Department of Computer Science and Information Technology

La Trobe University

CSE1OOF/4OOF Semester 1, 2020
Assignment Part C

Due Date: Tuesday, 9th June 2020, at 10.00 a.m.

First and Final date for SUBMISSION Tuesday 9th June 2020 at 10.00 am

Delays caused by computer downtime cannot be accepted as a valid reason for a late submission. Students must plan their work to allow for both scheduled and unscheduled downtime. **There are no days late or extensions on this assignment as execution test marking will begin Wednesday, 10th June 2020**.

This is an individual assignment. You are not permitted to work as a group when writing this assignment.

Copying, Plagiarism: Plagiarism is the submission of somebody else's work in a manner that gives the impression that the work is your own. The Department of Computer Science and Information Technology treats academic misconduct seriously. When it is detected, penalties are strictly imposed. Refer to the unit guide for further information and strategies you can use to avoid a charge of academic misconduct. ***All submissions will be electronically checked for plagiarism.***

Assessment Objectives:

- ◊ to design programs that conform to given specifications
- ◊ to practise combining multiple classes and methods into a whole program
- ◊ to implement programs in Java.

Submission Details: Full instructions on how to submit electronic copies of your source code files from your latcs8 account are given at the end. *If you have not been able to complete a program that compiles and executes containing all functionality, then you should submit a program that compiles and executes with as much functionality as you have completed. (You may comment out code that does not compile.)*

Note you must submit electronic copies of your source code files using the **`submit`** command on latcs8. Ensure you submit all required files, one file at a time. **For example**, the file **`Shuttle.java`** would be submitted with the command:

```
> submit OOF Shuttle.java
```

Do NOT use the LMS to submit your files, use latcs8 only

PLEASE NOTE*: While you are free to develop the code for this progress check on any operating system, your solution must run on the latcs8 system.*

Marking Scheme: *This assignment is worth 15% of your final mark in this subject.*

Implementation (Execution of code) 90%, explanation of code 10%

You may be required to attend a viva voce (verbal) assessment (to be used as a weighting factor on the final mark).

Return of Mark sheets:
The face to face execution test marking in the lab (Week 13) constitutes a return of the assignment mark, subject to the conditions on pages 11 - 12.

Please note carefully:

The submit server will close at 10:00 am on June 9th

After the submit server has closed, NO assignments can be accepted.

Please make sure that you have submitted **all** your assignment files before the submit server closes. (see page 12)

There can be NO extensions or exceptions.

Marking scheme:

90% for the code executing correctly

10%, you will be asked to explain (and / or alter) parts of your code.

Using code not taught in OOF - READ THIS

Please also note carefully that whilst we encourage innovation and exploring java beyond what has been presented in the subject to date, **above all, we encourage understanding**.

The assignment that follows can be solved using techniques that have been presented in lectures, lecture / workshops and labs so far.

These are the techniques and knowledge that we will later be examining in the Real Time Test (30% of final marks) and the exam (40% of final marks).

Code and techniques that are outside the material presented will not be examined, of course.

You are free to implement the program below in any way, with one condition.

Any assignment that uses code that is outside what has been presented to this point **must be <u>fully</u> explained at the marking execution test**. Not being able to <u>fully</u> explain code outside what has been presented in the subject so far will **result in the assignment being awarded a mark of 0**, regardless of the correctness of the program.

Submitting an assignment with code outside what has been presented so far and not attending the marking execution test will result in an automatic mark of 0, regardless of the correctness of the submission.

This assignment is based on Lab 7, in particular Question 3, but including question 1

Access Modifiers in objects

Please note that all object attributes must have **`private`** as their access modifier. Any classes that have any non-private object attributes will result in the whole assignment mark being heavily reduced, up to and including, being awarded 0, regardless of the correctness of the program. **<u>This includes omitting access modifiers,</u>** which means that the object attributes have the java default access modifier **`package`**. This was discussed in Week 7 lectures.

Problem Background

Humankind is pushing out into space. New planets are being discovered and explored with a view to colonizing suitable planets.

The agency tasked with this somewhat momentous job is the **International Space Centre,** also referred to by its better-known acronym, **ISC.**

ISC is staffed by intrepid personnel known as Crew. Crew use vehicles known as Shuttles to travel from their Space Station, which is a spaceship, down to a planet and back up to the spaceship.

When a Crew member first joins **ISC**, they have not been part of any missions, their skill level is Novice. A Crew member is also referred as a Crew.

As a Crew member carries out more missions, their skill level increases.

