We Keep Secrets

Object Oriented Programming
1st Project, version 1.6 – 2019-04-06 @ 18h00
Contact: carla.ferreira@fct.unl.pt

Important remarks

Deadline until 23h59 (Lisbon time) of **April 21, 2020.**

Team this project is to be MADE BY GROUPS OF 2 STUDENTS.

Deliverables: Submission and acceptance of the source code to **Mooshak** (score > 0). See the course web site for further details on how to submit projects to Mooshak.

Recommendations: We value the documentation of your source code, as well as the usage of the best programming style possible and, of course, the correct functioning of the project. Please carefully comment both interfaces and classes. The documentation in classes can, of course, refer to the one in interfaces, where appropriate. Please comment the methods explaining what they mean and defining preconditions for their usage. Students may and should discuss any doubts with the teaching team, as well as discuss this project with other students, but may not share their code with other colleagues. This project is to be conducted with full respect for the Code of Ethics available on the course web site.

1 Development of the application We Keep Secrets

1.1 Problem description

The goal of this project is to develop an application that manages documents with security concerns for a governmental organisation. To distinguish between documents that have no security constrains from those with sensitive information, documents are classified in two groups: *official* documents that are public and have no security constraints; and, *classified* documents that have an associated security level (*confidential*, *secret*, or *topsecret*) that describes the level of sensitivity of the document's information. To avoid information leakages, application users have an associated security clearance level that restricts the documents a user may manage and access. The application ensures that users only can manage and access documents with a security level equal to or lower than its own. However, this restriction might be too strong as in some situations users might need access to documents above their security clearance level. For these situations, a grant can be used to give a user access to a document she does not have security clearance. These grants are not permanent and can be revoked at any time. For auditing purposes and to detect potential information leaks, the application maintains information about all the accesses to classified documents, and grants given and revoked. The application also maintains information about the last 10 accesses to official documents.

For a registered user the application maintains a unique identifier (id), a security level, and a list of uploaded documents. The security level of a user restricts the documents that she can upload (manage) and access, as it will be explained in more detail later. Clerks are users that can only manage and access *official* documents, which are unclassified documents. Officers are users that manage and access a subset of documents according to their security clearance level. Security levels for classified documents are, in increasing order of security, *confidential*, *secret*, *topsecret*. An officer can manage and access documents with a security level equal to or lower than its own. For instance, an officer with a *topsecret* security

level may access all documents in the application and manage documents with any security level. While an officer with a *confidential* security level may only access and manage documents with security level *official* or *confidential*. As it was mentioned before, an officer managing a document can grant access to other officers with lower security clearance. These grants can later be revoked.

Each document has a manager (the user that uploaded the document), a name, a security level (*official*, *confidential*, *secret*, or *topsecret*), and the number of accesses. Official documents keep the last 10 users that have read it (this kind of documents can only be read). Classified documents keep track of all accesses, namely which user has accessed it and the type of access (*read* or *write*). Moreover, classified documents maintain which grants were issued and revoked.

2 Commands

In this section we present all the commands that the system must be able to interpret and execute. In the following examples, we differentiate `text written by the user` from the feedback written by the program in the console. You may assume that the user will make no mistakes when using the program other than those described in this document. In other words, you only need to take care of the error situations described here, in the exact same order as they are described.

Commands are case insensitive. For example, the `exit` command may be written using any combination of upper and lowercase characters, such as `EXIT`, `exit`, `Exit`, `exIT`, and so on. In the examples provided in this document, the symbol ↵ denotes a change of line. Whenever a command is expected from the user, the program should present a *prompt* denoted by the symbol >, which is always followed by a white space. The user input is written right after that white space, in the same line.

If the user introduces an unknown command, the program must write in the console the message Unknown command. Type help to see available commands. For example, the non existing command `some random command` would have the following effect:

> `some random command`↵
> Unknown command. Type help to see available commands.↵

Several commands have arguments. You may assume that the user will only write arguments of the correct type. However, some of those arguments may have an incorrect value. For that reason, we need to test each argument exactly by the order specified in this document. Arguments will be denoted **`with this style`**, in their description, for easier identification.

2.1 `exit` command

Terminates the execution of the program. This command does not require any arguments. The following scenario illustrates its usage.

