Chapter 7
Common APIs

Virtually every software platform includes a library of pre-built functionality for performing common tasks. The Java and .NET platforms are no exception. Each of these platforms offers a rich class library that provides support for many common functions. Among others, these functions include console I/O, string handling, regular expression processing, date/time functions, threading, logging, file I/O, networking, reflection, and serialization. This chapter focuses on some of the more basic APIs that are required for the majority of programs:

- Console Input/Output
- String handling
- Regular expressions
- Date and time functions
- Random numbers
- Timers
- Threads
- Collections
- File Input/Output

Console Input/Output

Most software platforms provide a way to collect input from and write output to the command (or shell) console. In contrast to a graphical user interface (GUI), a console-based application simply reads and outputs text characters and basic symbols. Because they do not rely on a complex windowing system, console-based applications have very low overhead. They typically load quicker, use memory more efficiently, and execute faster than their GUI counterparts.

Class Comparison

Both Java and .NET provide classes for reading input from and writing output to the command console. In Java, this class resides in the `java.lang` package and is named `System`. In .NET, this class is located in the `System` namespace and is called `Console`. One significant difference between these two classes is the fact that .NET's `System.Console` class has far fewer methods than `java.lang.System`. Unlike Java's `System` class, `Console` only contains methods pertaining to console I/O operations. `System` serves a utility class that implements many functions in addition to console I/O (e.g., retrieving system time and properties, invoking garbage collection, copying arrays, etc.). Table 7.1 compares the console I/O methods provided by these two classes.

Table 7.1 ***Console I/O Types***

Type **/ Member Description**	**Type**	**[Class] / [Struct]ure / [C]onstructor / [M]ethod / [P]roperty / [I]ndexer / [O]perator / [F]ield**
Console I/O Type	Class	`java.lang.`**`System`**
	Class	`System.`**`Console`**
Get the standard error output stream	F	`PrintStream` **`err`**
	P	`TextWriter` **`Error`**
	M	`Stream` **`OpenStandardError`**`(`*`[int bufferSize]`*`)`
Get the standard input stream	F	`InputStream` **`in`**
	P	`TextReader` **`In`**
	M	`Stream` **`OpenStandardInput`**`(`*`[int bufferSize]`*`)`
Get standard output stream	F	`PrintStream` **`out`**
	P	`TextWriter` **`Out`**
	M	`Stream` **`OpenStandardOutput`**`(`*`[int bufferSize]`*`)`
Set standard error output stream	M	`void` **`setErr`**`(PrintStream `*`error`*`)`
	M	`void` **`SetError`**`(TextWriter `*`error`*`)`
Set standard input stream	M	`void` **`setIn`**`(InputStream `*`input`*`)`
	M	`void` **`SetIn`**`(TextReader `*`input`*`)`
Set standard out	M	`void` **`setOut`**`(PrintStream `*`out`*`)`
	M	`void` **`SetOut`**`(TextWriter `*`out`*`)`
Read from input stream	F,M	`int` **`in.read`**`([[byte[] `*`buffer`*`], int `*`index`*`, int `*`length`*`])`
	M	`int` **`Read`**`()`
	M	`string` **`ReadLine`**`()`
Write to standard error	F,M	`void` **`err.print`**`[`**`ln`**`](boolean `*`b`*`)`
	F,M	`void` **`err.print`**`[`**`ln`**`](char `*`c`*`)`
	F,M	`void` **`err.print`**`[`**`ln`**`](char[] `*`c`*`)`
	F,M	`void` **`err.print`**`[`**`ln`**`](double `*`d`*`)`
	F,M	`void` **`err.print`**`[`**`ln`**`](float `*`f`*`)`
	F,M	`void` **`err.print`**`[`**`ln`**`](int `*`i`*`)`
	F,M	`void` **`err.print`**`[`**`ln`**`](long `*`l`*`)`
	F,M	`void` **`err.print`**`[`**`ln`**`](Object `*`o`*`)`
	F,M	`void` **`err.print`**`[`**`ln`**`](String `*`s`*`)`
	F,M	`void` **`err.println`**`()`
	F,M	`void` **`err.write`**`(int `*`byte`*`)`
	F,M	`void` **`err.write`**`(byte[] `*`buffer`*`, int `*`index`*`, int `*`length`*`)`
	F,M	`PrintStream` **`err.printf`**`(String `*`format`*`, Object... `*`args`*`)`
	F,M	`PrintStream` **`err.printf`**`(Locale `*`locale`*`, String `*`format`*`, Object... `*`args`*`)`
	F,M	`PrintStream` **`err.format`**`(String `*`format`*`, Object... `*`args`*`)`
	F,M	`PrintStream` **`err.format`**`(Locale `*`locale`*`, String `*`format`*`, Object... `*`args`*`)`
	P,M	`void` **`Error.Write`**`[`**`Line`**`](bool `*`b`*`)`
	P,M	`void` **`Error.Write`**`[`**`Line`**`](char `*`c`*`)`
	P,M	`void` **`Error.Write`**`[`**`Line`**`](char[] `*`c`*`)`

