# Java Programming Exercise

CE152

Assignment 2021

## WARNING AND ADVICE ABOUT POSSIBLE ACADEMIC OFFENCES

Your solutions should be your own unaided work. You can make use of any of the programs from the CE152 lecture notes, support classes and the lab solutions. You may use any features from the Java JDK API including those not covered in CE152.

You must NOT use any third-party classes (e.g. classes that are not provided as part of the Java JDK download). If you use any other sources, you must clearly indicate this as comments in the program, and the extent of the reference must be clearly indicated. For more information, please see the University pages on plagiarism and the Academic Offences Procedures.

***DO NOT COPY PROGRAM CODE FOR THIS ASSIGNMENT FROM ANOTHER STUDENT OR FROM THE INTERNET OR FROM ANY OTHER SOURCES. DO NOT LET OTHER STUDENTS COPY YOUR WORK.***

Deadline: see Faser.

### Submission

The assignment must be submitted via Faser. Your submission must comprise a **single zip file** created by **exporting your project as an archive**. No other files should be included in the zip file.

***You may receive a mark of zero if you fail to submit your solutions by the deadline. Please double check that you are submitting the correct files.***

## Extenuating Circumstances

The standard extenuating circumstances procedures will apply for those who - for circumstances beyond their control - are prevented from submitting work before the deadline or from attending the lab demonstration. Please see the Undergraduate Students' Handbook for the University policies regarding these matters.

## Introduction

The assignment contains Exercise 1 [15%], Exercise 2 [20%], Exercise 3 [40%] and Exercise 4 [20%] each focusing on a set of particular topics covered in the module. An additional [5%] will be awarded for commenting your code, adhering to the good coding practises mentioned in the lecture and submitting your project as an archive exported from your IDE.

Overall, this assignment will constitute 50% of your total grade.

Please add variables and methods only with the specified access modifiers. You may add additional variables and methods you find convenient.

# Exercise 1 [15%]

This exercise involves creating a class with variables, a constructor and methods. Additionally, you will use arrays and control statements.

## Exercise 1A [2%]

Create a class called `Rocket`. This class should have a `private` variable of type `int` array called `payload`.

Additionally create a class called `MainRocket` that contains a `main` method.

## Exercise 1B [2%]

Add a constructor that receives an `int` array as parameter and assigns it to `payload`.

## Exercise 1C [5%]

Add the following methods to `Rocket`:

- A `public` method called `getLaunchWeight`, without parameters that returns an `int` that contains the **sum** of all items in `payload`.
- A `public` method called `getAverageWeight`, without parameters that returns a `double` that contains the **mean** weight of all items in `payload`.
- A `public` method called `getMaxWeight`, without parameters that returns an `int` containing the **maximum** weight of an item in `payload`.
- A `public` method called `getMinWeight`, without parameters that returns an `int` containing the **minimum** weight of an item `payload`.

## Exercise 1D [3%]

Add a `public` method called `printCountdown`, that receives an `int` as parameter and returns `void`. That method should print a countdown from the `int` provided as parameter.

For example if 10 is provided the method should print:

```
10
9
8
7
6
5
4
3
2
1
Lift off!
```

If a parameter smaller than 1 is provided the method should print:

```
Invalid time.
```

## Exercise 1E [3%]

In the `main` method of `MainRocket` generate a random payload with ten elements and weights between 0 and 100 and store them in an `int` array.

Print the content of the array and instantiate a `Rocket` object with this array as parameter.

Print the maximum, mean, minimum and sum of the payload using the corresponding methods in `Rocket`.

For example:

```
Payload: [76, 60, 76, 36, 49, 62, 26, 13, 97, 17]
Max: 97
Mean: 51.2
Min: 13
Sum: 512
```

# Exercise 2 [20%]

This exercise involves overriding methods, implementing `Comparable` and using data structures.

## Exercise 2A [3%]

Create a class called `Astronaut`. This class should have:

- a `private` variable of type `String` called `nationality`
- a `private` variable of type `String` called `rank`
- a `private` variable of type `int` called `rankNum`
- a `private` variable of type `int` called `age`

Additionally, create a class called `Crew` with:

- a `private`, `static` and `final` variable of type `String` array called `RANKS` containing the elements "Commander", "Pilot", "Payload Commander", "Mission Specialist" and "Spaceflight Participant"
- a `private` variable of type `String` array called `nationalities`
- a `private` variable of type `List` for the datatype `Astronaut` called `crew`

Finally, create a class called `MainCrew` containing a `main` method.

