ESS 201: Programming in Java

Week 3
Term 1: 2018-19
T K Srikanth

Hiding the implementation

“separating the things that change from the things that stay the same.” - Bruce Eckel, Thinking in Java

The public methods that are used to create/access/modify objects should largely remain the same, although the implementation (including any other classes internal to this implementation) will likely change over time.

Access specifiers are one way to enable this separation.

Access specifiers

classes, methods and data fields can have specifiers added to them that control who all (which other classes) can access them

public, private (and later, **protected**, as well as a *default* (i.e. no specifier)) access

public: visible to all other classes

private: only visible from within (methods or class definition) of the same class

Does a *private class* make sense?

Public/private members

Visibility of members (methods or data) of a class A from within class B:

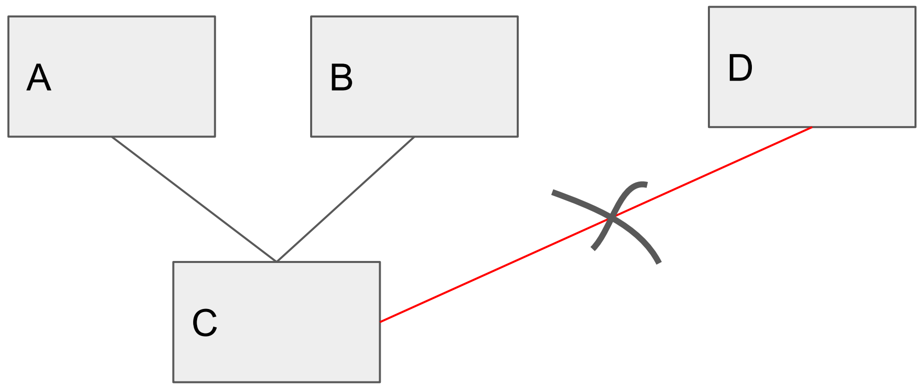
Specifier (in class A)	For class A	For class B
public	Y	Y
private	Y	N

Other controls of visibility

Public/private specifiers provide granular access to a class or its methods.

Need a mechanism to provide “partial” visibility

- A class (or a subset of its methods) visible to one set of classes but not to others



E.g. An algorithm that is needed by 2 or more developers of one team, but don’t want to expose it to others.

Packages

A mechanism for grouping classes together - typically a logical set of modules or library

Provides a *namespace*

- Can have the same class name in different packages
- Resolved by using <packageName>.<className>

Also provides another level of control over access

Package

Define a class (or rather the classes in a file) as belonging to a package p1 by adding the line

```
package p1;
```

as the first line of the file

By adding this in multiple files, all the classes in these files all now belong to the same package p1. (A file - i.e. the class in it - can be part of only one package)

Other packages import this using

```
import <package>.* ;
```

Not specifying a package name for a file implies it is part of the *default* package

Package

Packages can be nested into a hierarchy. So, we can have

```
package graphics.shapes;
```

Or

```
package a.b.c.d;
```

Source files are organized into directories that reflect this structure.

To avoid ambiguity, need a unique prefix for the package path, so that there is no chance of conflict. *What's a good scheme for this?*

You can import specific parts of packages with

```
import package.subpack.*
```

Package access specifiers

Package access is the default for class/method/data fields etc.

That is, unless specifically specified as public or private (or protected), access is provided to all other classes of that package - and to no other. Also called *package-private*

```
package aOne;
public class A {
   public A() {}            // visible to all classes
   f1() {}                  // visible to all classes of this package aOne
   private f2() {}          // visible only from within this class
}
```

Public/private and package(default) access

Visibility of members (methods or data) of a class A from within class B (both in package aOne) or class C (in package cOne):

	package aOne		package cOne
Specifier (in class A)	For class A	For class B	For class C
public	Y	Y	Y
<package-private>	Y	Y	N
private	Y	N	N

Same applies to classes in one package as seen by classes of the same or other package.

Constructors

Can we have constructors that are not public? Why would we need them.

- Package-private construcors?
- Private constructors?

Some common packages in Java

java.io	Provides for system input and output through data streams, serialization and the file system.
java.lang	Provides classes that are fundamental to the design of the Java programming language.
java.math	Provides classes for performing arbitrary-precision integer arithmetic (BigInteger) and arbitrary-precision decimal arithmetic (BigDecimal).
java.net	Provides the classes for implementing networking applications.
java.sql	Provides the API for accessing and processing data stored in a data source (usually a relational database) using the Java^{TM}programming language.
java.text	Provides classes and interfaces for handling text, dates, numbers, and messages in a manner independent of natural languages.
java.util	Contains the collections framework, legacy collection classes, event model, date and time facilities, internationalization, and miscellaneous utility classes (a string tokenizer, a random-number generator, and a bit array).

https://docs.oracle.com/javase/7/docs/api/overview-summary.html