Department of Computer Science and Information Technology

La Trobe University

CSE1OOF/4OOF Sydney SP1, 2017
Assignment Part A and Part B

Due Date: Wednesday,14th Feb, at 11.00 a.m.

First and Final date for SUBMISSION Wednesday,14th Feb, at 11.00 a.m.

Delays caused by computer downtime cannot be accepted as a valid reason for a late submission without penalty.

Please note carefully:

The submit server will close at 12:00 pm on Wednesday Feb 14th

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 17)

There can be NO extensions or exceptions.

Your assignment will be marked in your normal lab, week 7.

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**.

All of the Tasks that follow 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 solve the Tasks below in any way, with one condition.

Any assignment that uses code that is outside what has been presented to this point **must be fully explained at the marking execution test**. Not being able to **fully** 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.

An example would be the `split( )` method in the `String` class. The reason being that this method returns an array and we haven't studied arrays yet. So using the `split( )` method would require you to be prepared to explain to the marker how arrays worked in Java.

Another example is the use of **`BufferedReader`** and **`Integer.parseInt`**( ). **`BufferedReader`** is not being taught at all, so it won't be on the exam and **`Integer.parseInt`**( ) we touch only briefly, but not yet.

Task 1 – calcTickets.java

Write a Java program called **`calcTickets`** that prompts (asks) the user to enter a ticket number. The format of a valid ticket is CDDD[C][C][C]

where D = a digit and C = a character and [C] means that the character at this position
is optional.

The only valid characters at the start of the ticket are 'A', 'C', or 'T'

If there is only one optional character at the end of the ticket String, then that character may only be 'X', 'B' or 'K'

If there are two optional characters at the end of the ticket String, then the characters can only be 'B' **and** 'X', or 'K' **and** 'X', **in that order**.

If there are three optional characters at the end of the ticket String, then the characters can only be 'X', 'B' and 'K', **in that order**.

So an **example** of a valid ticket could be:
    T876BX

The user may enter a ticket String of any length. The valid length of a ticket is between 4 and 7 depending on the conditions above. Any ticket that is not in this length range is automatically invalid. A ticket that is invalid for any reason will result in a message being displayed to the screen saying that the ticket is invalid, and no processing will be done with that ticket.

Every effort should be made to give the exact reason why the ticket is invalid. For example, wrong length, starts with an invalid character, or ends with an invalid character. These are not the only reasons why a ticket is invalid, part of your task is to list all the conditions that make a ticket invalid and write code to cover those cases.

If a ticket is a valid length, you may assume that the digit part of the ticket is entered correctly. That is, it is exactly 3 digits and follows the character at the start of the ticket String.
Note that this means that the first character can be anything on the keyboard including a digit or a punctuation mark.

**You do not have to check that the contents in the 2nd, 3rd and 4th positions in the ticket String are digits. You do, however, need to convert the 3 character/digits into a
base 10 integer**.

Hint: consider that when we enter 0 at the keyboard we are entering the character '0'
    which has an ASCII/Unicode of 48. So what we are really storing is the
    base 10 number 48. We need to convert this from character 0 to integer 0. 1 is actually
    character '1' with ASCII/Unicode value 49 and so on.

If the ticket is valid, then the cost of the ticket is calculated and displayed to the screen.

An 'A' class ticket costs $50
A 'C' class ticket costs $100
A 'T' class ticket costs $200

**The 3 digits that have been converted into a base 10 integer are then added to the cost
of the ticket.** If the 3 digits were, for example, 155 then this would be converted in to the base 10 number one hundred and fifty-five, which would be added to the cost of the ticket, regardless of the class of the ticket.

The character 'X' at the end of the ticket String, **if it is the only optional character** adds $25.50 to the cost, regardless of the ticket class.

The character 'B' at the end of the ticket String, **if it is the only optional character** adds $33.75 to the cost, regardless of the ticket class.

The character 'K' at the end of the ticket String, **if it is the only optional character** adds $41.33 to the cost, regardless of the ticket class.

The characters 'B' and 'X', in that order, if there are the only optional characters at the end of the ticket String add $105.45 to the cost, regardless of the ticket class.

The characters 'K' and 'X', in that order, if there are the only optional characters at the end of the ticket String add $200.50 to the cost, regardless of the ticket class.

The characters 'X', 'B' and 'K', in that order, if there are the only optional characters at the end of the ticket String add $400.00 to the cost, regardless of the ticket class.

