Computer Science 2A

Practical Assignment 04

Assignment date:	2020-03-14
Deadline	2020-03-24 10h00

Marks: 115

This practical assignment must be uploaded to eve.uj.ac.za **before** 2020-03-24 10h00. Late[1] or incorrect submissions **will not be accepted**, and will therefore not be marked. You are **not allowed to collaborate** with any other student.

Good coding practices include a proper coding convention and a good use of documentation. Marks will be deducted if these are not present. Every submission **must** include a batch file unless stated otherwise.

The **reminder page** includes details for submission. Please ensure that **ALL** submissions follow the guidelines. The reminder page can be found on the last page of this practical.

This practical aims to introduce Advanced Object Orientation and Binary IO.

As previously discussed, your department at the **Milky Way Space Communication Board (MWSCB)** [2] will need to validate **`Message`**s. However, messages come in three categories, that is **`SOSMessage`**s, **`EncryptedMessage`**s and **`NormalMessage`**s. Each **`Message`** has the following properties:

- ID of type String
- Contents of type String
- SourcePlanet (Enum)
- DestinationPlanet (Enum)
- MessageType (Enum)

The three **`Message`** class categories have the following additional properties:

`SOSMessage`:

- Recipient (Enum - GOVERNMENT | PUBLIC)

`EncryptedMessage`:

- Public Key of type String (length of Public Key should be longer than 10 characters)

[1] Alternate arrangements for exceptional circumstances will been posted on eve.

[2] Disclaimer - This series of problem statements are a work of fiction. Names, characters, businesses, places, events and incidents are either the products of the author's imagination or used in a fictitious manner. Any resemblance to actual persons, living or dead, or actual events is purely coincidental.

`NormalMessage`:

- MESSAGE_LENGTH an integer

The **`Ship`** carries a **`Message`**s array. Additionally, each **`Employee`** will need to provide their details for validation before they can access **`Ship`** information. The **`Employee`** class has the following properties:

- EmployeeID of type String (with a minimum of 6 characters)
- FirstName of type String
- LastName of type String
- shipData of type **`Ship`**

Included in the library file **(jar)** is an Interface file with a method **validate()** that should be implemented differently by each **`Message`** category and **`Employee`** class. For a valid instance of,

- **`Employee`**, their **EmployeeID** should be of minimum length 6
- **`SOSMessage`**, the type of **Recipient** should be of either type provided in **jar** file. (GOVERNMENT | PUBLIC)
- **`EncryptedMessage`**, the **public Key** should be of length greater than 10
- **`NormalMessage`**, the **contents** length should be less than or equal to **MESSAGE_LENGTH** property

In a this practical, use the provided **MWSCB.jar** file and files. You are provided with two binary files inside of the **data** folder. One binary file contains **`Ship`** data and the other **`Message`**s that the Ship is carrying.

Create a **`DataReader`** class in the `acsse.csc2a.file` package with the following methods:

- A **readShip** method that takes in two parameters, a File that contains the **`Ship`** file name, and a File that contains the **`Message`**s file name. Reads the files and returns a Ship instance and respective messages.

The text files are structured as follows:

Ship File format

```
// SHIP_ID is a string
// SHIP_NAME is a string and can contain spaces
SHIP_ID SHIP_NAME
```

Message File format

```
// MESSAGE_TYPE (SOSMessage | EncryptedMessage | NormalMessage)
// ADDITIONAL_PROPERTY (Property specific to MESSAGE_TYPE)
ID CONTENTS SOURCE_PLANET DEST_PLANET MESSAGE_TYPE ADDITIONAL_PROPERTY
```

Create the respective classes in the `acsse.csc2a.model` package:

- A concrete implementation of the **validate** method in the respective classes.
- **printMessages** method in the **`Employee`** class to print out the **`Ship`** and **`Message`** details in a structured, readable format

- **sendMessages** method in the **`Employee`** class that validates the **`Message`**s (polymorphism) and returns **False** if any **`Message`** validation fails

To test the application, in the **`Main`**

- First create an **`Employee`** instance
  - Validate and print out a Success or Failure Message
  - Expected Output is Success
- Make use of the **readShip** method
- Make use of the **`Employee`** to call **printMessages** method
- Make use of the **`Employee`** to call **sendMessages** method
  - Validate and print out a Success or Failure Message
  - Expected Output is Success

Marksheet

1. Updated UML class diagrams for all classes. **[15]**
2. `DataReader`
   (a) `Ship` instance **[09]**
   (b) `Message`s array **[10]**
   (c) Error Handling **[10]**
3. `SOSMessage`
   (a) Implements *`validate`* method **[02]**
4. `EncryptedMessage`
   (a) Implements *`validate`* method **[02]**
5. `NormalMessage`
   (a) Implements *`validate`* method **[02]**
6. `Employee`
   (a) *`sendMessages`* method **[05]**
   (b) *`printMessages`* method **[05]**
   (c) Implements *`validate`* method **[02]**
7. `Main`
   (a) Create `Employee` and correct output **[02]**
   (b) Create `Ship` using *`readShip`* method **[02]**
   (c) *`printMessages`* with correct output **[02]**
   (d) Successfully send `Message`s using *`sendMessages`* **[02]**
8. Packages **[05]**
9. Coding convention (structure, layout, OO design) **[05]**
10. Commenting (normal and JavaDoc commenting) **[05]**
11. Correct execution (if it doesn't run from your batch file you get 0) **[30]**

NB

Submissions which **do not compile** will be capped at 40%!

Practical marks are awarded subject to the student's ability to explain the concepts and decisions made in preparing the practical assignment solution. (Inability to explain code = inability to be given marks.)

Execution marks are awarded for a correctly functioning application and not for having related code.

Reminder

Your submission must follow the naming convention below.

```
SURNAME_INITIALS_STUDENTNUMBER_SUBJECTCODE_YEAR_PRACTICALNUMBER
```

Example

Surname	Berners-Lee	**Module Code**	CSC02A2
Initials	TJ	**Current Year**	2023
Student number	209912345	**Practical number**	P04

```
Berners-Lee_TJ_209912345_ CSC02A2_2023_P04
```

Your submission must include the following folders:

Folder	**State**	**Purpose**
`bin`	*Required*	Should be empty at submission but will contain runnable binaries when your submission is compiled.
`docs`	*Required*	Contains the batch file to compile your solution, UML diagrams, and any additional documentation files. All files must be in **PDF** format. Your details must be included at the top of any **PDF** files submitted. **Do not include generated JavaDoc.**
`src`	*Required*	Contains all relevant source code. Source code must be places in relevant sub-packages! Your details must be included at the top of the source code.
`data`	*Optional*	Contains all data files needed to run your solution.
`lib`	*Optional*	Contains all libraries needed to compile and run your solution.

NB

Every submission **must** include a batch file that contains commands which will:

- Compile your Java application source code.
- Compile the associated application JavaDoc.
- Run the application.

Do not include generated JavaDoc in your submission. All of the classes/methods which were created/updated need to have JavaDoc comments.

Multiple uploads

Note that only **one** submission is marked. If you already have submitted once and want to upload a newer version then submit a newer file with the same name as the uploaded file in order to overwrite it.