ITSC315 – Security for Programmers

SAIT, Winter 2021

Assignment 1 – Secure Java Programming

<u>NOTE: ALL ASSIGNMENTS/ASSESSMENTS FOR THIS COURSE ARE *INDIVIDUAL* IN NATURE. EACH STUDENT MUST COMPLETE THE WORK FOR THE ASSIGNMENTS BY THEMSELVES</u>

<u>Due date: 11:59pm, Friday 26th February, 2021</u>.

<u>Submission:</u> Upload a file called *ITSCJavaAssign.zip* to BrightSpace containing the source code, and compiled versions, of your solution. See the "Application Submission Structure" section at the end of the document for details on what must be in this <u>.zip</u> file submission. Late submissions will not be accepted and will receive a mark of 0.

<u>Problem: Develop a console-based Java application using principles from the Java "Secure Coding Guidelines" document (as presented via PowerPoint and code examples in class)</u>

You are tasked with writing a console-based (text-based) Java application called **ManageComputers** that manages information about computers for a company. The company owns laptop and desktop computers and needs to store, display and manage the data about the specifications for the machines. You will use principles from the Java "*Secure Coding Guidelines*" document in the development of this application.

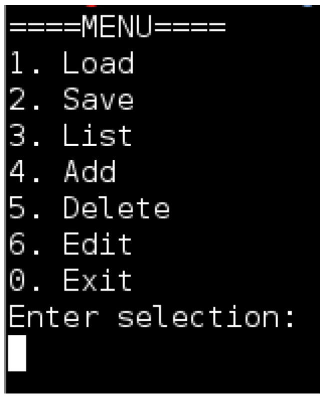
The use cases for this application include:

0. *Display a menu of options to the user and accept their selection*
1. Listing/displaying the details of all the computers stored in the system
2. Adding a new computer to the system
3. Editing an existing computer's data
4. Delete an existing computer from the system
5. Load the data on all of the computer from a drive
6. Save the data for all the computers (stored in memory) to the drive
7. Exit the application

Sample Application Run

(0) Showing the menu

When the application is run it should display the following menu:

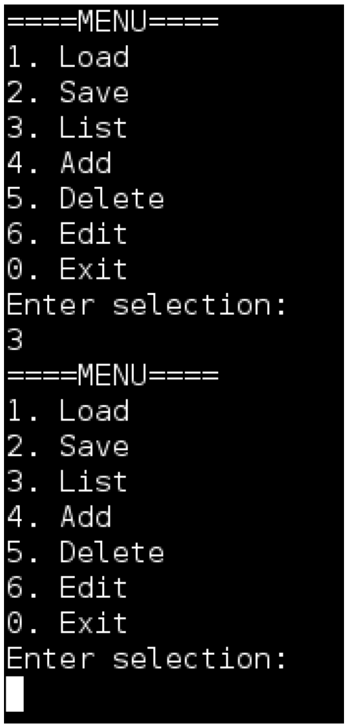


```
====MENU====
1. Load
2. Save
3. List
4. Add
5. Delete
6. Edit
0. Exit
Enter selection:
```

The user can enter a number for their selection, e.g. “**3**” to list the details of all the computers stored in the system.

(1) Listing the Computers (“List”)

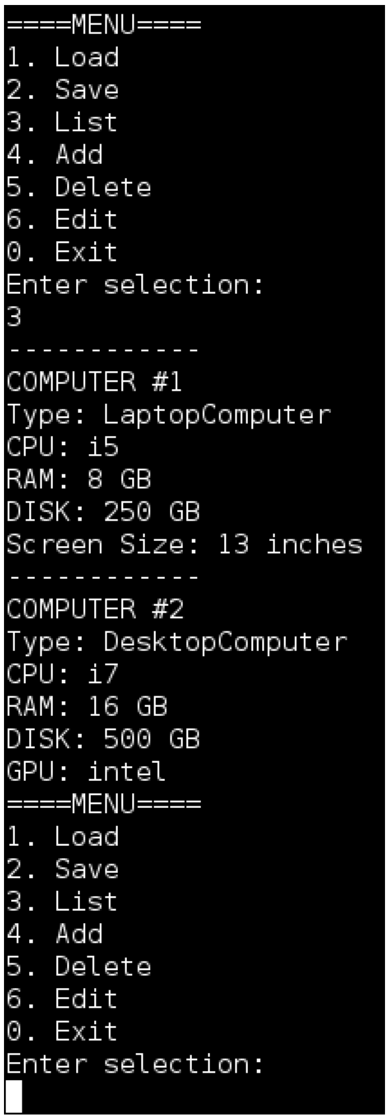
If the user enters “**3**” at the menu:



```
====MENU====
1. Load
2. Save
3. List
4. Add
5. Delete
6. Edit
0. Exit
Enter selection:
3
====MENU====
1. Load
2. Save
3. List
4. Add
5. Delete
6. Edit
0. Exit
Enter selection:
```

In this case there is no computer data available to the system yet, so the application simply redisplays the menu again.

In the event that there *is* data for a computer(s) in the system then it should display as follows:



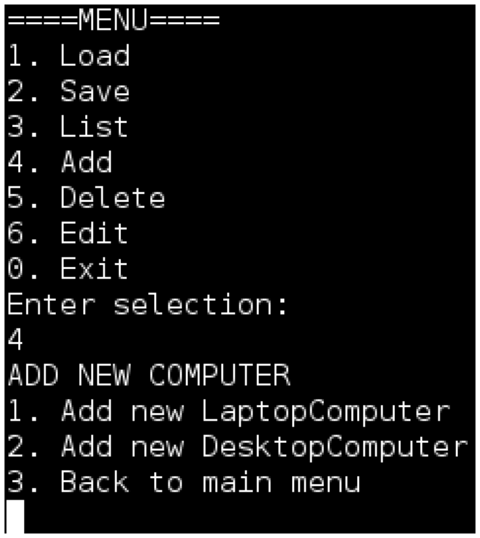
```
====MENU====
1. Load
2. Save
3. List
4. Add
5. Delete
6. Edit
0. Exit
Enter selection:
3
------------
COMPUTER #1
Type: LaptopComputer
CPU: i5
RAM: 8 GB
DISK: 250 GB
Screen Size: 13 inches
------------
COMPUTER #2
Type: DesktopComputer
CPU: i7
RAM: 16 GB
DISK: 500 GB
GPU: intel
====MENU====
1. Load
2. Save
3. List
4. Add
5. Delete
6. Edit
0. Exit
Enter selection:
```

In the example above there is data for a laptop and a desktop computer that can be displayed. The computers are numbered based upon the order they were stored in the system.

