Lab 1: Writing Java Program on GNU/Linux

Objectives

The main objectives of this laboratory are

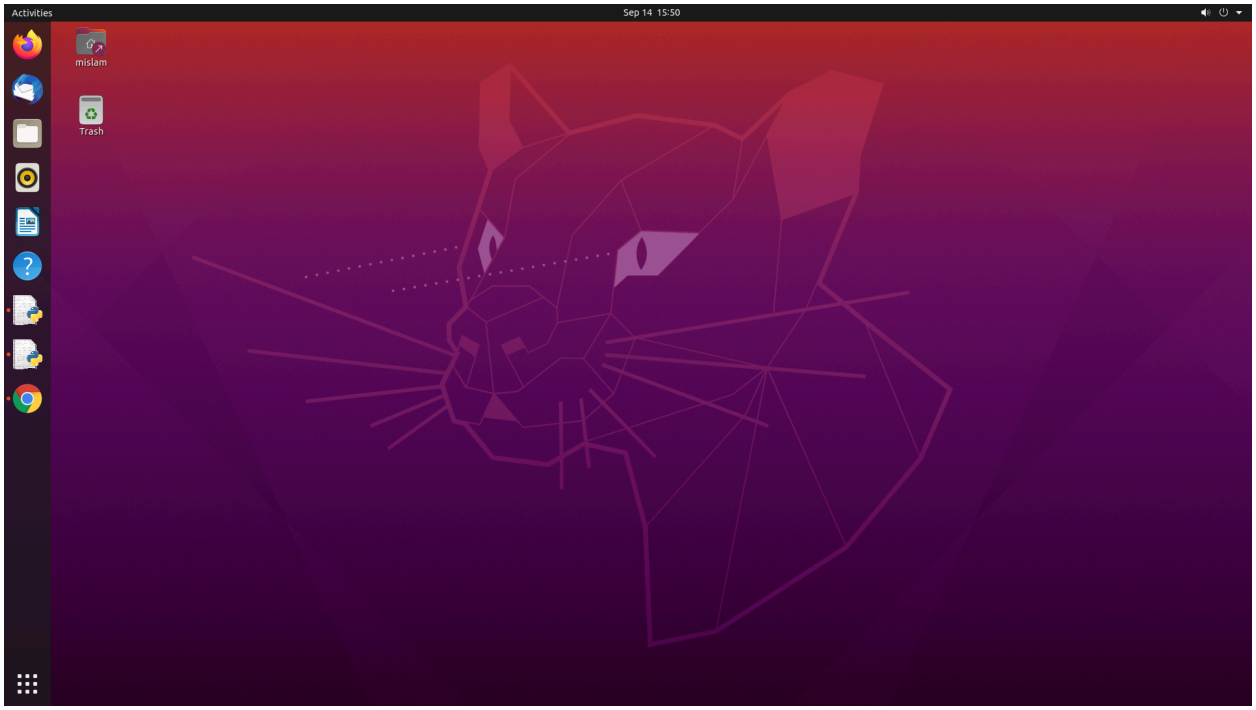
1. To familiarize yourself with the GNU/Linux operating system, and
2. To learn how to compile and run Java programs.
3. How to print in console/output window

1. GNU/Linux operating system

Introduction. In this course, we will be using the *GNU/Linux* operating system. It is a *free* and *open-source* operating system published under the *GNU General Public License* (GPL). Free and open source mean that, unlike *macOS* and *Windows*, anyone is free to use, modify and distribute any of the actual source code. GNU/Linux is touted for its speed, minimal hardware requirements, security and remote administration. It is a fully-featured operating system that does not have to cost a dime. Because of GNU/Linux's speed and stability, it has increasingly become the operating system of choice for many advanced users and permeates the entire web market.

