Department of Computer Science and Information Technology

La Trobe University

CSE1OOF/4OOF Semester 2, 2017
Progress Check Test Assignment

Due Date: Wednesday, 18th October 2017, at 10.00 a.m.

First and Final date for SUBMISSION Wednesday 18th October 2017 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 on Wednesday 18th October – in your normal, assigned, lab (Week 12)**
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.
- ◊ to practice using arrays of objects.

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. **<u>For example</u>**, the file **`Crew.java`** would be submitted with the command:

```
>  submit  OOF  Crew.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 10% 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 12) constitutes a return of the assignment mark, subject to the conditions on pages 15 - 16

Please note carefully:

<u>The submit server will close at 10:00 am on Wednesday October 18th</u>

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

Please make sure that you have submitted **<u>all</u>** your assignment files before the submit server closes. (see pages 15 -16)

There can be NO extensions or exceptions.

Your assignment will be marked in your normal lab, starting, Wednesday October 18th 2017.

You <u>must</u> attend the lab that you are allocated, according to the LMS page. Non-attendance will result in your assignment being awarded 0.

If you cannot attend your assigned lab, please email me (r.tresider@latrobe.edu.au) to arrange another time.

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 (20 marks) and the exam (60 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.

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

Humans are expanding into space. Cities in space are being established. Constructing and maintaining these cities is undertaken by Space Vehicles. Space Vehicles are equipped to handle any required task. Tasks are known as jobs.

Assignment C explored all aspects of these operations, using just 2 Space Vehicles. This proved to be such a success that now it has been decided to scale up the program to handle a full group of 30 Space Vehicles.

Each Space Vehicle can now have more than one Crew, but still must have at least one Crew assigned to the Space Vehicle before it can be assigned a job and the Space Vehicle must not already be on a job. When the Space Vehicle is on a job, it is assumed that the job is always completed. Each job counts as 1 hour for a Space Vehicle.

It is not possible to have a Crew without there being a Space Vehicle first. It is possible to have a Space Vehicle without a Crew.

When a Space Vehicle is first created, it is added to the Space City. This means, of course, that when a Space Vehicle is first added, it has no Crew, it is not on a job and its hours are 0.

As the Space Vehicle (and the assigned Crew) carry out more jobs, the level of the Crew increases. (Note that Crew here refers to a single Crew, not the whole group, within the whole group, there can be individual Crew with different levels)

If the Crew has carried out between 0 and 3 jobs (inclusive),
    then their level is **`trainee`**
If the Crew has carried out between 4 and 10 jobs (inclusive),
    then their level is **`trained worker`**
If the Crew has carried out between 11 and 16 jobs (inclusive),
    then their level is **`advanced worker`**
If the Crew has carried out 17, or more, jobs,
    then their level is **`specialist`**

To be assigned a job, firstly, the Space Vehicle must have a Crew (one or more) and not be on a job.

Secondly the level required for the job must be within the overall level of the whole Crew. A Crew (the whole group) with a higher overall level can undertake a job that requires a lesser level, but a Crew (the whole group) cannot undertake a job that requires a higher level.

The Crew (the whole group) must have, at least, the required overall level **before** being assigned the job. As soon as the Space Vehicle (and associated Crew) are assigned a job, the number of jobs that the Crew (each individual Crew) has carried out is incremented by 1. Note that adding 1 to the number of jobs for that particular Crew may move that particular Crew up into the next level, this has to be checked. There can also be bonus hours (jobs) associated with a job, this only applies to the Crew (the whole group), not to the Space Vehicle, see below.

To end a job, the Space Vehicle must actually be on a job.

Further changes from Assignment C

A SpaceVehicle can only take a job if the overall level of the Crew (the whole group) is greater than, or equal to, the user entered level.

How is this calculated?

One way of doing it is to assign 0 to the user entered level.

Then go through the array of Crew, one by one.

If the level of an individual Crew is the same as the user entered level add 0 to an accumulating total.

If the level of an individual Crew is one above the user entered level, then add 1, if it 2 levels above the user entered level, then add 2 and so on.

If the level of an individual Crew is 1 level below the user entered level, then subtract 1, if the level of the particular Crew is 2 levels below, subtract 2 and so on.

As before, if the user entered job level of the job is **`trainee`**, then every Space Vehicle, with at least one (individual) Crew, can undertake that job, provided, of course, that the Space Vehicle is not already on a job.

In a bit more detail

It is slightly more complicated if the user entered job is above **`trainee`**. One way to do it is to assign a local variable for the required overall job level with the starting value of 0.

Then go through the whole array of **`Crew`** (one by one) in that **`SpaceVehicle`**.

If the required job level (the one entered by the user) is **`"specialist"`**

If the level of an individual **`Crew`** is "**`trainee`**", then subtract 3 from the local required overall level variable.

If the level of an individual **`Crew`** is **`"trained worker"`**, then subtract 2 from the local required overall level variable.

If the level of an individual **`Crew`** is **`"advanced worker"`**, then subtract 1 from the local required overall level variable

If the level of an individual **`Crew`** is **`"specialist"`**, then subtract 0 from the local required overall level variable.

If the required job level (the one entered by the user) is **`"advanced worker"`**

If the level of an individual **`Crew`** is **`"trainee"`**, then subtract 2 from the local required overall level variable.

If the level of an individual **`Crew`** is **`"trained worker"`**, then subtract 1 from the local required overall level variable.

If the level of an individual **Crew** is **"advanced worker"**, then subtract 0 from the local required overall level variable

If the level of an individual Crew is **"specialist"**, then **add 1** to the local required overall level variable.

If the required job level (the one entered by the user) is **"trained worker"**

If the level of an individual **Crew** is **"trainee"**, then subtract 1 from the local required overall level variable.

If the level of an individual **Crew** is **"trained worker"**, then subtract 0 from the local required overall level variable.

If the level of an individual **Crew** is **"advanced worker"**, then **add 1** to the local required overall level variable

If the level of an individual **Crew** is **"specialist"**, then **add 2** to the required overall level variable.

After doing this for all the **Crew** in the selected **SpaceVehicle**, if the final overall required level local variable is greater than or equal to 0, then the **Crew** in that **SpaceVehicle** have an **overall job level** that is equal to or better than the required **job level**, as entered by the user, and so can take the **job**. As always, provided the **SpaceVehicle** is not already on a job.

The **Crew** (the whole group) must have the correct, required overall job level **before** being assigned the job. As soon as the **SpaceVehicle** (and associated **Crew**(s)) are assigned a **job**, the number of **jobs** that each **Crew** has carried out is incremented by 1. Note that adding 1 to the number of **jobs** for each **Crew** may move that individual **Crew** up to another **job level**, this has to be checked.

Being assigned a job also adds 1 to the number of hours that the **SpaceVehicle** has worked.

As in Assignment C, there may be bonus hours assigned to a job. Each **Crew** gets this bonus added to their total number of jobs (Recall that 1 job = 1 hour). This means checking whether that individual **Crew** as moved up to another level.

As before, this bonus applies only to the **Crew**, not to the **SpaceVehicle**.

To end a job, the **SpaceVehicle** must actually be on a job.

Program Requirements

You have been asked to write an interactive program, in Java, to aid in monitoring and maintaining all aspects of Space City Operations.

This program expands on the earlier work done in Assignment Part C and now needs to handle 30 SpaceVehicles. Each SpaceVehicle can have more than one Crew associated with it.

To aid in the rapid development of this program, 3 Java files and 1 sample input file are provided for you:

`Crew.java`, **`SpaceVehicle.java`**, **`SpaceCity.java`** and a sample input file **`q.dat`**

WARNING

You can re-use large parts of your Assignment Part C **`Crew.java`** and **`SpaceVehicle.java`** files and some parts of **`SpaceCity.java`**.

Before you copy the start-up files for this assignment from the directory listed below, MAKE SURE THAT YOU CREATE A SEPARATE DIRECTORY FOR THIS ASSIGNMENT! Change into this new directory first, then enter the copy command listed below.

If you do not create a separate directory and run the copy command, you will overwrite your Assignment Part C **`SpaceVehicle.java`** and **`Crew.java`** and we cannot recover these files. You will have to start all over again.

After you have copied the start-up files, compile them and run the program to see how the expanded menu structure fits together. The **`SpaceVehicle.java`** and **`Crew.java`** files from the library area are empty, just to get the driver class, **`SpaceCity.java`**, to compile.

After you compile and run the program, begin by reading the changes that are required in **`Crew.java`** (not many) and **`SpaceVehicle.java`** (quite a few) then copy your Assignment Part C **`Crew.java`** and **`SpaceVehicle.java`** into your new directory and re-compile the program, it should still compile, but won't do anything yet.

Copy them from the unit library area into your current directory using:

