COMP3310/etc – Programming Warm Up

Outline

The goals of this tutorial are:

• Make sure you have the necessary software for this course.

• Make sure you can compile and run programs from the command line.

• Introduce you to Inter Process Communication and the problems that can arise.

You don't have to finish this tutorial immediately. If you don't have the right software, or don't understand how to do something, you can catch up. Ask your tutor or your fellow students for help.

1 Software

Tutorials are designed and tested on **`Linux`**, as used in the CSIT Lab workstations. Usually everything will work on **`MS Windows`** or **`MacOS X`** (x86) without change, or with minor modifications.

`Python` and **`Java`** are the "official" programming languages for this course. Most tutorials are supplied in both these languages, and most students use one or the other for their assignments. Most tutorials require only the standard built in libraries.

For Python any version from about 3.7 on will work. You won't need type annotations, match-case statements, async code or other features from newer releases.

For Java the minimum is JDK 17, the long term release. You won't need virtual threads, records, string templates, or other new features.

Tutorials are designed for a command line environment. On Linux and MacOS this is the **`Terminal`** application. On MS Windows, use **`PowerShell`**, *not* the command prompt. We won't be writing shell scripts and usually the same commands work in all three systems.

Why a CLI, Command Line Environment? This is not to make things deliberately hard, or trying to bring back the computers of the previous century. Most networked computer programs, and most computer network administration, does not use a GUI. Anyone developing for the cloud (Azure, AWS, etc) will be using the command line. This course is about networks, not GUI programming.

You will need some way to edit your Python/Java programs. Any text editor will do: **`Sublime Text Editor`** or **`Atom`** are good choices on any system, **`BBEdit`** on Macs. You *can* use an IDE such as **`Eclipse`**, **`PyCharm`**, or **`Visual Studio`** if you prefer, but you *must* be able to run your programs from the command line. Assignments are submitted in source code form to your tutor, who probably won't have the same IDE as you. If your programs can't run from the command line, they won't be marked.

You need to know how to decompress / extract **`ZIP`** archives, as later on you will be submitting your assignments in ZIP form. ZIP, *not* 7z or RAR or tar.

Please note that all the files in these tutorials have names with just letters, numbers, underscores, and dashes. *Don't* put spaces or funny characters in file and folder names that will be used in a CLI.

2 Compile and run a program

> *Do, or do not. There is no try – Yoda.*
>
> *The tutorial exercises in this course are practical, not theoretical. If you think you already know everything written here, do it anyway. Just to make sure.*
>
> *If you can't do everything in one tutorial, that's OK too. The goal is to find out what you need to learn before starting the actual tutorials and assignments.*

The first program is **`peasant.py`** or **`Peasant.java`**. You can run from the command line with

```
python peasant.py
...
```

or

```
javac Peasant.java
java Peasant
...
```

> *This tutorial shows the actual commands you should type. In future we will just ask you to "compile and run the program" without this much detail.*

This program expects you to type something and press Enter, which will then be printed back to the terminal. It will keep doing so until End Of File, which in a terminal is you typing **`Control-D`**, or **`Control-Z`** then **`Enter`**, depending on your system. If you don't already know, find out which stops the program on your system.

The peasant program (Python or Java) has one optional command line argument. In the source code this is handled in **`processArgs`** and you can see the effect by running the program again. Some people prefer to study the code and then run the program; other people prefer to run the program first and then look at the code. There is no one right way.

```
python peasant.py Dennis
```

or

```
java Peasant Dennis
```

> *Don't worry about the name of the program or the command line argument. This is an old joke, because the tutorial was written by an old person.*

What happens if you type more than one name on the command line?

3 Second program

The second program is similar, in that it reads input and writes output until EOF, but is not identical. Open the source code in your editor / IDE and run the program.

```
python knight.py
```

or

```
javac Knight.java
java Knight
```

This program has new **`chooseResponse`** code that changes the output depending on the input. One special input will cause the program to crash, another will cause the program to loop forever. Read the code to find out what these strings are. Run the program a few times and try both values. If you don't already know, you can usually stop a runaway program with **`Control-C`**. (On MS Windows, first try **`Control-C`** and if that doesn't work, **`Control-Pause/Break`**).

4 Inter Process Communication

Send the output of the first program as the input to the second with command:

```
python peasant.py | python knight.py
```

or

```
java Peasant | java Knight
```

This is IPC, Inter Process Communication, one program communicating with another, although here without a computer network.

Type a few lines. Try the special input that crashes the knight program. (You may need to EOF as well.) What happens? Do both programs stop, or do you need to type **`Control-C`** as well?

Reverse the program order:

```
python knight.py | python peasant.py
```

or

```
java Knight | java Peasant
```

Now what happens if you type the special crash input?

5 Coding

Can the Java and Python programs interoperate? Use the IPC command, but instead of both being Java or both being Python, try one of each.

Modify one program to stop on an empty line, ie pressing Enter without any text.

Put the knight program into an infinite loop. Instead of Control-C, use the **`ps`** and **`kill`** commands (Linux and MacOS) or **`Task Manager`** (MS Windows) to stop it. If you don't know how to do this, now is a good time to learn.

Optional: Modify the peasant program to join all the command line arguments together (with a space in between each) to form a more complex name.