	P,M	`void` **`Error.Write`***`[`*`Line`*`]`*`(decimal` *`d`*`)`
	P,M	`void` **`Error.Write`***`[`*`Line`*`]`*`(double` *`d`*`)`
	P,M	`void` **`Error.Write`***`[`*`Line`*`]`*`(int` *`i`*`)`
	P,M	`void` **`Error.Write`***`[`*`Line`*`]`*`(long` *`l`*`)`
	P,M	`void` **`Error.Write`***`[`*`Line`*`]`*`(object` *`o`*`)`
	P,M	`void` **`Error.Write`***`[`*`Line`*`]`*`(float` *`f`*`)`
	P,M	`void` **`Error.Write`***`[`*`Line`*`]`*`(string` *`s`*`)`
	P,M	`void` **`Error.Write`***`[`*`Line`*`]`*`(unit` *`ui`*`)`
	P,M	`void` **`Error.Write`***`[`*`Line`*`]`*`(ulong` *`ul`*`)`
	P,M	`void` **`Error.Write`***`[`*`Line`*`]`*`(string` *`s`*`, object` *`o`*`)`
	P,M	`void` **`Error.Write`***`[`*`Line`*`]`*`(string` *`s`*`, params object[]` *`o`*`)`
	P,M	`void` **`Error.Write`***`[`*`Line`*`]`*`(char[]` *`c`*`, int` *`i1`*`, int` *`i2`*`)`
	P,M	`void` **`Error.Write`***`[`*`Line`*`]`*`(string` *`s`*`, object` *`o1`*`, object` *`o2`*`, object` *`o3`*`)`
Write to standard out	F,M	`void` **`out.print`***`[`*`ln`*`]`*`(boolean` *`b`*`)`
	F,M	`void` **`out.print`***`[`*`ln`*`]`*`(char` *`c`*`)`
	F,M	`void` **`out.print`***`[`*`ln`*`]`*`(char[]` *`c`*`)`
	F,M	`void` **`out.print`***`[`*`ln`*`]`*`(double` *`d`*`)`
	F,M	`void` **`out.print`***`[`*`ln`*`]`*`(float` *`f`*`)`
	F,M	`void` **`out.print`***`[`*`ln`*`]`*`(int` *`i`*`)`
	F,M	`void` **`out.print`***`[`*`ln`*`]`*`(long` *`l`*`)`
	F,M	`void` **`out.print`***`[`*`ln`*`]`*`(Object` *`o`*`)`
	F,M	`void` **`out.print`***`[`*`ln`*`]`*`(String` *`s`*`)`
	F,M	`void` **`out.println`**`()`
	F,M	`void` **`out.write`**`(int` *`byte`*`)`
	F,M	`void` **`out.write`**`(byte[]` *`buffer`*`, int` *`index`*`, int` *`length`*`)`
	F,M	`PrintStream` **`out.printf`**`(String` *`format`*`, Object...` *`args`*`)`
	F,M	`PrintStream` **`out.printf`**`(Locale` *`locale`*`, String` *`format`*`, Object...` *`args`*`)`
	F,M	`PrintStream` **`out.format`**`(String` *`format`*`, Object...` *`args`*`)`
	F,M	`PrintStream` **`out.format`**`(Locale` *`locale`*`, String` *`format`*`, Object...` *`args`*`)`
	P,M	`void` **`Out.Write`***`[`*`Line`*`]`*`(bool` *`b`*`)`
	P,M	`void` **`Out.Write`***`[`*`Line`*`]`*`(char` *`c`*`)`
	P,M	`void` **`Out.Write`***`[`*`Line`*`]`*`(char[]` *`c`*`)`
	P,M	`void` **`Out.Write`***`[`*`Line`*`]`*`(decimal` *`d`*`)`
	P,M	`void` **`Out.Write`***`[`*`Line`*`]`*`(double` *`d`*`)`
	P,M	`void` **`Out.Write`***`[`*`Line`*`]`*`(int` *`i`*`)`
	P,M	`void` **`Out.Write`***`[`*`Line`*`]`*`(long` *`l`*`)`
	P,M	`void` **`Out.Write`***`[`*`Line`*`]`*`(object` *`o`*`)`
	P,M	`void` **`Out.Write`***`[`*`Line`*`]`*`(float` *`f`*`)`
	P,M	`void` **`Out.Write`***`[`*`Line`*`]`*`(string` *`s`*`)`
	P,M	`void` **`Out.Write`***`[`*`Line`*`]`*`(unit` *`ui`*`)`
	P,M	`void` **`Out.Write`***`[`*`Line`*`]`*`(ulong` *`ul`*`)`

	P,M	void **Out.Write**[**Line**](string *s*, object *o*)
	P,M	void **Out.Write**[**Line**](string *s*, params object[] *o*)
	P,M	void **Out.Write**[**Line**](char[] *c*, int *i1*, int *i2*)
	P,M	void **Out.Write**[**Line**](string *s*, object *o1*, object *o2*, object *o3*)
	M	void **Write**[**Line**](bool *b*)
	M	void **Write**[**Line**](char *c*)
	M	void **Write**[**Line**](char[] *c*)
	M	void **Write**[**Line**](decimal *d*)
	M	void **Write**[**Line**](double *d*)
	M	void **Write**[**Line**](int *i*)
	M	void **Write**[**Line**](long *l*)
	M	void **Write**[**Line**](object *o*)
	M	void **Write**[**Line**](float *f*)
	M	void **Write**[**Line**](string *s*)
	M	void **Write**[**Line**](unit *ui*)
	M	void **Write**[**Line**](ulong *ul*)
	M	void **Write**[**Line**](string *s*, object *o*)
	M	void **Write**[**Line**](string *s*, params object[] *o*)
	M	void **Write**[**Line**](char[] *c*, int *i1*, int *i2*)
	M	void **Write**[**Line**](string *s*, object *o1*, object *o2*, object *o3*)