```
cp /home/1st/csilib/cse1oof/prog/* .
```

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` (almost the same as Assignment Part C) (this is still for one individual Crew)

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 <u>unique</u> id of the Crew, may be more than one word
• **`jobs`**	This is an integer and is the number of jobs that the Crew has undertaken.(See the explanation below)
• **`level`**	This is a String and is one of the four levels as described above. The level is always based on the **number of jobs that the Crew has undertaken and must be set by the program. The user must NEVER be able to enter the level, nor is it ever stored in a file.**

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

A constructor that takes **`name`**, **`Crew id`**, and **`jobs`** 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 page 11

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

Alternatively, you could write an overloaded constructor that just takes the first two values (**`name`** and **`Crew id`**) as parameters and assigns 0 to the number of **`jobs`**.

Either way, the **`level`** must be set by the constructor, not the user.

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

Consider also that you might find it useful to write a *copy constructor* in the **`Crew`** class. This can then be used in the **`SpaceVehicle`** class if you want to return a privacy leak free copy of the **`Crew`** array from the **`SpaceVehicle`** class. (Copy constructors were shown in lecture 20)

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

The **`Crew`** class also requires a method to increment the number of **`jobs`**. Calling this method adds 1 to the number of **`jobs`**. This method should then check whether that moves the **`Crew`** to the next level.

Jobs may attract a bonus which is added to the number of jobs, after adding 1 to the number of jobs. You will need an overloaded method that takes the bonus, an integer, as a parameter. This means that completing one job can increase the number of jobs undertaken by the **`Crew`** by more than 1. As with the first method to increment the number of jobs, that takes no parameters, the level of the **`Crew`** (each one) needs to be checked after this new figure for the total number of jobs has been worked out.

Finally the **`Crew`** class requires a **`toString`** method which returns a **`String`** with information about the **`Crew`** object, see page 18 for the format of the String to be returned.

Please note that this time you are free to format the screen output in any way, provided only that the user of the program can easily understand the information being displayed.

This is Space City operations after all, the users need to be able to take in the information at a glance, not spend time trying to decipher poorly formatted output.

Space City jobs don't stop while the user tries to read information.

The one addition to the functionality of this **`Crew`** class that you **might** (optional, not have to) consider is a method that can be called to write all of the **`Crew`** information to a text file.

There are other ways of doing this, this is not the only way and **is NOT** the "right" answer.

`SpaceVehicle.java` (<u>some major changes</u>)

<u>The SpaceVehicle class has the following attributes:</u>

- **`working`** This is a **`boolean`** variable, the value **`false`** indicates that the **`SpaceVehicle`** is NOT on a job. The value **`true`** indicates that the **`SpaceVehicle`** is on a job
- **`Space Vehicle id`** This is a String (text) and may consist of more than one word The Space Vehicle id is the **unique** identifier for a **`SpaceVehicle`**
- **`crew`** This is <u>now an array of **`Crew`**</u> class object references for the **`Crew`** objects associated with this **`SpaceVehicle`** (when the **`Crew`** objects are added and instantiated)
- **`maxCrew`** This is the maximum number of **`Crew`** that can be in this **`SpaceVehicle`**. This is set by the user when the **`SpaceVehicle`** object is created
- **`numOfCrew`** This is the current number of **`Crew`** actually in the **`SpaceVehicle`**, starts at 0 when the **`SpaceVehicle`** is instantiated
- **`hours`** This is an integer and is the number of hours that the **`SpaceVehicle`** has been working. Each job counts as just one hour. The bonus that the Crew may get for a job is NOT added on to the number of hours that the **`SpaceVehicle`** has been working.

The **`SpaceVehicle`** class still requires at least 2 overloaded constructors.

An overloaded, constructor for keyboard input for a **`SpaceVehicle`**. This constructor would take just the **`Space Vehicle id`** and the **`maximum number of Crew`** attributes as parameters. Since we have just created the **`SpaceVehicle`** object the value of **`working`** must be **`false`** (it cannot be on a job) and as there are no **`Crew`** associated with this **`SpaceVehicle,`** the **`numOfCrew`** is 0.

A second, overloaded, constructor would be used when reading from a text file. When reading from a text file, the **`SpaceVehicle`** id is in the file as is the **`maximum number`** of **`Crew`**. In addition there are values for **`working`** and the **`number of hours`** that the **`SpaceVehicle`** has worked.

<u>In both constructors, do not forget to correctly deal with the array of **`Crew`**.</u>

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

