ICA SPECIFICATION

This form should be attached to the front of the ICA specification and include all details.

Module Title: Java Programming 2	**Module Leader:** J Barker/E Diemoz
	Module Code: COM1054
Assignment Title: Car Parking System	**Deadline Date:** 16th May 2017
	Deadline Time: 23:59
	Submission Method: Online ■

Central Assignments Office (Middlesbrough Tower M2.08) Notes:

- All work (including CDs etc) needs to be secured in a plastic envelope or a folder and clearly marked with the student name, number and module title.
- An Assignment Front Sheet should be fully completed before the work is submitted.
- When an extension has been granted, a fully completed and signed Extension form must be submitted to the SCM Reception.

Online Submission Notes:

- Please follow carefully the instructions given on the Assignment Specification
- When an extension has been granted, a fully completed and signed Extension form must be submitted to the SCM Reception.

Library Support for Academic Skills

Did you know you can book an individual 30 minute tutorial in the Learning Hub with an adviser to help you with your academic skills, writing or numeracy? Or that there are loads of really useful workshops available to help you with your studies and assessments? Have a look at the Succeed @ Tees workshops for more details.

FULL DETAILS OF THE ASSIGNMENT ARE ATTACHED INCLUDING MARKING & GRADING CRITERIA

Overview

As part of a town planning scheme, a traffic department is modelling the use of a car park and the revenue that can be raised from car park charges. To assist with this, the planning department requires a prototype GUI application to record details of vehicles using the car park.

The prototype needs to be able to manage the following vehicles:

- Cars:

  Cars with length of 6 metres or less will be charged £1.00 per hour.
  Cars longer than 6 metres (camper-vans etc.) will be charged £1.50 per hour.
  Disabled badge holders are exempt from parking fees.

- Lorries:

  Lorries less than 20 tonnes are charged at £5.00 per day, those of 20 tonnes and over are charged at £8.00 per day. Lorries with a weight over 35 tonnes are not permitted to use the car park.

- Coaches:

  Coaches with 20 passengers or less are charged £4.50; those with more than 20 passengers are charged £6.00. The charge allows the coach to park for up to a week – for this application, assume coaches do not stay longer than a week. Tourist operators receive a 10% discount on parking charges.

The traffic department has provided the following GUI mock-up:

(Images from http://www.pureclipart.com/index.php)

The car park layout should show two separate regions: one for ‘cars’ and another region for ‘lorries and coaches’.

The region for parking cars should have at least 3 rows and 4 columns of ‘spaces’ for parking cars, and the lorry-coach region should have at least 4 places for parking lorries or coaches. These should appear as a larger space than the spaces for parking a car.

The actual number of spaces for cars and lorries-coaches is up to you, but the GUI should show at least 3 rows and 4 columns for the cars, and at least 4 lorry-coach spaces.

The following images are provided (on Blackboard):

car.png *campervan.png* *lorry.png* *coach.png*

Button Operation Summary

1. **Add Car**

   The user is prompted to input the registration number of a car and its length.

   If there is an appropriate empty space on the GUI, a Car is added to the vehicle list, totals updated and an appropriate image is displayed in the empty space.

   If there are no empty spaces in the car park for cars, a suitable message is displayed to the user and no further action is taken.

2. **Add Lorry**

   The user is prompted to input the registration number of a lorry and its weight.

   The Lorry is processed using a similar method to the one described for the car above.

3. **Add Coach**

   The user is prompted to input the registration number of a coach and the number of passengers.

   The Coach is processed using a similar method to the one described for the car above.

4. **Clear All**

   Clears all items from the application and resets the totals to zero.

5. **Save**

   The user is prompted to enter a file name (ending with the extension .dat) and all vehicles currently in the car park should be written to this file.

6. **Load**

   The user is prompted to input the name of an existing data file. The vehicle details are read in from the file and the display updated.

   Before reading the data file, any existing vehicle details should be cleared.

7. Current Total

The income expected from all the vehicles currently in the car park is calculated and displayed to the user.

8. Overall Total (inc. Removed Vehicles)

The total income from all vehicles that have used the car park since the application started, or the last 'Clear All' button event or the last 'Load' button event is displayed to the user.

