Fast Track to Java

Customized for Starbucks
Delivered by DevelopIntelligence

Inner Classes

Objectives

At the end of this module you should be able to:

- Understand the capabilities of inner classes
- Create code for inner classes and anonymous inner classes
- Recognize reasons for using inner classes

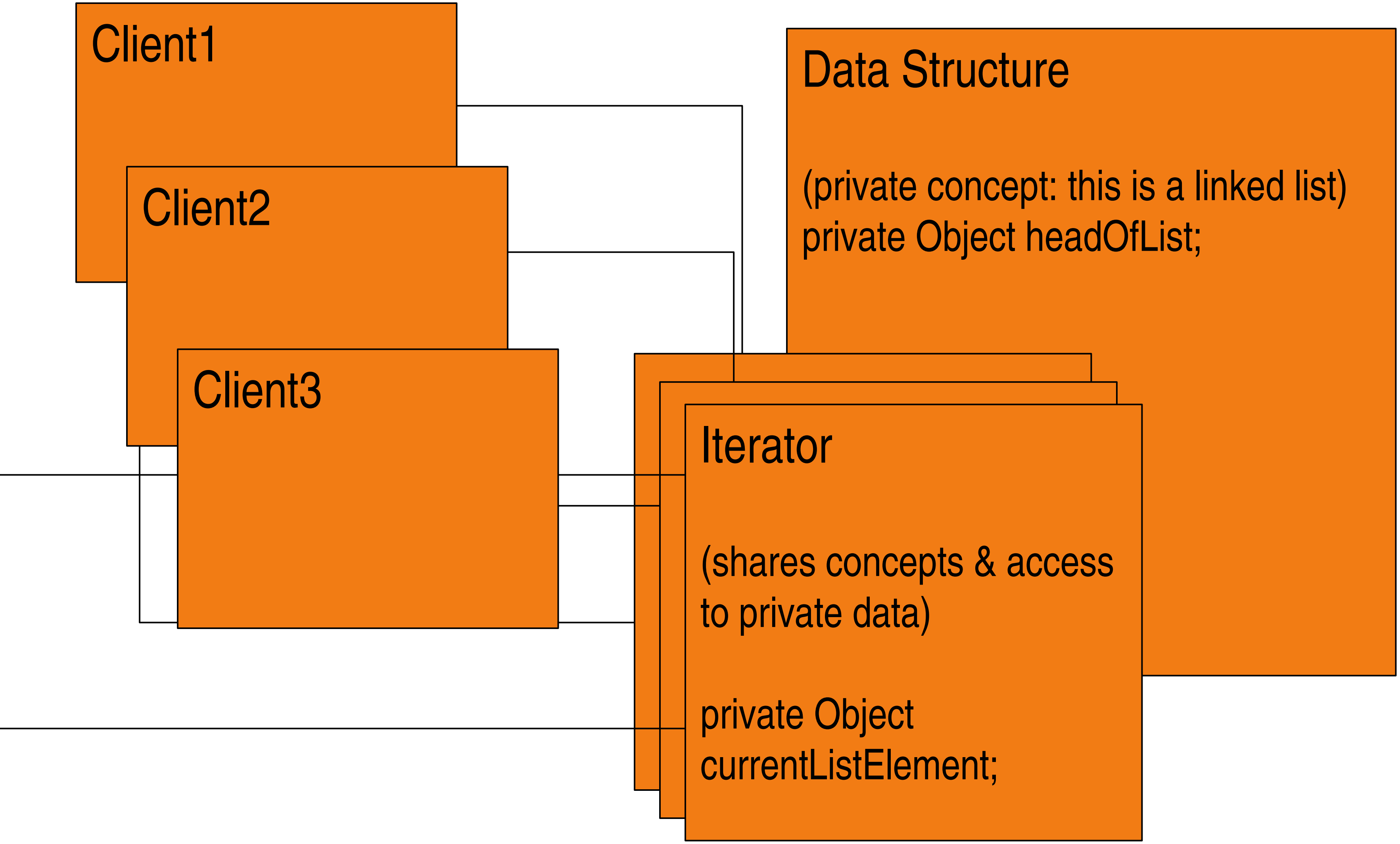
A Design Problem

- A data structure implements the List interface
- Clients of the structure wants to iterate over the contents
- Each client wants to keep track of its own progress
  - So the “cursor” must have a 1:1 association with the clients and cannot be stored in the data structure
- The client must not need to understand the implementation of the data structure
  - So, the iteration code needs to “belong” to the data-structure
- But the iteration code and the cursor belong together