One of those accessor methods may be a method to return a copy of the **`crew`** array. If you decide to write this accessor, remember to take privacy leaks into consideration.

The **`SpaceVehicle`** class also requires a **`toString`** method that returns a **`String`** with information about the state of that **`SpaceVehicle`** object. The format of the **`String`** is shown in the example output on page 20.

As with the **`Crew`** screen output, **this time you are free to format the output anyway you want.** (Provided that it is easy to read, output all on one line is NOT acceptable)

The **`SpaceVehicle`** class requires at least one method to add a **`Crew`** to the **`SpaceVehicle`**.

This method takes all of the relevant parameters for instantiating a `Crew` object from the keyboard, that is, `name` and `id`. Using the information in these parameters, instantiate a `Crew` object which is stored in the `crew` array, provided that there is a free space. **As before, the program first needs to check that the Crew id entered by the user is indeed unique.**

A second, overloaded method to add a `Crew` should be considered. This method will take all the information for adding a `Crew` from the text file. The difference is that, from the file, in addition to the `Crew id` and `name`, the `number of jobs` is included, whereas when reading the information from the keyboard, the `number of jobs` is, of course, 0.

There will be a number of mutator methods that you will find you need to write, amongst them a method to increment the `number of jobs` for all the `Crew`, in that `SpaceVehicle`, every time the `SpaceVehicle` starts a new `job`. This also includes an overloaded method that takes a parameter, the bonus hours.

As with the `Crew` class, you may want to consider writing a method that writes the details of the `SpaceVehicle` object to a text file. **Remember to write the actual number of crew figure (numOfCrew) into the text file.** This is required so that when we use that output file as an input file, the program knows how many `Crew` records it needs to read before the next `SpaceVehicle` record starts.

As discussed above, the **overall `level`** of all the `Crew`, in a `SpaceVehicle`, must be greater than or equal to the `level` of the job as entered by the user. You may want to consider writing a method that takes the user `level` as a parameter and returns `true` or `false` whether, or not, the overall `level` of the `Crew` is sufficient for the `SpaceVehicle` to be assigned the `job` (provided, first, that all the other conditions are met)

The `Crew` and `SpaceVehicle` 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.

`loading the array of SpaceVehicles from a text file`

The program now starts by asking the user for the name of a text file, see page 11 for the format. This file will contain a number of SpaceVehicle/Crew records. The file name, will as always, exist, and this time will not be empty.

The contents of this file is used to populate the array of SpaceVehicles, in SpaceCity. Once the contents of this file have been read into the array of SpaceVehicles, the file is closed. The only time that a file is opened again is if the user selects the save option.

Where the SpaceVehicle has a Crew (meaning any number of Crew greater than 0) then you want to call your overloaded addCrew method from the SpaceVehicle class, the one that takes all the Crew attributes. Use this method to add Crew to the correct SpaceVehicle.

This results in a change to the menu that is presented to the user. The menu is first presented to the user after the contents of the text file has been read into the array of SpaceVehicles.

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.

An example of the information, in the text file, on each `SpaceVehicle` consists of 4 lines. There may be any number of 3 line `Crew` records after the `SpaceVehicle` information. The `SpaceVehicle` and `Crew` information is collectively known as a **record**:

An example of an input file:

`T 23`	**unique** id of the SpaceVehicle
`2`	number of hours that the SpaceVehicle has worked
`false`	indicates the SpaceVehicle is not on a job
`4`	the maximum number of Crew for this SpaceVehicle
`0`	the actual number of Crew in this SpaceVehicle
`K 20`	**unique** id of the SpaceVehicle
`3`	number of hours that the SpaceVehicle has worked
`true`	indicates the SpaceVehicle is on a job
`6`	the maximum number of Crew for this SpaceVehicle
`2`	the actual number of Crew in this SpaceVehicle
`V 89`	**unique** id of the Crew
`First Person`	name of the Crew
`11`	number of jobs this Crew has undertaken
`V 90`	**unique** id of the Crew
`Second Person`	name of the Crew
`2`	number of jobs this Crew has undertaken

In the example above, the first 4 lines are required to instantiate the `SpaceVehicle` object reference in the `SpaceVehicle` object, the fifth line indicates the number of `Crew` objects to read, followed by this number of groups of 3 lines, which are required to instantiate the `Crew` object references in that `SpaceVehicle` object.

The file may contain any number of records.

Given that this method is called before the user gets to see the main menu, there is now no need to worry about unique `Crew id's` and `Space Vehicle id's` conflicting with anything that the user may enter.

The driver program, `SpaceCity.java` then presents the user with a menu, as follows:

