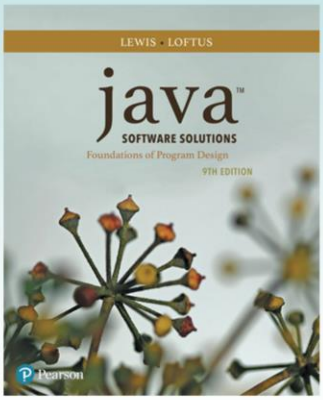
Chapter 5
Conditionals and Loops



Java Software Solutions

Foundations of Program Design

9th Edition

John Lewis
William Loftus

Outline

- Boolean Expressions
- The `if` Statement
- Comparing Data
- The `while` Statement
- ➡ Iterators
- The `ArrayList` Class

-When learning loops, we often have occasion to store data items each time through a loop

-In addition to storing multiple items, we also may want to process each item one at a time

-In this session, we'll be introduced to ways we can store multiple items of information

-We'll also be introduced to how we can step through each item in order to process an item one at a time

-The concepts in this session are meant as an introduction **only**

-We'll be looking at these concepts in depth in future studies

-Our goal in this session is to simply learn how to use the techniques introduced

Iterators

- An *iterator* is an **object** that allows you to process a collection of items one at a time
- It provides **methods** to allow you to step through each item in turn and process it as needed
- An iterator has a `hasNext` method that returns true if there is at least one more item to process
- The `next` method returns the next item
- Iterator objects are defined using the `Iterator` interface, which is discussed further in Chapter 7

-Note we refer to multiple items grouped together as a **collection**

-Note the term “interface” – this is one concept we’ll look at in-depth later

Iterators

- Several classes in the Java standard class library are iterators
- The `Scanner` class is an iterator
  - the `hasNext` method returns true if there is more data to be scanned
  - the `next` method returns the next scanned token as a string
- The `Scanner` class also has variations on the `hasNext` method for specific data types (such as `hasNextInt`)

-We've actually used iterators (without realizing it) as the Scanner class uses iterator methods

Iterators

- The fact that a `Scanner` is an iterator is particularly helpful when reading input from a file
- Suppose we wanted to read and process a list of URLs stored in a file
- One scanner can be set up to read each line of the input until the end of the file is encountered
- Another scanner can be set up for each URL to process each part of the path
- See `URLDissector.java`

```java
//*******************************************************************
//  URLDissector.java       Author: Lewis/Loftus
//
//  Demonstrates the use of Scanner to read file input and parse it
//  using alternative delimiters.
//*******************************************************************

import java.util.Scanner;
import java.io.*;

public class URLDissector
{
   //-----------------------------------------------------------------
   //  Reads urls from a file and prints their path components.
   //-----------------------------------------------------------------
   public static void main (String[] args) throws IOException
   {
      String url;
      Scanner fileScan, urlScan;

      fileScan = new Scanner (new File("urls.inp"));
```

continue

continue

```
      // Read and process each line of the file
      while (fileScan.hasNext())
      {
         url = fileScan.nextLine();
         System.out.println ("URL: " + url);

         urlScan = new Scanner (url);
         urlScan.useDelimiter("/");

         //  Print each part of the url
         while (urlScan.hasNext())
            System.out.println ("   " + urlScan.next());

         System.out.println();
      }
   }
}
```

continue

```
        // Rea
        while
        {
            url
            Sys

            url
            url

            //
            whi

            Sys
        }
    }
}
```

Sample Run

```
URL: www.google.com
   www.google.com

URL: www.linux.org/info/gnu.html
   www.linux.org
   info
   gnu.html

URL: thelyric.com/calendar/
   thelyric.com
   calendar

URL: www.cs.vt.edu/undergraduate/about
   www.cs.vt.edu
   undergraduate
   about

URL: youtube.com/watch?v=EHCRimwRGLs
   youtube.com
   watch?v=EHCRimwRGLs
```