If the Crew has been on between 0 and 6 missions (inclusive),
then their skill level is **`Novice`**
If the Crew has been on between 7 and 8 missions (inclusive),
then their skill level is **`Intermediate`**
If the Crew has been on between 9 and 14 missions (inclusive),
then their skill level is **`Advanced`**
If the Crew has been on 15, or more, missions,
then their skill level is **`Expert`**

To be assigned a mission, firstly, the Shuttle must have a Crew and not be on a mission.

Secondly, the skill level required for the mission must be within the skill level of the Crew. A Crew with a higher skill level can undertake a mission that requires a lesser skill level, but a Crew cannot undertake a mission that requires a higher skill level.

The Crew must have the required skill level **before** being assigned the mission.

Each time a Crew is assigned a mission, the number of missions they have been on is incremented by one. The program must then check if the Crew has accumulated enough missions to move to the next skill level.

To end a mission, the Shuttle must actually be on a mission.

Program Requirements

You have been asked to write an interactive program, in Java, to aid in monitoring and maintaining all aspects of Space Station operations for **ISC**.

This program is a prototype and there will be only 2 Shuttle objects in the program. If the prototype proves successful, this will be expanded in the next iteration (The Progress Check Test assignment)

Recall that a Shuttle object also has an associated Crew object (when the Crew object is assigned).

To aid in the rapid development of this program, 3 Java files and 3 sample input files are provided for you:

`Crew.java`, **`Shuttle.java`**, **`SpaceShip.java`** and sample input files **`a1.dat`**, **`b1.dat`** and **`c1.dat`**

These files are zipped and available with this assignment document for you to download and place it in your current directory.

In order to further speed up the development of this program, some of the files listed above have been partially implemented for you, read the comments in the files **and in this document**.

Crew.java

All Crew objects have the following object attributes:

- **`name`** This a String (text) and is the name of the Crew, may be more than one word
- **`Crew id`** This a String (text) and is the **unique** id of the Crew, may be more than one word
- **`missions`** This is an integer and is the number of missions that the Crew has participated in. (See the explanation below)
- **`skill level`** This is a String and is one of the four skill levels, as described above. The skill level is always based on the **number of missions that the Crew has participated in, and must be set by the program.**

  The user must NEVER be able to enter the skill level, nor is it ever stored in a file.

  The skill level is also known as the task level. Any references to task level in this document are the same as skill level.

The **`Crew`** class requires the following functionality:

A constructor that takes **`name`**, **`Crew id`**, and **`missions`** as parameters. This is the constructor that is called when a new **`Crew`** is added from the text file. From the file, all three of these parameters have values.

The format of the text file is shown on pages 8 - 9

Recall that it is the number of **`missions`** that determines the **`skill level`**, so there has to be a way for the constructor to set the **`skill level`**. If this constructor is also used for keyboard input, then the number of **`missions`** must be set to 0 as the **`Crew`** has just been created.

Alternatively, you could write an overloaded constructor, just for use with the keyboard, this just takes the first two values (**`name`** and **`Crew id`**) as parameters and assigns 0 to the number of **`missions`**.
Either way, the **`skill level`** must be set by the constructor, not the user.

This is where you write a private helper method in the **`Crew`** class, **<u>similar</u>** to the **`setCategory`** method in lab 7

The **`Crew`** class also requires accessor methods as you deem appropriate.

The **`Crew`** class also requires a method to increment the number of **`missions`**. Calling this method adds another mission to the total number of **`missions`**. This method should then check whether that moves the **`Crew`** to the next skill level.

Finally, the **`Crew`** class requires a **`toString`** method which returns a **`String`** with information about the **`Crew`** object, see pages 14 and 16 for the format of the **`String`** to be returned.

Please note that the output must follow the format exactly, this will be assigned marks in the execution test.

Shuttle.java

The Shuttle class has the following attributes:

- **`shuttle id`** This is a String (text) and may consist of more than one word
  The Shuttle id is the **<u>unique</u>** identifier for a **`Shuttle.`**
- **`on mission`** This is a **`boolean`** variable, the value **`false`** indicates that the **`Shuttle`** is NOT on a mission. The value **`true`** indicates that the **`Shuttle`** is on a mission.
- **`crew`** This is the **`Crew`** class object reference for the **`Crew`** object that is associated with this **`Shuttle`**
  (when the **`Crew`** object is added and instantiated)

The **`Shuttle`** class requires 2 overloaded constructors.

The first overloaded constructor takes just the **`Shuttle id`** for a **`Shuttle`** object. This constructor is called when adding a **`Shuttle`** object from the keyboard. The **`Shuttle`** cannot be on a mission as it has just been created and, for the same reason, it cannot have a **`Crew`** object associated with it.