```
Space City Menu
        1. Add Space Vehicle
        2. Add Crew
        3. Add Job
        4. End Job
        5. Display
        6. Save
        7. Exit
Enter choice >>
```

Note: SpaceCity class must NOT have Crew object attribute(s) or Crew object references. All interactions involving Crew will still be through the relevant SpaceVehicle object.

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

`1. Add Space Vehicle` (very much the same as Assignment C)

This menu option is chosen when the user attempts to add a **`SpaceVehicle`** object from the keyboard.

This time the difference is the program must first check if there is a free space in the SpaceVehicles array.

As before, the first thing that the user will enter is the **`Space Vehicle id`** of the **`SpaceVehicle`** to add. The program must check that this **`Space Vehicle id`** is not already assigned to any **`SpaceVehicle`** in the array of **`SpaceVehicles`**.

(Hint: write a **`search`** method that searches through the array of **`SpaceVehicles`** and returns the index of the requested **`SpaceVehicle`** or **`-1`** if the **`SpaceVehicle`** with the requested **`id`** is not found. See Lecture/Workshop 09)

If there is space in the array and the **`Space Vehicle id`** is unique, the user is asked for all the relevant **`SpaceVehicle`** information and the resultant **`SpaceVehicle`** object is added to the next free space in the **`array`** of **`SpaceVehicles`**. If there is no free space in the array, **the program returns to the main menu without asking the user for any information.**

If the **`Space Vehicle id`** is already assigned to a **`SpaceVehicle`**, the user is informed via a message to the screen and the program returns to the main menu.

Still NO Crew information is entered in this menu choice.

`2. Add Crew` (basically the same as Assignment C)

This menu choice adds a **`Crew`** to a **`SpaceVehicle`**, using information from the keyboard. First, of course, the program must check that there is actually at least one **`SpaceVehicle`** in the array. If there are no **`SpaceVehicles`** (that is, the array of **`SpaceVehicles`** is empty), 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 **`SpaceVehicle`** object in the array, then the user is asked for the **`Space Vehicle id`** of a **`SpaceVehicle`**. The program tries to find the **`SpaceVehicle`** with this **`Space Vehicle id`**. (Hint: use your **`search`** method again)

If the **`SpaceVehicle`** with the user entered **`Space Vehicle id`** is found, then there is one further check. This is to check that that **`SpaceVehicle`** does not already have its maximum number of Crew. If the **`SpaceVehicle`** already has its maximum number of **`Crew`**, 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 **`SpaceVehicle`** with the user entered **`Space Vehicle id`** is found and is not at its **`maximum Crew`**, 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.

This is a more complex problem than in Assignment C, although it is still the same condition. As each **SpaceVehicle** now has an array of **Crew**, the program has to go through all the **SpaceVehicle** objects in the array of **SpaceVehicle's** and check through the array of **Crew** in each **SpaceVehicle**. Only when this has been done and the **id** for that **Crew** (the one entered by the user) has not been found, in any of the **SpaceVehicles**, do we know that the **Crew id** entered by the user is indeed unique.

There are 2 basic strategies for doing this, the first is to return a (privacy leak free) copy of the **Crew** array for an individual **SpaceVehicle** to the main driver program (**SpaceCity**). The program can then go through this array comparing the **Crew id's** in that array with the **id** entered by the user. If the **id** is not found and there is another **SpaceVehicle** in the array of **SpaceVehicles**, the program moves to the next **SpaceVehicle**, gets a copy of the **Crew** array for that **SpaceVehicle** and searches through that array.

The alternative is to write a method in the **SpaceVehicle** class that takes the user entered **id** as a parameter. The method in the **SpaceVehicle** class searches through the **Crew** array and returns **true** or **false** as to whether the **Crew id** passed in as a parameter was found.

Either method is acceptable, write the one that you feel most comfortable with.

If the program gets to this point, then there is a **SpaceVehicle** object with the user entered **Space Vehicle id** and the user entered **Crew id** is unique. The user is then asked to enter the **name** of the **Crew** (you already know the **id**) and the **Crew** object is added to the **Crew** array for that **SpaceVehicle** object. (The number of jobs must be 0, as we have just instantiated the **Crew** and the **job level** is worked out by the **Crew** constructor, based on the number of **jobs**).

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

3. Add Job

This menu choice first checks that there is at least one **SpaceVehicle** object reference in the array. If not, 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 **SpaceVehicle** object reference in the array, then the user is prompted (asked) for the **Space Vehicle id** of a **SpaceVehicle**.

The program must then find the **Space Vehicle** that contains the **SpaceVehicle** object with this **Space Vehicle id**, again, you could use the **search** method. This could be anywhere in the array. If the **SpaceVehicle** with that **Space Vehicle id** is not found, then an appropriate message is displayed to the screen and the program returns to the main menu.

If the **SpaceVehicle** with that **Space Vehicle id** is found, then the program must check that this **SpaceVehicle** is available for a **job**.

To be available for a **job**, the **SpaceVehicle** must not be on a job **and** the **SpaceVehicle** **must** have a **Crew.** If this is true, the user enters the required **level** of the **job**. The **Crew** (the whole group) must have an overall **level** greater than or equal to the **level** entered by the user. This is explained in detail above.

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

If the **`SpaceVehicle`** can be assigned a **`job`**, then the appropriate changes are made to the **`SpaceVehicle`** object. These are that the number of **`jobs`** undertaken by **all** the **`Crew`**, of that **`SpaceVehicle`**, are incremented by 1 (do not forget to check whether this crosses one of the boundaries of the **`level`**, resulting in a change in **`level`**) and the **`working`** attribute is set to **`true`**, indicating that the **`SpaceVehicle`** is on a job.

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

<u>4. End Job</u>

This menu choice prompts (asks) the user for the **`Space Vehicle id`** of a **`SpaceVehicle`**, after making the same checks as in **`Add Job`**, that is, there is actually at least one non-null **`SpaceVehicle`** object in the array. As with **`Add Job`** appropriate messages should be displayed to the screen if the **`SpaceVehicle`** with the user entered **`Space Vehicle id`** is not found, the array of **`SpaceVehicles`** is empty, or if the **`Space Vehicle id`** is found but that **`SpaceVehicle`** is **not already on a Job**. (You can't end a job unless the **`SpaceVehicle`** is currently on a job.)

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

If the **`SpaceVehicle`** with the user entered **`Space Vehicle id`** exists and the **`SpaceVehicle`** is on a **`job`**, then the **`working`** attribute is set to **`false`**, indicating that the **`SpaceVehicle`** is not on a **`job`**.

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

<u>5. Display</u>

This menu choice displays the contents of the **non-null `SpaceVehicle`** object references to the screen. The format is shown in the sample run of the program on page 18. This time you are free to format the output any way you see fit, with a reminder that the output must be easy for the user of the program to **quickly** understand. **Calling any of these menu choices must not result in the program crashing (for example, NullPointerException)**

<u>This is now the top level menu for display. When the user selects this top level option, another sub menu will be presented.</u>