> `exit`↵
> Bye!↵

This command always succeeds. When executed, it terminates the program execution.

2.2 `help` command

Shows the available commands. This command does not require any arguments. The following scenario illustrates its usage.

```
> help↵
register - registers a new user↵
listusers - list all registered users↵
upload - upload a document↵
read - read a document↵
write - write a document↵
grant - grant access to a document↵
revoke - revoke a grant to access a document↵
userdocs - list the official or classified documents of an user↵
topleaked - list the top 10 documents with more grants↵
topgranters - list the top 10 officers that have given more grants↵
help - shows the available commands↵
exit - terminates the execution of the program↵
```

This command always succeeds. When executed, it shows the available commands.

2.3 `register` command

The command receives 3 parameters: its **`kind`** (either **`clerk`** or **`officer`**), the new **`user id`**, and its clearance level. For a clerk the clearance level has to be **`official`**, while for an officer the clearance level can either be **`confidential`**, **`secret`**, or **`topsecret`**. It can be assumed that the clearance level is always adequate when registering a user.
In case of success, the program presents the feedback message User <user id> was registered. Note that clerks can only upload documents with security level **`official`**, so this argument is unnecessary. However, it is used in order to maintain the same number of arguments for registering both kinds of users.
Suppose you want to create four users named *knowsNothing*, *knowsSomeStuff*, *knowsSecrets*, and *knowsTopSecrets*.

```
> register↵
clerk knowsNothing official↵
User knowsNothing was registered.↵
> register↵
officer knowsSomeStuff confidential↵
User knowsSomeStuff was registered.↵
> register↵
officer knowsSecrets secret↵
User knowsSecrets was registered.↵
> register↵
officer knowsTopSecrets topsecret↵
User knowsTopSecrets was registered.↵
```

If some parameter is incorrect, the user is not registered and an adequate error message is presented:

1. If there is already a registered user with the same identifier, the error message is (Identifier <user id> is already assigned to another user.).

The following example illustrates an interactive session where this error message would be generated. The problem with the input is highlighted in **red**.

```
> register↵
clerk knowsNothing official↵
User knowsNothing was registered.↵
> register↵
officer knowsNothing confidential↵
Identifier knowsNothing is already assigned to another user.↵
```

2.4 `listusers` command

Lists registered users. This command does not require any parameters and always succeeds. It lists all registered users in the order by which they were added. For each user it presents its kind, its identifier, and its security clearance level.
Suppose you want to create four users named *knowsNothing*, *knowsSomeStuff*, *knowsSecrets*, and *knowsTopSecrets* as shown for `register` command. The output of the `listusers` command would be as follows:

```
> listusers↵
clerk knowsNothing official↵
officer knowsSomeStuff confidential↵
officer knowsSecrets secrets↵
officer knowsTopSecrets topsecret↵
```

If there are no registered user, the program presents the message There are no registered users.

```
> listusers↵
There are no registered users.↵
```

2.5 `upload` command

Uploads a new document. The command receives 4 parameters: the new **`document name`**, the **`user id`** of person managing the document, the **`security level`** of the document (**`official`**, **`confidential`**, **`secret`**, or **`topsecret`**), and the description. In case of success, the program presents the feedback message Document <document name> was uploaded.
In the following example, users *knowsNothing*, *knowsSecrets*, and *knowsTopSecrets* upload several documents.

```
> listusers↵
clerk knowsNothing official↵
officer knowsSomeStuff confidential↵
officer knowsSecrets secret↵
officer knowsTopSecrets topsecret↵
> upload↵
officialDoc knowsNothing official↵
This document is an official document!↵
Document officialDoc was uploaded.↵
> upload↵
anotherOfficialDoc knowsTopSecrets official↵
This document is another official document...↵
Document anotherOfficialDoc was uploaded.↵
> upload↵
confidentialDoc knowsSecrets confidential↵
This document is a confidential document!↵
Document confidentialDoc was uploaded.↵
> upload↵
topSecretDoc knowsTopSecrets topsecret↵
This document is a topsecret document!↵
Document topSecretDoc was uploaded.↵
```

If some parameter is incorrect, the program is unable to upload the document:

1. The document's manager is not registered. The adequate error message is Not a registered user.
2. The author already has another document with the same name. The adequate error message is Document <name> already exists in the user account.
3. If the user has a lower security clearance than the document security level. The adequate error message is Insufficient security clearance.

