Import Declarations and Type Search Order

Import declarations are used to resolve simple names of types to their fully qualified names during compilation. The compiler uses predefined rules to resolve the simple names. Suppose the following statement appears in a Java program that uses a simple name A:

```
A var;
```

The Java compiler must resolve the simple name A to its fully qualified name during the compilation process. It searches for a type referenced in a program in the following order:

- The current compilation unit
- Single-type import declarations
- Types declared in the same package
- Import-on-demand declarations

The above list of type search is not complete. If a type has nested types, the nested type is searched before looking in the current compilation unit. I will defer the discussion of nested types until inner classes are discussed in Chapter 2 of the book Beginning Java Language Features (ISBN: 978-1-4302-6658-7).

Let's discuss the rules for a type search using some examples. Suppose you have a Java source file (a compilation unit) B.java whose content is as follows. Note that the file B.java contains declarations for two classes A and B.

```
// B.java
package p1;

class B {
    A var;
}

class A {
    // Code goes here
}
```

Class B refers to class A with its simple name when it declares an instance variable var of type A. When the B.java file is compiled, the compiler will look for a type with the simple name A in the current compilation unit (B.java file). It will find a class declaration whose simple name is A in the current compilation unit. The simple name A will be replaced with its fully qualified name p1.A. Note that both classes A and B are declared in the same compilation unit, and therefore they are in the same package, p1. The class B definition will be changed as follows by the compiler:

```
package p1;

class B {
        p1.A var; // A has been replaced by p1.A by the compiler
}
```

Suppose you want to use class A from package p2 in the previous example. That is, there is a class p2.A and you want to declare the instance variable var of type p2.A in class B instead of p1.A. Let's try to solve it by importing class p2.A using a single-type-import declaration, like so:

```
// B.java – Includes a new import declaration
package p1;

import p2.A;

class B {
        A var; // We want to use p2.A when you use A
}

class A {
        // Code goes here
}
```

When you compile the modified B.java file, you will get the following compilation error:

```
"B.java": p1.A is already defined in this compilation unit at line 2, column 1
```

What is wrong with the modified source code? When you remove the single-type-import declaration from it, it compiles fine. It means it is the single-type import declaration that is causing the error. Before you resolve this compiler error, you need to learn about a new rule about single-type import declarations. The rule is

> *It is a compile time error to import more than one type with the same simple name using multiple single-type-import declarations.*

Suppose you have two classes, p1.A and p2.A. Note that both classes have the same simple name A placed in two different packages. According to the above rules, if you want to use the two classes, p1.A and p2.A, in the same compilation unit, you cannot use two single-type-import declarations.

```
// Test.java
package pkg;

import p1.A;
import p2.A; // A compile-time error

class Test {
        A var1; // Which A to use p1.A or p2.A?
        A var2; // Which A to use p1.A or p2.A?
}
```

The reason behind this rule is that the compiler has no way to know which class (p1.A or p2.A) to use when you use simple name A in the code. Java might have solved this issue by using the first imported class or last imported class, which might have been error prone. Java decided to nip the problem in the bud by giving you a compiler error when you import two classes with the same simple names, so you cannot make silly mistakes like this and end up spending hours resolving them.