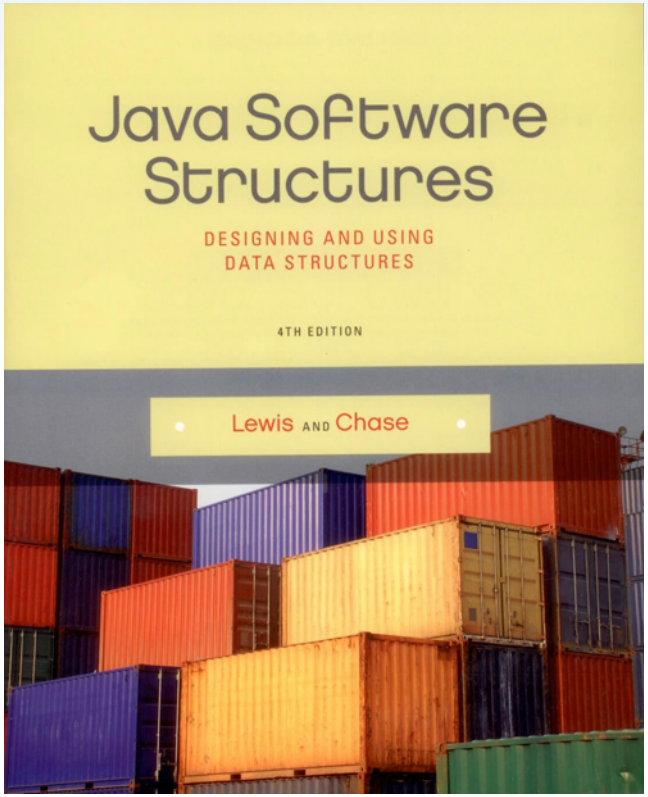

Chapter 7

Iterators

Chapter Scope

- The purpose of an iterator
- The `Iterator` and `Interable` interfaces
- The concept of fail-fast collections
- Using iterators to solve problems
- Iterator implementations

Iterators

- Using various methods, the user could write code to access each element in a collection, but it would be slightly different for each collection
- An *iterator* is an object that allows the user to acquire and use each element in a collection
- It works with a collection, but is a separate object
- An iterator simplifies the classic step of processing elements in a collection

Iterators

- There are two key interfaces in the Java API related to iterators:
  - `Iterator` – used to define an iterator
  - `Iterable` – used to define a collection that provides an iterator
- A collection is `Iterable`, which means it will provide an `Iterator` when requested

Iterators

- The `Iterator` interface:

Method	Description
`boolean hasNext()`	Returns true if the iteration has more elements.
`E next()`	Returns the next element in the iteration.
`void remove()`	Removes the last element returned by the iteration from the underlying collection.

- The `Iterable` interface:

Method	Description
`Iterator<E> iterator()`	Returns an iterator over a set of elements of type E.

Iterators

- Suppose `myList` is an `ArrayList` of `Book` objects

```
Iterator<Book> itr = myList.iterator();
while (itr.hasNext())
    System.out.println(itr.next());
```

- The first line obtains the iterator, then the loop uses `hasNext` and `next` to access and print each book

Iterators

- A for-each loop can be used for the same goal:

```
for (Book book : myList)
    System.out.println(book);
```

- The for-each loop uses an iterator behind the scenes
- The for-each loop can be used on any object that is `Iterable`

Iterators

- You may want to use an iterator explicitly if you don't want to process all elements
  - i.e., searching for a particular element
- You may also use an explicit iterator if you want to call the remove method
- The for-each loop does not give access to the iterator, so remove cannot be called

Iterators

- You shouldn't assume that an iterator will deliver the elements in any particular order unless the documentation explicitly says you can
- Also, remember that an iterator is accessing the elements stored in the collection
- The structure of the underlying collection should not be changed while an iterator is being used
- Most iterators in the Java API are *fail-fast*, meaning they throw an exception if the collection is modified while the iterator is active