Note that the data for laptop and desktop computers is a little different, i.e. laptops specify screen size as a specification detail whereas desktops specify GPU type instead.

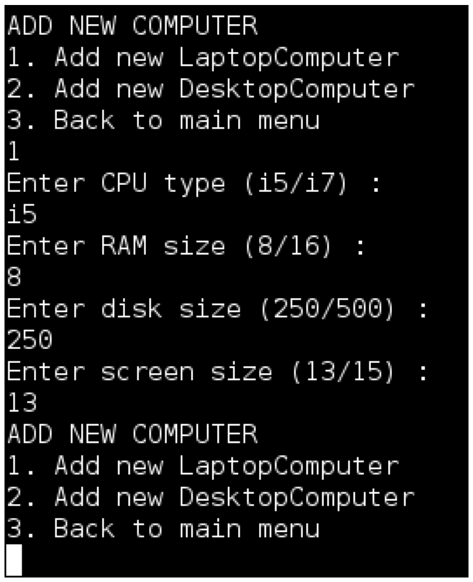
(2) Add a computer

If the user enters “**4**” and presses enter then the following sub-menu should be displayed:



```
====MENU====
1. Load
2. Save
3. List
4. Add
5. Delete
6. Edit
0. Exit
Enter selection:
4
ADD NEW COMPUTER
1. Add new LaptopComputer
2. Add new DesktopComputer
3. Back to main menu
```

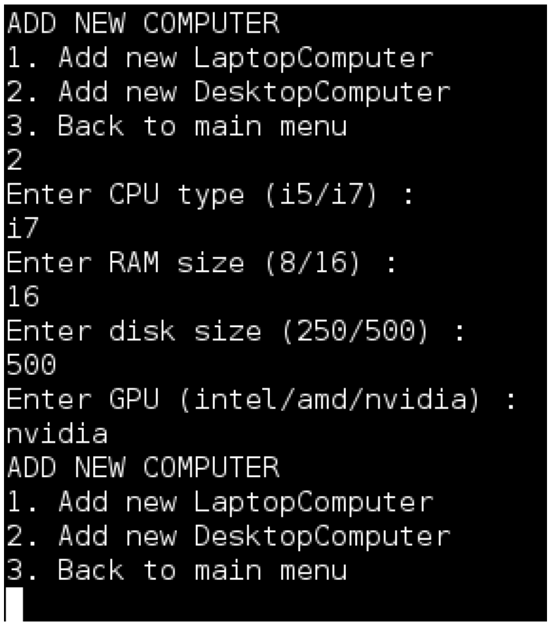
If the user enters “**1**” then s/he is prompted to enter the specification data for a laptop computer, e.g.:



```
ADD NEW COMPUTER
1. Add new LaptopComputer
2. Add new DesktopComputer
3. Back to main menu
1
Enter CPU type (i5/i7) :
i5
Enter RAM size (8/16) :
8
Enter disk size (250/500) :
250
Enter screen size (13/15) :
13
ADD NEW COMPUTER
1. Add new LaptopComputer
2. Add new DesktopComputer
3. Back to main menu
```

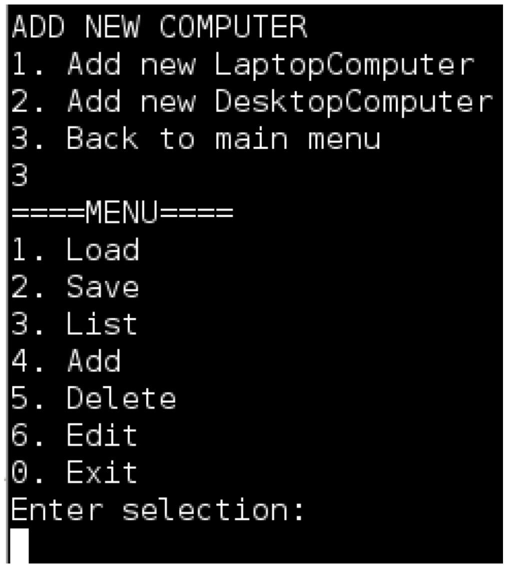
The prompts show the only valid options for each item, e.g. CPU’s can only be “**i5**” or “**i7**”, etc. Any other values should be rejected and an error message displayed.

Entering “**2**” at this sub-menu would allow the user to enter the data for a desktop computer, e.g.:



