Example: 1 Creating and writing file

```
// Writing data to a sequential text file with class Formatter.
import java.io.FileNotFoundException;
import java.lang.SecurityException;
import java.util.Formatter;
import java.util.FormatterClosedException;
import java.util.NoSuchElementException;
import java.util.Scanner;

public class CreateTextFile
{
   private static Formatter output; // outputs text to a file

   public static void main(String[] args)
   {
      openFile();
      addRecords();
      closeFile();
   }

   // open file clients.txt
   public static void openFile()
   {
      try
      {
         output = new Formatter("clients.txt"); // open the file
      }
      catch (SecurityException securityException)
      {
         System.err.println("Write permission denied. Terminating.");
         System.exit(1); // terminate the program
      }
      catch (FileNotFoundException fileNotFoundException)
      {
         System.err.println("Error opening file. Terminating.");
         System.exit(1); // terminate the program
      }
   }

   // add records to file
   public static void addRecords()
   {
      Scanner input = new Scanner(System.in);
      System.out.printf("%s%n%s%n? ",
```

```
            "Enter account number, first name, last name and balance.",
            "Enter end-of-file indicator to end input.");

         while (input.hasNext()) // loop until end-of-file indicator
         {
            try
            {
               // output new record to file; assumes valid input
               output.format("%d %s %s %.2f%n", input.nextInt(),
                  input.next(), input.next(), input.nextDouble());
            }
            catch (FormatterClosedException formatterClosedException)
            {
               System.err.println("Error writing to file. Terminating.");
               break;
            }
            catch (NoSuchElementException elementException)
            {
               System.err.println("Invalid input. Please try again.");
               input.nextLine(); // discard input so user can try again
            }

            System.out.print("? ");
         }
      }

      // close file
      public static void closeFile()
      {
         if (output != null)
            output.close();
      }
   } // end class CreateTextFile
```

End of file indicator for various operating systems:

Operating system	Key combination
UNIX/Linux/Mac OS X	*<Enter> <Ctrl> d*
Windows	*<Ctrl> z*

Example: 2 Reading from file

```
// This program reads a text file and displays each record.
import java.io.IOException;
import java.lang.IllegalStateException;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.util.NoSuchElementException;
import java.util.Scanner;

public class ReadTextFile
{
   private static Scanner input;

   public static void main(String[] args)
   {
      openFile();
      readRecords();
      closeFile();
   }

   // open file clients.txt
   public static void openFile()
   {
      try
      {
         input = new Scanner(Paths.get("clients.txt"));
      }
      catch (IOException ioException)
      {
         System.err.println("Error opening file. Terminating.");
         System.exit(1);
      }
   }

   // read record from file
   public static void readRecords()
   {
      System.out.printf("%-10s%-12s%-12s%10s%n", "Account",
         "First Name", "Last Name", "Balance");

      try
      {
```

```
            while (input.hasNext()) // while there is more to read
            {
               // display record contents
               System.out.printf("%-10d%-12s%-12s%10.2f%n", input.nextInt(),
                  input.next(), input.next(), input.nextDouble());
            }
         }
         catch (NoSuchElementException elementException)
         {
            System.err.println("File improperly formed. Terminating.");
         }
         catch (IllegalStateException stateException)
         {
            System.err.println("Error reading from file. Terminating.");
         }
      } // end method readRecords

      // close file and terminate application
      public static void closeFile()
      {
         if (input != null)
            input.close();
      }
   } // end class ReadTextFile
```

Enum

The `enum` declaration defines a *class* (called an *enum type*). The enum class body can include methods and other fields. The compiler automatically adds some special methods when it creates an enum. For example, they have a static `values` method that returns an array containing all of the values of the enum in the order they are declared. This method is commonly used in combination with the for-each construct to iterate over the values of an enum type. For example, this code from the `Planet` class example below iterates over all the planets in the solar system.

```
for (Planet p : Planet.values()) {
    System.out.printf("Your weight on %s is %f%n",
                      p, p.surfaceWeight(mass));
}
```

Using NIO Classes and Interfaces to Get File and Directory Information

Interfaces **Path** and **DirectoryStream** and classes **Paths** and **Files** (all from package java.nio.file) are useful for retrieving information about files and directories on disk:

- **Path** interface—Objects of classes that implement this interface represent the location of a file or directory. **Path** objects do not open files or provide any file-processing capabilities.

- **Paths** class—Provides static methods used to get a Path object representing a file or directory location.

- **Files** class—Provides static methods for common file and directory manipulations, such as copying files; creating and deleting files and directories; getting information about files and directories; reading the contents of files; getting objects that allow you to manipulate the contents of files and directories; and more

- **DirectoryStream** interface—Objects of classes that implement this interface enable a program to iterate through the contents of a directory.

Creating Path Objects

You'll use class static method get of class Paths to convert a String representing a file's or directory's location into a Path object. You can then use the methods of interface Path and class Files to determine information about the specified file or directory. We discuss several such methods momentarily. For complete lists of their methods, visit:

http://docs.oracle.com/javase/7/docs/api/java/nio/file/Path.html
http://docs.oracle.com/javase/7/docs/api/java/nio/file/Files.html

Absolute vs. Relative Paths

A file or directory's path specifies its location on disk. The path includes some or all of the directories leading to the file or directory. An absolute path contains all directories, starting with the root directory, that lead to a specific file or directory. Every file or directory on a particular disk drive has the same root directory in its path. A relative path is "relative" to another directory—for example, a path relative to the directory in which the application began executing.