Timed Lab 2
Polymorphism, Exceptions, and the CIA

Thomas Shields

1 Problem Description

You've recently been hired by the CIA to write some code for their new mission- managing system. Using the provided code, you will be writing a particular type of mission - a secret mission. You will only be modifying and submitting the provided `SecretMission.java` and `AccessDeniedException.java`.

Your boss lays out the specifications for you as follows:

1. **`SecretMission`**
   A `SecretMission` *is-a* `Mission` with the following special behavior:
   (a) Secret missions have two states - locked and unlocked. Secret missions behave differently when locked and unlocked. They default to locked.
   (b) Secret missions have a security clearance needed to view their information. We've provided an enumerated type for you to represent this.
   (c) When asked to change their description, if the secret mission is locked, the mission should refuse to change the description (i. e. do nothing). If the mission is unlocked, set the description as normal.
   (d) A `SecretMission` refuses to give up information. When asked for a String representation, if the mission is locked, it returns `CLASSIFIED_STRING`, a constant in the `SecretMission` class. If it is unlocked, instead return the default String representation (w/ title and description).
   (e) Since most secret missions will return `CLASSIFIED_STRING`, a secret mission should also provides the ability for someone to `unlockInfo` on it by passing in a `SecurityClearance`. If the clearance is at least as high as the mission's required clearance, the status of the mission is set to unlocked. Otherwise, it throws an `AccessDeniedException`.

2. **`AccessDeniedException`**
   (a) The `AccessDeniedException` should be a checked exception.
   (b) Each instance of `AccessDeniedException` should have a corresponding `SecurityClearance` - this should be given to it when it is created, and should be the clearance that was required to grant access. For example, if you attempted to unlock a locked mission using a `SecurityClearance.CONFIDENTIAL` but the mission required `SecurityClearance.TOP_SECRET`, the exception's clearance should be `SecurityClearance.TOP_SECRET`.
   (c) When caught, the exception's message should be "This mission requires a clearance of `CLEARANCE`", replacing `CLEARANCE` with the actual clearance, of course. *Hint: look at what the Exception constructor parameters are.*

2 Solution and Tips

This is a complex assignment. It's intentionally vague, but only to avoid giving away the answer, and you should find that there is really only one right way to accomplish it.

We've provided all the source files you will need, but you will need to modify them extensively. **Do not change any file names or visibility modifiers**.

We've included some hints in some of the files on how to accomplish some things.

It is absolutely vital that your submission compiles. Non-compiling solutions will be given a zero: no exceptions.

Make sure to test your code. The good news is that if you've finished and it all compiles, it's probably close to right, but make sure you consider all the possible cases.

3 Checkstyle

You must run checkstyle on your submission. The checkstyle cap for this assignment is **10** points.
Review the Style Guide and download the Checkstyle jar and associated XML file. Run Checkstyle on your code like so:

```
$ java -jar checkstyle-5.6-all.jar -c cs1331-checkstyle.xml *.java
Starting audit...
Audit done.
```

The message above means there were no Checkstyle errors. You can easily count Checkstyle errors by piping the output of Checkstyle through `wc -l` and subtracting 2 for the two non-error lines printed above (which is how we will deduct points). For example:

```
$ java -jar checkstyle-5.6-all.jar -c cs1331-checkstyle.xml *.java | wc -l
     2
```

Alternatively, if you are on Windows, you can use the following instead:

```
C:\> java -jar checkstyle-5.6-all.jar -c cs1331-checkstyle.xml *.java | findstr /v "Starting audit..." |
     findstr /v "Audit done" | find /c /v "hashcode()"
0
```

Food for thought: is there a one-liner like above that shows you only the number of errors? Hint: `man grep`.

The Java source files we provide contain no Checkstyle errors. For this assignment, there will be a maximum of **10** points lost due to Checkstyle errors (1 point per error). In future homeworks we will be increasing this cap, so get into the habit of fixing these style errors early!

4 Turn-in Procedure

Submit all of the Java source files you modified and resources your program requires to run to T-Square. Do not submit any compiled bytecode (`.class` files), the Checkstyle jar file, or the `cs1331-checkstyle.xml` file. When you're ready, double-check that you have submitted and not just saved a draft.

Please remember to run your code with Checkstyle!

Verify the Success of Your Submission to T-Square
Practice safe submission! Verify that your HW files were truly submitted correctly, the upload was successful, and that the files compile and run. It is solely your responsibility to turn in your homework and practice this safe submission safeguard.

1. After uploading the files to T-Square you should receive an email from T-Square listing the names of the files that were uploaded and received. If you do not get the confirmation email almost immediately, something is wrong with your HW submission and/or your email. Even receiving the email does not guarantee that you turned in exactly what you intended.
2. After submitting the files to T-Square, return to the Assignment menu option and this homework. It should show the submitted files.
3. Download copies of your submitted files from the T-Square Assignment page placing them in a new folder.
4. Recompile and test those exact files.
5. This helps guard against a few things.
   (a) It helps insure that you turn in the correct files.
   (b) It helps you realize if you omit a file or files. [1] (If you do discover that you omitted a file, submit all of your files again, not just the missing one.)
   (c) Helps find last minute causes of files not compiling and/or running.

[1] Missing files will not be given any credit, and non-compiling homework solutions will receive few to zero points. Also recall that late homework will not be accepted regardless of excuse. Treat the due date with respect. The real due date is midnight. Do not wait until the last minute!