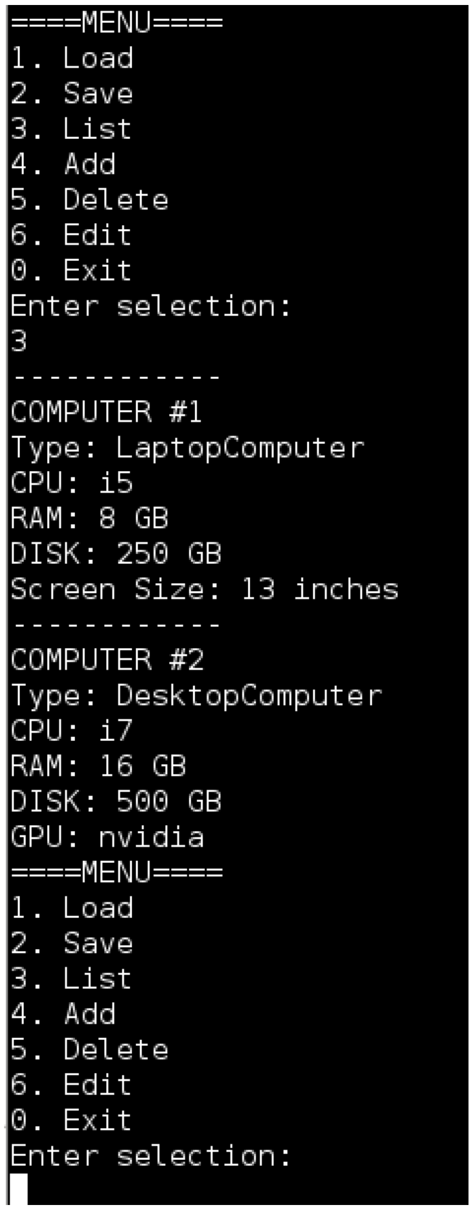
```
ADD NEW COMPUTER
1. Add new LaptopComputer
2. Add new DesktopComputer
3. Back to main menu
2
Enter CPU type (i5/i7) :
i7
Enter RAM size (8/16) :
16
Enter disk size (250/500) :
500
Enter GPU (intel/amd/nvidia) :
nvidia
ADD NEW COMPUTER
1. Add new LaptopComputer
2. Add new DesktopComputer
3. Back to main menu
```

Once the computers have been added to the system the user can press "**3**" and return to the main application window (or enter more computers using "**1**" and "**2**" in the sub-menu, if they so wish), e.g.:



```
ADD NEW COMPUTER
1. Add new LaptopComputer
2. Add new DesktopComputer
3. Back to main menu
3
====MENU====
1. Load
2. Save
3. List
4. Add
5. Delete
6. Edit
0. Exit
Enter selection:
```

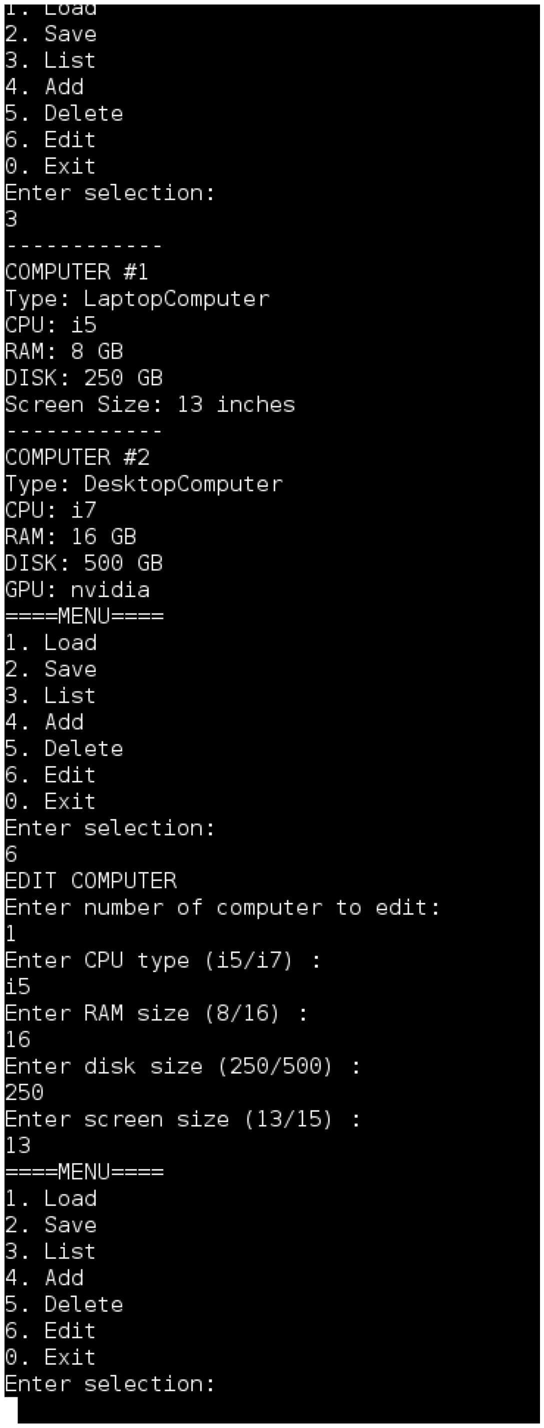
Entering "**3**" will list the data on the computers as entered, e.g.:



```
====MENU====
1. Load
2. Save
3. List
4. Add
5. Delete
6. Edit
0. Exit
Enter selection:
3
------------
COMPUTER #1
Type: LaptopComputer
CPU: i5
RAM: 8 GB
DISK: 250 GB
Screen Size: 13 inches
------------
COMPUTER #2
Type: DesktopComputer
CPU: i7
RAM: 16 GB
DISK: 500 GB
GPU: nvidia
====MENU====
1. Load
2. Save
3. List
4. Add
5. Delete
6. Edit
0. Exit
Enter selection:
```

