WORKED EXAMPLE 13.1 Finding Files

Your task is to print the names of all files in a directory tree that end in a given extension. To solve this task, you need to know two methods of the `File` class. A `File` object can represent either a directory or a regular file, and the method

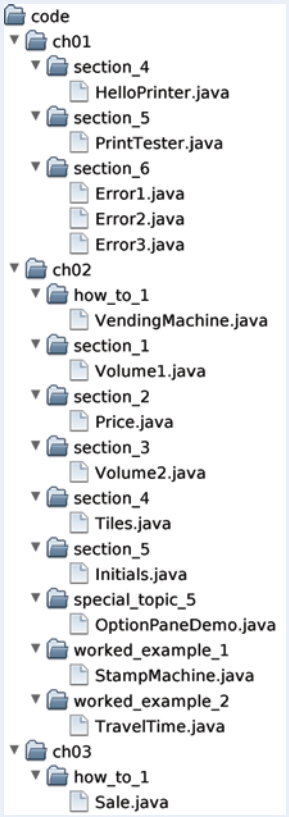
```
boolean isDirectory()
```

tells you which it is. The method

```
File[] listFiles()
```

yields an array of all `File` objects describing the files and directories in a given directory. These can be directories or files.

For example, consider the directory tree at right, and suppose the `File` object `f` represents the `code` directory. Then `f.isDirectory()` returns `true`, and `f.listfiles()` returns an array containing `File` objects describing `code/ch01`, `code/ch02`, and `code/ch03`.



Step 1

Consider various ways to simplify inputs.

Our problem has two inputs: A `File` object representing a directory tree, and an extension. Clearly, nothing is gained from manipulating the extension. However, there is an obvious way of chopping up the directory tree:

- Consider all files in the root level of the directory tree.
- Then consider each tree formed by a subdirectory.

This leads to a valid strategy. Find matching files in the root directory, and then recursively find them in each child subdirectory.

```
For each File object in the root
   If the File object is a directory
      Recursively find files in that directory.
   Else if the name ends in the extension
      Print the name.
```

Step 2

Combine solutions with simpler inputs into a solution of the original problem.

We are asked to simply print the files that we find, so there aren't any results to combine.

Had we been asked to produce an array list of the found files, we would place all matches of the root directory into an array list and add all results from the subdirectories into the same list.

Step 3 Find solutions to the simplest inputs.

The simplest input is a file that isn't a directory. In that case, we simply check whether it ends in the given extension, and if so, print it.

Step 4 Implement the solution by combining the simple cases and the reduction step.

We design a class `FileFinder` with a method for finding the matching files.

```
public class FileFinder
{
   private File[] children;

   /**
      Constructs a file finder for a given directory tree.
      @param startingDirectory the starting directory of the tree
   */
   public FileFinder(File startingDirectory)
   {
      children = startingDirectory.listFiles();
   }

   /**
      Prints all files whose names end in a given extension.
      @param extension a file extension (such as ".java")
   */
   public void find(String extension)
   {
      . . .
   }
}
```

In our case, the reduction step is simply to look at the files and subdirectories:

```
For each child in children
   If the child is a directory
      Recursively find files in the child.
   Else
      If the name of child ends in extension
         Print the name.
```

Here is the complete `FileFinder.find` method:

```
/**
   Prints all files whose names end in a given extension.
   @param extension a file extension (such as ".java")
*/
public void find(String extension)
{
   for (File child : children)
   {
      String fileName = child.toString();
      if (child.isDirectory())
      {
         FileFinder finder = new FileFinder(child);
         finder.find(extension);
      }
      else if (fileName.endsWith(extension))
      {
         System.out.println(fileName);
      }
   }
}
```

FileFinderDemo.java in the worked_example_1 directory completes the solution.

In this solution, we used a class for each directory. Alternatively, we can use a recursive static method:

```
/**
   Prints all files whose names end in a given extension.
   @param aFile a file or directory
   @param extension a file extension (such as ".java")
*/
public static void find(File aFile, String extension)
{
   if (aFile.isDirectory())
   {
      for (File child : aFile.listFiles())
      {
         find(child, extension);
      }
   }
   else
   {
      String fileName = aFile.toString();
      if (fileName.endsWith(extension))
      {
         System.out.println(fileName);
      }
   }
}
```

The basic approach is the same. For a file, we check whether it ends in the given extension. If so, we print it. For a directory, we look at all the files and directories inside.

Here, we chose to accept either a file or directory. For that reason, the calling pattern is subtly different. In our first solution, recursive calls are only made on directories. In the second solution, the method is called recursively on all elements of the array returned by `listFiles()`. The recursion ends right away for files.

A complete solution is in worked_example_1/FileFinder2.java:

```
import java.io.File;

public class FileFinder2
{
   public static void main(String[] args)
   {
      File startingDirectory = new File("/home/myname");
      find(startingDirectory, ".java");
   }

   /**
      Prints all files whose names end in a given extension.
      @param aFile a file or directory
      @param extension a file extension (such as ".java")
   */
   public static void find(File aFile, String extension)
   {
      if (aFile.isDirectory())
      {
         for (File child : aFile.listFiles())
         {
            find(child, extension);
         }
```

```
            }
            else
            {
               String fileName = aFile.toString();
               if (fileName.endsWith(extension))
               {
                  System.out.println(fileName);
               }
            }
         }
      }
```