COMS2013: Mobile Computing

Laboratory 1

1 The Linux Command Line

In this section some of the basic commands used in the linux terminal are outlined.

- `ls` - Lists files in a directory. An example of the output of `ls` is shown below.

  ```
  Desktop    Downloads  Pictures  Public     test.txt
  Documents  Music      projects  Templates  Videos
  ```

- `cd` - Changes to the specified subdirectory *folder*. In the example above, you could type `cd projects` to move to the projects folder. Your shell shows you the current path as shown below:

  ```
  pravesh@pc81 ~/projects $
  ```

  In order to go back to the parent folder, you can type `cd ..`

- `mkdir` - Makes a subfolder with the specified name. You can then change to the created subfolder with `cd`

- `mv` - Moves a specified file to a specified folder. In the example shown for `ls`, typing the command `mv test.txt projects` would move the file `test.txt` to the `projects` folder. Typing `mv test.txt ..` would move the file `test.txt` to the parent folder.

- You can use the / character to refer to a path. Assuming you were in the `Pictures` folder, in which you have a file called `3.jpg`, you could move the file to the `projects` folder by typing:

  ```
  mv 3.jpg ../projects
  ```

  `../projects` indicates that it has to move up a folder and then down into the projects folder.

Try out these commands by opening a terminal and traversing the folder structure.

2 Compiling and Running

1. Download the files `Examplea.java` and `Exampleb.java` from Moodle.
2. Open a terminal, create a folder called `labs`, and go to that folder.
3. In the `labs` folder, create a folder called `lab1`.
4. In the lab1 folder, create a folder called `example`
5. Copy the two downloaded files into the `example` folder.

6. In order to compile a file, we use the javac command. As an example, to compile Examplea.java, we would type the following command: `javac Examplea.java` in a terminal, making sure that we are in the `example` folder.

7. In order to run a java program, we invoke the java virtual machine using the java command. As an example, to run `Examplea`, we would type: `java Examplea`

8. Now compile `Exampleb.java` and run the resulting program.

3 Introduction to Java

While an IDE such as Netbeans or Eclipse is a great way to program, it is important to know how to program without one.

1. Make a folder in your `lab1` folder called `lab1a`

2. Next, open a normal text editor. To launch gedit, either open Text Editor from the applications menu or type the following command in a terminal: `gedit &`

3. In the resulting window, type the code of the program below:

```
public class Program{
   public static void main(String args[]){
      double pi=3.1415, area, volume;
      int r=5;
      area = pi * r * r;
      double height = 24;
      volume = height * area;
      System.out.println("The volume is : " + volume);
   }
}
```

4. Make sure you save this file as `Program.java` in your `lab1a` folder. Note that the java file must always have the same name as the class it contains

5. Now run this program using the method you used for `Examplea` and `Exampleb`

6. For the moment, you can think of a **class** as a java module. This file then defines a java module called Program (`public class Program`), and must always be in a file called `Program.java` as a result. The program has a **method** (which is the Java terminology for a function) called `main` which is always the method that java looks for when you run the class using the `java` command. This means that when you type `java Program`, the Java interpreter will look in the `Program` class for a method defined as `public static void main(String args[])` and will call it.

4 Creating zip files to submit to the marking system

1. Since java is object oriented, it's quite possible to use multiple files to represent one program. To make submission easier in this case, we need to package our files into a zip file.

2. Note that in order for the Java Virtual Machine to know which file to execute when there are many files, we need to define which file is our main class, which is the one containing a main method. In order to simplify this process, the marking system will look for a file called `Program.java` unless stated otherwise in your lab sheet

3. In a terminal, go to the `lab1a` folder that you previously used to compile the `Program.java` file.

4. In this folder, type the following command to zip up all the java files into a zip file called `submit.zip`

   ```
   zip submit.zip *.java
   ```

5. Now try submitting this zip file to the Moodle system.

6. After submitting the file, try hitting refresh. If your submission was successful, the submission status should change to `Accepted`

5 Lab1b - Input and Output

- When making this program, it may help to look at the example code you had in `Lab1a`, `Examplea` and `Exampleb`

- In your `lab1` folder, create a folder called `lab1b`

- In the `lab1b` folder, edit a file called `Program.java`

- Remember to call your class `Program`

- Note that your code needs to be placed inside a function called `main`, which is declared in the following way:

  ```
  public static void main(String args[]){
     //code goes here...
  }
  ```

- This class needs to read a person's name as a String from the keyboard, using:

  ```
  Scanner in =new Scanner(System.in);
  String name = in.nextLine();
  ```

- You can think of the first line (`Scanner in =new Scanner(System.in);`) as creating a helper module for you that will handle reading input. In a way, this is like summoning a genie called `in`, who will then be waiting for commands. The second line (`String name = in.nextLine();`) then tells `in` to read a line from standard input and store it in a String object.

- If you then wanted to read another line, you do not need to create another `Scanner` using the first line, you can instead just tell the same `Scanner` to read another line for you using the second line.

- It must then output a greeting to the person, saying `Hello, person`, using the `System.out.println()` function.
- Package this up in a zip file and upload it to the marking system.

5.1 Example input

```
Pravesh
```

5.2 Example output

```
Hello, Pravesh
```

6 Lab1c

- In your `lab1` folder, create a folder called `lab1c`
- In the `lab1c` folder, edit a file called `Program.java`
- Remember to call your class `Program`
- This class needs to read an integer from the keyboard, using:

```
Scanner in =new Scanner(System.in);
String s = in.nextLine();
int i = Integer.parseInt(s);
```

- The third line finds the integer equivalent of String `s`, which it stores in variable `i`
- It must output `true` if the number entered is a multiple of 3, and `false` otherwise.
- (Hint : Use the modulus function (%) which returns the remainder of an integer division)
- Package this up in a zip file and upload it to the marking system.

6.1 Example input

```
4
```

6.2 Example output

```
false
```

7 Lab1d

- In order to do this lab, you need to know some methods that the String class has. In order to find this information, check the Java API documentation, which can be found at

  ```
  http://docs.oracle.com/javase/7/docs/api/
  ```

- As an example, take a look at the `endsWith` function. According to the API documentation, it takes in a String and returns a boolean, returning `true` if the String ends with the specified suffix. A code snippet is presented below as an example.

  ```
  String myString = "This is not a drill";
  boolean myBool = myString.endsWith("ill");
  ```

  The code above would set myBool to `true`, as it called the `endsWith` function on the `myString` variable, sending `ill` as the parameter for the function. This means that the String we are checking is `myString`, and we are checking whether it ends with `ill`, which it does.

- Now, In your `lab1` folder, create a folder called `lab1d`
- In the `lab1d` folder, edit a file called `Program.java`
- Remember to call your class `Program`
- This class needs to read in a String from the keyboard, and return the second character of that String.
- Read through the functions that the String class has in order to figure out which one you need. All of these can be found in the API documentation.
- Package this up in a zip file and upload it to the marking system.

7.1 Example input

```
This is a test
```

7.2 Example output

```
h
```