(3) Edit a computer

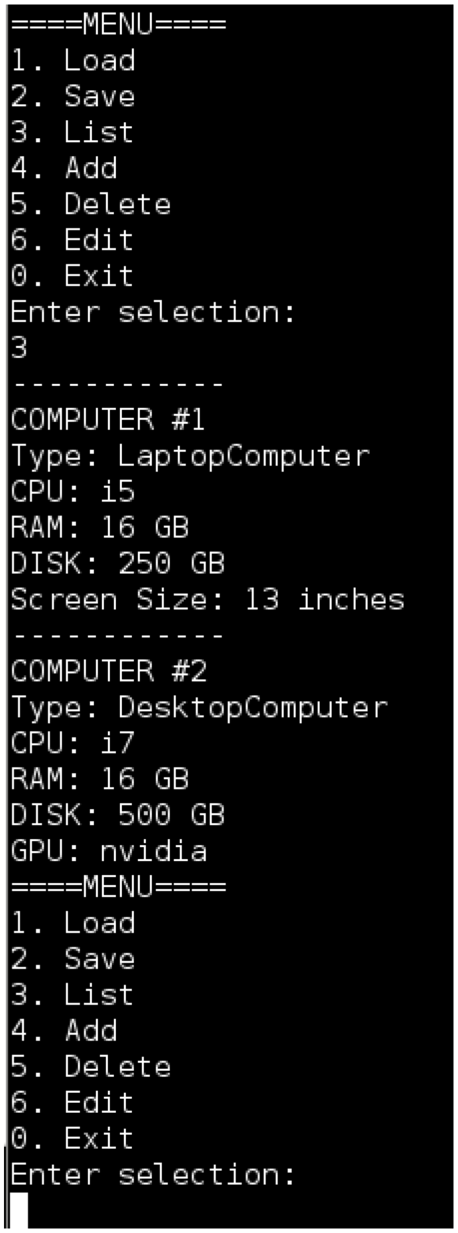
To edit the data for an existing computer, enter “**6**” at the main menu, and then the number of the computer to edit as shown in the computer listing (by entering “**1**”), e.g. to edit the data for computer number *1* (the laptop in the sample data above):



```
1. Load
2. Save
3. List
4. Add
5. Delete
6. Edit
0. Exit
Enter selection:
3
------------
COMPUTER #1
Type: LaptopComputer
CPU: i5
RAM: 8 GB
DISK: 250 GB
Screen Size: 13 inches
------------
COMPUTER #2
Type: DesktopComputer
CPU: i7
RAM: 16 GB
DISK: 500 GB
GPU: nvidia
====MENU====
1. Load
2. Save
3. List
4. Add
5. Delete
6. Edit
0. Exit
Enter selection:
6
EDIT COMPUTER
Enter number of computer to edit:
1
Enter CPU type (i5/i7) :
i5
Enter RAM size (8/16) :
16
Enter disk size (250/500) :
250
Enter screen size (13/15) :
13
====MENU====
1. Load
2. Save
3. List
4. Add
5. Delete
6. Edit
0. Exit
Enter selection:
```

In the example above the user is upgrading the RAM in the laptop to 16GB (but s/he still needs to enter the other data values as well, even if their values are the same as before).

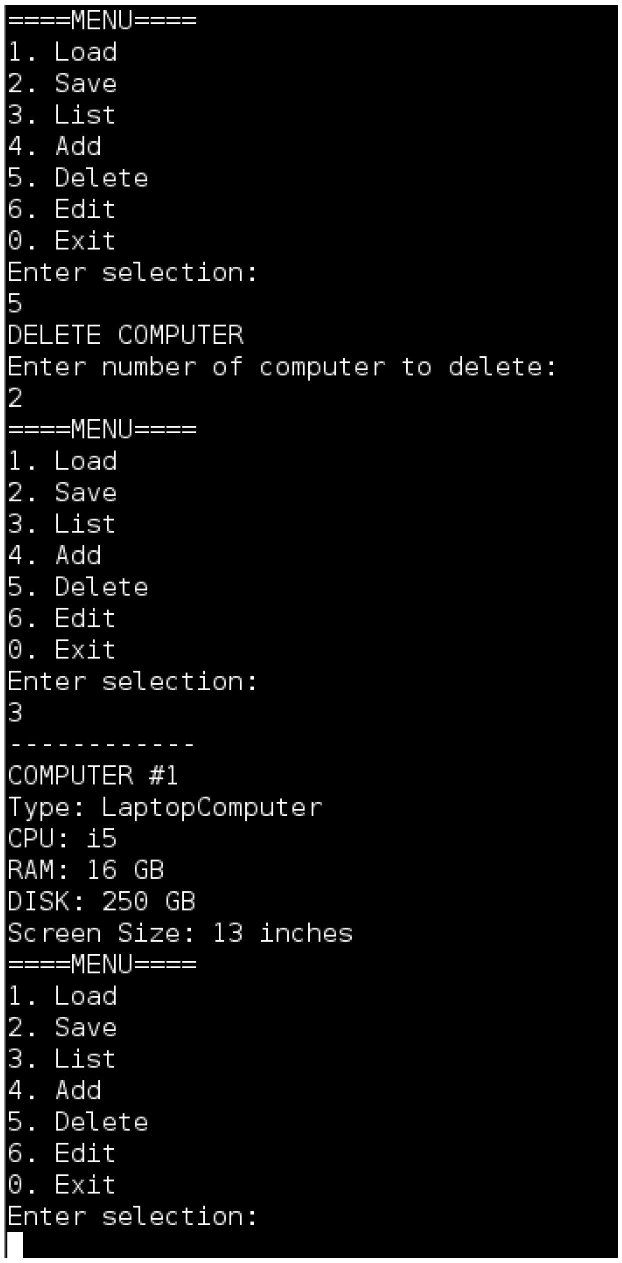
Redisplaying the computer data after the laptop has been edited would show the following:



```
====MENU====
1. Load
2. Save
3. List
4. Add
5. Delete
6. Edit
0. Exit
Enter selection:
3
------------
COMPUTER #1
Type: LaptopComputer
CPU: i5
RAM: 16 GB
DISK: 250 GB
Screen Size: 13 inches
------------
COMPUTER #2
Type: DesktopComputer
CPU: i7
RAM: 16 GB
DISK: 500 GB
GPU: nvidia
====MENU====
1. Load
2. Save
3. List
4. Add
5. Delete
6. Edit
0. Exit
Enter selection:
```