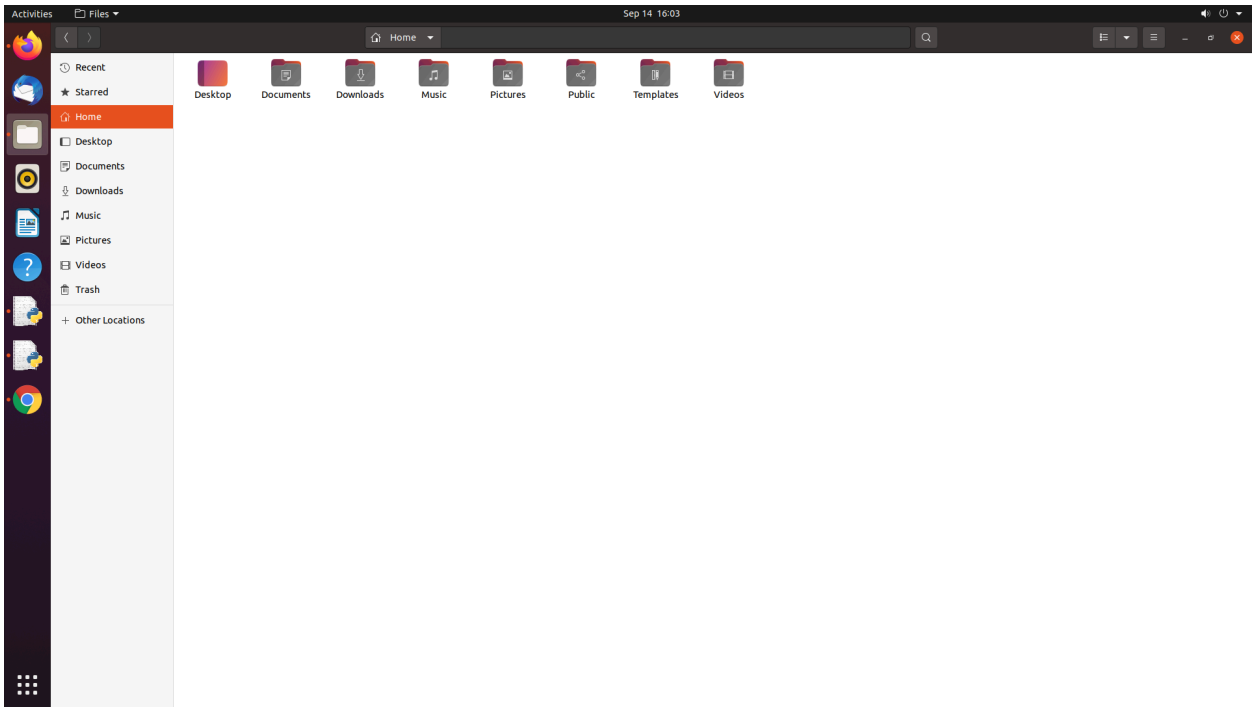
Ubuntu. There are a number of different distributions of GNU/Linux available on the Internet. For its elegant design and robustness, the Trinity College Computer Science Department has chosen the *Ubuntu* distribution (which is based on the popular *Debian* distribution of GNU/Linux). Ubuntu is currently the most popular Linux distribution and widely-regarded as the best alternative to macOS and Windows.

Logging in. To use GNU/Linux, you must first identify yourself to the system. This is done so that the system knows who you are, what permissions you have, and what your preferences are. To identify who you are, you will use your usual Trinity username and password. Enter your username and password. You should then see the following *Ubuntu* 14.04 *LTS* (*Trusty Tahr*) desktop:



This is a free and open-source desktop environment for GNU/Linux, similar to the desktop environments found under macOS and Windows. The main desktop components are the menus and launcher located at the left and bottom ends of the screen. The launcher gives you quick access to popular applications. The third top item of the menu is the Files cabinet button.

Creating a cpsc115 directory: You will now create a directory (i.e., folder) to keep all of your course-related files. First, click the file-cabinet button. You should then see the following *File*



Browser window.