Mouse Operation Summary

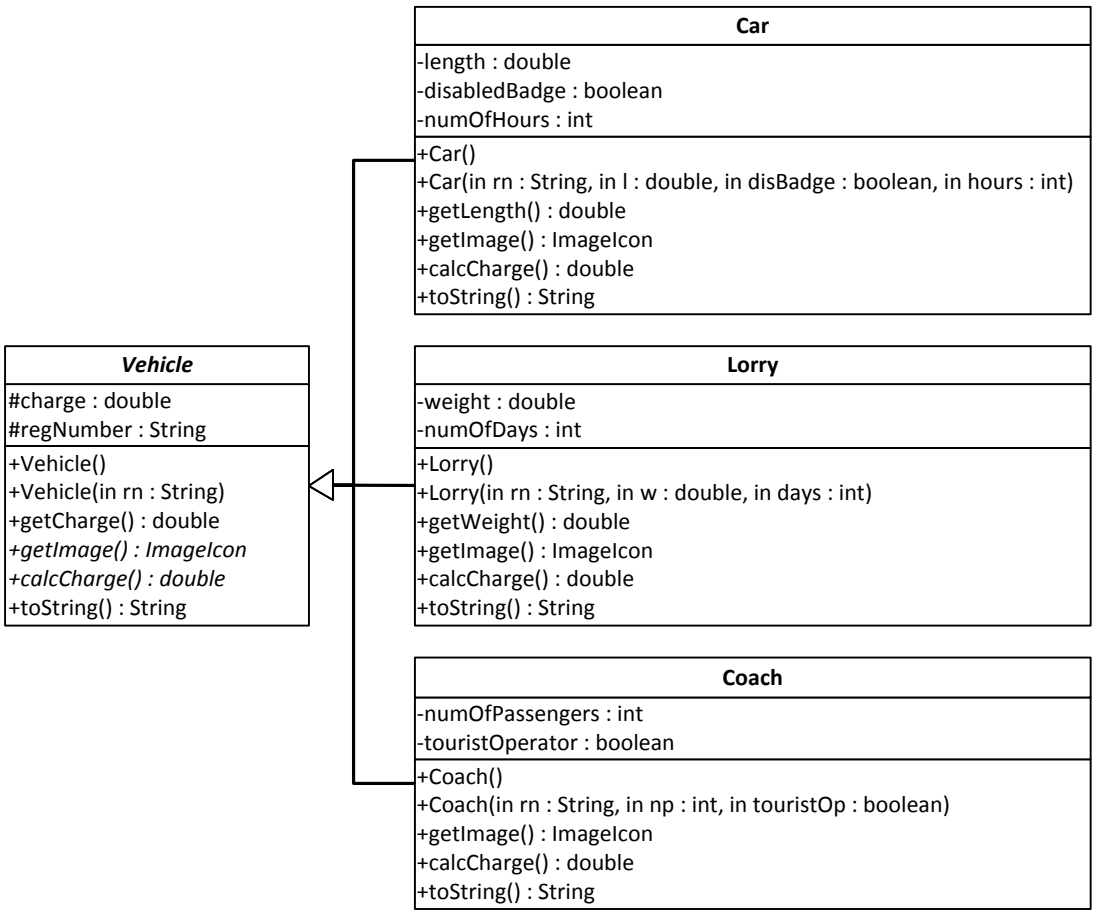
When the user clicks on an item the following operations should be performed:

- Left Click: Display details of the parked vehicle.
- Right Click: Remove the vehicle from the car park (after confirmation from the user).
- Middle Click: Change/update vehicle attributes (length, weight, etc) and update charge.

If an empty cell (space) is clicked a message should be displayed to state that the space is empty.

Technical Information

After analysing the requirements, the following class and subclass UML diagram has been prepared:



Notes:

1. All default (no argument) constructors should set the instance variable to suitable values: zero for numeric, null for objects.

2. The **Vehicle** class:

   a) Is to be abstract, with an abstract methods getImage() and calcCharge.

   b) The Vehicle constructor sets the registration number to the value given by the parameter and sets charge to zero and image to null.

   c) The Vehicle toString() method will return a string with the vehicle's registration number and the charge for parking the vehicle.

   d) Implements the serializable interface.

   e) getCharge just returns the charge. calcCharge calculates, stores and returns the charge. calcCharge is used to update the charge if the data is changed with the edit operation.

3. The **Car** Class

   a) The constructor sets the class variables and calculates the charge.

   b) The toString() method overrides Vehicle's toString(): the string it returns has the car's registration number, charge and car's length.

4. The **Lorry** class

   a) The constructor sets the class variables and calculates the charge.

   b) The toString() method overrides Vehicle's toString(): the string it returns has the lorry's registration number, charge and lorry's weight.

5. The **Coach** class:

   a) The constructor sets the class variables and calculates the charge.

   b) The Coach's toString() method overrides Vehicle's toString(): the string it returns has the coach's registration number, charge and number of passengers.

Tasks

1. Class and Subclasses [10%]

The classes that need to be used in the application are given in the UML class diagram above.

Create these classes in **Java** and write a **separate class** with a main method that will test the subclasses. In the test class you will create objects of each subclass, using the classes' constructors, and output the result of calling the objects' getCharge() methods and toString() methods. A command line output will be sufficient for this.

Marks will be awarded for appropriate use of access modifiers, abstraction, serialization, use of super method to link subclass methods to the parent class methods.

