Class declaration

Writing a Java class source file

Elements in a Java class source code file

These are the elements which can occur in a Java source code file for a class:

- A package declaration - declares logical association expressed as a package (which corresponds to the path to the class file)
- import statements (if needed - paths to stuff used in this class)
- Class declaration with the class name and the block of code for the class
- Variables for keeping the state of objects created from this class (instance variables)
- The variables shared by all instances created from this class (static variables)
- The constructor(s) providing means to create instances from this class
- The methods of instances of this class - what objects of this class can do

Elements in a Java class source code file (cont.)

- The methods which exist independently from instances of this class (methods which don't involve instances of this class, static methods)

Actually also:

- Nested and inner classes - a class definition can actually contain other classes (this is mostly outside of the scope for this book)
- Initializers - you can actually put blocks of code directly in the class body (this is outside of the scope for this book)

Package declaration

If we have the following directory structure:

```
.
`-- org
    `-- progund
        `-- Game.java
```

Then we could say that the Game class belongs to the package

```
org.progund
```

The package declaration is the first statement

The source code for Game would then start with the following statement:

```
package org.progund;
```

The source code file would then be in the following path:

org/progund/Game.java

And after compilation, the class file would end up in the same path:

org/progund/Game.class

(doesn’t have to be the same directory, but the same path!)

Possible division of source and binaries

```
.
|-- bin
|   `-- org
|       `-- progund
|           `-- Game.class
`-- src
    `-- org
        `-- progund
            `-- Game.java
```

The first path is relevant for java (when running) and the second for the compiler, javac .

Compiling with destination, running with classpath

To compile source and putting the class file in a new directory structure:

```
$ javac -d bin src/org/progund/Game.java
```

To run a Java program (class file with main) from a directory above the classes directories:

```
$ java -cp bin org.progund.Game
```

Advantages of having packages

Having directories for related classes allows for a good structure and a logical division of your source code files. You will also get a way to keep class names short, and let the package be part of the full name.

In the Java API, there are two classes representing “Dates”:

`java.util.Date`	(general representation of a Date)
`java.sql.Date`	(a date as a datatype for databases and the SQL language)

The nameless package

If you don’t put your source code in a package, the class declared will actually belong to a package anyway. This is sometimes called the nameless package and it corresponds to a directory path similar to “.” (current directory).

While this is common in small stand alone example programs in books and education, it is neither common in the real world, nor is it recommended.

We like to organize files in folders and source code files are no exception.