Delete a computer

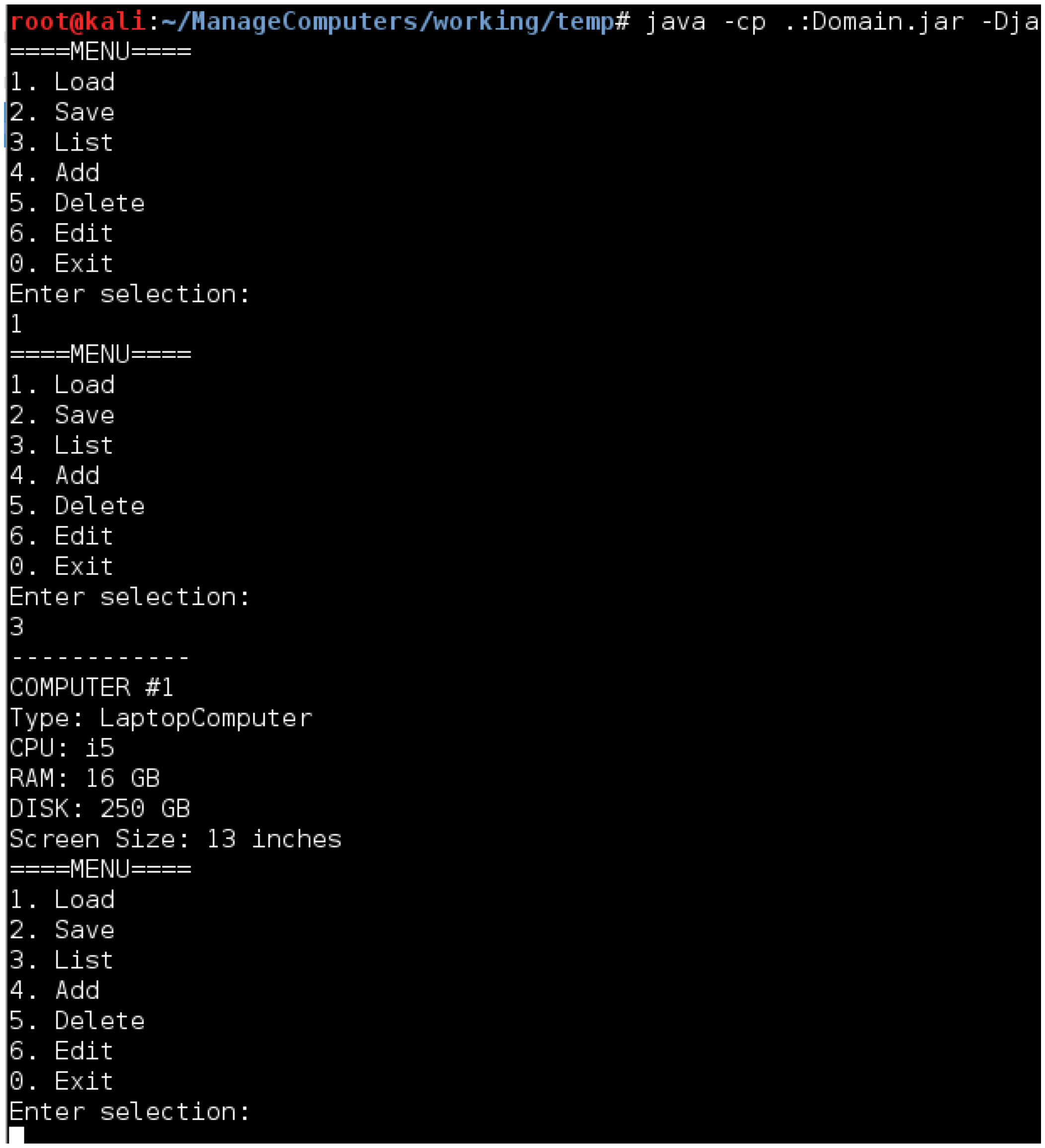
To delete a computer, enter "**5**" at the main menu, and then the number of the computer to delete, e.g. to delete the desktop computer from our sample data, and show the revised list of computers:



```
====MENU====
1. Load
2. Save
3. List
4. Add
5. Delete
6. Edit
0. Exit
Enter selection:
5
DELETE COMPUTER
Enter number of computer to delete:
2
====MENU====
1. Load
2. Save
3. List
4. Add
5. Delete
6. Edit
0. Exit
Enter selection:
3
------------
COMPUTER #1
Type: LaptopComputer
CPU: i5
RAM: 16 GB
DISK: 250 GB
Screen Size: 13 inches
====MENU====
1. Load
2. Save
3. List
4. Add
5. Delete
6. Edit
0. Exit
Enter selection:
```

(4) Load computer data from drive

To load previously-saved computer data from the drive select “**1**” from the menu. This will load the data for those computers into memory, replacing any computer data that was already in memory with the data loaded from the drive.



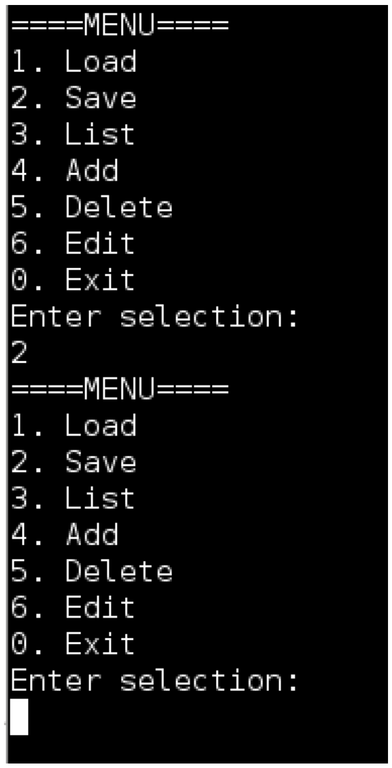
```
root@kali:~/ManageComputers/working/temp# java -cp .:Domain.jar -Dja
====MENU====
1. Load
2. Save
3. List
4. Add
5. Delete
6. Edit
0. Exit
Enter selection:
1
====MENU====
1. Load
2. Save
3. List
4. Add
5. Delete
6. Edit
0. Exit
Enter selection:
3
------------
COMPUTER #1
Type: LaptopComputer
CPU: i5
RAM: 16 GB
DISK: 250 GB
Screen Size: 13 inches
====MENU====
1. Load
2. Save
3. List
4. Add
5. Delete
6. Edit
0. Exit
Enter selection:
```

The screenshot above shows the user running the application. Then the user selected “**1**” from the menu, which loaded the data on any computers stored on the drive back into memory.