Code Examples

The following code examples demonstrate some of the console I/O functions provided by the Java and .NET platforms. Specifically, these operations are presented:

- Basic Input/Output
- Formatted Output

Basic Input/Output

Basic console I/O involves reading data from the keyboard and outputting information to the console. Java uses the `System.in` and `System.out` streams to perform basic console I/O. In .NET, the `Console` class is used for the same purpose. Listings 7.1 and 7.2 demonstrate basic console I/O in Java and .NET.

Java

Listing 7.1 *Basic console I/O (Java)*

```
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.BufferedReader;

public class BasicConsoleIO
{
  public static void main(String[] args) throws IOException
  {
    //get input stream from standard input
    InputStreamReader isr = new InputStreamReader(System.in);
```

```
    //chain to buffered reader for read line functionality
    BufferedReader br = new BufferedReader(isr);

    System.out.print("First Name: ");
    String firstName = br.readLine();

    System.out.print("Last Name: ");
    String lastName = br.readLine();

    System.out.printf("Hello %s %s!\n", firstName, lastName);
  }
}
```

C#

Listing 7.2 *Basic console I/O (C#)*

```
using System;

public class BasicConsoleIO
{
  public static void Main()
  {
    Console.Write("First Name: ");
    String firstName = Console.ReadLine();

    Console.Write("Last Name: ");
    String lastName = Console.ReadLine();

    Console.WriteLine("Hello {0} {1}!", firstName, lastName);
  }
}
```

Output

The output produced by Listings 7.1 and 7.2 looks like this:

```
First Name: Erin
Last Name: Callaway
Hello Erin Callaway!
```

Formatted Output

Java and .NET provide similar features for formatting output. The formatting syntax, however, differs significantly. In contrast to .NET, Java designers made a concerted effort to follow the standard `printf` formatting syntax introduced by the C programming language. This similarity makes Java formatting easy to learn for C programmers. On the other hand, .NET pioneered its own text formatting syntax that programmers not familiar with C might find more intuitive.

Java

Listing 7.3 *Formatted console output (Java)*

```
import java.io.IOException;
import java.util.GregorianCalendar;

public class FormattedConsoleOutput
{
  public static void main(String[] args) throws IOException
  {
    //specify index and change order
```

```
    System.out.printf("%3$s %2$s %1$s %2$s %3$s\n", "a", "b",
      "c");

    //specify width, right justified
    System.out.printf("|%2s%3s%4s|\n", "a", "b", "c");

    //specify width, left justified
    System.out.printf("|%-2s%-3s%-4s|\n", "a", "b", "c");

    //format date to MMMM DD, YYYY  hh:mm ap
    System.out.printf("%1$tB %1$te, %1$tY  %1$tl:%1$tM %1$tp\n",
      GregorianCalendar.getInstance());

    //divide first into groupings, display 2 decimal places for
    //second and enclose negative number in parentheses
    int i = 123456789;
    float f = -100.5F;
    System.out.printf("int: %,d  float: %(.2f\n", i, f);

    //display +/- signs for first and pad second with zeroes
    System.out.printf("int: %+d  float: %010.1f\n", i, f);
  }
}
```

The Java formatting syntax may appear very complex. However, once you understand the rules, you'll find that it is not difficult to use. Java formatting follows this format:

```
%[index$][flags][width][.precision]type
```

The formatting strings all begin with a `%`. `index` refers to the position of a variable in the parameter list. The `flags` position conveys formatting information such as text justification, case, date/time elements, numeric signs, and text padding. `width` indicates the width of the string while `precision` designates the number of decimal places to display. Finally, the `type` position indicates the data type of the associated variable. These data types include values like string (`%s`), integer (`%d`), and float (`%f`).

Of course, we have only scratched the surface of this powerful formatting feature. For a comprehensive description of the formatting syntax, see the JavaDocs for the `java.util.Formatter` class that ship with the JDK.

