Object-Oriented Programming

Nested Classes

Dr. Seán Russell
sean.russell@ucd.ie

School of Computer Science,
University College Dublin

October 30, 2018

Learning outcomes

After this lecture and the related practical students should...

- understand the concept of nested classes
- be able to declare nested classes in Java
- understand the use difference between static nested classes and non-static nested (inner) classes

Table of Contents

Multiple Classes

- We can declare multiple classes in the same file
- These can defined one after another
- However, when a class is public then the name of the class must match the name of the file
- This means that any other classes that we declare in this way cannot be declared as pubic

Multiple Classes

```
1 public class FirstClass{
2     ...
3 }
4 class SecondClass{
5     ...
6 }
```

Multiple Classes

- This is useful for placing a lot of code together, but it makes understanding the code more difficult
- Normally, to find code for a class we only need to go to the correct file in the correct package
- With multiple classes in the same file, we have to search multiple files until I find the one containing `SecondClass`
- This is not a recommended way of structuring your code

Nested Classes

- A nested class is when one class is defined inside another class
- It is a way of logically grouping classes that are only used in one place:
  - If a class is used by only one other class, then it is good to embed it in that class and keep the two together
- Nesting such "**helper classes**" makes their package more streamlined

Nested Classes and Encapsulation

- Consider two classes, A and B, where B needs access to instance variables of A that would otherwise be declared private
- By nesting class B within class A, A's instance variables can be declared private and B can access them
- In addition, B itself can be private
- Nesting small classes within top-level classes places the code closer to where it is used

Categories of Nested Classes

- Nested classes are divided into two categories
  1. Nested classes that belong to the class
  2. Nested classes that belong to an instance of the class
- These are known as static and non-static nested classes
- Non-static nested classes are normally called inner classes

Table of Contents

Inner Classes

- An inner class must belong to an instance of the outer class
- This means that if we want to create an instance of the inner class, we must
  - Create it from inside a non-static part of the outer class, or
  - Have a reference to an instance of the outer class
- An inner class is used to define something that will only be used in one place

Inner Class Example

- Consider the `Node` class in most link based data structures
- They are used to store the relative positioning of elements within the data structure
- No other class need to know about this
- When we are using a link based stack, we only ever interact with the stack, never with a node directly

Inner Node Class Example

```
1  public class LinkStack implements Stack {
2
3    private class Node {
4      private Object element;
5      private Node next;
6
7      public Node(Object e) {
8        element = e;
9      }
10   }
11
12   private Node top;
13   private int size;
```

Example Explained

- This code declares a private inner class named `Node`
- This means that the `LinkStack` class can know what a `Node` object is but to any other class it does not exist
- Our node class cannot be confused with similar classes from another link based data structure
- Additionally, we know that the stack interface stores and returns objects
- To any other class using a link based stack object, they do not need to know how it is represented inside the class

Table of Contents

Accessing Data

- Because an inner class is non-static it can access the data of the class it is declared inside, even if the data is private
- To give an example of this, we are going to create an Iterator class for our link based stack implementation
- We are first, going to cover/revise what an iterator is

Iterators

- An iterator is an object that can be used to access all of the elements in a data structure one by one
- The `Iterator` interface is in `java.util`
- We need to implement two methods;
  - `hasNext` - returns a boolean value to tell us if there are any more items we have not accessed
  - `next` - returns the next item in the data structure that we have not already accessed

Iterator Example

If a data structure has three items, we can use an iterator to access them. The first time we call `next`, it will give us the first item, the second time we call next it will give us the second item and so on

Iterable

- A related interface to `Iterator` is `Iterable`
- This interface is generally implemented by a data structure
- The interface requires one method
  - `iterator` - Returns a `Iterator` object that we can use to view the objects in this data structure

Iterable Implementation Example

```
1 public Iterator iterator(){
2   return new StackIterator();
3 }
```

Inner Iterator Classes

- If we are designing an iterator for a link based implementation of the stack, we will need to know which Node we should return the data from
- Whenever we are asked for an item, we return the item and change the location where the next item will be taken from
- This means the for the link based stack we start at the `Node top` and then the next node and the next node
- This means we need to remember what the current node we are looking at is

Inner Iterator class for Link Based Stack

```
1  public class LinkStack implements Stack, Iterable {
2    private Node top;
3    private int size;
4    private class StackIterator implements Iterator{
5      private Node current;
6      private StackIterator(){
7        current = top;
8      }
9      public boolean hasNext() {
10       return current != null;
11     }
12     public Object next() {
13       Object o = current.element;
14       current = current.next;
15       return o;
16     }
17   }
18 ...
```

Accessing Data from the Outer Class

- Inner classes can access and change the variables of the outer class that created them or was used to create them
- This is not true for a static nested class
- In the last example, the constructor of the iterator could just copy the node that was at the top

StackIterator Constructor

```
1     private StackIterator(){
2       current = top;
3     }
```

- We do not need a reference to the outer class to do this

Table of Contents

Public Inner Classes

- Inner classes can be declared with any visibility
- Public inner classes can be constructed by any class to a reference to an outer class
- For example, if I have a reference to an `LinkStack` object and the `StackIterator` was public, then I could create an instance of the `StackIterator` class

Creating an Instance of an Inner class

```
1 LinkStack stack = new LinkStack();
2 LinkStack.StackIterator it = stack.new StackIterator();
```

- This is not a very common operation

Table of Contents

Anonymous Inner Classes

- If an inner class is only going to be used in a single place, then we have no need to give it a name
- This can only be done, if the class is to implement an interface or abstract class
- We can declare the class directly where it is used

Anonymous Inner Class Example

```
1 InterfaceName iN = new InterfaceName() {
2   // implementation !!!
3 };
```

- Important parts to notice are;
  - We need brackets before the class definition - `(){`
  - We need a semicolon after the class definition - `};`

Iterator as Anonymous Inner Class

- Typical examples of classes like this would be iterators and event listeners such as mouse and key listeners
- We could replace our iterator for the `LinkStack` class by declaring an anonymous inner class inside the iterator method of the LinkStack class

Anonymous Iterator Inner Class

```
1 public Iterator iterator() {
2   Iterator it = new Iterator() {
3     Node current = top;
4     public boolean hasNext() {...}
5     public Object next() {...}
6   };
7   return it;
}
```

Table of Contents

Static Nested Classes

- Static nested classes are slightly from inner classes
- A static nested class cannot access the instance variables of the outer class
- Static nested classes can be constructed without an instance of the outer class
- E.g. `OuterClass.InnerClass l = new OuterClass.InnerClass(parameters);`
- An example of a static nested class is `Double`, which is nested in the abstract class `Line2D` in the package `java.awt.geom`

Static Nested Classes Example

- The `Double` class is actually a subclass of the `Line2D` class, it provides an implementation of the abstract class where all of the information is represented as doubles
- There is another nested class called `Float`, which does the same but represented as floats
- These classes are useful for performing some calculations related to lines and line segments, which can be useful in the development of 2D applications or games
- To create an instance of this class we use the code

```
Line2D.Double l = new Line2D.Double(0,0,10,10);
```

Further Types

- In Java, there are also related topics such as Lambda expressions and local inner classes
- However, because this course is focusing on the key concepts in OOP, not just in Java, these topics are left to you to read about further