User input should be accepted in both upper and lower case and any combination of the two

Some **sample** runs of the program are shown below (user input is in **bold**):
NOTE: the samples DO NOT cover all of reasons a ticket can be invalid, that is, they do not cover all test cases. Identifying and covering all of the cases is part of your Task.
The samples do not always display appropriate invalid messages in most cases, your assignment submission will display the appropriate messages.

```
> java calcTicket
Enter ticket >> A020
The cost of the ticket is $70.0

> java calcTicket
Enter ticket >> A400Z
If there is only one optional character, it must be one of X, B or K
Z is not a valid character in this position

> java calcTicket
Enter ticket >> T090XB
XB is not valid for 2 optional characters

> java calcTicket
Enter ticket >> Z999
First character must be one of A, C or T only
Z is not a valid first character
> java calcTicket
Enter ticket >> T100XBK
The cost of the ticket is $700.0

> java calcTicket
Enter ticket >> T55
Ticket format is CDDD[C][C][C]
This ticket is the wrong length
```

Note: **`System.exit()`** is not be used anywhere in any of the Tasks.
(Marks will be deducted if it is used anywhere in this assignment.)

An example input file for Task 2 may be copied from the csilib area

don't forget the dot

```
cp /home/student/csilib/cse1oof/SOFassignAB/stars1.dat .
```

don't forget the dot

An example input file for Task 3 may be copied from the csilib area

```
cp /home/student/csilib/cse1oof/SOFassignAB/stars3.dat .
```

These files will also be made available by your lecturer.

Task 2 – Planets1.java

Write a Java program called `Planets1.java` that, firstly, prompts (asks) the user to enter an input file name. This is the name of a text file that can contain any number of records. A record in this file is a single line of text in the following format:

Star name;Alternate star name;HR code

where

Star name, this is a String (text) and is the name of a Star, maybe more than one word and **is immediately followed by a semi-colon (;) and no space**.

Alternate star name, this is a String (text) and is the alternate name of a Star, maybe more than one word and **is immediately followed by a semi-colon (;) and no space**.

HR code consists of the uppercase characters HR followed by a space and then an integer. This is the Harvard Revised catalogue number

An **example** of some of the lines of this file might be:

```
Acamar;Theta 1 Eridani;HR 897
Ain Al Rami;Nu 1 Sagittarii;HR 7116
Aldebaran;Alpha Tauri;HR 1457
Ras Al Muthallah;Alpha Trianguli;HR 544
```

The input file may have 0 to any number of records. **The format of the input file is guaranteed to be correct.** Your program does not have to check the format.

Also, your program must work with any file name of the correct format.
(Do not hard code the file name.)

Once this file has been opened, the user is then prompted for an HR code.

The program then reads through the file. If a matching HR code is found in the file, all the information for that star is displayed to the screen. The order of display is the HR code (surrounded by " "), the name of the star and then the alternate name.

HR codes are unique in the file, so there will be at most only one match.

User entered HR codes must be case insensitive, that is, any combination of uppercase and lowercase letters must give the same result.

If the entire contents of the file has been read and no match is found, then an appropriate message is displayed to the screen.

Some sample runs of the program are included below (user input is in **bold**):
(Note that the sample runs do not necessarily show all the functionality required)

```
> java Planets1
Enter file name >> stars1.dat
Enter HR code >> HR 6746
The planet with HR code "HR 6746" is called Zujj Al Nushshabah
Also known as Gamma 2 Sagittarii

> java Planets1
Enter file name >> stars1.dat
Enter HR code >> HR 15
No planet with HR code "HR 15" was found
```

Task 3 – Planets2.java

Write a Java program called `Planets2.java` that, firstly, prompts (asks) the user to enter an input file name. This is the name of a text file that can contain any number of records (lines).

Each record has the format

Star name;HR code;Star name;HR code

where

Star name, is a String (text) and is the name of a Star, maybe more than one word and **is immediately followed by a semi-colon (;) and no space**.

HR code consists of the uppercase characters HR followed by a space and then an integer and **is immediately followed by a semi-colon (;) and no space then the second Star name immediately followed by a semi-colon (;) and no space, followed by the HR code for the second star**.

As in Task 2, the HR code is the Harvard Revised catalogue number.