C#

Listing 7.4 *Formatted console output (C#)*

```
using System;

public class FormattedConsoleOutput
{
  public static void Main()
  {
    //specify index and change order
    Console.WriteLine("{2} {1} {0} {1} {2}", "a", "b", "c");

    //specify width, right justified
    Console.WriteLine("|{0,2} {1,3} {2,4}|", "a", "b", "c");

    //specify width, left justified
    Console.WriteLine("|{0,-2} {1,-3} {2,-4}|", "a", "b", "c");

    //format date to MMMM DD, YYYY  hh:mm ap
```

```
    Console.WriteLine("{0:MMMM d, yyyy  h:mm tt}", DateTime.Now);

    //divide first into groupings, display 2 decimal places for
    //second and enclose negative number in parentheses
    int i = 123456789;
    float f = -100.5F;
    Console.WriteLine("int: {0:#,##0}  float: {1:0.00;(0.00)}",
      i, f);

    //display +/- signs for first and pad second with zeroes
    Console.WriteLine("int: {0:+0;-0}  float: {1:0000000.0}",
      i, f);
  }
}
```

The .NET formatting syntax may appear a bit cryptic at first. However, with a little practice, formatting text in .NET can become almost effortless. The format string follows this format:

```
{index[,alignment][:format[;negativeFormat[;zeroFormat]]]}
```

The `index` position refers to the position of a variable in the parameter list. `alignment` indicates how the text is justified (e.g., left or right) and `formatString` conveys formatting information like date/time elements, numeric signs, and text padding. The `negativeFormat` and `zeroFormat` positions provide a means of formatting negative numbers and zero differently than positive values. For comprehensive documentation of the .NET formatting syntax, see the “Composite Formatting” section in the *.NET Framework Developer’s Guide* that ships with the .NET SDK.

Output

The output generated by Listings 7.3 and 7.4 looks like this:

```
c b a b c
| a  b   c|
|a b  c   |
February 18, 2004  10:53 PM
UPPER CASE
int: 123,456,789  float: (100.50)
int: +123456789  float: -0000100.5
```

String Handling

String handling refers to the various operations that may be performed on text strings. These operations include string concatenation, comparison, parsing, encoding, and formatting.

In Java and .NET, strings are represented by the `java.lang.String` and `System.String` objects, respectively. These strings are *immutable*. That is, once created, they cannot be altered. Some operations, such as concatenation or case conversion (changing to lower- or upper-case), may appear to alter a string. However, this is not the case. Rather, these operations simply cause a new string to be created. In some circumstances, creating a new string object for each modification operation can be very inefficient. For these situations, Java and .NET provide mutable string classes (`java.lang.StringBuffer` and `System.Text.StringBuilder`, respectively) that can be altered without creating a new string object for each modification.

Class Comparison

The Java and .NET platforms provide functionality to perform all types of string manipulation. The following tables document common classes and methods that are applicable to string handling. Specifically, Table 7.2 documents the immutable string classes provided by both platforms and Table 7.3 documents the mutable string classes that allow for dynamic string building.

Table 7.2 ***Immutable String Types***

Type **/ Member Description**	**Type**	**[Class] / [Struct]ure / [C]onstructor / [M]ethod / [P]roperty / [I]ndexer / [O]perator / [F]ield**
Immutable string type	Class	`java.lang.`**`String`**
	Class	`System.`**`String`** *(aliased as* `string`*)*
Extract specified character	M	`char` **`charAt`**`(int` *`index`*`)`
	I	`char` *`string`*`[int` *`index`*`]`
Compare strings lexicographically	M	`int` **`compareTo`**`(String` *`compareString`*`)`
	M	`int` **`compareToIgnoreCase`**`(String` *`compareString`*`)`
	M	`int` **`CompareTo`**`(string` *`compareString`*`)`
Concatenate strings	M	`String` **`concat`**`(String` *`appendString`*`)`
	M	`string String.`**`Concat`**`(string` *`string1`*`, string` *`string2`*`)`
Compare end of string	M	`boolean` **`endsWith`**`(String` *`suffix`*`)`
	M	`bool` **`EndsWith`**`(string` *`suffix`*`)`
Check for equality	M	`boolean` **`equals`**`(String` *`compareString`*`)`
	M	`bool` **`Equals`**`(string` *`compareString`*`)`
	O	*`string1`* **`==`** *`string2`*
Format string	*M*	`String` **`format`**`(String` *`format`*`, Object...` *`args`*`)`
	M	`String` **`format`**`(Locale` *`locale`*`, String` *`format`*`, Object...` *`args`*`)`
	M	`string` **`Format`**`(string` *`format`*`, object` *`arg`*`)`
	M	`string` **`Format`**`(string` *`format`*`, params object[]` *`args`*`)`
	M	`string` **`Format`**`(IFormatProvider` *`provider`*`, string` *`format`*`, params object[]` *`args`*`)`
	M	`string` **`Format`**`(string` *`format`*`, object` *`arg1`*`, object` *`arg2`*`)`
	M	`string` **`Format`**`(string` *`format`*`, object` *`arg1`*`, object` *`arg2`*`, object` *`arg3`*`)`
Get unique hash code for string	M	`int` **`hashCode`**`()`
	M	`int` **`GetHashCode`**`()`
Locate character in string	M	`int` **`indexOf`**`(char` *`searchChar[, int startIndex]`*`)`
	M	`int` **`IndexOf`**`(char` *`searchChar[, int startIndex[, int length]]`*`)`
Locate last occurrence of character in string	M	`int` **`lastIndexOf`**`(char` *`searchChar[, int startIndex]`*`)`
	M	`int` **`LastIndexOf`**`(char` *`searchChar[, int startIndex[, int length]]`*`)`
Locate string in string	M	`int` **`indexOf`**`(String` *`searchString[, int startIndex]`*`)`
	M	`int` **`IndexOf`**`(string` *`searchString[, int startIndex[, int length]]`*`)`
Locate last occurrence of string in string	M	`int` **`lastIndexOf`**`(String` *`searchString[, int startIndex]`*`)`
	M	`int` **`LastIndexOf`**`(string` *`searchString[, int startIndex[,`*