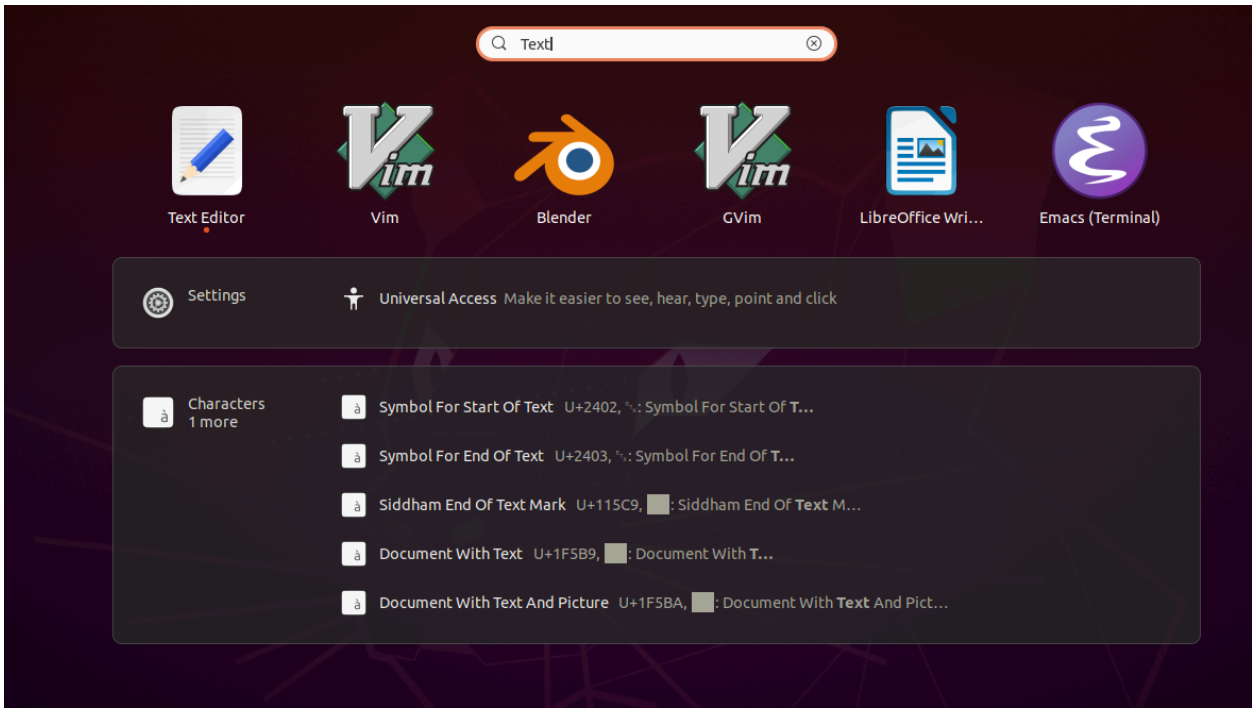
Now, right-click anywhere inside the window and select *New Folder*. Name the new folder cpsc115, *all in the lowercase without any space* (in general, files and directories should be named all in the lowercase without any space). You should save all course-related files in this directory. Since this is your first lab, create a *lab1* folder inside the *cpsc115* and save all files related to the first lab inside the *lab1* folder. You should do the same thing for future labs. Remember how to find this directory because you are going to need it every week.

