Inner Classes, part 1

An inner class is a class defined inside another class.

A class inside a class is a member of the containing class. Just like methods and fields, it can be public or private (or package or protected), and static or non-static.

Static Inner Classes

Please see the class example List1 for an example of our linked list implementation, but with the linked list node and iterator as static inner classes.

The inner class is public to make it easy to demonstrate how inner classes work, but normally we would make it private because we do not want someone

using the linked list class to be able to access the individual nodes.

Note that, becaue the inner class is static, we need to define a generic for the inner class. We could have used T, but I chose to instead use E for the Node class in order to be

clear that the inner class and outer class generics are different. Why do we need to define another generic? Because the inner class is static. The outer

class generic is only specified on instances of the outer class. The inner class belongs to the entire class, not to a specific instance of the outer class.

Because the inner class is static, an instance of the inner class belongs to the outer class, not an instance of the outer class. So, "this" refers to the

inner class instance.

To create a member of the inner class Node from outside the List1 class, we can use exactly the same Java terminology we use to create other objects and to

access static fields and methods:

List1.Node<String> node = new List1.Node<String>("Hi", null);

Note that the generic goes on the Node class. We do not need to specify the generic for List1 because we are not creating an instance of List1.

(It is not an error to specify the generic for List1, but Java will ignore that generic since no instance is being created.)

Non-static Inner Classes

Please see the class example List2 for an example of our linked list implementation, but with the linked list node as a non-static inner class.

Now note that we do not specify a generic for the inner class.

This is because the inner class belongs to a specific instance of the outer class, and the outer class instance has already specified the generic.

In general, non-static inner classes can be more useful because they have access to all the non-static fields and method of the containing class, but they

can be a little trickier to use. We will do a little more on non-static inner classes in the next lecture.

To create a member of the inner class Node from outside the List2 clas, we do something similar as with the non-static inner class, but now we must create

an instance of the outer class first. Note the difference with the new operator and note that we now specify the generic with the outer class:

List2<String> list = new List2<String>();

List2<String>.Node node = list.new Node("Hi", null);

Anonymous Classes

An anonymous class is defined in a statement, wherever an instance of a class is needed.

The class has no name.

new InterfaceToImplement() { code to override the various methods here };

or

new ClassToExtend() { code to extend the class by overriding methods here };

Note, that you should not create new non-private methods when writing an inner class, but you should just override existing methods.

To see why, think about the polymorphism rules. What is the current-type of an instance of the anonymous class?

How does the compiler decide what methods are allowed to be called?

We created an anonymous class for the Iterator of List2.

public Iterator<T> iterator() {

return new Iterator<T>() {

/\* the methods of Iterator go here \*/

}; <- note the semi-colon terminating the return statement

}

Note that inside both non-static inner classes and anonymous classes, there are two this references.

One refers to the instance of the inner class, and one refers to the instance of the outer class.

this refers to the anonymous class this. To get the outer this, give it the type name: List2.this

Anonymous classes are very useful when we only need to create a single instance of a class and we are only overriding methods.

In these cases, why should we create a separate public class file when we can just define the class where needed in the code.