```
Space City Display Menu
        1. Display all
        2. Display Space Vehicles (no crews)
        3. Display Single Space Vehicle
        4. Display Single Crew
        5. Display working Space Vehicles
        6. Return to main menu
Enter choice >>
```

1. **`Display all`**
   Choosing this menu option, all of the **`SpaceVehicles`**, **with all their Crew**, are displayed to the screen.

2. **`Display Space Vehicles (no Crew)`**
   Choosing this menu option, all of the **`SpaceVehicles`**, **this time without any Crew information**, are displayed to the screen.

3. **`Display Single Space Vehicle`**
   Choosing this menu option, the user is prompted (asked) for the **`Space Vehicle id`** of a **`SpaceVehicle`**. If the **`SpaceVehicle`** with that **`Space Vehicle id`** is found in the array, then all the information about that **`SpaceVehicle`**, including its **`Crew(s)`**, are displayed to the screen.
   If the **`SpaceVehicle`** with that **`Space Vehicle id`** is not found in the array, then an appropriate message is displayed to the screen.

4. **`Display Single Crew`**
   Choosing this menu option, the user is prompted (asked) for the **`id`** of a **`Crew`**. If the **`Crew`** with that **`id`** is found, then the information for that one **`Crew`** is displayed to the screen. If the **`Crew`** with the requested **`id`** is not found, then an appropriate message is displayed to the screen.

5. **`Display working Space Vehicles`**
   Choosing this menu option, only those **`SpaceVehicles`** that are on a **`job`**, with their **`Crew`** information, are displayed to the screen

6. **`Return to main menu`**

`6. Save` (see pages 27 - 28 for how to do this)

Choosing this menu option, the user is prompted (asked) for the name of an output text file and all the information in the array is written to this text file.

File names must NOT be hard coded.

Also recall that this output file must be able to be used as input file the next time that the program is run, so the output file needs to be written to the file in the format as shown on page 11.

`7. Exit`

This menu choice closes the program, without asking the user if they would like to save their changes.

Electronic Submission of the Source Code

- Submit all the Java files that you have developed in the jobs 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 SpaceVehicle.java
submit OOF SpaceCity.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.

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 12 (Oct 18 2017)

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.

How the assignment is marked

Please also 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.

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

user input in bold

