Generic Programming and Inner classes

gen·er·ic 1a : relating or applied to or descriptive of all members of a genus, species, class, or group; common to or characteristic of a whole group or class: typifying or subsuming: not specific or individual.

Reading: Weiss: 118-130 (Section 4.6)
ProgramLive: Sections 12.3 and 12.4.

A use of generic: We like to buy generic drugs, instead of brand-name drugs, because they are cheaper.

poly·mor·phism: the quality or state of being polymorphous:
as (1) : capability of assuming different forms : capability of wide variation

Recall linear search code

// = “v is in b”
static boolean linearSearch (int b, int v) {
    for (int k= 0;  k < b.length;  k= k+1)
        if (b[k] == v)
            return true;
    return false;
}

This linear search code not dependent on types

// = “v is in b”
static boolean linearSearch (Comparable[] b, Object v) {
    for (int k= 0; k < b.length; k= k+1)
        if (b[k].compareTo(v) == 0)
            return true;
    return false;
}

More generic, because it is independent of the type of the array. This is polymorphism.
But: still relies on data being stored in a 1-D array.
This code will not work if data is stored in a more general data structure such as a 2-D array.

Goal

• First version of linear search
  – Input was array of int
• More generic version of linear search
  – Input was array of Comparable
• Can we write a still more generic version of linear search that is independent of data structure?
  – For example, work even with 2-D arrays of Comparable

Key ideas in solution

• Iterator interface
  • Linear search written once and for all using Iterator interface
  • Data class that wants to support linear search must implement Iterator interface
  • Implementing Iterator interface
    – Inner classes provide elegant solution

Rewrite linear search using while-loop

// = “v is in b”
static boolean linearSearch (Comparable[] a, Object v) {
    int k= 0;
    // invariant: v is not in a[0..k-1]
    while (k < a.length) {
        if (a[k].compareTo(v) == 0) return true;
        k= k+1;
    }
    return false;
}

Intuitively, linear search needs to know:
- are there more elements to look at?
- if so, get me the next element?
**Intuitive idea of generic linear search**

- Data is contained in some object.
- Object has an *adapter* that permits data to be enumerated in some order.
- Adapter has two buttons
  - `boolean hasNext()`: are there more elements to be enumerated?
  - `Object Next()`: if so, give me a new element that has not been enumerated so far

**Iterator interface**

```java
public interface Iterator {
    boolean hasNext();
    Object next();
    void remove(); //we will not use this
}
```

This interface is predefined in package java.util. Linear search is written using this interface. Data class must provide an implementation of this interface.

**Generic Linear Search**

```java
static boolean linearSearch(Iterator b, Object v) {
    while (b.hasNext())
        if (((Comparable) b.next()).compareTo(v) == 0)
            return true;
    return false;
}
```

```java
static boolean linearSearch(Comparable[] k, Object v) {
    int k = 0;
    while (k < a.length) {
        if (a[k].compareTo(v) == 0) return true;
        k = k + 1;
    }
    return false;
}
```

**How does data class implement Iterator interface?**

Let us look at a number of solutions.

1. Adapter code is part of class containing data
2. Adapter is a separate class that is hooked up to data class
3. Adapter is an inner class in class containing data

**Adapter (version 1)**

```java
public class Crock1 implements Iterator {
    private SomeClass[] a;  //Class SomeClass must implement Comparable
    private int cursor;  //index of next element to be enumerated

    /** Constructor: an instance with …
    public Crock1() {
        …store data in array a…
    }

    public boolean hasNext() { return (cursor < a.length); }
    public Object next() { cursor = cursor + 1; return a[cursor-1]; }
    public void remove() {} //unimplemented
}
```

**Critique**

- As shown, client class can enumerate elements only once!
  - How do we reset the cursor?
- Making the data class implement Iterator directly is something of a crock because its concern should be with data, rather than enumeration of data.
• One solution to resetting the cursor:
  – Data class implement a method `void reset()` which resets all internal cursor(s)
  – Method must be declared in Iterator interface
• But we still cannot have multiple enumerations of elements going on at the same time
  • Remember: only one cursor….
• **Problem:** cannot create new cursors on demand
• **To solve this problem,** cursor must be part of a different class that can be instantiated any number of times for a single data object.

---

**Sharks and remoras**

Iterator implementation is like a remora.

Data class is like shark

**Single shark must allow us to hook many remoras to it.**

---

**Adapter (version 2)**

```java
public class Remora implements Iterator {
    // Constructor:
    public Remora(Shark s) {
        myShark = s; cursor = 0;    }
    public boolean hasNext() {
        return (cursor < myShark.a.length);    }
    public Object next() {
        cursor = cursor + 1;
        return myShark.a[cursor - 1];    }
    public void remove() {} // unimplemented
}
```

**Client code:**

```java
Shark s = new Shark(); // object containing data
Iterator enum = new Remora(s)….
boolean b = linearSearch( new Remora(s), v);
```

**Solution: use an inner class**

A class can be defined in several places:
- **By itself**
- **As a component of a class**
  - just like a field or method
- **Within a method**
  - just like a local variable
- **Expression-level**
  - Inner class defined as it were part of an expression
  - Called anonymous classes

We focus on a class as a component of a class.

---

**Critique**

• **Good:**
  – Class Shark focuses on data, class Remora focuses on enumeration
• **Bad:**
  – Remora code relies on being able to access Shark variables such as array `a`
    • What if `a` is declared private (which we want)?
  – Remora is specialized to Shark but code appears outside Shark class
    • 2-D array Shark will require a different Remora
    • We may change class Shark and forget to update Remora.
**Adapter (version 2)**

class Shark{
    public Comparable[] a;
    // Constructor: …
    public Shark() {
        …get data into a…}
}

Shark s= new Shark(…); …
Iterator enum= new Remora(s);
while (enum.hasNext())      {    …   }

**Adapter (version 3)**

class Shark{
    private Comparable[] a;
    // Constructor: …
    public Shark() {
        …get data into a…}
    
    private class Remora implements Iterator{
        int cursor;
        Shark myShark;
        public Remora(Shark s) {
            myShark= s;
            cursor= 0;
        }
        public boolean hasNext() {
            return (cursor < myShark.a.length);
        }
        public Object next() {
            cursor= cursor+1;
            return myShark.a[cursor-1];
        }
        public void remove() {} //unimplemented 
    }
    // end Remora
    public Iterator getSharkIterator() {
        return new Remora();
    }
} // end Shark

**Important Points**

- Instances of inner class have access to all components of containing outer class instance

- Inner class can be declared to be public, private, or protected
  - Inner class name is visible accordingly

- We choose to make the inner class private and to provide a public method that returns an instance of the inner class.

**Client code**

…
Shark s = new Shark(); //object containing data
// manipulate the data in Shark s
Object v = ….;

// Set b to “v is somewhere in the data of s”
boolean b = linearSearch(s.getSharkIterator(), v);
…..

**Website**

In the lecture notes page, where we list lectures: for lecture 10, a file innerclass.zip, which contains:

1. List211. Our interface for lists
2. Stack2. Our implementation of a stack in an array, with an inner class that is an Iterator over the stack elements.
3. Generics. Methods for doing a linear search of an Iterator, finding the max of an Iterator, and finding the min of an Iterator.

We urge you to download this program and study it!!!
Load it into DrJava and do calls on it in the Interactions pane.