## Exercise 2B [3%]

In the `Astronaut` class add a constructor that receives parameters for `nationality`, `rank`, `rankNum` (this should contain the index of the astronaut's rank in the `RANKS` variable in the `Crew` class) and age. Assign the parameters to the corresponding variables of the class.

Override `toString` to return a `String` containing `rank`, `nationality` and age that confirms to the following examples:

```
Pilot (Bolivian, 54)
Spaceflight Participant (Omani, 29)
Payload Commander (Canadian, 36)
```

Implement `Comparable` for the datatype `Astronaut` and override the `compareTo()` method. Sorting should be according to `rankNum`, `nationality` and finally age (all in ascending order).

Finally override the `equals` method to return true if `rank`, `nationality` and age are identical.

## Exercise 2C [4%]

In the `Crew` class, add a constructor that receives a `String` as parameter. The `String` should give the path to the file `nationalities.txt` that you can download from Moodle. This file contains one nationality per line.

The information was obtained from:

https://assets.publishing.service.gov.uk/government/uploads/system/uploads/attachment_data/file/664133/CH_Nationality_List_20171130_v1.csv/preview

Read this file and store it in the array `nationalities`.

Now initialise one `Astronaut` object per element in `RANKS` and `nationalities`. Generate a random age for each `Astronaut` in a range you find suitable for your crews. Set `rankNum` according to the index of the astronaut's rank in `RANKS`. Add each `Astronaut` to the `List crew`. This should produce a `List` with 1125 elements.

- Add a `public` method `printCrew` that returns `void` and prints every `Astronaut` in `crew` to the command line.
- Add a `public` method `sortCrew` that returns `void` and sorts the `crew` `List`
- Add a `public` method `shuffleCrew` that returns `void` and shuffles the `crew` `List`

Finally, add a `public` method `assembleMissionCrew` that returns a `Map` with `String` as key and `Astronaut` as value that contains one `Astronaut` of each rank (so one of each "Commander", "Pilot", "Payload Commander", "Mission Specialist" and "Spaceflight Participant") chosen randomly from the `crew` list. The `String` used as key should be the rank of the `Astronaut`. For example:

```
{Commander=Commander (Albanian, 34),
  Mission Specialist=Mission Specialist (Chinese, 37),
  Payload Commander=Payload Commander (Croatian, 61),
  Pilot=Pilot (Liechtenstein citizen, 25),
  Spaceflight Participant=Spaceflight Participant (Ivorian, 27)}
```

## Exercise 2D [10%]

In the main method of `MainCrew` add code that via the command line:

- Asks the user for a filename to read the nationalities from
- If that filename cannot be found asks the user again until the file can be read and creates a new `Crew` using this file
- Asks the user to enter:

  ```
  Please enter:
  1 to print crew (sorted)
  2 to print crew (shuffled)
  3 to assemble and print mission crew
  quit to quit
  ```

- Calls the corresponding `Crew` methods for if the user enters 1-3 and exits the program if the user enter quit
- Upon any other entry the program should respond with "Invalid entry!"

For example:

```
Please enter crew file name:
ce152
data/ce152 (No such file or directory)
Please enter crew file name:
nationalities
Please enter:
1 to print crew (sorted)
2 to print crew (shuffled)
3 to assemble and print mission crew
quit to quit
3
{Commander=Commander (American, 24),
  Mission Specialist=Mission Specialist (Uruguayan, 22),
  Payload Commander=Payload Commander (Panamanian, 24),
  Pilot=Pilot (Costa Rican, 39),
  Spaceflight Participant=Spaceflight Participant (Salvadorean, 50)}
Please enter:
1 to print crew (sorted)
2 to print crew (shuffled)
3 to assemble and print mission crew
quit to quit
ce152
Invalid entry!
Please enter:
1 to print crew (sorted)
2 to print crew (shuffled)
3 to assemble and print mission crew
```

```
quit to quit
quit
Bye!
```

# Exercise 3 [40%]

For this exercise you need to read a CSV file, store the data in suitable data structures and visualise the data in a `JFrame`.