Solution: the Iterator Pattern

- The Iterator pattern addresses this problem
- An Iterator is an intermediate class
  - Has privileged access to the members of the data-structure, so it can perform the iteration effectively, but prevents the client needing to know how this happens
  - Is instantiated by the data-structure on behalf of the client, on a 1:1 basis. The iterator maintains the cursor, which ensures that each client has its own progress
- Remaining issue is how to grant the iterator object privileged access to the data structure

Iterator Pattern



Client1

Client2

Client3

Data Structure

(private concept: this is a linked list)
private Object headOfList;

Iterator

(shares concepts & access to private data)

private Object currentListElement;

Granting Privileged Data Access

- Could use package level (default) access, but that grants access to too many other classes
- Using an “inner” class, we can achieve this

```
public class MyDataStructure implements List {
  private Object headOfList;
  public class MyDSIterator implements Iterator {
    public Object next() { /* ... */ }
    // remaining Iterator code
  }
  public boolean add(Object e) { /* ... */ }
  // ... remaining MyDataStructure code
```

Instance and `static` Inner Classes

- Instance variables are associated with one instance of their class
- Instance inner-class objects are too
  - This allows them to determine which “outer” data they are accessing
- They must be created where `this` has meaning
  - Or have an explicit outer instance for the constructor
- You can define an inner class as static
  - Such a class cannot refer to instance variables in the enclosing object (there is no enclosing object)

The Enclosing Instance

- Create an inner class instance:
- Anywhere this has meaning
  - `new MyInnerClass();`
- Is equivalent to:
  - `this.new MyInnerClass();`
- If there's no current instance (no `this`)
  - `new MyOuterClass().new MyInnerClass();`
- Or:
  - `outerReference.new MyInnerClass();`

The Enclosing Instance

- To access enclosing instance:
  - Field/member access is generally automatic
  - Explicit access can be specified using:

```
MyOuterClass.this
```

Anonymous Inner Classes

- Often, an inner class object implements an interface (e.g. Iterator) and has no need for any special identity
- For this case, Java provides anonymous inner classes
  - These are probably the most common inner classes
- To create an anonymous inner, we simply call `new`, specify the interface we want to implement, and define the implementation right there

Anonymous Inner Class Example

```
public class IterableFixedArrayList<E>
  implements Iterable<E> {
    private E [] storage = (E[])(new Object[10]);
    private int count = 0;

    public void put(E element) {
        if ((count + 1) < storage.length) {
            storage[count++] = element;
        }
    }
```

Anonymous Inner Class Example

```
// method to get an iterator for this list
public Iterator<E> iterator() {
  return new Iterator() { // specify interface
    private int cursor = 0;
    public boolean hasNext() {
      return (cursor) < count;
    }
    public E next() {
      if (hasNext()) {
        return storage[cursor++];
      } else { return null; }
    }
    public void remove() { /* . . . */ }
  }; // note semicolon terminating “new” statement
}
```

Anonymous Inner Class Example

```
IterableFixedArrayList<String> ifal =
  new IterableFixedArrayList<String>();
  ifal.put("Hello");    ifal.put("World");
  ifal.put("How’re");   ifal.put("You?");
  Iterator i1 = ifal.iterator();
  Iterator i2 = ifal.iterator();
  System.out.println("Iter1> " + i1.next());
  System.out.println("Iter2>     " + i2.next());
  System.out.println("Iter2>     " + i2.next());
  System.out.println("Iter1> " + i1.next());
  System.out.println("Iter1> " + i1.next());
  System.out.println("Iter2>     " + i2.next());
  System.out.println("Iter2>     " + i2.next());
  System.out.println("Iter1> " + i1.next());
```

Designing With Inner Classes

- Not always obvious when inner classes apply
- OO says “keep together what belongs together” and “keep apart what changes independently”
  - Inner classes can help with this:
  - Keep “code that is triggered by this UI button” right next to the button
  - As a way to keep code that must be in a separate class in the same source file (so you know where to look)
- Controlled access to members from helpers
- Avoid cluttering namespace (anonymous)

Summary

In this module, we covered:

- Understand the capabilities of inner classes
- Create code for inner classes and anonymous inner classes
- Recognize reasons for using inner classes