The second overloaded constructor is for creating a **`Shuttle`** object from the input text file. In the text file there will be the **`Shuttle id`** and a **`boolean`** value to indicate whether or not the **`Shuttle`** is currently on a mission. There will also always be another boolean, directly after this **`on mission boolean`** value. If this has the value **`true`**, then it indicates there is information for the **`Crew`** object associated with this **`Shuttle`**. This will consist of the Crew's **`Crew id`**, **`name`** and the number of **`missions`** that they have worked. If the **`boolean`** is **`false`**, then there is no **`Crew`** associated with this **`Shuttle`**. See pages 8 and 9 for the file format.

The Shuttle class must never have a Crew object reference as a parameter in any of its methods, including the constructor(s). Even when reading from the file, if there is a **`Crew`** associated with the **`Shuttle`**, then the **`Crew's name`**, **`id`** and the number of **`missions`** they have worked will have been read out after the **`Shuttle`** object is created. Then that **`Shuttle's addCrew`** method will be called. The **`addCrew`** method will take all three **`Crew`** parameters.

The **`Shuttle`** class will require accessor methods as you deem appropriate.

The **`Shuttle`** class also requires a **`toString`** method that returns a **`String`** with information about the state of that **`Shuttle`** object. The format of the **`String`** is shown in the example output on pages 14 and 16. As with the **`Crew`** screen output, **you must follow the format shown exactly, marks will be assigned to following the format.**

The **`Shuttle`** class requires a method to add a **`Crew`** to the **`Shuttle`**. (This was discussed above) This method takes all of the relevant parameters for instantiating a **`Crew`** object **(but doesn't pass in a Crew object reference)**. The resultant object is stored in the **`Crew`** object reference. **Each Shuttle can have only one Crew. When adding from the keyboard, the program needs to check that the Shuttle object does not already have a Crew object.** This method is called when the user attempts to add a **`Crew`** from the keyboard or from the file.

The **`Shuttle`** also needs methods to start and end a mission. Recall that when all the conditions are met and a mission is assigned to a **`Shuttle`**, the number of **`missions`** for the associated **`Crew`** has to be incremented. Starting a mission sets the **`on mission`** attribute of the **`Shuttle`** to **`true`** and ending a mission sets this attribute to **`false`**.

The **`skill level`** attribute is **`private`** to the **`Crew`** class, so it cannot be called directly from the **`Shuttle`** class. All interactions (message passing) between the **`Shuttle`** and **`Crew`** classes must be through the **`public`** methods of the **`Crew`** class.

If the skill level attribute can be set directly from the Shuttle class, or any other attributes of the Crew class can be directly access from the Shuttle class, the whole assignment will be awarded 0, regardless of the functionality of the program.

The **`Shuttle`** class could also use a method that takes the required **`skill level`** of the mission as a parameter and returns **`true`** or **`false`** as to whether the associated **`Crew`** has the required **`skill level`**. Note that this is could also be done in the main driver class, **`SpaceShip`**, which is described below.

You might find it useful to write a method in the **`Shuttle`** class that returns a **`boolean`** to indicate if the **`Shuttle`** already has a **`Crew`** or not, and another one to indicate if the Shuttle is on a mission or not.

The `Crew` and `Shuttle` classes do NOT ask the user for any input, either keyboard or file. There must not be any input objects in these classes such as Scanner or BufferedReader but not limited to these 2. Another way of saying this is that these classes are not interactive.

The driver program, **`SpaceShip.java`** starts by presenting the user with a menu, as follows:

```
ISC Planetary Explorer Main menu
        1. Load from file
        2. Add Shuttle
        3. Start Mission
        4. End Mission
        5. Display
```

Note: SpaceShip class must NOT have Crew object attribute(s) or Crew object references. All interactions involving a Crew will be through the relevant Shuttle object.

```
        6. Add Crew
        7. Close the program
Enter choice >>
```

Implement the functionality of each menu choice as follows (assume user input is always an integer):

1. Load from file

First the program must check that there is at least one Shuttle object free. If both Shuttle object references already have objects attached to them, the program prints an appropriate message. **If this is the case, the user is NOT asked for an input file name, the program simply returns to the main menu.**

Assuming there is at one free Shuttle object reference, then the user is prompted for the name of an input text file. You may assume that this file always exists and is correctly formatted. The input file will hold at least one `Shuttle` record (A record is shown in the table below), your program may assume that the user never enters a file name that exists but is empty.

The program must work with any file name entered by the user (of the correct format), that is, the file name must not be hard coded.

Consider that you may open an input text file with 3 records in it and the user has already entered information from the keyboard for one `Shuttle` object. In that case the first record would be read from the input text file and assigned to the second `Shuttle` object reference. After this the file could be closed or you could keep reading through the text file, but any further records would be discarded, they would not overwrite the information for existing `Shuttle` objects.