2. Writing first java program

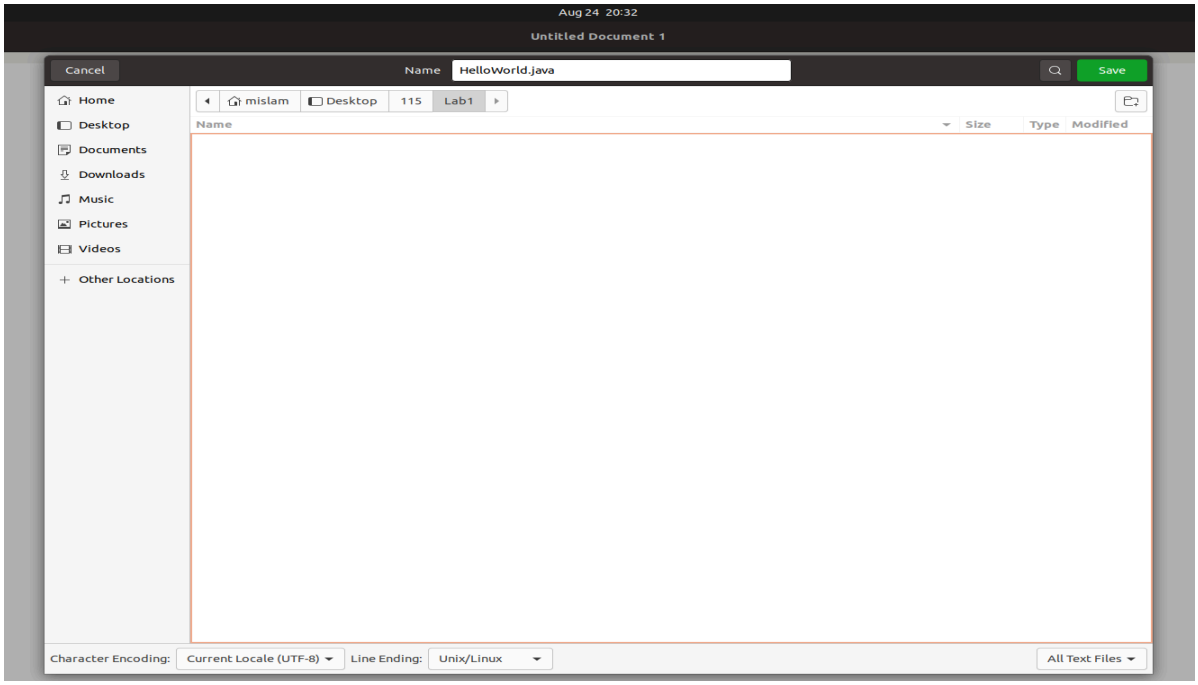
In this course, to learn how to program in java, we will be using a simple text editor to write code that is already installed in the lab computers.

Launching Text Editor. To launch the text editor on your laboratory workstation, all you need to do is to click on the Ubuntu button, type “text” in the Dash, and select “Text Editor”, as

shown below:



To write Java code, you have to create a java file. After launching the text editor and click 'Save' which will prompt you for a file name and location. Give the name *HelloWorld.java* and save the file inside the *lab1* folder.



Now, we are ready to write our first Java Program.

You can use any IDE i.e. Atom, SublimeText, etc. to write your code. For example, you can launch Atom which is a popular IDE to write java code. Type “Atom” clicking on the Ubuntu icon and select the IDE from the pop-up list.

Hello, World! Traditionally (at least dating back to the instructor’s undergraduate years), the first program students write involves instructing a computer to output “Hello, World!”. In this course, we will also follow this tradition. So, we will write a Java program that prints “Hello World!” in the output window.

While coding, it's a given that errors can occur. The computer won't correct these errors for you; rather, you'll need to address them yourself. Rectifying these mistakes can be simpler if you don't attempt to write the entire program at once. Instead, write a few lines, then compile them to identify any syntax errors. If you find any, address them; if not, you can proceed to add a few more lines.

We will follow the same methodology. Everything we write in a Java program must be inside a class. The syntax of writing a class in Java is given below:

```
class <class name>
{
}
```

A java file can contain multiple classes. However, we will be working with a single class in a Java file. In that case, the class name must match with the file name without the extension(.java). Since our filename is `HelloWoorld.Java`, the class name must be `HelloWorld`. Remember that Java is case sensitive. Hence, write the following class in `HelloWorld.java`.

```
class HelloWorld
{
}
```

By the pair of curly braces, we define the boundary of a class, method etc. that we will be learning in future. What we want to put inside a class must be inside the boundary.

Compiling and running Java a Java program. When we write a Java program, we first need to make sure that our code doesn’t contain any syntax error. We use the Java compiler to check the syntax error. We will be running the Java compiler from the *terminal*. To launch the *terminal*, similar to launching a text editor, type `terminal` clicking the Ubuntu icon and select the *terminal* option.

`javac` is the command to run the java compiler in the *terminal* where `javac` stands for Java Compiler. Hence, you have to write the `javac` command in the terminal. We are running the Java compiler to compile a Java file. We need to mention which file we would like to compile.

Hence, write the `javac HelloWorld.java` command in the *terminal*. If your Java file contains any error, it will be shown in the *terminal*. If you see any error, try to fix it by reading the error message or ask any of the TAs or your instructor to address it. If there is no error, the compiler will generate an intermediate file(*HelloWorld.class*) in your current directory. If you look inside the lab1 folder, you should see the intermediate file. The intermediate file contains the byte code that will be run by *JVM*(*Java Virtual Machine*) or *Java Runtime*. To run the *JVM*, run `java HelloWord`(java is the JVM and hence, you are telling the *JVM* to run the `HelloWorld.class` file) in the terminal.