2. Build GUI and Initialise Variables/Objects [15%]

Design and build the Graphical User Interface (GUI) with appropriate panels and components. An average solution will just replicate this layout. A good solution will improve on it. To achieve this layout you can use any layout manager you think appropriate.

You will also need to decide how to represent the information required for this application and create/initialise appropriate variables and objects. This will include:

- Counters and totals.
- Arrays or collections to represent lists of data. You will also need to consider how this data links to the GUI.

3. Event Handling for Buttons [20%]

Implement actions for the buttons (or alternative event source) described in the overview. Marks will be awarded for:

- Creating appropriate handlers to meet the requirements.
- Including appropriate logic to update the data as operations are performed by the user.
- Suitable exception handlers for the file read and write operations.

4. Mouse Events [25%]

Write the event handling for the mouse clicks as described in the requirements. Where summary text is required use the appropriate subclass toString() method. Marks will be awarded for:

- Creating appropriate handlers to meet the requirements.
- Including appropriate logic to update the data as operations are performed by the user.

An average solution would use a dialog to describe the state of the object. A good solution would add another panel to the GUI which displays the properties of the object. The choice is up to you, but be aware that the more advanced solutions attract better marks.

5. Code Efficiency and Documentation [20%]

The code in your application should:

- Be well commented **using JavaDoc style** comments (no superfluous comments).
- Be consistently indented, making good use of blank lines.
- Use suitable identifiers, adopting the house style conventions.
- Be as efficient as possible, well designed, using internal methods where appropriate.
- Demonstrate use of a range of GUI components, and other classes from the Java API.

6. The Report [10%]

You must also submit a written report (approximately 500 words) which is to contain:

- A description of how feedback from the first assessment has been used to improve this assessment.
- An honest evaluation of your final application. An account of which parts of the assessment have been successfully implemented and those parts which have not been fully successful. You are NOT required to provide a Test Plan, but you are expected to have tested your code thoroughly.
- An evaluation of your program development, i.e. how you approached the problem.

Java Programming 2 ICA 2016/17 – where the marks are

General: (25)

- Your application must have at least one abstract, serializable Class that has at least 3 subclasses (1)
- Each subclass must use the correct constructor call using the correct logic (3)
- Each subclass must have the appropriate modifiers (private, public or protected) set and must provide the correct overrides (3)
- You should provide at least two test classes for each subclass (3)
- You will need to use appropriate Layout Managers (10)
- You must include a range of appropriate collections and instance variables (5)

Button Events (20)

Your application must include some button events to demonstrate your knowledge, this must include

- The ability to search for a space on a grid (3)
- Be able to select a particular item from a list, create the object (item), add it to a list and display an appropriate image for that item (4)
- You should also include a button to clear the grid (which should also empty the collection/list and update the GUI (3)
- Include a button to save/write the data to a file using object serialization (3)
- Include a button to load data back into your collection, and update the GUI (4)
- Handle exceptions as appropriate (3)

Mouse Events (25):

Your application should also include some mouse event handling to demonstrate your knowledge. The mouse events should include:

- The ability to click on an item and have the correct details displayed for that item (4)
- A clear distinction of the use for each mouse button (1)
- The ability to click on the middle button to remove an item and close gap (4)
- The ability to change the details for a particular item on the Grid (Edit on right click) - inc. GUI (12)
- The ability to display using Information (left click) - inc. GUI (4)

Additional Marks (20)

- Additional marks will be awarded for Code Efficiency and Documentation – including code layout, comments, code quality & efficiency (including the use of additional internal methods)
- Produce software documentation using the JavaDoc tool, and you must adopt a professional approach to software development with appropriate consideration of legal and professional issues.

Report (10)

A further 10 marks can be obtained by providing a report to reflect upon the effectiveness of your work, identify skills you need to develop to enhance your employability

General Notes/Advice

- DESIGN your solution using pseudo code or flow charts. Resist the urge to start hacking away immediately.
- You are NOT expected to validate user input, apart from normal exception handling when working with files. Assume that only valid data will be input by the user.
- Your application must demonstrate the use of inheritance for main class and subclasses, exception handling and event handling.
- Do not devote too much time to the GUI at the expense of the other sections.
- If in doubt keep the GUI simple. Don't be too ambitious at an early stage.
- Design your GUI using sketches on paper.
- You must use **NetBeans** to develop your code but do **not** use the **NetBeans GUI designer** to produce the GUI for this assignment. You **must** produce the GUI manually using the house style.
- **You are strongly advised to retain all versions of your code and your development notes, so that you can show the development of your work. This may be required as evidence in the event of any query about the authorship of your work.**

What to Submit

Your assignment work must be submitted electronically as a single zip file to the module support area on Blackboard. The zip file must contain the code and document files and be called K1234567-ICA2 (where K1234567 is your user id).

IF THE ZIP FILE IS UNREADABLE OR BLANK YOU WILL RECEIVE A ZERO MARK! Once you have completed the upload, download the file to check contents.