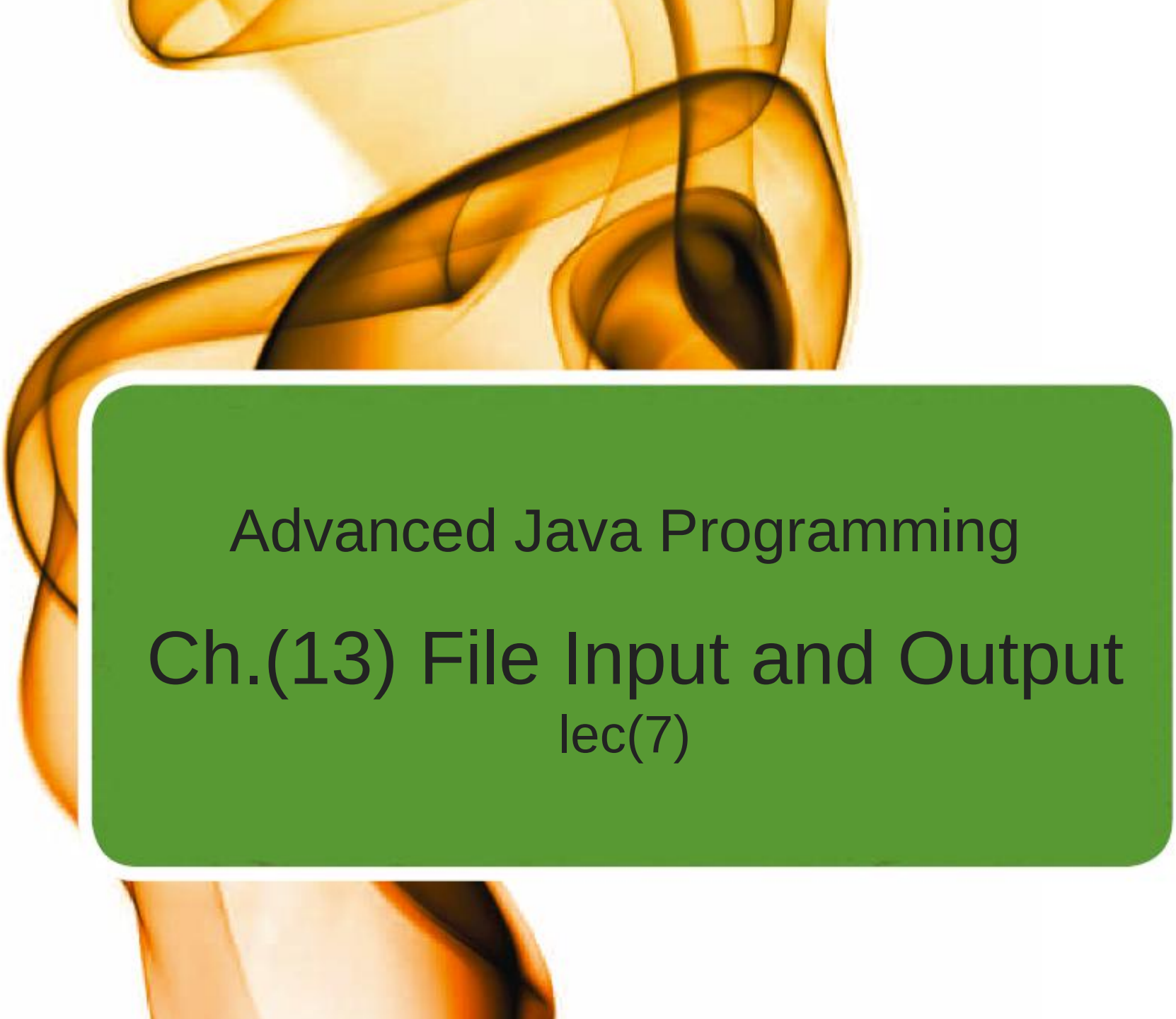


Advanced Java Programming

Ch.(13) File Input and Output

lec(7)

Understanding Computer Files

- **A computer file** is a collection of data stored on a nonvolatile device.
- **Types** :
  - **Text files** contain data that can be read in a text editor. such as a payroll file, program files...etc.
  - **Binary files** contain data that has not been encoded as text.e.g: images, music, and the compiled program files
- **Organization** : computer users organize their files into **folders** or **directories and** also can create folders within folders to form a hierarchy

Understanding Computer Files

- **Path**: A complete list of the disk drive plus the hierarchy of directories in which a file resides
- Typical File Operations:
  - Determining whether and where a path or file exists
  - Opening a file
  - Writing to a file
  - Reading from a file
  - Closing a file
  - Deleting a file
- Java provides built-in classes that contain methods to help you with these tasks

Using the `Path` Class

- Use the `Path` class to create objects that contain information about files or directories, such as their locations, sizes, creation dates, and whether they even exist.
- The `Path` class is brand new in Java 7; it replaces the functionality of the `File` class used in older Java versions.
- `java.nio.file` package

Creating a Path

- First determine the default file system on the host computer

```
FileSystem fs = FileSystems.getDefault();
```

- Define a `Path` using the `getPath()` method

```
Path path =
fs.getPath("C:\\Java\\Chapter.13\\Data.txt");
```

- What is file system ?