The format of the file is:

`Shuttle Id`	- **`String`**
`onmission`	- **`boolean`**
`hasCrew`	- **`boolean`**, **`true`** means that there is a Crew, **`false`** means not
`Crew name - String`	Only if the **`boolean`** on the line above is **`true`**, otherwise it is either the end of file or the start of the next **`Shuttle`**.
`Crew id - String`	
`Crew missions - int`	

An example of the information, in the text file, on each `Shuttle` consists of 6 lines if there is a `Shuttle` and `Crew` as follows in the example below, **or just 3 lines if there is just a Shuttle,** either way, the complete 6 lines, or 3 lines, is known as a <u>**record**</u>:

`A 04 S23`	**unique** id of the Shuttle
`true`	indicates the Shuttle is on a mission
`true`	indicates that there IS a Crew object associated with this Shuttle object
`Carl Ton`	name of the Crew
`A 01`	**unique** id of the Crew
`5`	number of missions that the Crew has participated in
`CV 64`	**unique** id of the Shuttle
`false`	indicates the Shuttle is NOT on a mission
`false`	indicates that there is NOT a Crew object associated with this Shuttle object

Another example of an input file:

`C33 7`	**unique** id of the Shuttle
`false`	indicates the Shuttle is NOT on a mission
`false`	indicates that there is NO Crew object associated with this Shuttle object
`D 45 ERT`	**unique** id of the Shuttle
`false`	indicates the Shuttle is NOT on a mission
`true`	indicates that there is a Crew object associated with this Shuttle object
`ISC Ness`	name of the Crew
`B 12`	**unique** id of the Crew
`3`	number of missions that the Crew has worked

In both the examples, the first 2 lines are required to instantiate the **`Shuttle`** object reference for the **`Shuttle`** object, the third line indicates whether, or not, there is a **`Crew`** object to read. If there is, then the next 3 lines are required to instantiate the **`Crew`** object reference in the same **`Shuttle`** object.

The file may contain any number of records.

As a final note, consider that when the user enters a **`Crew id`** or a **`Shuttle id`** from the keyboard, **to add a Crew or add a Shuttle**, the program must check that the **`Crew id`** or **`Shuttle id`** are not already in use. If they are, then the user is informed with an appropriate message to the screen and the method returns to the main menu.

`Crew id's` and **`Shuttle id's`** in the input file are guaranteed to be unique and you may assume that the user will not enter a **`Crew id`** or a **`Shuttle id`** from the keyboard that is already in the input text file, even though you have not read the text file. Your program does NOT need to check for this.

The user can select this add from input file option at any time whilst the program is running, it does not have to be at the start.

2. Add Shuttle

This menu option is chosen when the user attempts to add a **`Shuttle`** object from the keyboard. If both of the **`Shuttle`** object references **`shuttle1`** and **`shuttle2`** already have **`Shuttle`** objects, then the user is informed via a message to the screen **and the program returns to the main menu without asking the user for any information.**

If at least one of the **`Shuttle`** object references does not have a **`Shuttle`** object, then the user is asked to enter the **`id`** of a **`Shuttle`**.

The program must check that this **`Shuttle id`** is not already assigned to a **`Shuttle`**. If the **`Shuttle id`** is already assigned to another **`Shuttle`**, the user is informed via a message to the screen and the program returns to the main menu. If the user entered **`Shuttle id`** is unique, then the appropriate **`Shuttle`** constructor is called and a new **`Shuttle`** object is instantiated.

NO Crew information is entered in this menu choice.

If both **`Shuttle`** object references are free (do not have **`Shuttle`** objects attached to them), then **shuttle1 is always used first.** If **`shuttle1`** is not free, then **`shuttle2`** is used.

How can you tell if a **`Shuttle`** object reference is "free", that is, does not have a **`Shuttle`** object attached? We have studied this ☺

Regardless of the action taken in this menu choice, the program returns to the main menu when this menu choice is completed.

3. Start Mission

This menu choice first checks that there is at least one non-null **`Shuttle`** object reference. If both **`Shuttle`** object references are **`null`**, then an appropriate message is displayed to the screen and the program returns to the main menu. If there is at least one non-null **`Shuttle`** object reference, then the user is prompted (asked) for the **`id`** of a **`Shuttle`**.

The program must then try to find the **`Shuttle`** that contains the **`Shuttle`** object with this **`Shuttle id`**. This could be the first **`Shuttle`** object or the second **`Shuttle`** object (if it exists). If the **`Shuttle id`** is not found, then an appropriate message is displayed to the screen and the program returns to the main menu.
If the **`Shuttle id`** is found, then the program must check that this **`Shuttle`** is available for a mission.

To be available for a mission, the **`Shuttle`** must not on a mission (working) **and** the **`Shuttle`** **must** have a **`Crew`**. That is, the **`Crew`** object reference in the **`Shuttle`** object in which the **`Shuttle`** with the user entered **`Shuttle id`** was found, must have a **`Crew`** object attached to it.