## Exercise 3A [2%]

Create a class called `MarsData`. This class should have

- a private two dimensional array of type `double` called `arrayOfMars`

Additionally, create a class called `MarsDisplay`. This class should inherit from the `Plot` used in the lecture and have

- a `private` variable of type `MarsData` called `md`

Finally, create a class called `MainMars` with a `main` method.

You may add any other variables, methods or even classes you may find helpful.

## Exercise 3B [8-15%]

In `MarsData` you should read the data from `marsPolarSmall.csv` or `marsPolarMedium.csv`. The difference between the two files is the resolution. Thus, testing your program with the small file will be faster whereas you can create a higher resolution image using the data from the medium file. Nonetheless, you should ensure your program will run on either file without errors. As the names imply there is also a higher resolution file which I can make available upon request.

The files contain coordinates in longitude (-180 to 180), latitude (-90 to 90) and altitude (in meters) of the surface of the planet Mars.

For example:

```
-179.75,-89.75,-1965
-179.75,-89.5,-2011
-179.75,-89.25,-2140
-179.75,-89,-2162
```

In terms of screen coordinates to top left point would be -180, 90 and the bottom right would be 180, -90.

This data was collected during the Mars Global Surveyor Mission and is available from:

https://pds-geosciences.wustl.edu/missions/mgs/megdr.html

You have two options to chose from when storing the data that differ in the marks they award (**you will receive marks only for one option**). Keeping track of the minimum and maximum altitude you encounter will be helpful later in both versions.

### Option 3B.1 [8%]

Implement a method called `readDataN3` that stores each line from the file in a line of `arrayOfMars` and the three values in the second dimension. This will result in an array of dimension N x 3 (hence the name).

If you choose this option you may make `arrayOfMars` public to be able to access it in `MarsDisplay`.

### Option 3B.2 [15%]

Implement a method called `readData2D` that stores each altitude in an array that is longitude by latitude. The challenge is that longitude and latitude are floating point numbers so you cannot directly use them for indexing. Thus you will have to somehow map longitude and latitude values to array indices.

If you choose option 3B.2 you should keep `arrayOfMars` private and access the data through a method called `getAltitude` which returns a `double` containing the `altitude` closest to the longitude and latitude provided to `getAltitude` as parameters using two variables of type `double`.

## Exercise 3C [8-18%]

Visualise the data stored in `MarsData` using one of the following three options (**you will receive marks only for one**).

### Exercise 3C.1 [8%]

Map longitude and latitude to screen coordinates and altitude from black (lowest altitude) to white (highest altitude). For this purpose it is useful to keep track of the minimum and maximum values when reading the data file in `MarsData`. See Figure 1 for an example.