```
java SpaceCity
Enter file name >> city01.dat
Space City Menu
    1. Add Space Vehicle
    2. Add Crew
    3. Add Job
    4. End Job
    5. Display
    6. Save
    7. Exit
Enter choice >> 5

Space City Display Menu
    1. Display all
    2. Display Space Vehicles (no crews)
    3. Display Single Space Vehicle
    4. Display Single Crew
    5. Display working Space Vehicles
```

```
     6. Return to main menu
Enter choice >> 1

Here is the list of Space Vehicles

SpaceVehicle
[
    Id : T 23
    Hours: 2
    Currently not working
    Has a maximum crew capacity of 4
    This Space Vehicle has no crew
]
SpaceVehicle
[
    Id : K 20
    Hours: 3
    Currently working
    Has a maximum crew capacity of 6
Crew [
    Name:  First Person
    Level: advanced worker
    Id:    V 89
    Jobs:  11
     ]
Crew [
    Name:  Second Person
    Level: trainee
    Id:    V 90
    Jobs:  2
     ]
]

Space City Display Menu
     1. Display all
     2. Display Space Vehicles (no crews)
     3. Display Single Space Vehicle
     4. Display Single Crew
     5. Display working Space Vehicles
     6. Return to main menu
Enter choice >> 2

Here is the list of Space Vehicles only (no crew)

SpaceVehicle
[
    Id : T 23
    Hours: 2
    Currently not working
    Has a maximum crew capacity of 4
    Currently has 0 crew
SpaceVehicle
[
    Id : K 20
    Hours: 3
    Currently working
    Has a maximum crew capacity of 6
    Currently has 2 crew

Space City Display Menu
     1. Display all
     2. Display Space Vehicles (no crews)
     3. Display Single Space Vehicle
     4. Display Single Crew
     5. Display working Space Vehicles
     6. Return to main menu
Enter choice >> 3

Enter Space Vehicle Id >> A 45

No Space Vehicle with that id was found

Space City Display Menu
     1. Display all
     2. Display Space Vehicles (no crews)
     3. Display Single Space Vehicle
     4. Display Single Crew
     5. Display working Space Vehicles
     6. Return to main menu
Enter choice >> 3

Enter Space Vehicle Id >> K 20
SpaceVehicle
[
    Id : K 20
    Hours: 3
    Currently working
    Has a maximum crew capacity of 6
Crew [
    Name:  First Person
    Level: advanced worker
    Id:    V 89
    Jobs:  11
     ]
Crew [
    Name:  Second Person
    Level: trainee
    Id:    V 90
    Jobs:  2
     ]
]

Space City Display Menu
     1. Display all
     2. Display Space Vehicles (no crews)
     3. Display Single Space Vehicle
     4. Display Single Crew
     5. Display working Space Vehicles
     6. Return to main menu
Enter choice >> 4

Enter crew Id >> B 12

No Crew with that id was found

Space City Display Menu
     1. Display all
     2. Display Space Vehicles (no crews)
     3. Display Single Space Vehicle
     4. Display Single Crew
     5. Display working Space Vehicles
     6. Return to main menu
Enter choice >> 4

Enter crew Id >> V 89
```

```
Here is the Crew information

Crew [
    Name:  First Person
    Level: advanced worker
    Id:    V 89
    Jobs:  11
     ]

Space City Display Menu
     1. Display all
     2. Display Space Vehicles (no crews)
     3. Display Single Space Vehicle
     4. Display Single Crew
     5. Display working Space Vehicles
     6. Return to main menu
Enter choice >> 5
SpaceVehicle
[
    Id : K 20
    Hours: 3
    Currently working
    Has a maximum crew capacity of 6
    Currently has 2 crew

Space City Display Menu
     1. Display all
     2. Display Space Vehicles (no crews)
     3. Display Single Space Vehicle
     4. Display Single Crew
     5. Display working Space Vehicles
     6. Return to main menu
Enter choice >> 6

Space City Menu
     1. Add Space Vehicle
     2. Add Crew
     3. Add Job
     4. End Job
     5. Display
     6. Save
     7. Exit
Enter choice >> 4
Enter Space Vehicle id >> T 23

This Space Vehicle is not currently on a job, so cannot end job

Space City Menu
     1. Add Space Vehicle
     2. Add Crew
     3. Add Job
     4. End Job
     5. Display
     6. Save
     7. Exit
Enter choice >> 1
Enter Space Vehicle id >> K 20

That Space Vehicle ID is already assigned.
Space Vehicle Id's must be unique

Space City Menu
     1. Add Space Vehicle
     2. Add Crew
     3. Add Job
     4. End Job
     5. Display
     6. Save
     7. Exit
Enter choice >> 1
Enter Space Vehicle id >> HF 001
Enter max crew >> 5
Space City Menu
     1. Add Space Vehicle
     2. Add Crew
     3. Add Job
     4. End Job
     5. Display
     6. Save
     7. Exit
Enter choice >> 5

Space City Display Menu
     1. Display all
     2. Display Space Vehicles (no crews)
     3. Display Single Space Vehicle
     4. Display Single Crew
     5. Display working Space Vehicles
     6. Return to main menu
Enter choice >> 1

Here is the list of Space Vehicles

SpaceVehicle
[
    Id : T 23
    Hours: 2
    Currently not working
    Has a maximum crew capacity of 4
    This Space Vehicle has no crew
]
SpaceVehicle
[
    Id : K 20
    Hours: 3
    Currently working
    Has a maximum crew capacity of 6
Crew [
    Name:  First Person
    Level: advanced worker
    Id:    V 89
    Jobs:  11
     ]
Crew [
    Name:  Second Person
    Level: trainee
    Id:    V 90
    Jobs:  2
     ]
]
SpaceVehicle
[
    Id : HF 001
    Hours: 0
    Currently not working
    Has a maximum crew capacity of 5
    This Space Vehicle has no crew
]

Space City Display Menu
     1. Display all
     2. Display Space Vehicles (no crews)
     3. Display Single Space Vehicle
     4. Display Single Crew
     5. Display working Space Vehicles
     6. Return to main menu
Enter choice >> 6
Space City Menu
     1. Add Space Vehicle
     2. Add Crew
     3. Add Job
     4. End Job
     5. Display
     6. Save
     7. Exit
Enter choice >> 3
Enter Space Vehicle id >> T 23