Another way to create a Path

- Use the Paths class
- The Paths class is a helper class that eliminates the need to create a FileSystem object.
- Create a Path object by using the following statement:

```
Path filePath = Paths.get("C:\\Java\\Chapter.13\\SampleFile.txt");
```

Retrieving Information about a Path
Path class methods

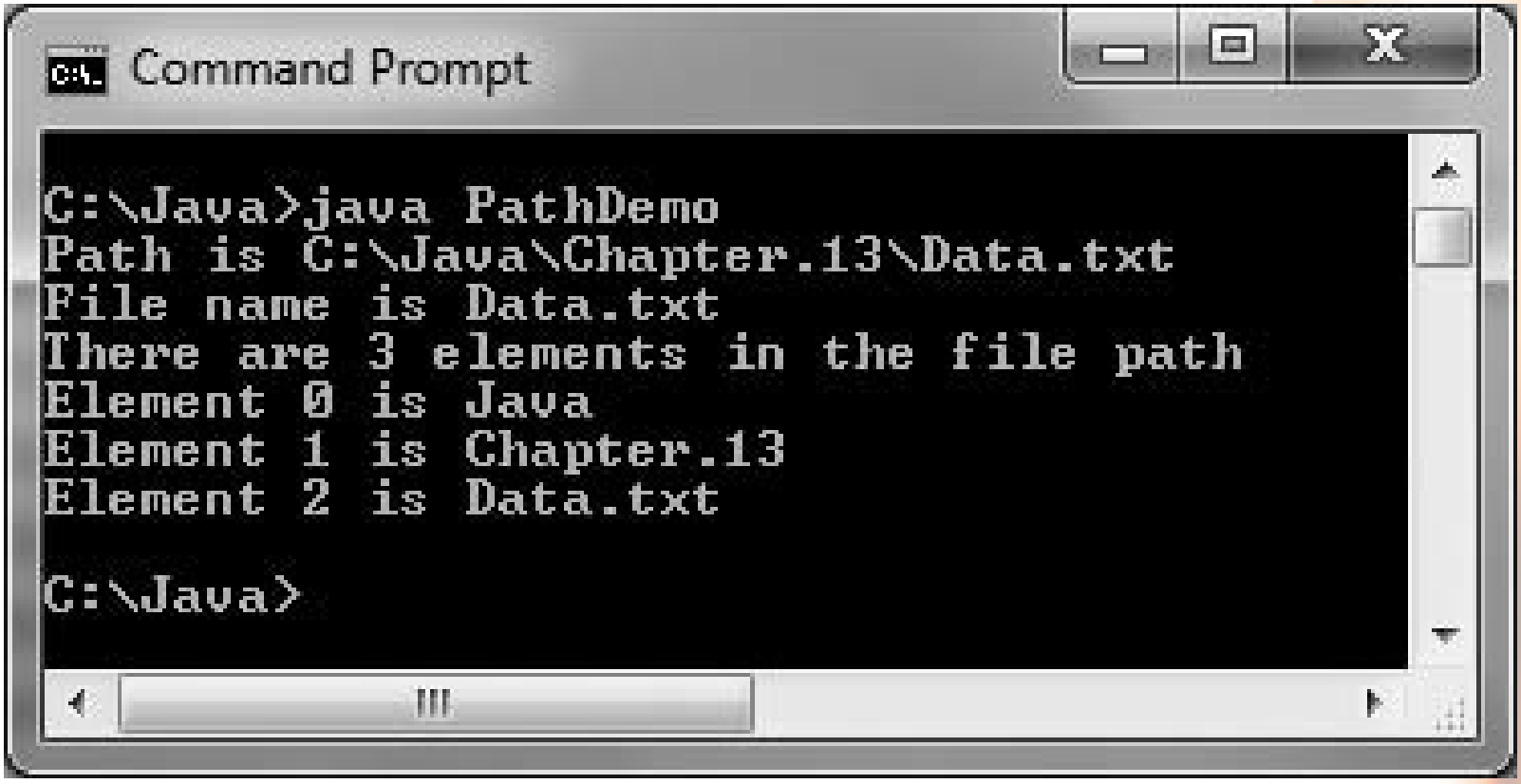
Method	Description
`String toString()`	Returns the `String` representation of the `Path`, eliminating double backslashes
`String getName()`	Returns the last item in the sequence of name elements
`int getNameCount()`	Returns the number of name elements in the `Path`
`String getName(int)`	Returns the name in the position of the `Path` specified by the integer parameter
`boolean exists()`	Returns `true` if the `Path` exists
`boolean notExists()`	Returns `true` if the `Path` does not exist
`void delete()`	Deletes a `Path` and throws an exception if the `Path` doesn't exist
`void deleteIfExists()`	Deletes a `Path` but throws no exception whether the `Path` exists or not