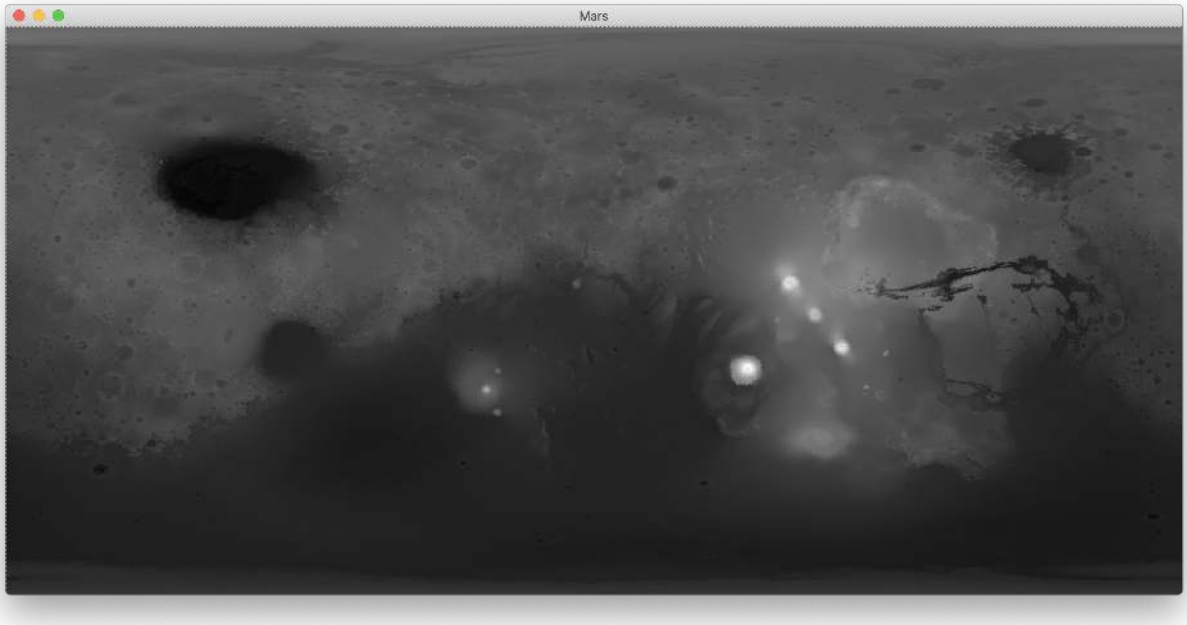


Figure 1: Surface of Mars rendered in gray scale (3C.1)

### Exercise 3C.2 [12%]

Map longitude and latitude to screen coordinates and altitude from dark brown (lowest altitude) to light brown (highest altitude). See Figure 2 for an example. You do not have to match the colour in the example. The criterion is that you do not map to a single channel (red, green or blue), two channels or all three (gray scale).

### Exercise 3C.3 [18%]

Proceed as in 3C.2 but apply a Robinson projection. You can retrieve the required coefficients from the file called `robinson`. You must research yourself how to do this (for example on Wikipedia). See Figure 3 for an example.

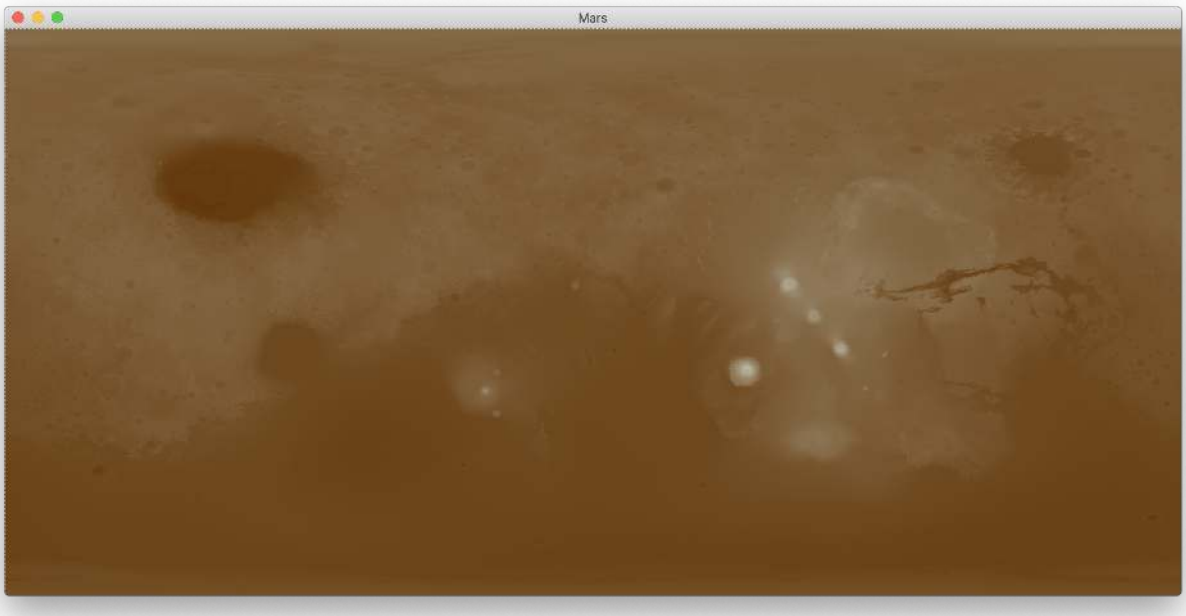


Figure 2: Surface of Mars rendered in colour (3C.2)