Entering “**3**” after reloading the computer data displayed the newly-loaded computer data, in this example for a single laptop computer.

(5) Save computer data to a drive

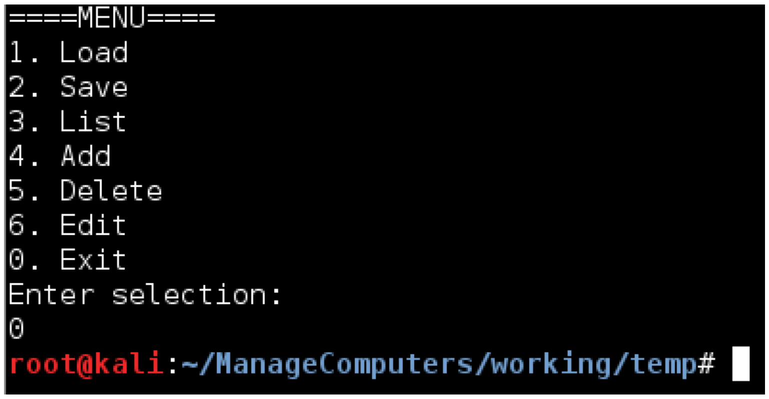
To save the data for any computers that is currently held in memory the user selects “**2**” from the menu. The application saves copies of the data for each computer (held in memory) in separate serialized data files on the drive, and then redisplays the main menu:



```
====MENU====
1. Load
2. Save
3. List
4. Add
5. Delete
6. Edit
0. Exit
Enter selection:
2
====MENU====
1. Load
2. Save
3. List
4. Add
5. Delete
6. Edit
0. Exit
Enter selection:
```

(6) Exit the application

Selecting "**0**" terminates the application and drops the user back to the command prompt, e.g.:



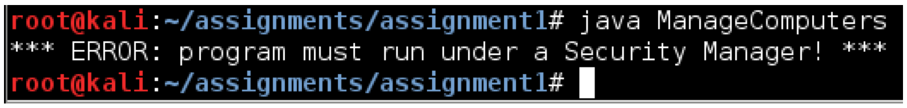
```
=====MENU=====
1. Load
2. Save
3. List
4. Add
5. Delete
6. Edit
0. Exit
Enter selection:
0
root@kali:~/ManageComputers/working/temp# 
```

Exiting from the application should ***not*** auto-save any unsaved computer data in memory.

__**Error Messages**__

The application must output the following error messages[1] as required:

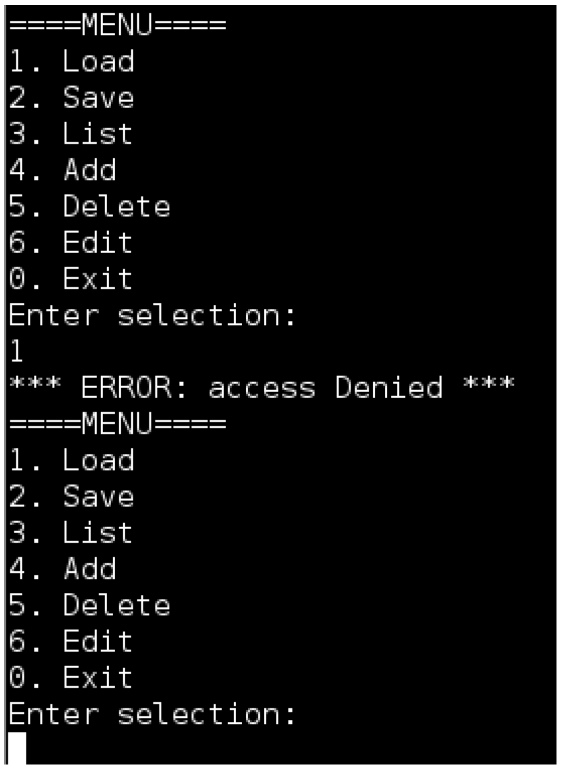
- If the application is run without the supervision of a ***SecurityManager***:



```
root@kali:~/assignments/assignment1# java ManageComputers
*** ERROR: program must run under a Security Manager! ***
root@kali:~/assignments/assignment1# 
```

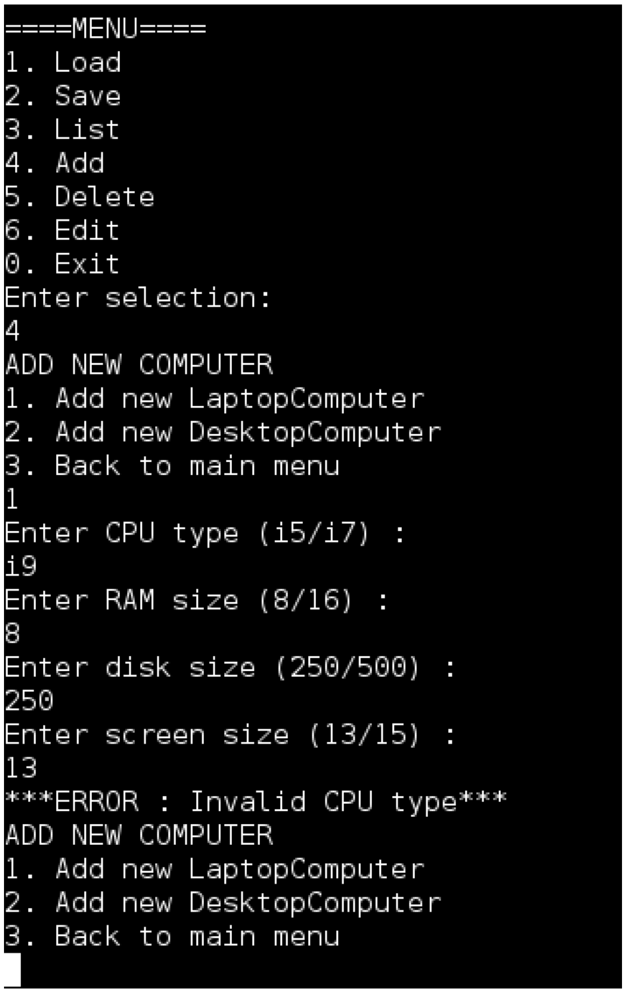
[1] Note that error messages displayed on-screen must all be ***sanitized*** of sensitive information, e.g. file names and paths, etc.