If the **`Shuttle`** is not available for a mission, then an appropriate message is displayed to the screen and the program returns to the main menu.

After passing the checks above, the user is asked to enter the required **`skill level`** of the mission. If the **`Crew`** has the required **`skill level`**, then and only then, is the **`Shuttle`** assigned that mission.

If the **`Shuttle`** can be assigned a mission, then the appropriate changes are made to the **`Shuttle`** and its associated **`Crew`** object. These are that the number of **`missions`** worked by the **`Crew`** is incremented (do not forget to check whether this crosses one of the boundaries of the **`skill levels`**, resulting in a change in **`skill level`**) and the **`on mission`** attribute of the **`Shuttle`** is set to **`true`**, indicating that the **`Shuttle`** is on a mission.

The program always returns to the main menu at the end of the menu choice, regardless of the action taken or not taken.

4. End Mission

This menu choice prompts (asks) the user for the **`id`** of a **`Shuttle`**, after making the same checks as in **`Start Mission`**, that is, there is actually at least one non-null **`Shuttle`** object reference. As with **Add `Mission`** appropriate messages should be displayed to the screen if the **`Shuttle`** with the user entered **`Shuttle id`** is not found, there are no non-null **`Shuttle`** object references or if the `Shuttle id` is found but that `Shuttle` is **not already on a mission**. (You can't end a mission unless the `Shuttle` is currently on a mission.)

If any of the above are `true,` then the program returns to the main menu.

If the `Shuttle` with the user entered `Shuttle id` exists and the `Shuttle` is on a mission, then the `on mission` attribute is set to `false`, indicating that the `Shuttle` is not on a `mission`.

After the conclusion of any and all actions in this menu choice, the program returns to the main menu.

5. Display

This menu choice displays the contents of the **non-null** `Shuttle` object references to the screen. The format is shown in the sample run of the program on pages 14 and 16. If both `Shuttle` object references are `null` then an appropriate message is displayed to the screen.
Calling this menu choice must not result in the program crashing (for example, NullPointerException)

6. Add Crew

This menu choice adds a `Crew` to a `Shuttle`, using information from the keyboard. First, of course, the program must check that there is actually a `Shuttle` to which we can add a `Crew`, If there are no `Shuttle's` (that is, there are no `Shuttle` objects), then the user is informed with a message to the screen and the program returns to the main menu. **The user is not asked for any further information**.

If there is at least one `Shuttle` object, then the user is asked for the `id` of a `Shuttle`.
The program tries to find the `Shuttle` with this `Shuttle id`.

If the `Shuttle` with the user entered `Shuttle id` is found, then there is one further check. This is to check that that `Shuttle` does not already have a `Crew` object. Recall that there can only be one `Crew` object in a `Shuttle` object. If there is already a `Crew` object, then the user is informed via a message to the screen, **no further information is asked from the user**, and the program returns to the main menu.

If the `Shuttle` with the user entered `Shuttle id` is found and it does not have a `Crew` object, then the user is asked to enter the `id` of a `Crew`. There is one final check. The program must check that this user entered `Crew id` is not already in use. If the user entered `Crew id` is already in use, then the user is informed via a message to the screen, **no further information is requested from the user**, and the program returns to the main menu.

If the program gets to this point, then there is a `Shuttle` object with the user entered `Shuttle id` and the user entered `Crew id` is unique. The user is then asked to enter the `name` of the `Crew` and the `Crew` object is added to that `Shuttle` object. (The number of `missions` must be 0, as we have just instantiated the `Crew` and the `skill level` is worked out by the `Crew` constructor, based on the number of `missions`).

If there is not a `Shuttle` with the user entered `Shuttle id`, then a message is displayed to the screen and the program returns to the main menu, **without asking for any more information**.

7. Exit the program

This menu choice closes the program.

Electronic Submission of the Source Code

- Submit all the Java files that you have developed in the missions above.
- The code has to run under Unix on the latcs8 machine.
- You submit your files from your latcs8 account. Make sure you are in the same directory as the files you are submitting. Submit each file separately using the **`submit`** command.

```
        submit OOF Crew.java
        submit OOF Shuttle.java
        submit OOF SpaceShip.java
```

After submitting the files, you can run the following command that lists the files submitted from your account:

```
        verify
```

You can submit the same filename as many times as you like before the assignment deadline; the previously submitted copy will be replaced by the latest one.

Note: in the above requirements, where it talks about Shuttle id's and Crew id's being unique, this applies on a case insensitive basis.

That is, `A 10 N23` and `a 10 n23` are to be treated as exactly the same id.

