCOM:120 Ca:3 NPM:15
org.osgi.service.coordinator.Participant->WMC:2 DIT:1 NOC:0 CBO:1 RFC:2 LCOM:1 Ca:2 NPM:2
org.osgi.service.coordinator.SignerProperty->WMC:5 DIT:1 NOC:0 CBO:2 RFC:19 LCOM:0 Ca:1 NPM:3
org.osgi.service.coordinator.CoordinationPermission->WMC:15 DIT:3 NOC:0 CBO:6 RFC:38 LCOM:33 Ca:2 NPM:7
org.osgi.service.coordinator.Coordinator->WMC:8 DIT:1 NOC:0 CBO:2 RFC:8 LCOM:28 Ca:0 NPM:8
org.osgi.service.coordinator.CoordinationPermissionCollection->WMC:8 DIT:2 NOC:0 CBO:3 RFC:40 LCOM:0 Ca:1 NPM:4
org.osgi.service.coordinator.CoordinationPermission$1->WMC:2 DIT:1 NOC:0 CBO:3 RFC:10 LCOM:0 Ca:1 NPM:1
org.osgi.service.coordinator.CoordinationException->WMC:5 DIT:4 NOC:0 CBO:1 RFC:10 LCOM:0 Ca:0 NPM:5
org.knopflerfish.bundle.logcommands.LogConfigCommandGroup->WMC:15 DIT:0 NOC:0 CBO:9 RFC:79 LCOM:105 Ca:1 NPM:6
org.knopflerfish.bundle.logcommands.LogCommands->WMC:4 DIT:1 NOC:0 CBO:8 RFC:17 LCOM:4 Ca:1 NPM:3
org.knopflerfish.bundle.logcommands.LogCommandGroup->WMC:5 DIT:0 NOC:0 CBO:4 RFC:14 LCOM:10 Ca:2 NPM:1
org.knopflerfish.bundle.logcommands.LogCommandGroup$1->WMC:3 DIT:1 NOC:0 CBO:7 RFC:48 LCOM:1 Ca:1 NPM:2
org.osgi.util.measurement.Measurement->WMC:24 DIT:1 NOC:0 CBO:1 RFC:42 LCOM:0 Ca:0 NPM:24
org.osgi.util.measurement.State->WMC:8 DIT:1 NOC:0 CBO:0 RFC:15 LCOM:0 Ca:0 NPM:8
org.osgi.util.measurement.Unit->WMC:14 DIT:1 NOC:0 CBO:0 RFC:26 LCOM:53 Ca:1 NPM:3
```

	WMC	DIT	NOC	CBO	RFC	LCOM	Ca	NPM
repository_api-1.0.0.jar	1.0	0.67	0.0	0.33	1.33	0.0	0.0	0.67
kxml-2.3.0.kf4-001.jar	21.8	1.0	0.13	1.93	40.67	287.8	1.93	18.6
monitor_api-4.0.0.jar	8.67	1.17	0.0	0.67	17.0	19.0	0.67	7.17
crimson-2.1.0.kf4-001.jar	4.5	1.17	0.17	1.0	16.5	10.17	0.0	1.17
coordinator_api-1.0.0.jar	7.63	1.75	0.0	2.5	18.0	22.75	1.25	5.63
logcommands-5.0.0.jar	6.75	0.5	0.0	7.0	39.5	30.0	1.25	3.0

```
WMC: Mean Weighted Methods Per Class=8.56
DIT: Mean Depth of Inheritance Tree=0.83
NOC: Mean Number of Children=0.27
CBO: Mean Coupling between Object Classes=5.92
RFC: Mean Response for a Class=20.59
LCOM: Mean Lack of Cohesion of Methods=65.59
Ca: Mean Afferent couplings=5.52
NPM: Mean Number of Public Methods=6.9
```

Figure 3: Analysis output from `CKMetrics`

of 150 words. [**5 Marks**]

4. Propose and implement four refactorings for the class identified in Task 3 to enhance quality factors mentioned in Task 2. In maximum of 200 words, discuss the extent suggested refactorings enhance encapsulation and modularity, reduce complexity, and enhance the quality factors mentioned in Task 2. Justify your choices by comparing the results of applying CKMetrics to `VehicleControlSystem` before and after the refactorings have been implemented. [**10 Marks**]

5. Draw a new class diagram for `VehicleControlSystem` after implementing your refactorings in Task 4. In maximum of 150 words, discuss any structural differences in the design of `VehicleControlSystem` in your new implementation. [**5 Marks**]

Deliverables

1. Submit a single pdf document electronically via Moodle for Assessed Exercise 1. You must clearly state your name and registration number at the top of the document. The document should also contain your **solution to tasks 1 - 5** with relevant **screenshots of CKMetrics** and **class diagrams**.
2. A source folder for `Vehicle Control System` containing the implementation of your refactoring tasks.
3. A runnable jar file of the `Vehicle Control System` after implementing your proposed refactoring tasks.

Assessment

Submissions is due by **16.30** on Friday **12th February 2020**. You should submit your solution through the class Moodle page.

Tutors and demonstrators will be in your allocated Lab to offer assistance. **Endeavour to attend your labs to discuss issues rather than sending emails**.

As per the Code of Assessment policy regarding late submissions, submissions will be accepted for up to 5 working days beyond due date. Any late submissions will be marked as if submitted on time, but reduced by 1 mark for each additional day. Submissions received more than 5 working days after the due date will receive an H (band value of 0).

Dr Graham McDonald, 406 School of Computing Science, University of Glasgow