- If the application does not have permissions to read, write and delete files in the **/root/assign1Data** directory, where computer object data must be stored:



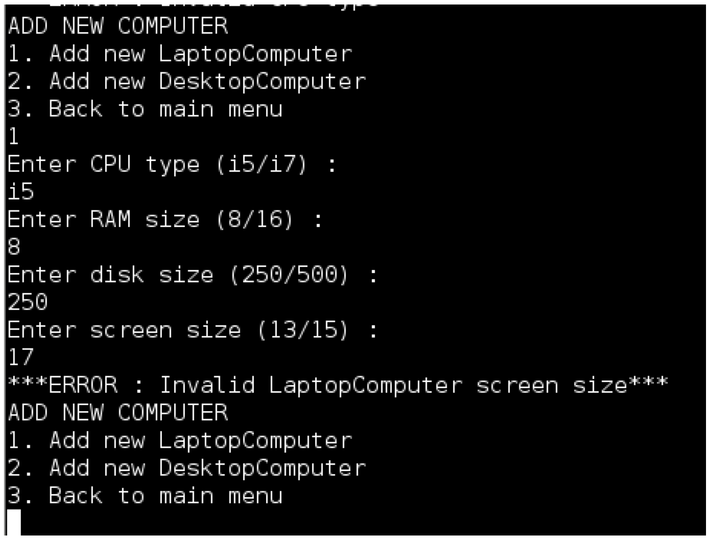
```
====MENU====
1. Load
2. Save
3. List
4. Add
5. Delete
6. Edit
0. Exit
Enter selection:
1
*** ERROR: access Denied ***
====MENU====
1. Load
2. Save
3. List
4. Add
5. Delete
6. Edit
0. Exit
Enter selection:
```

- If the user enters invalid data for a new computer, e.g. an “**i9**” processor:



```
====MENU====
1. Load
2. Save
3. List
4. Add
5. Delete
6. Edit
0. Exit
Enter selection:
4
ADD NEW COMPUTER
1. Add new LaptopComputer
2. Add new DesktopComputer
3. Back to main menu
1
Enter CPU type (i5/i7) :
i9
Enter RAM size (8/16) :
8
Enter disk size (250/500) :
250
Enter screen size (13/15) :
13
***ERROR : Invalid CPU type***
ADD NEW COMPUTER
1. Add new LaptopComputer
2. Add new DesktopComputer
3. Back to main menu
```

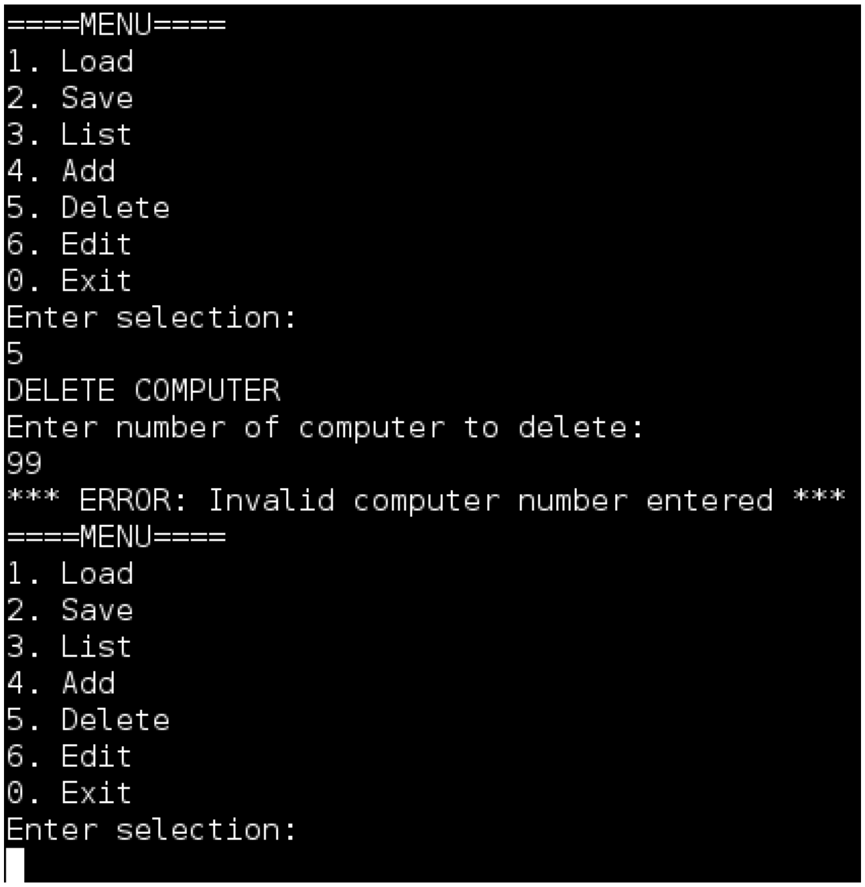
Or, an invalid laptop screen size (“**17**”):



```
ADD NEW COMPUTER
1. Add new LaptopComputer
2. Add new DesktopComputer
3. Back to main menu
1
Enter CPU type (i5/i7) :
i5
Enter RAM size (8/16) :
8
Enter disk size (250/500) :
250
Enter screen size (13/15) :
17
***ERROR : Invalid LaptopComputer screen size***
ADD NEW COMPUTER
1. Add new LaptopComputer
2. Add new DesktopComputer
3. Back to main menu
```

<u>Note:</u> Display equivalent error messages for other invalid input values also.