Please make sure that you have read page 2 about the submission close off date and time and the compulsory requirement to attend the execution test during Week 10
Failure to do both of these things will result in your assignment be awarded a mark of 0, regardless of the correctness of the program.

Execution test marks are provisional and subject to final plagiarism checks and checks on the compliance of your code to this assignment document.

As such final assignment marks may be lower or withdrawn completely.

Example run of the program (NOTE not all functionality and error checks/messages are shown)

user input in bold

```
> java SpaceShip

ISC Planetary Explorer Main menu
     1. Load from file
     2. Add Shuttle
     3. Start Mission
     4. End Mission
     5. Display
     6. Add Crew
     7. Close the program
Enter choice >> 1

Enter file name >> c1.dat

ISC Planetary Explorer Main menu
     1. Load from file
     2. Add Shuttle
     3. Start Mission
     4. End Mission
     5. Display
     6. Add Crew
     7. Close the program
Enter choice >> 5

Here are the Shuttle details

Shuttle
[ Id: C33 7
  is not on a mission
  This shuttle has no Crew assigned
]
Shuttle
[ Id: D 45 ERT
  is not on a mission
 Crew
   [ name: ISC Ness id: B 12
     Missions: 3 level: Novice
   ]
]

ISC Planetary Explorer Main menu
     1. Load from file
     2. Add Shuttle
     3. Start Mission
     4. End Mission
     5. Display
     6. Add Crew
     7. Close the program
Enter choice >> 3
Enter Shuttle id >> C33 7

This shuttle does not have a Crew so cannot start a mission

ISC Planetary Explorer Main menu
     1. Load from file
     2. Add Shuttle
     3. Start Mission
     4. End Mission
     5. Display
     6. Add Crew
     7. Close the program
Enter choice >> 3
Enter Shuttle id >> D 45 ERT
Enter required skill level >> expert

The Crew of this Shuttle does not have required level of expertise
to carry out this mission.

ISC Planetary Explorer Main menu
     1. Load from file
     2. Add Shuttle
     3. Start Mission
     4. End Mission
     5. Display
     6. Add Crew
     7. Close the program
Enter choice >> 4
Enter Shuttle id >> C33 7

The shuttle with id C33 7 is not on a mission, so cannot end mission
```

```
ISC Planetary Explorer Main menu
     1. Load from file
     2. Add Shuttle
     3. Start Mission
     4. End Mission
     5. Display
     6. Add Crew
     7. Close the program
Enter choice >> 4
Enter Shuttle id >> D 45 ERT

The shuttle with id D 45 ERT is not on a mission, so cannot end mission

ISC Planetary Explorer Main menu
     1. Load from file
     2. Add Shuttle
     3. Start Mission
     4. End Mission
     5. Display
     6. Add Crew
     7. Close the program
Enter choice >> 3
Enter Shuttle id >> d 45 ert
Enter required skill level >> novice

ISC Planetary Explorer Main menu
     1. Load from file
     2. Add Shuttle
     3. Start Mission
     4. End Mission
     5. Display
     6. Add Crew
     7. Close the program
Enter choice >> 5

Here are the Shuttle details

Shuttle
[ Id: C33 7
  is not on a mission
  This shuttle has no Crew assigned
]
Shuttle
[ Id: D 45 ERT
  is on a mission
 Crew
   [ name: ISC Ness id: B 12
     Missions: 4 level: Novice
   ]
]
```