The following scenario illustrates these issues:

```
> upload↵
anotherDoc bob confidential↵
There is something wrong with this command...↵
Not a registered user.↵
> upload↵
officialDoc1 knowsSecrets official↵
This document is an official document.↵
Document officialDoc1 was uploaded.↵
> upload↵
officialDoc1 knowsSecrets official↵
This document is an official document.↵
Document officialDoc1 already exists in the user account.↵
> upload↵
anotherTopSecretDoc knowsSecrets topsecret↵
There is something wrong with this command...↵
Insufficient security clearance.↵
```

2.6 `write` command

Updates the document (description). The command receives the following parameters: the **`document name`**, the **`user id`** of the user managing the document, the **`user id`** of the user updating the document, and the updated description. In case of success, the program presents the feedback message Document <document name> was updated.
In the following example a document is uploaded and updated twice.

```
> upload↵
confidentialDoc knowsSecrets confidential↵
This document is a confidential document!↵
Document confidentialDoc was uploaded.↵
> write↵
confidentialDoc knowsSecrets knowsSecrets↵
This document is a confidential document that was updated!↵
Document confidentialDoc was updated.↵
> write↵
confidentialDoc knowsSecrets knowsTopSecrets↵
This document is a confidential document that was updated twice!↵
Document confidentialDoc was updated.↵
```

If some parameter is incorrect, the program is unable to update the document:

1. The manager of the document or the user trying to write the document are not registered. The adequate error message is Not a registered user.
2. The document does not exist in the user account. The adequate error message is Document <name> does not exist in the user account.
3. The document is an official document. The adequate error message is Document <name> cannot be updated.
4. If the user has a lower security clearance than the document security level. The adequate error message is Insufficient security clearance.

The following scenario illustrates these issues:

```
> write↵
confidentialDoc knowsSecrets alice↵
There is something wrong with this command...↵
Not a registered user.↵
> write↵
confidentialDoc bob knowsSecrets↵
There is something wrong with this command...↵
Not a registered user.↵
> write↵
anotherOfficialDoc knowsSecrets knowsSecrets↵
There is something wrong with this command...↵
Document anotherOfficialDoc does not exist in the user account.↵
> upload↵
officialDoc knowsNothing official↵
This document is an official document from clerk knowsNothing!↵
Document officialDoc was uploaded.↵
> write↵
officialDoc knowsNothing knowsTopSecrets↵
There is something wrong with this command...↵
Document officialDoc cannot be updated↵
> write↵
topSecretDoc knowsTopSecrets knowsNothing↵
There is something wrong with this command...↵
Insufficient security clearance.↵
```

2.7 `read` command

Read a document (description). The command receives the following parameters: the **document name**, the **user id** of the user managing the document, and the **user id** of the user reading the document. In case of success, the program presents the feedback message Document: <description> that includes the document description.
In the following example, users *knowsNothing*, *knowsSecrets*, and *knowsTopSecrets* upload and read several documents.

```
> upload↵
officialDoc knowsNothing official↵
This document is an official document!↵
Document officialDoc was uploaded.↵
> upload↵
confidentialDoc knowsSecrets confidential↵
This document is a confidential document!↵
Document confidentialDoc was uploaded.↵
> read↵
officialDoc knowsNothing knowsSecrets↵
Document: This document is an official document!↵
> read↵
confidentialDoc knowsSecrets knowsTopSecrets↵
Document: This document is a confidential document!↵
```

If some parameter is incorrect, the program is unable to read the document:

1. The manager of the document or the user trying to read the document are not registered. The adequate error message is Not a registered user.
2. The document does not exist in the user account. The adequate error message is Document <name> does not exist in the user account.
3. If the user has a lower security clearance than the document security level and was not granted access to the document. The adequate error message is Insufficient security clearance.