This Space Vehicle has no Crew cannot add job

Space City Menu
     1. Add Space Vehicle
     2. Add Crew
     3. Add Job
     4. End Job
     5. Display
     6. Save
     7. Exit
Enter choice >> 2
Enter Space Vehicle id >> T 23
Enter crew id >> A 01
Enter crew name >> Third Person
Space City Menu
     1. Add Space Vehicle
     2. Add Crew
     3. Add Job
     4. End Job
     5. Display
     6. Save
     7. Exit
Enter choice >> 3
Enter Space Vehicle id >> T 23
Enter job level >> specialist

The Crew of this Vehicle does not have the required job level

Space City Menu
     1. Add Space Vehicle
     2. Add Crew
     3. Add Job
     4. End Job
     5. Display
     6. Save
     7. Exit
Enter choice >> 3
Enter Space Vehicle id >> T 23
Enter job level >> trainee
Are there bonus hours for this job [yes/no] ? no
Space City Menu
```

```
     1. Add Space Vehicle
     2. Add Crew
     3. Add Job
     4. End Job
     5. Display
     6. Save
     7. Exit
Enter choice >> 5

Space City Display Menu
     1. Display all
     2. Display Space Vehicles (no crews)
     3. Display Single Space Vehicle
     4. Display Single Crew
     5. Display working Space Vehicles
     6. Return to main menu
Enter choice >> 1

Here is the list of Space Vehicles

SpaceVehicle
[
    Id : T 23
    Hours: 3
    Currently working
    Has a maximum crew capacity of 4
Crew [
    Name:  Third Person
    Level: trainee
    Id:    A 01
    Jobs:  1
     ]
]
SpaceVehicle
[
    Id : K 20
    Hours: 3
    Currently working
    Has a maximum crew capacity of 6
Crew [
    Name:  First Person
    Level: advanced worker
    Id:    V 89
    Jobs:  11
     ]
Crew [
    Name:  Second Person
    Level: trainee
    Id:    V 90
    Jobs:  2
     ]
]
SpaceVehicle
[
    Id : HF 001
    Hours: 0
    Currently not working
    Has a maximum crew capacity of 5
    This Space Vehicle has no crew
]

Space City Display Menu
     1. Display all
     2. Display Space Vehicles (no crews)
     3. Display Single Space Vehicle
     4. Display Single Crew
     5. Display working Space Vehicles
     6. Return to main menu
Enter choice >> 6

Space City Menu
     1. Add Space Vehicle
     2. Add Crew
     3. Add Job
     4. End Job
     5. Display
     6. Save
     7. Exit
Enter choice >> 4
Enter Space Vehicle id >> K 20

Space City Menu
     1. Add Space Vehicle
     2. Add Crew
     3. Add Job
     4. End Job
     5. Display
     6. Save
     7. Exit
Enter choice >> 2
Enter Space Vehicle id >> T 23
Enter crew id >> A 02
Enter crew name >> Fourth Person

Space City Menu
     1. Add Space Vehicle
     2. Add Crew
     3. Add Job
     4. End Job
     5. Display
     6. Save
     7. Exit
Enter choice >> 4
Enter Space Vehicle id >> K 20

This Space Vehicle is not currently on a job, so cannot end job

Space City Menu
     1. Add Space Vehicle
     2. Add Crew
     3. Add Job
     4. End Job
     5. Display
     6. Save
     7. Exit
Enter choice >> 3
Enter Space Vehicle id >> K 20
Enter job level >> advanced worker

The Crew of this Vehicle does not have the required job level

Space City Menu
     1. Add Space Vehicle
     2. Add Crew
     3. Add Job
     4. End Job
     5. Display
     6. Save
     7. Exit
Enter choice >> 3
Enter Space Vehicle id >> K 20
Enter job level >> trained worker
Are there bonus hours for this job [yes/no] ? yes
Enter hours >> 5

Space City Menu
     1. Add Space Vehicle
     2. Add Crew
     3. Add Job
     4. End Job
     5. Display
     6. Save
     7. Exit
Enter choice >> 5

Space City Display Menu
     1. Display all
     2. Display Space Vehicles (no crews)
     3. Display Single Space Vehicle
     4. Display Single Crew
     5. Display working Space Vehicles
     6. Return to main menu
Enter choice >> 3

Enter Space Vehicle Id >> K 20
SpaceVehicle
[
    Id : K 20
    Hours: 5
    Currently working
    Has a maximum crew capacity of 6
Crew [
    Name:  First Person
    Level: specialist
    Id:    V 89
    Jobs:  17
     ]
Crew [
    Name:  Second Person
    Level: trained worker
    Id:    V 90
    Jobs:  8
     ]
]
Space City Display Menu
     1. Display all
     2. Display Space Vehicles (no crews)
     3. Display Single Space Vehicle
     4. Display Single Crew
     5. Display working Space Vehicles
     6. Return to main menu
Enter choice >> 6

Space City Menu
     1. Add Space Vehicle
     2. Add Crew
     3. Add Job
     4. End Job
     5. Display
     6. Save
     7. Exit
Enter choice >> 2
Enter Space Vehicle id >> T 23
Enter crew id >> A 03
Enter crew name >> Fifth Person

Space City Menu
     1. Add Space Vehicle
     2. Add Crew
     3. Add Job
     4. End Job
     5. Display
     6. Save
     7. Exit
Enter choice >> 3
Enter Space Vehicle id >> T 23