Outline

- Boolean Expressions
- The `if` Statement
- Comparing Data
- The `while` Statement
- Iterators
- The `ArrayList` Class

The ArrayList Class

- An `ArrayList` object stores a list of objects, and is often processed using a loop
- The `ArrayList` class is part of the `java.util` package
- You can reference each object in the list using a numeric index
- An `ArrayList` object grows and shrinks as needed, adjusting its capacity as necessary

-As with other classes we've learned, the ArrayList class is extremely useful

-It can be used to store multiple items of a specific object type

-The general term we use to store multiple items is the term **array**

-We'll look in detail at arrays in a later chapter

-Think of an array as a container used to store multiple items

-For now, we just need to know that it is a technique used to store multiple items

-The ArrayList class is a type of **dynamic container** to store multiple items

-As items are added, the capacity of the container grows

-As items are removed, the capacity of the container shrinks

The ArrayList Class

- Index values of an `ArrayList` begin at 0 (not 1):

0	"Bashful"
1	"Sleepy"
2	"Happy"
3	"Dopey"
4	"Doc"

- Elements can be inserted and removed
- The indexes of the elements adjust accordingly

ArrayList Methods

- Some `ArrayList` methods:

```
boolean add (E obj)
void add (int index, E obj)
Object remove (int index)
Object get (int index)
boolean isEmpty()
int size()
```


-Note that some argument types in these methods are of type “E”

-This designation is a placeholder for **any** object type

-In other words, we can substitute any object type in a method where we see “E”

-We also refer to this “E” term as what we call a **generic** since it is a general designation for any type

-This generic type is an advanced concept covered thoroughly in the next class

-For now, we just need to know that when we see this, we can substitute any type of object

The ArrayList Class

- The type of object stored in the list is established when the `ArrayList` object is created:

```
ArrayList<String> names = new ArrayList<String>();

ArrayList<Book> list = new ArrayList<Book>();
```

- This makes use of Java *generics*, which provide additional type checking at compile time
- An `ArrayList` object cannot store primitive types, but that's what wrapper classes are for
- See `Beatles.java`

-The generic “E” designation is also used when we create an ArrayList object

-We substitute the object type we want to use in the <> brackets as shown above

-This defines the type of object that is stored in the ArrayList collection

-In the “names” ArrayList above, String objects will be stored in the ArrayList

-In the “list” ArrayList above, Book objects (a class we might write) will be stored in the ArrayList

```java
//********************************************************************
//  Beatles.java       Author: Lewis/Loftus
//
//  Demonstrates the use of a ArrayList object.
//********************************************************************

import java.util.ArrayList;

public class Beatles
{
   //-----------------------------------------------------------------
   //  Stores and modifies a list of band members.
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      ArrayList<String> band = new ArrayList<String>();

      band.add ("Paul");
      band.add ("Pete");
      band.add ("John");
      band.add ("George");
```

continue

continue

```
      System.out.println (band);
      int location = band.indexOf ("Pete");
      band.remove (location);

      System.out.println (band);
      System.out.println ("At index 1: " + band.get(1));
      band.add (2, "Ringo");

      System.out.println ("Size of the band: " + band.size());
      int index = 0;
      while (index < band.size())
      {
         System.out.println (band.get(index));
         index++;
      }
   }
}
```

continue

```
        System.out.p
        int locatior
        band.remove

        System.out.p
        System.out.p                                    (1));
        band.add (2,

        System.out.p                                    nd.size());
        int index =
        while (index < band.size())
        {
            System.out.println (band.get(index));
            index++;
        }
    }
}
```

Output

```
[Paul, Pete, John, George]
[Paul, John, George]
At index 1: John
Size of the band: 4
Paul
John
Ringo
George
```


-Note how the toString method in the ArrayList prints the elements as Strings surrounded by brackets

-Recall the toString method prints a representation of object data as Strings