Program of Study Revisited

- The `ProgramOfStudy` class was introduced in the last chapter
- It implements the `Iterable` interface
- Its `iterator` method returns the iterator provided by the list
- The `POSGrades` class uses a for-each loop to print courses with a grade of A or A-

```
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.io.Serializable;
import java.util.Iterator;
import java.util.LinkedList;
import java.util.List;

/**
 * Represents a Program of Study, a list of courses taken and planned, for an
 * individual student.
 *
 * @author Lewis and Chase
 * @version 4.0
 */
public class ProgramOfStudy implements Iterable<Course>, Serializable
{
    private List<Course> list;

    /**
     * Constructs an initially empty Program of Study.
     */
    public ProgramOfStudy()
    {
      list = new LinkedList<Course>();
    }
```

```
/**
 * Adds the specified course to the end of the course list.
 *
 * @param course the course to add
 */
public void addCourse(Course course)
{
  if (course != null)
        list.add(course);
}

/**
 * Finds and returns the course matching the specified prefix and number.
 *
 * @param prefix the prefix of the target course
 * @param number the number of the target course
 * @return the course, or null if not found
 */
public Course find(String prefix, int number)
{
  for (Course course : list)
        if (prefix.equals(course.getPrefix()) &&
                  number == course.getNumber())
              return course;

  return null;
}
```

```
/**
 * Adds the specified course after the target course. Does nothing if
 * either course is null or if the target is not found.
 *
 * @param target the course after which the new course will be added
 * @param newCourse the course to add
 */
public void addCourseAfter(Course target, Course newCourse)
{
 if (target == null || newCourse == null)
      return;

 int targetIndex = list.indexOf(target);
 if (targetIndex != -1)
      list.add(targetIndex + 1, newCourse);
}
```

```
/**
 * Replaces the specified target course with the new course. Does nothing if
 * either course is null or if the target is not found.
 *
 * @param target the course to be replaced
 * @param newCourse the new course to add
 */
public void replace(Course target, Course newCourse)
{
 if (target == null || newCourse == null)
       return;

 int targetIndex = list.indexOf(target);
 if (targetIndex != -1)
       list.set(targetIndex, newCourse);
}

/**
 * Creates and returns a string representation of this Program of Study.
 *
 * @return a string representation of the Program of Study
 */
public String toString()
{
 String result = "";
 for (Course course : list)
       result += course + "\n";
 return result;
}
```

```
/**
 * Returns an iterator for this Program of Study.
 *
 * @return an iterator for the Program of Study
 */
public Iterator<Course> iterator()
{
 return list.iterator();
}

/**
 * Saves a serialized version of this Program of Study to the specified
 * file name.
 *
 * @param fileName the file name under which the POS will be stored
 * @throws IOException
 */
public void save(String fileName) throws IOException
{
 FileOutputStream fos = new FileOutputStream(fileName);
 ObjectOutputStream oos = new ObjectOutputStream(fos);
 oos.writeObject(this);
 oos.flush();
 oos.close();
}
```

```
    /**
     * Loads a serialized Program of Study from the specified file.
     *
     * @param fileName the file from which the POS is read
     * @return the loaded Program of Study
     * @throws IOException
     * @throws ClassNotFoundException
     */
    public static ProgramOfStudy load(String fileName) throws IOException,
    ClassNotFoundException
    {
      FileInputStream fis = new FileInputStream(fileName);
      ObjectInputStream ois = new ObjectInputStream(fis);
      ProgramOfStudy pos = (ProgramOfStudy) ois.readObject();
      ois.close();

      return pos;
    }
}
```

```
import java.io.FileInputStream;
import java.io.IOException;
import java.io.ObjectInputStream;

/**
 * Demonstrates the use of an Iterable object (and the technique for reading
 * a serialzed object from a file).
 *
 * @author Lewis and Chase
 * @version 4.0
 */
public class POSGrades
{
    /**
     * Reads a serialized Program of Study, then prints all courses in which
     * a grade of A or A- was earned.
     */
    public static void main(String[] args) throws Exception
    {
      ProgramOfStudy pos = ProgramOfStudy.load("ProgramOfStudy");

      System.out.println(pos);

      System.out.println("Classes with Grades of A or A-\n");

      for (Course course : pos)
      {
            if (course.getGrade().equals("A") || course.getGrade().equals("A-"))
                  System.out.println(course);
      }
    }
}
```