The input file may have 0 to any number of records. **The format of the input file is guaranteed to be correct.** Your program does not have to check the format.

Also, your program must work with any file name of the correct format.
(Do not hard code the file name.)

An **example** of an input file is shown below:

```
Alpha 1 Crucis;HR 4730;Epsilon Canis Majoris;HR 2618
Xi Andromedae;HR 390;Nu 1 Sagittarii;HR 7116
Al Minliar Al Ghurab;HR 4623;Aldebaran;HR 1457
```

Once this file has been opened, the program checks if this file is empty (you may assume that the user always enters a valid file name). If the file is empty the program displays an appropriate message to the screen and closes, **without using System.exit( ).**

If the file is not empty, then the user is presented with a menu, as shown below:

```
Main Menu
1. Show Planet Connections
2. Exit
Enter choice >>
```

1. Show Planet Connections

If the user selects this choice, then the program prompts (asks) the user for the name of a Star.
User entered Star names must be case insensitive, that is, any combination of uppercase and lowercase letters must give the same result.

The program then reads the contents of the file, line by line, and displays to the screen all the Stars that connect to the Star entered by the user. To connect means to find the line(s) in the file that contains the Star entered by the user and display the other Star on the line.

Unlike Task 2, this will require that your program reads the entire file. **There may be more than one match for the user entered Star name in the input file.**

If there is no match for the requested Star name, then an appropriate message is displayed to the screen.

Note that you need to consider that the Star name entered by the user may the first or second Star name on the line from the file. If the user entered Star name is matched as the first one on the line, then the second Star name and HR code are displayed to the screen, indented by a tab. If the user entered Star name is matched as the second one on the line then the first Star Name and HR code are displayed to the screen, indented by a tab.

The user is then presented with the menu again. The program continues to loop until the user selects choice 2, when the program closes. You may assume that the user always enters an integer, but, your program must correctly deal with an invalid integer entered by the user.

Some sample runs of the program are included below (user input is in **bold**):
(Note that the sample runs do not necessarily show all the functionality required)

```
> java Planets2
Enter file name >> stars3.dat

Main Menu
1. Show Planet Connections
2. Exit
Enter choice >> 1
Enter planet start name >> Alpha 1 Crucis

Planet "Alpha 1 Crucis" (HR 4730) connects to:
     Epsilon Canis Majoris(HR 2618)
     Algol(HR 936)
     Al Minliar Al Ghurab(HR 4623)
     Becrux(HR 4853)

Main Menu
1. Show Planet Connections
2. Exit
Enter choice >> 6

6 is not a valid choice

Main Menu
1. Show Planet Connections
2. Exit
Enter choice >> 1

Enter planet start name >> Arrakis

Planet "Arrakis" (HR 6370) connects to:
     Becrux(HR 4853)
     Gamma Cephei(HR 8974)

Main Menu
1. Show Planet Connections
2. Exit
Enter choice >> 1

Enter planet start name >> Pern

Pern not a valid planet name

Main Menu
1. Show Planet Connections
2. Exit
Enter choice >> 2

Program is now closing
```

As a final note for this task, consider that your program has to read through the entire file every time the user chooses option 1. So your program has to re-open the file each time option 1 is chosen.

See Lecture/Workshop 2 for revision on the String class.

Task 4 (Part B) – DOTP.java

Write a Java program called `DOTP.java` that defends a planet against asteroid strikes.

You are in charge of Defence Of The Planet, an organization that defends your planet from asteroid strikes.

Asteroids always come from the left or right and follow an angled path down to the surface of your planet. If an asteroid reaches the ground, then the results will catastrophic. You DOTP organization fires special missiles, which if they hit the asteroid before the asteroid reaches the ground, will completely destroy the asteroid.

The area into which the asteroids enter is marked out as a grid on your DOTP screens.
The ground (where you are) is at y coordinate 0 and asteroids are detected at y coordinate 10 (everything is in integer units). The grid extends along the ground -10 to +10 units from a
fixed central point.

If asteroids enter from the right, then their initial coordinates will be x = +5, y = +10 Each time through the loop in your program, the asteroid will move to the left 1 unit and down 1 unit. So after the first iteration, the asteroid will be at x = +4, y = +9, after the second iteration, the asteroid will be at x = +3, y = +8 and so on, until the asteroid is destroyed or the asteroid hits the ground.