Please note: it is a marking requirement that the format of the Shuttle and the Crew display is exactly as shown in this example run.

```
ISC Planetary Explorer Main menu
     1. Load from file
     2. Add Shuttle
     3. Start Mission
     4. End Mission
     5. Display
     6. Add Crew
     7. Close the program
Enter choice >> 6
Enter shuttle id >> d 45 ert

This shuttle already has a Crew, cannot add Crew
```

```
ISC Planetary Explorer Main menu
     1. Load from file
     2. Add Shuttle
     3. Start Mission
     4. End Mission
     5. Display
     6. Add Crew
     7. Close the program
Enter choice >> 6
Enter shuttle id >> c33 7
Enter crew id >> b 12

This crew id is already in use.
Crew id's must be unique

ISC Planetary Explorer Main menu
     1. Load from file
     2. Add Shuttle
     3. Start Mission
     4. End Mission
     5. Display
     6. Add Crew
     7. Close the program
Enter choice >> 2

Already at maximum Shuttle capacity cannot add any more Shuttles

ISC Planetary Explorer Main menu
     1. Load from file
     2. Add Shuttle
     3. Start Mission
     4. End Mission
     5. Display
     6. Add Crew
     7. Close the program
Enter choice >> 1

All Shuttles have been allocated

ISC Planetary Explorer Main menu
     1. Load from file
     2. Add Shuttle
     3. Start Mission
     4. End Mission
     5. Display
     6. Add Crew
     7. Close the program
Enter choice >> 6
Enter shuttle id >> c33 7
Enter crew id >> C 16 T
Enter name >> Second Crew

ISC Planetary Explorer Main menu
     1. Load from file
     2. Add Shuttle
     3. Start Mission
     4. End Mission
     5. Display
     6. Add Crew
     7. Close the program
Enter choice >> 5
```

```
Here are the Shuttle details

Shuttle
[ Id: C33 7
  is not on a mission
 Crew
   [ name: Second Crew id: C 16 T
     Missions: 0 level: Novice
   ]
]
Shuttle
[ Id: D 45 ERT
  is on a mission
 Crew
   [ name: ISC Ness id: B 12
     Missions: 4 level: Novice
   ]
]

ISC Planetary Explorer Main menu
     1. Load from file
     2. Add Shuttle
     3. Start Mission
     4. End Mission
     5. Display
     6. Add Crew
     7. Close the program
Enter choice >> 4
Enter Shuttle id >> d 45 ert

ISC Planetary Explorer Main menu
     1. Load from file
     2. Add Shuttle
     3. Start Mission
     4. End Mission
     5. Display
     6. Add Crew
     7. Close the program
Enter choice >> 5

Here are the Shuttle details

Shuttle
[ Id: C33 7
  is not on a mission
 Crew
   [ name: Second Crew id: C 16 T
     Missions: 0 level: Novice
   ]
]
Shuttle
[ Id: D 45 ERT
  is not on a mission
 Crew
   [ name: ISC Ness id: B 12
     Missions: 4 level: Novice
   ]
]

ISC Planetary Explorer Main menu
     1. Load from file
     2. Add Shuttle
     3. Start Mission
     4. End Mission
     5. Display
     6. Add Crew
     7. Close the program
Enter choice >> 7
```

Another run of the program

```
> java SpaceShip

ISC Planetary Explorer Main menu
     1. Load from file
     2. Add Shuttle
     3. Start Mission
     4. End Mission
     5. Display
     6. Add Crew
     7. Close the program
Enter choice >> 5

There are no Shuttles to display
Add at one least one Shuttle


ISC Planetary Explorer Main menu
     1. Load from file
     2. Add Shuttle
     3. Start Mission
     4. End Mission
     5. Display
     6. Add Crew
     7. Close the program
Enter choice >> 6

There are no Shuttles!
Cannot add Crew until there is at least one Crew

ISC Planetary Explorer Main menu
     1. Load from file
     2. Add Shuttle
     3. Start Mission
     4. End Mission
     5. Display
     6. Add Crew
     7. Close the program
Enter choice >> 3

Cannot start mission as there are no Shuttles.
Add at least one Shuttle

ISC Planetary Explorer Main menu
     1. Load from file
     2. Add Shuttle
     3. Start Mission
     4. End Mission
     5. Display
     6. Add Crew
     7. Close the program
Enter choice >> 4
```

```
Cannot end mission as there are no Shuttles.
Add at least one Shuttle

ISC Planetary Explorer Main menu
     1. Load from file
     2. Add Shuttle
     3. Start Mission
     4. End Mission
     5. Display
     6. Add Crew
     7. Close the program
Enter choice >> 2
Enter Shuttle id >> A 09 D45

ISC Planetary Explorer Main menu
     1. Load from file
     2. Add Shuttle
     3. Start Mission
     4. End Mission
     5. Display
     6. Add Crew
     7. Close the program
Enter choice >> 5

Here are the Shuttle details

Shuttle
[ Id: A 09 D45
  is not on a mission
  This shuttle has no Crew assigned
]

ISC Planetary Explorer Main menu
     1. Load from file
     2. Add Shuttle
     3. Start Mission
     4. End Mission
     5. Display
     6. Add Crew
     7. Close the program
Enter choice >> 2
Enter Shuttle id >> A 09 D45

The requested shuttle id is already in use. Shuttle id's must be unique.
Rerun the menu choice with valid shuttle id

ISC Planetary Explorer Main menu
     1. Load from file
     2. Add Shuttle
     3. Start Mission
     4. End Mission
     5. Display
     6. Add Crew
     7. Close the program
Enter choice >> 2
Enter Shuttle id >> A 10 Z6

ISC Planetary Explorer Main menu
     1. Load from file
     2. Add Shuttle
     3. Start Mission
     4. End Mission
     5. Display
     6. Add Crew
     7. Close the program
Enter choice >> 5

Here are the Shuttle details

Shuttle
[ Id: A 09 D45
  is not on a mission
  This Shuttle has no Crew assigned
]
Shuttle
[ Id: A 10 Z6
  is not on a mission
  This shuttle has no Crew assigned
]

ISC Planetary Explorer Main menu
     1. Load from file
     2. Add Shuttle
     3. Start Mission
     4. End Mission
     5. Display
     6. Add Crew
     7. Close the program
Enter choice >> 6
Enter shuttle id >> A 10 z6
Enter crew id >> ISC 01
Enter name >> First Crew

ISC Planetary Explorer Main menu
     1. Load from file
     2. Add Shuttle
     3. Start Mission
     4. End Mission
     5. Display
     6. Add Crew
     7. Close the program
Enter choice >> 5

Here are the Shuttle details

Shuttle
[ Id: A 09 D45
  is not on a mission
  This shuttle has no Crew assigned
]
Shuttle
[ Id: A 10 Z6
  is not on a mission
 Crew
   [ name: First Crew id: ISC 01
     Missions: 0 level: Novice
   ]
]

ISC Planetary Explorer Main menu
     1. Load from file
     2. Add Shuttle
     3. Start Mission
     4. End Mission
     5. Display
     6. Add Crew
     7. Close the program
Enter choice >> 1

All Shuttles have been allocated

ISC Planetary Explorer Main menu
     1. Load from file
     2. Add Shuttle
     3. Start Mission
     4. End Mission
     5. Display
     6. Add Crew
     7. Close the program
Enter choice >> 2

Already at maximum Shuttle capacity cannot add any more Shuttles

ISC Planetary Explorer Main menu
     1. Load from file
     2. Add Shuttle
     3. Start Mission
     4. End Mission
     5. Display
     6. Add Crew
     7. Close the program
Enter choice >> 3
Enter Shuttle id >> B 1

No shuttle with that id was found

ISC Planetary Explorer Main menu
     1. Load from file
     2. Add Shuttle
     3. Start Mission
     4. End Mission
     5. Display
     6. Add Crew
     7. Close the program
Enter choice >> 3
Enter Shuttle id >> A 10 z6
Enter required skill level >> novice

ISC Planetary Explorer Main menu
     1. Load from file
     2. Add Shuttle
     3. Start Mission
     4. End Mission
     5. Display
     6. Add Crew
     7. Close the program
Enter choice >> 5

Here are the Shuttle details

Shuttle
[ Id: A 09 D45
  is not on a mission
  This shuttle has no Crew assigned
]
Shuttle
[ Id: A 10 Z6
  is on a mission
```

```
 Crew
   [ name: First Crew id: ISC 01
     Missions: 1 level: Novice
   ]
]

ISC Planetary Explorer Main menu
     1. Load from file
     2. Add Shuttle
     3. Start Mission
     4. End Mission
     5. Display
     6. Add Crew
     7. Close the program
Enter choice >> 7
```