		int *length]]*)
Store string in or retrieve string from the global string pool	M	String **intern**()
	M	string String.**Intern**(string *stringToIntern*)
Get length of string	M	int **length**()
	P	int **Length**
Replace character in string	M	String **replace**(char *oldChar*, char *newChar*)
	M	string **Replace**(char *oldChar*, char *newChar*)
Replace string in string	M	String **replaceAll**(String *regExp*, String *newString*)
	M	string **Replace**(string *oldString*, string *newString*)
Parse string according to specified delimiter	M	String[] **split**(String *regExp[*, int *maxItems]*)
	M	string[] **Split**(char[] *delimiter*, int *maxItems*)
Compare beginning of string	M	boolean **startsWith**(String *prefix*)
	M	bool **StartsWith**(String *prefix*)
Extract a portion of the string	M	String **substring**(int *startIndex[*, int *endIndex]*)
	M	string **Substring**(int *startIndex[*, int *length]*)
Convert to character array	M	char[] **toCharArray**()
	M	char[] **ToCharArray**(*[*int *startIndex*, int *length]*)
Convert to lower-case	M	String **toLowerCase**(*[*Locale *locale]*)
	M	string **ToLower**(*[*CultureInfo *culture]*)
Convert to upper-case	M	String **toUpperCase**(*[*Locale *locale]*)
	M	string **ToUpper**(*[*CultureInfo *culture]*)
Remove leading and trailing whitespace	M	String **trim**()
	M	String **Trim**(*[*char[] *trimChars]*)

Table 7.3 ***Mutable String Types***

Type **/ Member Description**	**Type**	**[Class] / [Struct]ure / [C]onstructor / [M]ethod / [P]roperty / [I]ndexer / [O]perator / [F]ield**
Mutable string type	Class	java.lang.**StringBuffer**
	Class	System.Text.**StringBuilder**
Append a boolean value	M	StringBuffer **append**(boolean *value*)
	M	StringBuilder **Append**(bool *value*)
Append a char value	M	StringBuffer **append**(char *value*)
	M	StringBuilder **Append**(char *value*)
Append a char array	M	StringBuffer **append**(char[] *array[*, int *startIndex*, int *length]*)
	M	StringBuilder **Append**(char[] *array[*, int *startIndex*, int *length]*)
Append a double value	M	StringBuffer **append**(double *value*)
	M	StringBuilder **Append**(double *value*)
Append a float value	M	StringBuffer **append**(float *value*)
	M	StringBuilder **Append**(float *value*)
Append an integer value	M	StringBuffer **append**(int *value*)
	M	StringBuilder **Append**(int *value*)
Append a long value	M	StringBuffer **append**(long *value*)

	M	StringBuilder **Append**(long *value*)
Append string representation of object	M	StringBuffer **append**(Object *value*)
	M	StringBuilder **Append**(object *value*)
Append a string value	M	StringBuffer **append**(String *value*)
	M	StringBuilder **Append**(string *value*)
Get capacity	M	int **capacity**()
	P	int **Capacity**
Extract specified character	M	char **charAt**(int *index*)
	I	char *stringBuilder*[int *index*]
Delete a range of characters	M	StringBuffer **delete**(int *startIndex*, int *endIndex*)
	M	StringBuilder **Remove**(int *startIndex*, int *length*)
Ensure minimum capacity	M	void **ensureCapacity**(int *minimumCapacity*)
	M	int **EnsureCapacity**(int *minimumCapacity*)
Insert a boolean value	M	StringBuffer **insert**(int *index*, boolean *value*)
	M	StringBuilder **Insert**(int *index*, bool *value*)
Insert a char value	M	StringBuffer **insert**(int *index*, char *value*)
	M	StringBuilder **Insert**(int *index*, char *value*)
Insert a char array	M	StringBuffer **insert**(int *index*, char[] *array[*, int *startIndex*, int *length]*)
	M	StringBuilder **Insert**(int *index*, char[] *array[*, int *startIndex*, int *length]*)
Insert a double value	M	StringBuffer **insert**(int *index*, double *value*)
	M	StringBuilder **Insert**(int *index*, double *value*)
Insert a float value	M	StringBuffer **insert**(int *index*, float *value*)
	M	StringBuilder **Insert**(int *index*, float *value*)
Insert an integer value	M	StringBuffer **insert**(int *index*, int *value*)
	M	StringBuilder **Insert**(int *index*, int *value*)
Insert a long value	M	StringBuffer **insert**(int *index*, long *value*)
	M	StringBuilder **Insert**(int *index*, long *value*)
Insert string representation of object	M	StringBuffer **insert**(int *index*, Object *value*)
	M	StringBuilder **Insert**(int *index*, object *value*)
Insert a string value	M	StringBuffer **insert**(int *index*, String *value*)
	M	StringBuilder **Insert**(int *index*, string *value*)
Get length	M	int **length**()
	P	int **Length**
Replace string with string	M	StringBuffer **replace**(int *startIndex*, int *endIndex*, String *newString*)
	M	StringBuilder **Replace**(string *oldString*, string *newString[*, int *startIndex*, int *length]*)
Alter character	M	void **setCharAt**(int *index*, char *value*)
	I	*stringBuilder*[int *index*] = *value*
Set length	M	void **setLength**(int *length*)
	P	int **Length**
Extract a portion of the	M	String **substring**(int *startIndex[*, int *endIndex]*)

string	M	`string ToString(int startIndex, int length)`
Convert to a string	M	`String toString()`
	M	`string ToString()`