The following scenario illustrates these issues:

```
> read↵
confidentialDoc knowsSecrets alice↵
Not a registered user.↵
> read↵
confidentialDoc bob knowsSecrets↵
Not a registered user.↵
> read↵
anotherOfficialDoc knowsSecrets knowsSecrets↵
Document anotherOfficialDoc does not exist in the user account.↵
> read↵
confidentialDoc knowsSecrets knowsNothing↵
Insufficient security clearance.↵
```

2.8 `grant` command

A document manager grants access to another user with insufficient security clearance. Users can only access documents within their security clearance. However, officers that manage secret and top secret documents can grant read access to other officers with lower security clearance. Note that if a grant that has been revoked, a new grant can be issued again. The command receives 3 parameters: the **`document name`**, the **`id`** of the officer managing the document and granting the access, and the **`id`** of the officer to be granted access. In case of success, the program presents the feedback message Access to document <document name> has been granted.
The following example illustrates how an officer that manages a document can grant access to another officer with insufficient security clearance.

```
> listusers↵
officer knowsSomeStuff confidential↵
officer knowsSecrets secrets↵
officer knowsTopSecrets topsecret↵
> upload↵
topSecretDoc knowsTopSecrets topsecret↵
This document is a TOP secret document!↵
Document topSecretDoc was uploaded.↵
> upload↵
secretDoc knowsSecrets secret↵
This document is a secret document!↵
Document secretDoc was uploaded.↵
> grant↵
secretDoc knowsSecrets knowsSomeStuff↵
Access to document secretDoc has been granted.↵
> read↵
secretDoc knowsSecrets knowsSomeStuff↵
Document: This document is a secret document!↵
> grant↵
topSecretDoc knowsTopSecrets knowsSomeStuff↵
Access to document topSecretDoc has been granted.↵
> read↵
topSecretDoc knowsTopSecrets knowsSomeStuff↵
Document: This document is a TOP secret document!↵
```

If some parameter is incorrect, the program is unable to grant access to the document:

1. The manager of the document or the officer being granted access are not registered. The adequate error message is Not a registered user.

2. The manager of the document or the user requesting access is a clerk. The adequate error message is Grants can only be issued between officers.

3. The document does not exist in the officer account. The adequate error message is Document <name> does not exist in the user account.

4. If the officer requesting access already has security clearance to access the document, either by its own security level or a previous grant. The adequate error message is Already has access to document <name>.

The following scenario illustrates these issues:

```
> listusers↵
clerk knowsNothing official↵
officer knowsSomeStuff confidential↵
officer knowsSecrets secrets↵
officer knowsTopSecrets topsecret↵
> grant↵
secretDoc knowsSecrets alice↵
Not a registered user.↵
> grant↵
secretDoc bob knowsSomeStuff↵
Not a registered user.↵
> grant↵
secretDoc knowsSecrets knowsNothing↵
Grants can only be issued between officers.↵
> grant↵
anotherSecretDoc knowsSecrets knowsSomeStuff↵
Document anotherSecretDoc does not exist in the user account.↵
> grant↵
secretDoc knowsSecrets knowsTopSecrets↵
Already has access to document secretDoc.↵
> grant↵
topSecretDoc knowsTopSecrets knowsSomeStuff↵
Access to document topSecretDoc has been granted.↵
> grant↵
topSecretDoc knowsTopSecrets knowsSomeStuff↵
Already has access to document topSecretDoc.↵
```

2.9 `revoke` command

A document owner revokes a previous given grant. The command receives 3 parameters: the **`document name`**, the **`id`** of the officer managing the document, and the **`id`** of the officer which was previously given a grant access. In case of success, the program presents the feedback message Access to document <document name> has been revoked.
The following example illustrates how an officer that manages a document can revoke a grant.

```
> listusers↵
officer knowsSomeStuff confidential↵
officer knowsSecrets secrets↵
> upload↵
secretDoc knowsSecrets secret↵
This is a secret document!↵
Document secretDoc was uploaded.↵
> grant↵
secretDoc knowsSecrets knowsSomeStuff↵
Access to document secretDoc has been granted.↵
> read↵
secretDoc knowsSecrets knowsSomeStuff↵
Document: This is a secret document!↵
> revoke↵
secretDoc knowsSecrets knowsSomeStuff↵
Access to document secretDoc has been revoked.↵
> read↵
secretDoc knowsSecrets knowsSomeStuff↵
Insufficient security clearance.↵
```

If some parameter is incorrect, the program is unable to revoke the access grant:

1. The manager of the document or the user being revoked the grant are not registered. The adequate error message is Not a registered user.
2. The manager of the document or the user being revoked the grant is a clerk. As these users cannot issue or receive grants, the adequate error message is Grants can only be issued between officers.
3. The document does not exist in the officer account. The adequate error message is Document <name> does not exist in the user account.
4. The access grant does no exist, so it cannot be revoked. The adequate error message is Grant for officer <id> does not exist.
5. The grant was already revoked. The adequate error message is Grant for officer <id> was already revoked.