How the assignment is marked

Please note that the assignment is marked, face to face, in your lab. You are required to run your program. This means that you need to have code that compiles, runs and displays to the screen. Code that does not compile will achieve a mark of 0. We cannot award marks for non-compiling, non-running or non-displaying code, that is, look through your code and award marks on what might have happened if you could have gotten your code to run and/or display.

It is essential that your code displays to the screen the outcome of the actions that you have coded. Without this display, we have no way of checking that your assignment is in fact meeting the requirements. Please refer to the paragraph above.

The smallest amount of code that compiles, runs and displays an outcome will get more marks than a whole assignment that does not compile, does not run or does not display anything to the screen.

Final Notes: - transferring files between Windows and Unix

Be very careful transferring files from Windows to Unix. If you do transfer a file from Windows to Unix open the file, in Unix, using vi.

For example, if you transferred a file named **`b.txt`** from Windows to Unix

open the file in Unix with the command

`vi –b b.txt`

you will (might) see a lot of **`^M`**'s at the end of each line.

These MUST be removed using the command shown below or else your input file will have too many newline characters and will not translate properly. That is, your code will not correctly read the input file(s).

Your code will work on Windows but NOT on Unix.

Still in vi, in command mode (press the Esc key first) do the following

```
:%s/ctrl-v ctrl-m//g
```

`ctrl-v ctrl-m` means hold down the control key and with the control key down press v then press m.

Final, final notes

Don't let anyone look at your code and definitely don't give anyone a copy of your code. The plagiarism checker will pick up that the assignments are the same and you will both get 0 (it doesn't matter if you can explain all your code).

There will be consultation sessions for the assignment, the times will be posted on LMS. If you have problems come to consultation.

And a final, final, final note, "eat the dragon in little bits". Do a little bit every night; before you know it you will be finished. The assignment is marked with running code, so you are better to have 2 or 3 parts completed that actually compile and run, rather than a whole lot of code that doesn't compile.

The execution test is done on latcs8 so please make sure that your code runs on latcs8 before you submit.