Code Examples

In the sections that follow, Java and C# code examples are presented to illustrate how string handling functions are implemented on each platform. Specifically, the following string functions are presented:

- Concatenation
- Equality Comparison
- Lexicographic Comparison
- Parsing
- Encoding
- Formatting

Concatenation

Concatenation is the process of creating a new string by appending one string to another. In both Java and C#, strings can be concatenated using the + operator like this:

```
String example = "Hello, " + "World" + "!";
```

Or the += operator like this:

```
String example2 = "Hello, ";
example2 += "World";
example2 += "!";
```

However, string concatenation can usually be performed far more efficiently using the Java and C# mutable string classes (`StringBuffer` and `StringBuilder`, respectively). Code listings 7.5 and 7.6 demonstrate how strings can be concatenated using these classes.

Java

Listing 7.5 ***String concatenation using StringBuffer (Java)***

```
public class StringConcatenation
{
  public static void main(String[] args)
  {
    StringBuffer example = new StringBuffer("Hello, ");
    example.append("World").append("!");

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

C#

Listing 7.6 ***String concatenation using StringBuilder (C#)***

```
using System;
using System.Text;

public class StringConcatenation
```

```
{
  public static void Main()
  {
    StringBuilder example = new StringBuilder("Hello, ");
    example.Append("World").Append("!");

    Console.WriteLine(example.ToString());
  }
}
```

Output

The output from both listings looks like this:

```
Hello, World!
```

Equality Comparison

String equality comparisons are similar in Java and C# except for the fact that C# overloads the == operator so that it checks for string equality rather than object equality. Therefore, while the `equals()` method is used to perform string comparisons in Java, the == operator can be used to perform similar comparisons in C#. Listings 7.7 and 7.8 demonstrate ways to perform string comparisons in Java and C#.

Java

Listing 7.7 ***Equality string comparison example (Java)***

```
public class EqualityStringComparison
{
  public static void main(String[] args)
  {
    String s1 = "Java";
    String s2 = "Java";
    String s3 = "java";

    //s1 and s2 are equal because they contain the same string
    System.out.println("s1.equals(s2): " + s1.equals(s2));

    //s1 and s3 are not equal because their strings differ
    System.out.println("s1.equals(s3): " + s1.equals(s3));

    //s1 and s3 are equal if case is ignored
    System.out.println("s1.equalsIgnoreCase(s3): " +
      s1.equalsIgnoreCase(s3));
  }
}
```

The output from Listing 7.7 looks like this:

```
s1.equals(s2): true
s1.equals(s3): false
s1.equalsIgnoreCase(s3): true
```

C#

Listing 7.8 ***Equality string comparison example (C#)***

```
using System;
```

```
public class EqualityStringComparison
{
  public static void Main()
  {
    String s1 = "C#";
    String s2 = "C#";
    String s3 = "c#";

    //the == operator is overloaded to check for string equality
    //so it is equivalent to the Equals() method
    Console.WriteLine("s1 == s2: " + (s1 == s2));

    //s1 and s3 are not equal because their strings differ
    Console.WriteLine("s1 == s3: " + (s1 == s3));

    //s1 and s2 are equal because they contain the same string
    //(same as == operator)
    Console.WriteLine("s1.Equals(s2): " + s1.Equals(s2));

    //s1 and s3 are not equal because their strings differ
    Console.WriteLine("s1.Equals(s3): " + s1.Equals(s3));

    //s1 and s3 are equal if case is ignored
    Console.WriteLine("s1.ToLower().Equals(s3.ToLower()): " +
      s1.ToLower().Equals(s3.ToLower()));
  }
}
```

The output from Listing 7.8 looks like this:

```
s1 == s2: True
s1 == s3: False
s1.Equals(s2): True
s1.Equals(s3): False
s1.ToLower().Equals(s3.ToLower()): True
```

Lexicographic Comparison

A *lexicographic comparison* between strings indicates the order in which the strings would appear relative to one another in the dictionary (i.e., sorted alphabetically). In Java and C#, a lexicographic comparison returns one of three possible integer values. A negative integer indicates that the current string precedes the argument string in lexicographic order while a positive integer means that the argument string lexicographically precedes the current string. A zero value indicates that the strings are equal. Listings 7.9 and 7.10 demonstrate the Java and C# lexicographic comparison methods `compareTo()` and `CompareTo()`.

Java

Listing 7.9 *Lexicographic string comparison example (Java)*

```
public class LexicographicStringComparison
{
  public static void main(String[] args)
  {
    String a = "apple";
    String b = "banana";
    String c = "cherry";

    System.out.println("apple and cherry: " + a.compareTo(c));
    System.out.println("apple and banana: " + a.compareTo(b));
```