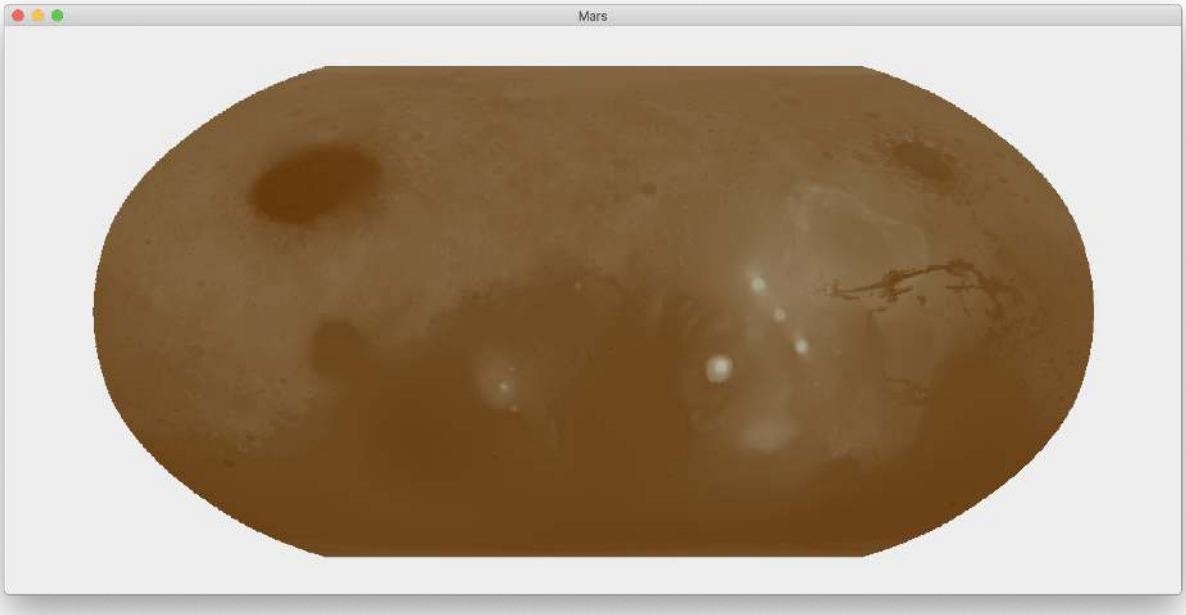


Figure 3: Surface of Mars rendered in colour and using Robinson projection (3C.3)

### Exercise 3D [5%]

In `MainMars` add a `MarsDisplay` to a `JFrame` and display. You can do this even if you do not complete Exercise 3C.

## Exercise 4 [20%]

For this exercise you will need to implement an animation, a key and a mouse listener.

### Exercise 4A [2%]

Create a class called `SolarDisplay`. This class should extend `JComponent`.

Create a class called `MainSolar`. This class should contain a `main` method.

You may add any number of variables you find helpful.

### Exercise 4B [8%]

Override the `paintComponent` method in `SolarDisplay` such that a yellow circle representing the Sun is in the center and a orange circle representing Mercury, a yellow circle representing Venus, a blue circle representing Earth and a red circle representing Mars are placed at distances corresponding to the relative orbits given in the table below.

| Planet | Orbit | Year | Speed |
|---|---|---|---|
| Mercury | 0.387 | 0.2409 | 1.607 |
| Venus | 0.723 | 0.616 | 1.174 |
| Earth | 1.0 | 1.0 | 1.000 |
| Mars | 1.524 | 1.9 | 0.802 |

Values from:

https://www.sjsu.edu/faculty/watkins/orbital.htm

Whenever `paintComponent` is called move the planets in their orbit (which may be perfectly circular) using the relative speeds also provided in the table. The planets may all be the same size and the sun is not required to be to scale.

### Exercise 4C [3%]

In `MainSolar` add `SolarDisplay` to a `JFrame` and display it. Add a `while` loop for the animation.

### Exercise 4D [7%]

Add a `MouseListener` to `SolarDisplay` that pauses the animation when the left mouse button is clicked.

Add a `KeyListener` to `SolarDisplay` that speeds up the animation when the plus (+) key is pressed and slows it down to a minimum of zero when the minus (-) key is pressed.