If the asteroids comes from the left, then their initial coordinates will be x = -5, y = +10 Each time through the loop in your program, the asteroid will move to the right 1 unit and down 1 unit. So after the first iteration, the asteroid will be at x = -4, y = +9, after the second iteration, the asteroid will be at x = -3, y = +8 and so on, until the asteroid is destroyed or the asteroid hits the ground.

Your planet has some very usual ground conditions which means that base, and the missile, starts at a random position between x = -10 and x = +10 (y, of course = 0, you're on the ground).

After DOTP fires the missile it tracks both the missile and the asteroid, issuing commands to the missile to, hopefully, bring the missile and the asteroid into exactly the same coordinates before the asteroid hits the ground.

As the asteroid is steadily moving across the sky towards the ground, DOTP has only 10 iterations (commands) to direct the missile into the asteroid. If the missile has not destroyed the asteroid after 10 turns, the asteroid hits the ground.

There is one further problem, if your missile goes outside x = -10 or x = +10, then DOTP can no longer track the missile, so DOTP can no longer issue commands. The missile self-destructs, but, of course, the asteroid hits the ground.

Note the missile can still be tracked at x = -10 or x = +10, it is when x = -11 or less or x = +11 or greater that the missile can no longer be tracked.

The missile is capable of being tracked and surviving when its y coordinate = 0, but any command that causes the y coordinate of the missile to become negative, means that the missile has impacted with the ground and blown up - asteroid hits the ground.

There is no limit however to how high (positive y coordinate) at which the missile can still be tracked.

The program starts by assigning a starting position for the asteroid, either left or right. This should be done by generating a random number. How you decide whether the random number means left or right, is up to you. Recall that this means the asteroid starts at either
x = +5, y = +10 or x = -5, y = +10 and determines which way the asteroid travels.

Then the program generates a random number for the x coordinate of the base, and hence the missile. This x coordinate must be between x = -10 to x = +10, the y coordinate is, of course, 0.

The program then displays to the screen the initial position of the asteroid and the missile (base).

The program then prompts (asks) the user to enter a command (explained below). The program calculates the new position (x and y coordinates) of the missile and the new position (x and y coordinates) of the asteroid.

If the x and y coordinates of the missile and the asteroid are the same, then the program displays a success message to the screen and the program ends.

If the updated x and y coordinates (from the previous command) of the missile are not the same as the asteroid, the program loops around and asks for another command for the missile. (This may be the same command as the previous command or a different command) and the program calculates the new x and y coordinates of the missile and the new x and y coordinates of the asteroid, checks to see if the missile has destroyed the asteroid (success message, program ends), the missile hits the ground, goes out of tracking range or the asteroid hits the ground (fail message, program ends) or loops back again for another command.

The missile (user) has a maximum of 10 attempts to destroy the asteroid, else the asteroid hits the ground. Each time a valid command is entered, the calculations made, the check done and the missile has not destroyed the asteroid, 1 is subtracted from the remaining attempts.

Only certain commands are allowed, these are entered as text (**not numbers**)

The allowed commands and the change they make to the x and y coordinates of the missile (subject to the boundary condition above) are:

`up`	y coordinate increased by 2, x coordinate stays the same
`down`	y coordinate decreased by 2, x coordinate stays the same
`left`	y coordinate stays the same, x coordinate decreased by 2
`right`	y coordinate stays the same, x coordinate increased by 2
`left up`	y coordinate increased by 1, x coordinate decreased by 1
`left down`	y coordinate decreased by 1, x coordinate decreased by 1
`right up`	y coordinate increased by 1, x coordinate increased by 1
`right down`	y coordinate decreased by 1, x coordinate increased by 1

For example, if `Left` command is entered, the nett effect would be

```
new value of x = old value of x - 2
new value of y = old value of y  (unchanged)
```

There must be a space between any commands that have more than one word and all commands must be **case insensitive**. This means that **`Down`** or **`DOwn`** or **`down`** should all produce the same result.

Your program must correctly deal with the user entering an invalid command (not one of the commands listed above). If the user enters an invalid command, an appropriate message is displayed to the screen, **but neither the missile nor the asteroid have their x and y coordinates updated, they stay at the same positions.**

To destroy the asteroid, the missile must be at the same coordinates as the asteroid in the same turn.

coordinates are always integers, including the initial coordinates.