Table 13-1 Selected `Path` class methods

Retrieving Information about a Path

```
import java.nio.file.*;
public class PathDemo
{
   public static void main(String[] args)
   {
      Path filePath =
         Paths.get("C:\\Java\\Chapter.13\\Data.txt");
      int count = filePath.getNameCount();
      System.out.println("Path is " + filePath.toString());
      System.out.println("File name is " + filePath.getName());
      System.out.println("There are " + count +
         " elements in the file path");
      for(int x = 0; x < count; ++x)
        System.out.println("Element " + x + " is " +
           filePath.getName(x));
   }
}
```

J **Figure 13-1** The PathDemo class


Command Prompt
C:\Java>java PathDemo
Path is C:\Java\Chapter.13\Data.txt
File name is Data.txt
There are 3 elements in the file path
Element 0 is Java
Element 1 is Chapter.13
Element 2 is Data.txt
C:\Java>

Absolute and Relative Path

- An absolute path is a complete path; it does not need any other information to locate a file on a system
- A relative path depends on other information.
- The toAbsolutePath() method converts a relative path to an absolute path
- It does not modify the input, but if the input represents a relative Path, it then creates an absolute path by assigning the file name to the current directory.

Converting a Relative Path to an Absolute One

```
import java.util.Scanner;
import java.nio.file.*;
public class PathDemo2
{
   public static void main(String[] args)
   {
        String name;
        Scanner keyboard = new Scanner(System.in);
        System.out.print("Enter a file name >> ");
        name = keyboard.nextLine();
        Path inputPath = Paths.get(name);
        Path fullPath = inputPath.toAbsolutePath();
        System.out.println("Full path is " +  fullPath.toString());
   }
}
```

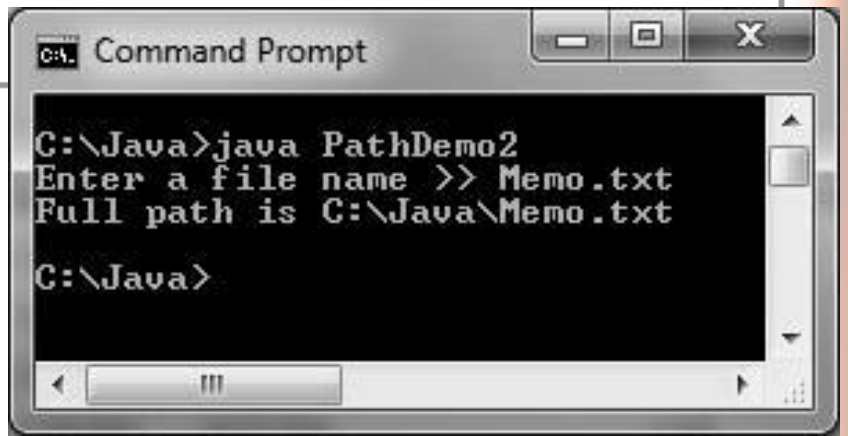


Figure 13-3 The PathDemo2 class

Checking File Accessibility

- Use the **checkAccess()** method.
- Arguments to the checkAccess() method:
  - ✓No argument - Checks that the file exists;
  - ✓READ - Checks that the file exists and that the program has permission to read the file.
  - ✓WRITE - Checks that the file exists and that the program has permission to write to the file.
  - ✓EXECUTE - Checks that the file exists and that the program has permission to execute the file
- To access constants that can be used as arguments to the method checkAccess() use the following :

  Import static java.nio.file.AccessMode.*;

```
import java.nio.file.*;
import static java.nio.file.AccessMode.*;
import java.io.IOException;
public class PathDemo3
{
   public static void main(String[] args)
   {
      Path filePath =
         Paths.get("C:\\Java\\Chapter.13\\Data.txt");
      System.out.println("Path is " + filePath.toString());
      try
      {
         filePath.checkAccess(READ, EXECUTE);
         System.out.println("File can be read and executed");
      }
      catch (IOException e)
      {
         System.out.println
           ("File cannot be used for this application");
      }
   }
}
```

Figure 13-5 The PathDemo3 class

Deleting a Path

- The delete() method deletes a file or throws an exception if the deletion fails.

