Working with Multiple Files

In Java, each public class resides in its own file. As each program is usually composed of many classes, this means you will have many different files.

Compiling Multiple Files

In order to compile the files A.java, B.java and C.java, you can use the command:

```
> javac A.java B.java C.java
```

Alternatively, in order to compile all the files with the extension `.java` in the current directory, you can use:

```
> javac *.java
```

Or, if all source files are at a `src` sub-directory (as should be the structure of your projects):

```
> javac src/*.java
```

Compiling like is done above will put the compiled `.class` files alongside the source `.java` files. It is sometimes convenient to separate the compiled files from the source files. You can use the `-d` switch to tell the compiler to put the class files in a specific folder.

```
> javac -d bin src/*.java
```

will tell the compiler to create the `.class` files in the `bin` directory.

Compiling and Running Inter-Dependent Classes

The above commands will serve you well in cases where each class is standalone, and no class uses any other class (as you had for Assignment-1). However, this is far from being the common case.

Let's say we have a class `A` with a `void main(String[] args)` we want to run. When doing its work, class `A` would probably want to "communicate" with other classes (for example `B` and `C`) - that it, to use them in its code.
This is not a problem, as long as java knows where the files for the `B` and `C` classes are.
By default, java looks for the class files in the current directory.
You can tell java (and javac) to look at other directories as well, using the `-cp` (=class-path) commandline switch, as illustrated in the following example:

Compiling

```
> javac src/A.java -cp src
```

This tells the java compiler (`javac`) to compile the class written in `src/A.java`. If this code is assuming the existence of other classes (e.g. it instanciates a `B` object with `B b = new B()`), javac will search the required `.java` files (e.g. `B.java`) in the class-path. If the compiler can't find `B.java`, compilation would fail with a `cannot find symbol` error message -- it just does'nt understand what your code means by `B`.

Running

```
> java A -cp bin
```

This tells the Java Runtime Environment to load the class `A` and execute its `main`. Similar to what we had with javac, the JRE searches required `.class` files in the class-path. So `A.class`, along with the `.class` files of any class it uses, should be in the class-path (here, `bin`).

So what should I run?

Taken together, in your assignments you should compile and run with the following commands (assuming you are in the assignment's *root directory*, you have *biuoop-1.4.jar* in it, and all `.java` files are in `src`):

```
> javac -d bin -cp biuoop-1.4.jar:src src/*.java
> java -cp biuoop-1.4.jar:bin MainClassName
```

> ***Note**: In Windows, the separator in the `-cp` argument is `;` instead of `:`, so your lines should look like: `java -cp biuoop-1.4.jar;. SimpleGuiExample`.*
>
> *These differences are part of the reason we use Ant, so that we have a unified cross-platform configuration for building and running your code that would work anywhere.*
>
> *Ant will also take care of these compiling and running tasks for you, so that eventually you can simply type `ant compile` and `ant run` instead of the complicated commands we presented here. However, we **strongly encourage** you to also play around with the raw `javac`/`java` commands in order to gain a good understanding of the Java environment.*