Some sample runs of the program are included below (user input is in **bold**):
(Note: not all options, functionality and messages are shown.)

```
> java DOTP
Initial Asteroid position is x: -5 y: 10
Initial Missile position is x: 5 y: 0

Available Missile Commands
     up
     down
     left
     right
     left up
     left down
     right up
     right down
Enter command >> **Up**

Missile is now at x: 5 y: 2

Asteroid is now at x: -4 y: 9

Available Missile Commands
     up
     down
     left
     right
     left up
     left down
     right up
     right down
Enter command >> **LEfT**

Missile is now at x: 3 y: 2

Asteroid is now at x: -3 y: 8



Available Missile Commands
     up
     down
     left
     right
     left up
     left down
     right up
     right down
Enter command >> **left up**

Missile is now at x: 2 y: 3

Asteroid is now at x: -2 y: 7

Available Missile Commands
     up
     down
     left
     right
     left up
     left down
     right up
     right down
Enter command >> **left up**

Missile is now at x: 1 y: 4

Asteroid is now at x: -1 y: 6

Available Missile Commands
     up
     down
     left
     right
     left up
     left down
     right up
     right down
Enter command >> left up

Missile is now at x: 0 y: 5

Asteroid is now at x: 0 y: 5
Congratulations, you have saved the planet !!
Your name will long be remembered
```

Another run of the program

```
> java DOTP
Initial Asteroid position is x: 5 y: 10
Initial Missile position is x: 1 y: 0

Available Missile Commands
     up
     down
     left
     right
     left up
     left down
     right up
     right down
Enter command >> coffee time
That is not a valid command

Available Missile Commands
     up
     down
     left
     right
     left up
     left down
     right up
     right down
Enter command >> down

Missile is now at x: 1 y: -2

Asteroid is now at x: 4 y: 9

You crashed the missile into the ground, now we're all doomed
```

Another run of the program

```
> java DOTP
Initial Asteroid position is x: 5 y: 10
Initial Missile position is x: 7 y: 0

Available Missile Commands
     up
     down
     left
     right
     left up
     left down
     right up
     right down
Enter command >> Up

Missile is now at x: 7 y: 2

Asteroid is now at x: 4 y: 9
```

```
Available Missile Commands
     up
     down
     left
     right
     left up
     left down
     right up
     right down
Enter command >> up

Missile is now at x: 7 y: 4

Asteroid is now at x: 3 y: 8

Available Missile Commands
     up
     down
     left
     right
     left up
     left down
     right up
     right down
Enter command >> left

Missile is now at x: 5 y: 4

Asteroid is now at x: 2 y: 7

Available Missile Commands
     up
     down
     left
     right
     left up
     left down
     right up
     right down
Enter command >> left

Missile is now at x: 3 y: 4

Asteroid is now at x: 1 y: 6

Available Missile Commands
     up
     down
     left
     right
     left up
     left down
     right up
     right down
Enter command >> left

Missile is now at x: 1 y: 4

Asteroid is now at x: 0 y: 5

Available Missile Commands
     up
     down
     left
     right
     left up
     left down
     right up
     right down
Enter command >> right up

Missile is now at x: 2 y: 5

Asteroid is now at x: -1 y: 4

Available Missile Commands
     up
     down
     left
     right
     left up
     left down
     right up
     right down
Enter command >> right down

Missile is now at x: 3 y: 4

Asteroid is now at x: -2 y: 3

Available Missile Commands
     up
     down
     left
     right
     left up
     left down
     right up
     right down
Enter command >> right

Missile is now at x: 5 y: 4

Asteroid is now at x: -3 y: 2

Available Missile Commands
     up
     down
     left
     right
     left up
     left down
     right up
     right down
Enter command >> down

Missile is now at x: 5 y: 2

Asteroid is now at x: -4 y: 1

Available Missile Commands
     up
     down
     left
     right
     left up
     left down
     right up
     right down
Enter command >> left
```

```
Missile is now at x: 3 y: 2

Asteroid is now at x: -5 y: 0
The Asteroid hit the planet, the cockroaches take over
```

Electronic Submission of the Source Code

- Submit all the Java files that you have developed in the tasks 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 SOF calcTicket.java
        submit SOF Planets1.java
        submit SOF Planets2.java
        submit SOF DOTP.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.

You will be informed if there is an alternate submission process.

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 in Week 7

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.

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 for Tasks 2 and 3.

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 2 or 3 Tasks 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.