This Space Vehicle is already working, cannot add job

Space City Menu
     1. Add Space Vehicle
     2. Add Crew
     3. Add Job
     4. End Job
     5. Display
     6. Save
     7. Exit
Enter choice >> 2
Enter Space Vehicle id >> T 23
Enter crew id >> A 04
Enter crew name >> Sixth Person

Space City Menu
     1. Add Space Vehicle
     2. Add Crew
     3. Add Job
     4. End Job
     5. Display
     6. Save
     7. Exit
Enter choice >> 2
Enter Space Vehicle id >> T 23

This Space Vehicle already has its maximum crew

Space City Menu
     1. Add Space Vehicle
     2. Add Crew
     3. Add Job
     4. End Job
     5. Display
     6. Save
     7. Exit
Enter choice >> 5

Space City Display Menu
     1. Display all
     2. Display Space Vehicles (no crews)
     3. Display Single Space Vehicle
     4. Display Single Crew
     5. Display working Space Vehicles
     6. Return to main menu
```

```
Enter choice >> 1

Here is the list of Space Vehicles

SpaceVehicle
[
    Id : T 23
    Hours: 3
    Currently working
    Has a maximum crew capacity of 4
Crew [
    Name:  Third Person
    Level: trainee
    Id:    A 01
    Jobs:  1
     ]
Crew [
    Name:  Fourth Person
    Level: trainee
    Id:    A 02
    Jobs:  0
     ]
Crew [
    Name:  Fifth Person
    Level: trainee
    Id:    A 03
    Jobs:  0
     ]
Crew [
    Name:  Sixth Person
    Level: trainee
    Id:    A 04
    Jobs:  0
     ]
]
SpaceVehicle
[
    Id : K 20
    Hours: 5
    Currently working
    Has a maximum crew capacity of 6
Crew [
    Name:  First Person
    Level: specialist
    Id:    V 89
    Jobs:  17
     ]
Crew [
    Name:  Second Person
    Level: trained worker
    Id:    V 90
    Jobs:  8
     ]
]
SpaceVehicle
[
    Id : HF 001
    Hours: 0
    Currently not working
    Has a maximum crew capacity of 5
    This Space Vehicle has no crew
]
Space City Display Menu
```

```
     1. Display all
     2. Display Space Vehicles (no crews)
     3. Display Single Space Vehicle
     4. Display Single Crew
     5. Display working Space Vehicles
     6. Return to main menu
Enter choice >> 6

Space City Menu
     1. Add Space Vehicle
     2. Add Crew
     3. Add Job
     4. End Job
     5. Display
     6. Save
     7. Exit
Enter choice >> 6
Enter output file name >> city02.dat

Space City Menu
     1. Add Space Vehicle
     2. Add Crew
     3. Add Job
     4. End Job
     5. Display
     6. Save
     7. Exit
Enter choice >> 7
```

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.

The PrintWriter class (writing to text files)

We have covered reading from files using the **`Scanner`** class in lectures and laboratories. Writing to files is just as easy.

`System.out` is an object of the **`PrintStream`** class which is connected to the standard output (screen). The **`PrintWriter`** class is similar.

We can create other instances of the **`PrintWriter`** class and connect them to files. Using these other instances, programs can send their output to files rather than the screen.

To create a **`PrintWriter`** object connected to an output file we first create a **`File`** object (as we did with input files):

```
File myFile = new File("output.txt");
```

(The above example uses a **hard coded** file name, which your program must not do. We have been getting the user to enter filenames since Lab 4 so you should know how to do it ☺)

Then we create a **`PrintWriter`** object using this **`File`** object:

```
PrintWriter fout = new PrintWriter(myFile);
```

Alternatively, we can do this in one statement:

```
PrintWriter fout
          = new PrintWriter(new File("output.txt"));
```

We can use the same **`print`**, **`println`** and **`printf`** methods we used with **`System.out`** with the **`fout`** object:

```
fout.println("Hello");
```

After you have finished with your output file, you must use the close method to close the file. If you do not close the file sometimes the file is incomplete.

```
    fout.close();
```

Basically it is the same as **`System.out.print`**, **`System.out.println`** and **`System.out.printf`**. If you know how to write to the screen, then you know how to write to a text file. The only difference is that you have to replace **`System.out`** with your **`PrintWriter`** object reference.

Many of the methods in **`SpaceCity`** require that the user enter a **`Space Vehicle id`** and then the program has to find the **`SpaceVehicle`** with that **`Space Vehicle id`** in the array, or report that the **`SpaceVehicle`** with that **`Space Vehicle id`** is not in the array.

Rather than repeating code (copy and paste) consider writing a method that takes the **`Space Vehicle id`** as a parameter and returns the **`index`** of the **`SpaceVehicle`** with that **`Space Vehicle id`** in the array, or **`-1`** if the **`SpaceVehicle`** with that **`Space Vehicle id`** was not found.

You can then just call this method to return the **`index`**. For example, when you want to **add** a **`SpaceVehicle`**, you want this method to return **`-1`** because then it indicates that the **`Space Vehicle id`** entered by the user is indeed unique.

When you want to **add** a Crew, you want this method to return the **`index`** of the **`SpaceVehicle`** with that **`Space Vehicle id`**. Now you know the **`index`** in the **`array`** of **`SpaceVehicles`** (that is, for which **`SpaceVehicle`**) to call the **`addCrew`** method. If the search method returns **`-1`**, then you know that the **`SpaceVehicle`** with the requested **`Space Vehicle id`** is not in the array and an appropriate message can be displayed to the screen.