The `File` Class

An abstract representation of file and directory pathnames.

Construction: `File(String pathname)`

Some useful methods:

```
boolean exists()
boolean createNewFile()
boolean delete()
long    length()
```

Working with Text Files

These remainder of these slides deal only with useful classes and methods for reading/writing data in text files.

A *text file* is one in which all data values are represented as sequences of characters (encoded in some common scheme like ASCII or Unicode).

A *binary file* is one in which all data values are represented by the same bit patterns used to represent them in machine memory.

The `FileWriter` Class

For writing sequentially to a text file, the `FileWriter` class is usually sufficient.

Construction: `FileWriter(String fileName)`

`FileWriter(File file)`

Some useful methods:

```
void write(char[] cbuf)
void write(char[] cbuf, int offset, int length)
void write(String str)
void flush()
void close()
```

The `FileReader` Class

For reading sequentially from a text file, the `FileReader` class is often sufficient.

Construction: `FileReader(String fileName)`

`FileReader(File file)`

Some useful methods:

```
int  read()
int  read(char[] cbuf)
int  read(char[] cbuf, int offset, int length)
void close()
```

The `RandomAccessFile` Class

Supports reading/writing to a random access file; extremely useful when you need to both read and write the same file or when you need to seek to selected locations within a file and then read or write there.

Construction:

```
RandomAccessFile(File file, String mode)
RandomAccessFile(String name, String mode)
mode:  "r"   "rw"   ("rws"   "rwd")
```

Logical view is that underlying file is a sequence (i.e., array) of bytes.

Each byte occurs at a unique offset from the beginning of the file.

Maintains an internal *file pointer* to the current location within the file.

Reads/writes advance the file pointer.

Writes at the end of the file cause the file to be extended.

The `RandomAccessFile` Class

Some useful methods:

```
int  read()

int  read(byte[] b)

int  read(char[] cbuf, int offset, int length)

String readLine()

void write(byte[] b)

void write(byte[] b, int offset, int length)

long length()

int  getFilePointer()

void seek(long offset)

void close()
```

Be very careful about other methods… some work with two-byte representations and some are intended for binary I/O.

RandomAccessFile Example

```
public class rafExample {

   public static void main(String[] args) {
      try {
         long offset = 0;
         RandomAccessFile raf = new RandomAccessFile(args[0], "r");

         //Get the position of the first record (should be 0):
         offset = raf.getFilePointer();

         //Grab first line (first complete record):
         String record = raf.readLine();
         //Tell the world:
         System.out.println("The record offset is: " + offset);
         System.out.println("The record is: " + record);

      } catch (FileNotFoundException e) {
         System.err.println("Could not find file: " + args[0]);
      } catch (IOException e) {
         System.err.println("Writing error: " + e);
      }
    }
}
```

The `Scanner` Class

A simple text scanner which can parse primitive types and strings using regular expressions.

A `Scanner` breaks its input into tokens using a delimiter pattern, which by default matches whitespace. The resulting tokens may then be converted into values of different types using the various next methods.

Construction: `Scanner(InputStream source)`

`Scanner(String source)`

Configuration: `useDelimiter(String pattern)`

The `Scanner` Class

Some useful methods:

```
String next()
byte   nextByte()
int    nextInt()
. . .
boolean hasNext()              boolean hasNextByte()
boolean hasNextInt()           boolean hasNextLine()
. . .
void close()
```

Scanner Example

```
public class scannerExample {

   public static void main(String[] args) {

      String line = "foo\tbar\twidget";

      Scanner s = new Scanner(line);
      s.useDelimiter("\t");
      String  token1  = s.next();
      String  token2  = s.next();
      String  token3  = s.next();

      System.out.println(token1 + "  " + token2 + "  " + token3);
   }
}
```

`String` method `split`

```
// Pre:
//    row is a string made up of comma-separated integer values
//
public int sumRow( String row ) {

   int sum = 0;

   String[] values = row.split(",");

   for (int idx = 0; idx < values.length; idx++) {

      sum += Integer.parseInt( values[idx] );
   }

   return sum;
}
```

If `row` is: `"18, -5, 10, 7, 25"`

then `values` would be:

"18"	"-5"	"10"	"7"	"25"

The `Formatter` Class

```
class Buffer {

   long    offset;
   String  data;
   boolean dirty;

   . . .

   public String toString() {

      Formatter f = new Formatter();
      f.format("%12d:  ", offset);

      return ( f.toString() + data );
   }
}
```

Also, see the `format` method in the `String` class.