```
    System.out.println("banana and banana: " + b.compareTo(b));
    System.out.println("cherry and banana: " + c.compareTo(b));
    System.out.println("cherry and apple: " + c.compareTo(a));
  }
}
```

The output from Listing 7.9 looks like this:

```
apple and cat: -2
apple and banana: -1
banana and banana: 0
cat and banana: 1
cat and apple: 2
```

Notice that the negative and positive integer values indicate the relative position between the strings (i.e., the further apart lexicographically, the higher or lower the number).

C#

Listing 7.10 ***Lexicographic string comparison example (C#)***

```
using System;

public class LexicographicStringComparison
{
  public static void Main()
  {
    string a = "apple";
    string b = "banana";
    string c = "cherry";

    Console.WriteLine("apple and cherry: " + a.CompareTo(c));
    Console.WriteLine("apple and banana: " + a.CompareTo(b));
    Console.WriteLine("banana and banana: " + b.CompareTo(b));
    Console.WriteLine("cherry and banana: " + c.CompareTo(b));
    Console.WriteLine("cherry and apple: " + c.CompareTo(a));
  }
}
```

The output from Listing 7.10 looks like this:

```
apple and cherry: -1
apple and banana: -1
banana and banana: 0
cherry and banana: 1
cherry and apple: 1
```

Notice that, unlike Java, the C# `CompareTo()` method does not indicate the relative position between two strings. Rather, only the integer values `-1`, `0`, and `1` are used.

Parsing

Parsing is the process of breaking a string into its component parts. Java and C# provide methods for locating text within a string, extracting portions of a string, and splitting a string into discrete parts based on a specified delimiter. Listings 7.11 and 7.12 demonstrate these operations.

Java

Listing 7.11 ***String parsing example (Java)***

```
public class StringParsing
```

```
{
  public static void main(String[] args)
  {
    String s1 = "The quick brown fox jumped over the lazy dog";

    String s2 = s1.substring(16); //get substring from 16 to end
    System.out.println(s2);

    int index = s1.indexOf("quick"); //find location of string
    String s3 = s1.substring(index, index+5); //get substring
    System.out.println(s3);

    String s4 = "apples,oranges,plums,pears";
    String[] s5 = s4.split(","); //split using comma delimiter

    for (int i = 0; i < s5.length; i++)
    {
      System.out.println(s5[i]);
    }
  }
}
```

Note that the start index accepted by the Java `substring()` method is zero-based while the end index is one-based. For example, consider the following code:

```
String abc = "abc";

abc.substring(0, 1); //extracts a
abc.substring(1, 2); //extracts b
abc.substring(2, 3); //extracts c
```

This zero-based/one-based approach allows the end index to be calculated by simply adding the length of the string to the start index.

> **NOTE**
> The Java class `java.util.StringTokenizer` can also be used to parse strings like this:
>
> ```
> import java.util.*;
>
> public class StringTokenizerParsing
> {
>   public static void main(String[] args) throws Exception
>   {
>     String s = "apples,oranges,plums,pears";
>
>     StringTokenizer st = new StringTokenizer(s, ",");
>     while (st.hasMoreTokens())
>     {
>       System.out.println(st.nextToken());
>     }
>   }
> }
> ```

C#

Listing 7.12 ***String parsing example (C#)***

```
using System;

public class StringParsing
```

```
{
  public static void Main()
  {
    string s1 = "The quick brown fox jumped over the lazy dog";

    string s2 = s1.Substring(16); //get substring from 16 to end
    Console.WriteLine(s2);

    int index = s1.IndexOf("quick"); //find location of string
    string s3 = s1.Substring(index, 5); //get substring
    Console.WriteLine(s3);

    string s4 = "apples,oranges,plums,pears";
    char[] delim = {','};
    string[] s5 = s4.Split(delim); //split using comma delimiter

    for (int i = 0; i < s5.Length; i++)
    {
      Console.WriteLine(s5[i]);
    }
  }
}
```

Notice that while the Java `substring()` accepts a start and end index, the C# `Substring()` method accepts a start index and a length.

Output

The output from Listing 7.11 and 7.12 looks like this:

```
fox jumped over the lazy dog
quick
apples
oranges
plums
pears
```

Encoding

Encoding refers to the process of representing text characters as numbers. A *character set* defines the mapping between each character and its corresponding number. Different character sets may represent each character as a different number. Therefore, in order to build a string of text from an array of bytes (i.e., numbers), it is necessary to know the character set encoding used to produce the bytes in the first place.

In Java, a character encoding can be specified in the `String` class's constructor. On the other hand, C# uses the `String.Text.Encoding` class to build a string based on a given character encoding and byte array. Listings 7.13 and 7.14 demonstrate how to encode strings using different character encodings in Java and C#.

Java

Listing 7.13 ***String encoding example (Java)***

```
public class StringEncoding
{
  public static void main(String[] args) throws
     java.io.UnsupportedEncodingException
  {
    byte[] asciiEncoded = { 74, 97, 118, 97, 32, 67, 35 };
    byte[] unicodeEncoded = { 0, 74, 0, 97, 0, 118, 0, 97,
```

```
      0,32, 0, 67, 0, 35 };