## Marks for comments and style [5%]

Marks will be awarded for comments, good coding style (for example sensible exception handling) and exporting and submitting your project as a zip file.

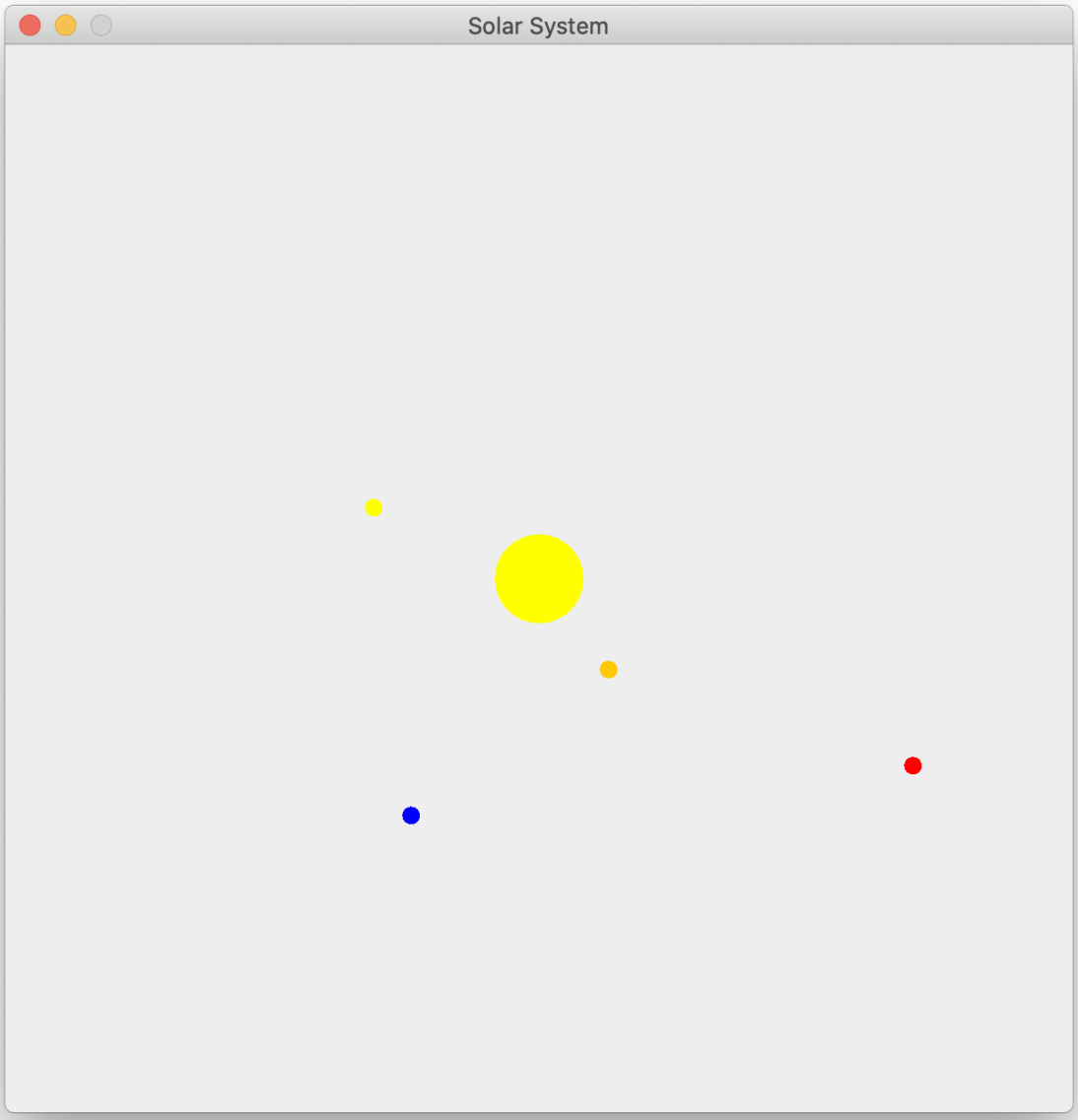


Figure 4: Example of the solar system animation