How should your project look like

You are not required to define and use packages throughout your exercises. However, since your code will get more and more complex, containing few dozens of classes, you may want to organize your classes into packages, as programmers do "in the real world".

In Java, the packaging system is intertwined within the directory hierarchy of your project. In other words, the directory hierarchy corresponds to- and defines the packages. Both your code files (`.java`) and your binary files (`.class`) must reside in a directory hierarchy that reflects the package names you declare and use in your Java code.

A package is basically a directory containing classes or other packages. The code files in that directory must declare that they belong to the package, using the statement `package package_name;` in the first line of the file. `package_name` must also be the name of the directory containing this file.

The package name may also reflect a deeper hierarchy - for example the package `gui.shapes.squares`, containing the classes `Square` and `Cube`, reflect the following directory hierarchy:

```
> gui
   > shapes
      > squares
         > Square.java
         > Cube.java
```

Using classes from other packages

Now, the fully-qualified name of a class includes its package full name, e.g. `gui.shapes.squares.Square`. When using the Square class in a code that is not part its package, we must use its fully-qualified name or import it first. Read more about [[Using Packages]].

`.java` files or `.class` files?

The above-mentioned directory hierarchy requirement for Java packages holds both for your source code files (`.java`) and for your bytecode files (`.class`). The Java compiler, which reads and translates `.java` files, expects this correspondence between the directory hierarchy and the package names in order to find the code for the classes you are using; and same is true for the Java Runtime Environment (JRE) - the program running your bytecode - which reads and executes `.class` files.

There are two conventional ways of achieving the same directory hierarchy for `.java` and `.class` files. One option is to place the bytecode for each class within the same directory as its source code. The second option is to have two directories - typically `src` for source code and `bin` for bytecode - which both correspond to the same hierachy.

Compiling projects with a package structure

When using packages, the use of `*.java` in the compilation command is no longer enough. To avoid listing all the java files manually, the compiler provides a way to get the list of files it needs to compile from a file. Combining this together with a handy **linux `find` command**, creates a convenient way of compiling all files, without specifying the internal structure of our source folder. Here is an example of how the combination of these commands will look:

```
# write into sources.txt, the path to all java files inside src folder and any sub folder it may contain
find src -name "*.java" > sources.txt
# compile all files that are listed in sources.txt file
javac @sources.txt
```

This will place the generated bytecode side by side with the source code (the first option mentioned above), which may be inconvenient and messy. We recommend using the second option, which only adds specifying a destination folder:

```
# compile all files that are listed in sources.txt file into 'bin' directory
javac @sources.txt -d bin
```

This will result in the following project structure (using our example from above):

```
> ass99 (root directory of the project)
   > src
      > gui
         > shapes
            > squares
               > Square.java
               > Cube.java
   > bin
      > gui
         > shapes
            > squares
               > Square.class
               > Cube.class
```

Note: Remember that you still need to provide the other compiler flags.