Program of Study Revisited

- Now we'll use an iterator to remove any course in the program of study that doesn't already have a grade
- Since the iterator's `remove` method will be used, we cannot use a for-each loop

```
import java.io.FileInputStream;
import java.io.ObjectInputStream;
import java.util.Iterator;

/**
 * Demonstrates the use of an explicit iterator.
 *
 * @author Lewis and Chase
 * @version 4.0
 */
public class POSClear
{
    /**
     * Reads a serialized Program of Study, then removes all courses that
     * don't have a grade.
     */
    public static void main(String[] args) throws Exception
    {
        ProgramOfStudy pos = ProgramOfStudy.load("ProgramOfStudy");

        System.out.println(pos);

        System.out.println("Removing courses with no grades.\n");
```

```
        Iterator<Course> itr = pos.iterator();
        while (itr.hasNext())
        {
            Course course = itr.next();
            if (!course.taken())
                itr.remove();
        }

        System.out.println(pos);

        pos.save("ProgramOfStudy");
    }
}
```

Implementing Array-based Iterators

- Our `ArrayList` class contains a private inner class that defines an iterator for the list
- An iterator is an appropriate use for an inner class because of its intimate relationship with the outer class (the collection)
- It maintains a modification count that is initialized to the current number of elements in the collection
- If those counts get out of a sync, the iterator throws a `ConcurrentModificationException`

```
/**
 * ArrayListIterator iterator over the elements of an ArrayList.
 */
private class ArrayListIterator implements Iterator<T>
{
 int iteratorModCount;
 int current;

 /**
  * Sets up this iterator using the specified modCount.
  *
  * @param modCount the current modification count for the ArrayList
  */
 public ArrayListIterator()
 {
      iteratorModCount = modCount;
      current = 0;
 }
```

```
/**
 * Returns true if this iterator has at least one more element
 * to deliver in the iteration.
 *
 * @return  true if this iterator has at least one more element to deliver
 *          in the iteration
 * @throws  ConcurrentModificationException if the collection has changed
 *          while the iterator is in use
 */
public boolean hasNext() throws ConcurrentModificationException
{
     if (iteratorModCount != modCount)
          throw new ConcurrentModificationException();

     return (current < rear);
}
```

```
/**
 * Returns the next element in the iteration. If there are no
 * more elements in this iteration, a NoSuchElementException is
 * thrown.
 *
 * @return  the next element in the iteration
 * @throws  NoSuchElementException if an element not found exception occurs
 * @throws  ConcurrentModificationException if the collection has changed
 */
public T next() throws ConcurrentModificationException
{
     if (!hasNext())
          throw new NoSuchElementException();

     current++;

     return list[current - 1];
}
```

```
    /**
     * The remove operation is not supported in this collection.
     *
     * @throws UnsupportedOperationException if the remove method is called
     */
    public void remove() throws UnsupportedOperationException
    {
        throw new UnsupportedOperationException();
    }
  }
}
```

Implementing Linked-Based Iterators

- Similarly, an iterator can use links
- Like the previous example, the `LinkedListItertor` class is implemented as a private inner class

```
/**
 * LinkedIterator represents an iterator for a linked list of linear nodes.
 */
private class LinkedListIterator implements Iterator<T>
{
 private int iteratorModCount;  // the number of elements in the collection
 private LinearNode<T> current;  // the current position

 /**
  * Sets up this iterator using the specified items.
  *
  * @param collection  the collection the iterator will move over
  * @param size        the integer size of the collection
  */
 public LinkedListIterator()
 {
      current = head;
      iteratorModCount = modCount;
 }
```

```
/**
 * Returns true if this iterator has at least one more element
 * to deliver in the iteration.
 *
 * @return  true if this iterator has at least one more element to deliver
 *          in the iteration
 * @throws  ConcurrentModificationException if the collection has changed
 *          while the iterator is in use
 */
public boolean hasNext() throws ConcurrentModificationException
{
     if (iteratorModCount != modCount)
          throw new ConcurrentModificationException();

     return (current != null);
}
```

```
/**
 * Returns the next element in the iteration. If there are no
 * more elements in this iteration, a NoSuchElementException is
 * thrown.
 *
 * @return the next element in the iteration
 * @throws NoSuchElementException if the iterator is empty
 */
public T next() throws ConcurrentModificationException
{
     if (!hasNext())
          throw new NoSuchElementException();

     T result = current.getElement();
     current = current.getNext();
     return result;
}
```

```
    /**
     * The remove operation is not supported.
     *
     * @throws UnsupportedOperationException if the remove operation is called
     */
    public void remove() throws UnsupportedOperationException
    {
        throw new UnsupportedOperationException();
    }
  }
}
```