The following scenario illustrates these issues:

```
> listusers↵
clerk knowsNothing official↵
officer knowsSomeStuff confidential↵
officer knowsSecrets secrets↵
officer knowsTopSecrets topsecret↵
> revoke↵
secretDoc knowsSecrets alice↵
Not a registered user.↵
> revoke↵
secretDoc bob knowsSomeStuff↵
Not a registered user.↵
> revoke↵
secretDoc knowsSecrets knowsNothing↵
Grants can only be issued between officers.↵
> revoke↵
anotherSecretDoc knowsSecrets knowsSomeStuff↵
Document anotherSecretDoc does not exist in the user account.↵
> revoke↵
topSecretDoc knowsTopSecrets knowsSomeStuff↵
Grant for officer knowsSomeStuff does not exist.↵
> grant↵
topSecretDoc knowsTopSecrets knowsSomeStuff↵
Access to document topSecretDoc has been granted.↵
> revoke↵
topSecretDoc knowsTopSecrets knowsSomeStuff↵
Access to document secretDoc has been revoked.↵
> revoke↵
topSecretDoc knowsTopSecrets knowsSomeStuff↵
Grant for officer knowsSomeStuff was already revoked.↵
```

2.10 `userdocs` command

Lists the official or classified documents of a user. The command receives as arguments the user **`id`** and the **`type`** of documents to list (**`official`** or **`classified`**) In the following example, user *knowsNothing* has uploaded 3 documents which have multiple read accesses. For each document the program presents its name, the number of accesses, followed by the ids and security levels of the last 10 users that have read the document, with the more recent accesses shown first. The documents should be presented in the order they where uploaded. Note that in the case a document has no accesses, the message shown is There are no accesses.
In the following example, user *knowsNothing* has uploaded 3 documents which have multiple accesses.

```
> userdocs↵
knowsNothing official↵
officialDoc1 0: There are no accesses.↵
officialDoc2 5: alice [secret], bob [confidential], alice [secret], jane [official], knowsNothing [official]↵
officialDoc3 4: knowsNothing [official], bob [confidential], alice [secret], jane [official]↵
```

In the following example, user *knowsTopSecrets* has uploaded 2 documents which have multiple read and write accesses, and grant and revoke actions. For each document, it presents 3 lines: the first line presents its name, its security level, and the number of accesses. The second line presents all accesses to the document, where for each access it is shown the user id, its security level, and the type of access (**`read`** or **`write`**), with the more recent accesses shown last. The third line presents the grants given and revoked. For each grant, it should be shown the user id and its security level, with the more recent actions shown last. Again, the documents should be presented in the order they where uploaded. If a document has no associated grants, the following message should be shown There are no grants. Moreover, in the case a document has no accesses, the message shown is There are no accesses.

```
> userdocs↵
knowsTopSecrets classified↵
secretDoc secret 3↵
alice [secret, read], alice [secret, write], bob [confidential, read]↵
bob [confidential, grant], john [confidential, grant], bob [confidential, revoked]↵
topSecretDoc topsecret 2↵
007 [topsecret, read], M [topsecret, write]↵
There are no grants.↵
anotherSecretDoc secret 0↵
There are no accesses.↵
There are no grants.↵
```

If the user has no documents of the type being listed, the program presents the message There are no <type> documents.

```
> userdocs↵
knowsSomeSecrets official↵
There are no official documents.↵
> userdocs↵
userdocs knowsSecrets classified↵
There are no classified documents.↵
```

If some parameter is incorrect, the program is unable to upload the document:

1. The user is not registered. The adequate error message is Not a registered user.
2. If the user has a lower security clearance than the given security level given as argument. The adequate error message is Inappropriate security level.