You can learn how to navigate to the appropriate directory into the Ubuntu terminal to run the java files from this document.

Writing the main method. When you run the intermediate file, Java shows an error saying *the main method is not found in class HelloWorld*. When we run a Java file, it starts the execution from the class that matches with the file name and that's why you have to match the class name and the file name. We will see in the future that we can have multiple methods inside a class. The question is, which method Java executes first or from which method, Java starts the execution? Java set the rule that the class must have a main method from where it will start the execution. Hence, the next task is to add the main method. The syntax of writing a main method is given below:

```
public static void main(String[] args)
{
}
```

We will discuss in future why we have so many keywords in the main method. Again, the curly braces define the boundary of the main method where the opening curly brace defines the start of the method and the closing curly brace defines the end of the method. Whatever we write inside the main method will be executed sequentially.

So, put the main method inside the class and your class should look like this:

```
class HelloWorld
{
    public static void main(String[] args)
    {

    }
}
```

Now, compile and run the program again, you should see no error now. However, we will not see any output in the terminal.

Let's print `Hello World!` in the console/terminal. `System.out.println()` is used to print a message in the console. Hence, modify the program as shown below:

```
class HelloWorld
{
      public static void main(String[] args)
      {
            System.out.println("Hello World!");
      }
}
```

When you compile and run the above program, it should print `Hello World!` in the terminal. In Java, you have to enclose a message within a pair of double quotations and put the message inside the parentheses of *println()* method to print the message. Remember that `System.out.println()` can be used to print a blank line in the output window when there is no message inside the parentheses. We can write multiple print statements inside the main method. You can also use `System.out.print()` to print messages. Java will print the next message in the same line when you use `System.out.print()` instead of `System.out.println()`. For example, replace the *HelloWorld* program

Now, Modify the above *HelloWorld* program with the code given below and compile and run to see how the print statements work.

```
class HelloWorld
{
      public static void main(String[] args)
      {
            System.out.println();
            System.out.println("Welcome to 115!");
            System.out.println();
            System.out.print("We will be learning object ");
            System.out.print("oriented programming using");
            System.out.println("Java in this course.");
            System.out.println();
            System.out.print("The more you practice, the ");
```

```
            System.out.print("more you will enjoy the course!");
            System.out.print("Good luck!");
            System.out.println();
        }
}
```

Understanding code indentation is key to writing clean and readable code. Please read this document to learn how to indent java code.

3. Writing first java program

We know how to print a message in Java. However, we have to be careful when we would like to print certain characters. For example, double quotation marks are used to represent a string/message in Java and hence, double quotations are treated as special characters and we cannot directly print double quotation marks. We use a backslash (\) to print a double quotation mark (") where the backslash acts as an escape character. In many programming languages, including Java, an escape character is used to indicate that the character following the escape character should be treated specially or differently than its literal meaning. In Java (and in many other programming languages), the backslash (\) is used as an escape character.

When we want to include certain special characters within a string, such as double quotation marks ("), single quotation marks ('), or newline characters (\n), we need to escape them using a backslash. For example, the following print statement will print a double quotation.

```
System.out.println("\"");
```

Similarly, backslash (\) is also treated as a special character in Java. We have to put another backslash as an escape character if we would like to print a backslash. For example, the print statement written below prints a backslash.

```
System.out.println("\\");
```

Now, write a java program in *CatFace.java* that prints the following cat face in the console

```
   /\_/\
  /  o o \
(     "   )
  \~(*)~/
   //  \\
```

What to hand in

Upon completion of your laboratory, you have to upload the following java files:

- Last modified *HelloWorld.java*
- *CatFace.java*

Create a folder by your last name, copy your files inside the folder. Then create a zip file(your_last_name.zip) and upload the zip file in the moodle. Before you upload, check with one of the TAs to make sure that your programs are working perfectly and you uploaded the correct files.