- The deletion fails If you try to delete :
  - a file that does not exist.
  - a directory that contains files.
  - a file but you don’t have permission.

```
import java.nio.file.*;
import java.io.IOException;
public class PathDemo4
{
   public static void main(String[] args)
   {
      Path filePath =
         Paths.get("C:\\Java\\Chapter.13\\Data.txt");
      try
      {
         filePath.delete();
         System.out.println("File or directory is deleted");
      }
      catch (NoSuchFileException e)
      {
         System.out.println("No such file or directory");
      }
      catch (DirectoryNotEmptyException e)
      {
         System.out.println("Directory is not empty");
      }
      catch (IOException e)
      {
         System.out.println("No permission to delete");
      }
   }
}
```

Determining File Attributes

- Use **readBasicFileAttributes()** method of the **Attributes** class to retrieve useful information about a file.
- The method takes a Path argument and returns an instance of the BasicFileAttributes class. For example:

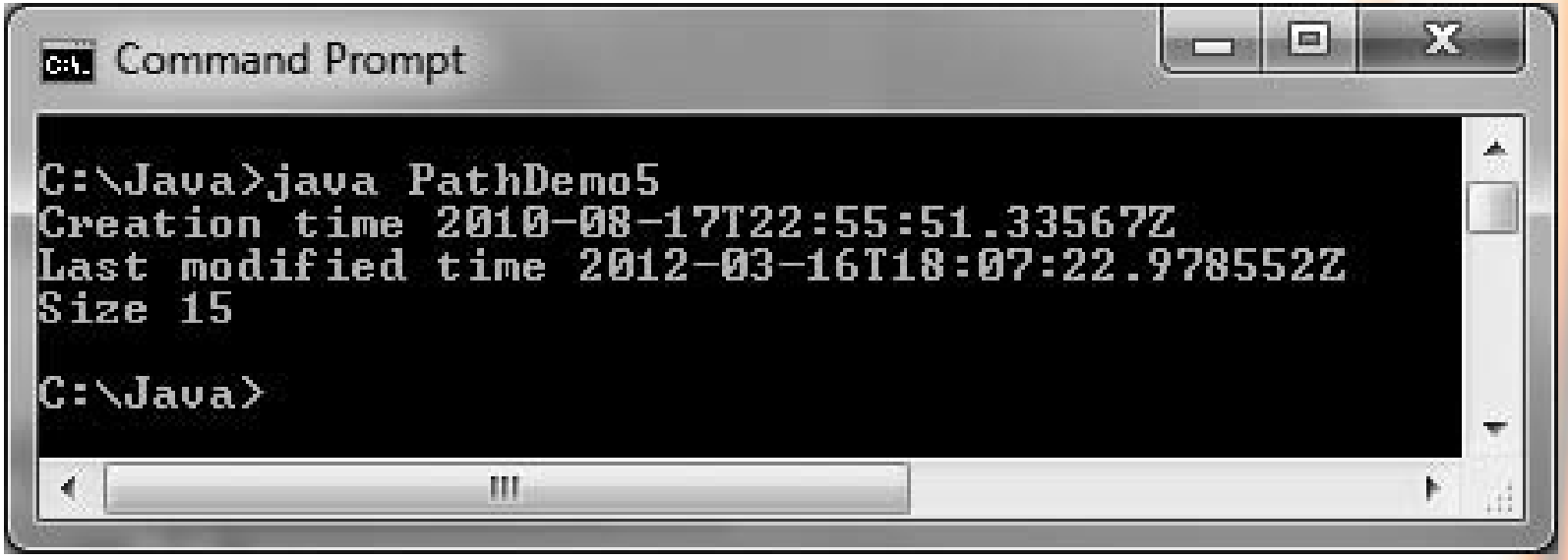
```
BasicFileAttributes attr =Attributes.readBasicFileAttributes(filePath);
```

- Methods for retrieving information about a file include:

  size(), creationTime() and lastModifiedTime()

```
import java.nio.file.*;
import java.nio.file.attribute.*;
import java.nio.file.attribute.Attributes;
import java.io.IOException;
public class PathDemo5
{
   public static void main(String[] args)
   {
      Path filePath =
         Paths.get("C:\\Java\\Chapter.13\\Data.txt");
      try
      {
         BasicFileAttributes attr =
            Attributes.readBasicFileAttributes(filePath);
         System.out.println("Creation time " + attr.creationTime());
         System.out.println("Last modified time " +
            attr.lastModifiedTime());
         System.out.println("Size " + attr.size());
      }
      catch (IOException e)
      {
          System.out.println("IO Exception");
      }
   }
}
```

Figure 13-8 The PathDemo5 class


Command Prompt
C:\Java>java PathDemo5
Creation time 2010-08-17T22:55:51.33567Z
Last modified time 2012-03-16T18:07:22.9785522
Size 15
C:\Java>