The following scenario illustrates these issues:

```
> userdocs↵
alice classified↵
Not a registered user.↵
 > userdocs↵
knowsNothing classified↵
Inappropriate security level.↵
```

2.11 `topleaked` command

Lists the top 10 documents what where "leaked". In here it is assumed that a document was "leaked" if access was granted to lower security officers. This command does not require any parameters and always succeeds. It lists the documents by decreasing order of the number of access grants. For each document it presents the document name, the user id, the security level, the number of accesses, and the number of grants given and revoked. Ties are addressed using the alphabetic order of the documents name. If the number of documents with grants is lower than 10, the program should only present documents with at least one grant. For instance, if 200 documents were uploaded and of these only 5 documents have grants, then the output of the command will just present these 5 documents.
The following example illustrates the command where only 4 documents have security clearance grants.

```
> topleaked↵
leakedALotDoc bob secret 200 5 1↵
shareWithFriendsDoc alice secret 100 5 2↵
importantInfoDoc eve secret 1000 2 0↵
shouldNotBeDisclosedDoc bob topsecret 3 1 1↵
```

If there are no documents with security clearance grants, the program presents the message There are no leaked documents.

```
> topleaked↵
There are no leaked documents.↵
```

2.12 `topgranters` command

Lists the top 10 officers with more grants given to lower security officers. This command does not require any parameters and always succeeds. It lists the officers by decreasing order of access grants given. Ties are addressed using the alphabetic order of the officer id. For each user it presents the user id, its clearance security level, the number of documents, grants given, and grants revoked.
If the number of officers that have given grants is lower than 10, the program should only present officers with associated security clearance grants. The following example illustrates the command where only 3 officers have issued grants to their documents.

```
> topgranters↵
bob secret 4 3 0↵
alice topsecret 6 2 2↵
eve secret 4 1 1↵
```

If no users have assigned grants to their documents, the program presents the message No officer has given grants.

```
> topgranters↵
No officer has given grants.↵
```

3 Developing this project

Your program should take the best advantage of the elements taught up until now in the Object-Oriented programming course. You should make this application as **extensible as possible** to make it easier to add, for instance, new kinds of documents and users. You should build your program so that changes like this one would require as little changes as possible to the rest of your code.

You can start by developing the main user interface of your program, clearly identifying which commands your application should support, their inputs and outputs, and preconditions. Then, you need to identify the entities required for implementing this system. Carefully specify the **interfaces** and **classes** that you will need. You should document their conception and development using a class diagram, as well as documenting your code adequately, with Javadoc.

It is a good idea to build a skeleton of your `Main` class, to handle data input and output, supporting the interaction with your program. In an early stage, your program will not really do much. Remember the **stable version rule**: do not try to do everything at the same time. Build your program incrementally, and test the small increments as you build the new functionalities in your new system. If necessary, create small testing programs to test your classes and interfaces.

Have a careful look at the test files, when they become available. You should start with a really bare bones system with the `help` and `exit` commands, which are good enough for checking whether your commands interpreter is working well, to begin with. Then, you register users, and the user listing operation, so that you can create users and then check they are ok. So far, users don't have any documents. The next step is to start uploading documents. And so on. Step by step, you will incrementally add functionalities to your program and test them. **Do not try to make all functionalities at the same time. It is a really bad idea.**

In this project you will handle several collections. These offer you several opportunities for reusing code, rather than repeating it over and over again. Last, but not the least, **do not underestimate the effort for this project**.

4 Submission to Mooshak

To submit your project to Mooshak, please register in the Mooshak contest POO1920-TP1 and follow the instructions that will be made available on the Moodle course website.

4.1 Command syntax

For each command, the program will only produce one output. The error conditions of each command have to be checked in the exact same order as described in this document. If one of those conditions occurs, you do not need to check for the other ones, as you only present the feedback message corresponding to the first failing condition. For example, if you attempt to upload a document of a user that doesn't exist, the remaining conditions do not need to be checked. However, the program does need to consume all the remaining input parameters, even if they are to be discarded.

4.2 Tests

The Mooshak tests verify incrementally the implementation of the commands. They will be made publicly available on April 3. When the sample test files become available, use them to test what you already have implemented, fix it if necessary, and start submitting your partial project to Mooshak. Do it from the start, even if just implemented the exit and help commands. By then you will probably have more than those to test, anyway. Good luck!