Java modifiers

Some of the material today is taken from the Readings BackgroundJava.pdf which I posted early in the course. I have added further details, some of which involve concepts from inheritance.

Packages

A package is a particular set of classes. Go to the Java API

http://java.sun.com/j2se/1.5.0/docs/api/

and notice that in the upper left corner of that page is a listing of dozens of packages. A few of them that you are familiar with are:

- java.lang - familiar classes such as Object, String, Math, ...
- java.util - has many of the data structure classes that we use in this course, such as LinkedList, ArrayList, HashMap

The full name of a class is the name of the package following by the name of the class, for example java.lang.Object or java.util.LinkedList. You can have packages within packages e.g. java.lang is a package within the java package.

You can also make your own packages, of course. For example, the classes Beagle and Dog might belong to package comp250.lectures, and so the full name of the classes would be comp250.lectureExamples.Beagle and comp250.lectures.Dog.

Packages have a tree structure. Packages are directories, typically internal nodes]. The classes within a package are leaves. The issue of how file systems and packages are organized is important in practice, but I did not discuss it in class. If you want to learn about package trees and how they correspond to file directories (and to www URLs) then you read through the following:

http://cs.armstrong.edu/liang/intro6e/supplement/packages.pdf

Visibility Modifiers

You are familiar with defining classes to be public and defining methods and fields to be public or private. You learned, for example, that fields in a class are usually defined to be private, and the fields are accessed using public getter and setter methods.

The question of public vs. private gets a bit more complicated when you consider inheritance relationships and packages.

Visibility modifiers for classes

If you want a class to be visible to all other classes, regardless of which package the other classes is in, declare the class public. If you want a class B to be visible only to class A, then define B inside A – so that B is an inner class or nested class of A – and declare class B to be private. This is only way it makes sense to have a private class.

[1]the only exception being if you have an empty package in which case it would be a leaf
Class visibility modifiers also determine whether a class can extend another. If a class A is declared without a modifier (package visibility) then this class can only be extended by a class B within the same package. If you want a class B to extend a class A from a different package, then A can be defined as public.

In fact, there is one other possibility, namely to define A to be protected. This allows A to be visible to an class in the same package (default package visibility), but it also allows A to be extended by classes in other packages. I will not discuss the protected modifier further in these lecture notes. You don’t need to know the rules of this modifier for the final exam.

Visibility modifiers for fields and methods

Let’s next consider modifiers for class fields and methods. Let’s deal with methods first. Methods contain instructions which invoke methods. Suppose that method mA is within class A and method mB is within class B and that class A is visible to class B. (Note that visibility is not a symmetric relationship. Its possible for class A to be visible to class B but for B to not be visible to A, e.g. A might be public and B might be package, and A and B might be in different packages.

For an instruction in method mB to invoke method mA, method mA in class A needs to be visible to class B. For this, it is necessary that class A is visible to class B. But this is not sufficient, since we also require that the method mA itself is visible. So we have a situation as follows. Note that I am not specifying the visibility of method mB here since it is irrelevant to the current question.

```java
void mB( A v ){
    v.mA(); // or attempt to access field v.fA in class A
}
```

The left column of the table below shows the four possible method visibility modifiers for method mA or field fA in class A. The other columns consider whether the compiler allows mB to invoke method mA, as in the code above. A 1 means yes and 0 means no.

<table>
<thead>
<tr>
<th>mA/mA modifier</th>
<th>A &amp; B same class</th>
<th>A&amp;B same package, but</th>
<th>A and B in diff packages</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>same class</td>
<td>diff classes (even if B is is subclass of A)</td>
<td>diff packages</td>
</tr>
<tr>
<td>public</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>(package)</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>private</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

One possible source of confusion is that these visibility relationships are between classes. (This is easy to understand if we keep in mind that these visibility relationships as defined at compile time, not at runtime.) For example, suppose the Dog class has a private field:

```java
private String name
```

Then any methods that are defined with the Dog class can use this field. Methods defined within the dog class do not have to use the getName() or setName() methods. For the example the Dog class might have a method
by which a Dog could say its own name or the name of another Dog. Such a method can be inherited by a subclass e.g. Doberman, even though the instructions in the method use the private field of Dog. This might be counter-intuitive, since any method that is defined explicitly in the subclass (say Doberman) is not allowed to explicitly use private fields like name from the superclass Dog and instead it must use the getter/setter.

Use modifiers

static

The modifier static specifies that a variable or method is associated with the class, not with particular instances. It is possible to define a static class, but only in the special case that the class is an inner class.

More often, the static modifier is used for methods and fields. It is often used, for example, to keep track of the number of instances of a class.

```java
static int numberOfObjects; // A constructor would increment
static int getNumberOfObjects(){ // this variable.
    return numberOfObjects;
}
```

Another example of a static method is main(). Note that you invoke this method when you “run the class” and you do this without instantiating the class.

A static method or field is often called a “class method” or field. Examples of static methods are sin() or exp in the java.lang.Math class. Note that to use such methods, in your class definition you would specify that you are using the package java.lang and then you would invoke the method just by stating the classes name and the method e.g. Math.cos( 0.5 ). Note that the class name is used instead of a reference variable (to an object) which is why we say that static methods are “class methods” rather than “instance methods”.

final

The final modifier means different things, depending on whether it is used for a class, a method, or a variable.

- public final class A - means that the class cannot be extended (a compiler error results if you write B extends A.)
- final double PI = 3.1415. The value cannot be changed. Note that you need to assign a value for this to make sense.
- final int myMethod() - means that the method (which happens to return an int) cannot be overridden in a subclass.

String and Math are example of a final class.