- If an invalid computer number is entered during an operation, e.g. delete:



```
====MENU====
1. Load
2. Save
3. List
4. Add
5. Delete
6. Edit
0. Exit
Enter selection:
5
DELETE COMPUTER
Enter number of computer to delete:
99
*** ERROR: Invalid computer number entered ***
====MENU====
1. Load
2. Save
3. List
4. Add
5. Delete
6. Edit
0. Exit
Enter selection:
```

Application Requirements

General Requirements

This application must be written using the following Java "*Secure Coding Guidelines*" (as covered in class):

- **No subclassing** allowed (code reuse is to be done via ***composition***)
- Classes can only be instantiated using **static constructor** methods, e.g***. getInstance()***. Normal constructors must be declared as **private**
- Objects must be **immutable** (editing an object means replacing it with a new instance)
- **User input** to the application must be **validated**. It must not be possible to store invalid attribute values in objects used in the system. Attempting to do so should **display an error** on-screen, and redisplay the menu to the user
- The application must run under the supervision of a ***SecurityManager*** with an appropriate *policy file* to allow data saving and loading
- The ***BaseComputer***, ***LaptopComputer*** and ***DesktopComputer*** classes must be accessed from a **sealed** .jar file called **Domain.jar** when the application is run
- Exceptions must be sanitized of sensitive data, e.g. file names and paths

Development Platform

It is recommended to develop this application in the Kali VM using **gedit** as the code editor. This will help you understand the structure of Java code for an application like this, i.e. how packages relate to subdirectories, etc., as well as the compilation and packaging of a Java application from the command line. Using an IDE hides these details from you as a developer, and these will be important at times.

NOTE: Do **not** submit an IDE-based project in completion of this assignment, i.e. from NetBeans or any other IDE. See the "*Application submission structure*" section at the end of this document for details on expected submitted code structure). ***Submitting an IDE-based solution will result in a decreased mark for this assignment!***

Runtime Platform

The application must run in a **terminal window** in the Kali VM provided for the course. The current directory in the terminal when the command line below is executed must be **/root/ManageComputers**. The application must run using the following command line:

```
java -cp .:Domain.jar -Djava.security.manager -Djava.security.policy=security.policy ManageComputers
```

In the **/root/ManageComputers** directory, when the command line is executed, must be the following files:

- The **Domain.jar** file (a **sealed** .jar file that contains the classes in the **domain** package classes: ***BaseComputer***, ***LaptopComputer*** and ***DesktopComputer***)
- The **security.policy** file is a ***SecurityManager*** policy file that contains the permission settings for the application (specify one permission stating that it can read and write the **/root/assign1Data** directory, and a separate one that states it can perform read, write and delete operations on any files in that directory)
- The **ManageComputers.class** file

Application Structure

The following files must exist in your solution:

- **ManageComputers.java**: this contains the **main()** method for your application, and other methods as required
  - When run the **main()** method creates the ***ArrayList*** used to store the computer data for the application. This is where user-entered data, and data loaded from a file, is stored and accessed by the application
- **domain** package:
  - ***BaseComputer.java***: holds the CPU type (*String*), RAM size (*int*) and disk size (*int*) for any computer
  - ***LaptopComputer.java***: holds an instance of ***BaseComputer***, and additionally the screen size (*int*) for the laptop
  - ***DesktopComputer.java***: holds an instance of ***BaseComputer***, and additionally the GPU type (*String*) for the desktop
- **MANIFEST.MF**: contains the information used to seal the **domain** package
- **security.policy**: the ***SecurityManager*** policy file containing permissions allowing the application to work with files in the **/root/assign1Data** directory

Data Storage and Retrieval

Computer data (i.e. for **LaptopComputer** and **DesktopComputer** objects) must be stored in serialized form on a drive. The data must be held in files in **/root/assign1Data**. Each laptop and desktop object in the ***ArrayList*** maintained by the application must be stored separately in its own serialized file in the **assign1Data** directory (named **1.txt**, **2.txt**, **3.txt** etc.). Any ***String*** attribute data must be encrypted so that it cannot be easily read in the serialized data files. Numeric attribute data is not easily readable in serialized files and does not have to be explicitly encrypted.

Unsaved data in memory will be <u>*lost*</u> if the user closes the application without saving it.

Data in memory will be _*completely replaced*_ with data loaded from the drive when the user chooses to load data.

The **assign1Data** directory will be completely cleared of any existing files, and have new files written into it from the ***ArrayList***, when the user chooses to save the in-memory application data.

Object String Data Encryption

When serializing objects (during a save operation), any ***String*** attribute data should be encrypted by shifting all of the characters in the ***String*** one place to the right in the alphabet (a "Caesar Cipher" approach to data encryption). Looking directly at the contents of the serialized object files on the drive should not show the unencrypted ***String*** data in the files.

When deserializing object data (during a load operation) the ***String*** character data loaded from the files should be shifted one place to the left (to decrypt the String data).

Hint: The best way to do this is to implement private **readObject()** and **writeObject()** methods in any class that is going to be serialized/deserialized.

Application Submission Structure

Your submission for this assignment must follow the following standard structure:

- An overall file called **ITSCassign1.zip** which contains:
  - A subdirectory called **source**. This contains all of the source code files for your solution (including correct subdirectories for packages, etc.), **MANIFEST.MF** and **security.policy** file. All the files required to build your application must be located here
  - Another subdirectory called **ManageComputers**. This directory must contain the compiled **ManageComputers.class** file, the pre-built **Domain.jar** file, and the **security.policy** file
    - It must be possible to change into this directory and enter the command line specified earlier to run your application without further work (you can assume that the **/root/assign1Data** directory already exists)