    //convert bytes to string
    String ascii = new String(asciiEncoded, "US-ASCII");
    System.out.println("ASCII: " + ascii);
    String unicode = new String(unicodeEncoded, "UTF-16");
    System.out.println("Unicode: " + unicode);

    //convert string to bytes
    byte[] asciiBytes = ascii.getBytes("US-ASCII");
    byte[] unicodeBytes = unicode.getBytes("UTF-16");
  }
}
```

The most common character sets supported by the Java `String` class are `US-ASCII`, `ISO-8859-1`, `UTF-8`, and `UTF-16`.

C#

Listing 7.14 ***String encoding example (C#)***

```
using System;
using System.Text;

public class StringEncoding
{
  public static void Main()
  {
    byte[] asciiEncoded = { 74, 97, 118, 97, 32, 67, 35 };
    byte[] unicodeEncoded = { 0, 74, 0, 97, 0, 118, 0, 97,
      0, 32, 0, 67, 0, 35 };

    //convert bytes to string
    string ascii = Encoding.ASCII.GetString(asciiEncoded);
    Console.WriteLine(ascii);
    string unicode = Encoding.Unicode.GetString(unicodeEncoded);
    Console.WriteLine(unicode);

    //convert string to bytes
    byte[] asciiBytes = Encoding.ASCII.GetBytes(ascii);
    byte[] unicodeBytes = Encoding.Unicode.GetBytes(unicode);
  }
}
```

The C# `Encoding` class supports ASCII, Unicode, UTF7, and UTF8 character sets.

Output

The output produced by Listings 7.13 and 7.14 looks like this:

```
Java C#
Java C#
```

Formatting

Formatting is the process of altering the manner in which a string is displayed. The Java and .NET string classes both support methods for formatting strings. The Java format string follows this format:

```
%[index$][flags][width][.precision]type
```

The .NET format string follows this format:

```
{index[,alignment][:format[;negativeFormat[;zeroFormat]]]}
```

See the “Formatting Output” part of the “Console Input/Output” section for more information regarding the format of Java and .NET formatting strings. Listings 7.15 and 7.16 demonstrate string formatting in Java and C#.

Java

Listing 7.15 ***String formatting (Java)***

```
import java.io.IOException;
import java.util.GregorianCalendar;

public class Test
{
  public static void main(String[] args) throws IOException
  {
    //specify index and change order
    String index = String.format("%3$s %2$s %1$s %2$s %3$s", "a",
      "b", "c");

    //specify width, right justified
    String width = String.format("|%2s%3s%4s|", "a", "b", "c");

    //specify width, left justified
    String leftJustified = String.format("|%-2s%-3s%-4s|", "a",
      "b", "c");

    //format date to MMMM DD, YYYY  hh:mm ap
    String date = String.format("%1$tB %1$te, %1$tY  " +
      "%1$tl:%1$tM %1$tp", GregorianCalendar.getInstance());

    //divide first into groupings, display 2 decimal places for
    //second and enclose negative number in parentheses
    int i = 123456789;
    float f = -100.5F;
    String numbers1 = String.format("int: %,d  float: %(.2f", i,
      f);

    //display +/- signs for first and pad second with zeroes
    String numbers2 = String.format("int: %+d  float: %010.1f",
      i, f);

    System.out.println(index);
    System.out.println(width);
    System.out.println(leftJustified);
    System.out.println(date);
    System.out.println(numbers1);
    System.out.println(numbers2);
  }
}
```

C#

Listing 7.16 ***String formatting (C#)***

```
using System;

public class Temp
```

```
{
  public static void Main()
  {
    //specify index and change order
    string index = String.Format("{2} {1} {0} {1} {2}", "a", "b",
      "c");

    //specify width, right justified
    string width = String.Format("|{0,2} {1,3} {2,4}|", "a", "b",
      "c");

    //specify width, left justified
    string lftJustified = String.Format("|{0,-2} {1,-3} {2,-4}|",
      "a", "b", "c");

    //format date to MMMM DD, YYYY  hh:mm ap
    string date = String.Format("{0:MMMM d, yyyy  h:mm tt}",
      DateTime.Now);

    //divide first into groupings, display 2 decimal places for
    //second and enclose negative number in parentheses
    int i = 123456789;
    float f = -100.5F;
    string numbers1 = String.Format("int: {0:#,##0}  float: " +
      "{1:0.00;(0.00)}", i, f);

    //display +/- signs for first and pad second with zeroes
    string numbers2 = String.Format("int: {0:+0;-0}  float: " +
      "{1:0000000.0}", i, f);

    Console.WriteLine(index);
    Console.WriteLine(width);
    Console.WriteLine(lftJustified);
    Console.WriteLine(date);
    Console.WriteLine(numbers1);
    Console.WriteLine(numbers2);
  }
}
```

Output

The output produced by Listings 7.15 and 7.16 looks like this:

```
c b a b c
| a  b   c|
|a b  c   |
February 21, 2004  0:32 PM
int: 123,456,789  float: (100.50)
int: +123456789  float: -0000100.5
```

Regular Expressions

A *regular expression* is a sequence of characters that describes the pattern (or format) of a string. For example, using a special syntax, a regular expression could express that a valid e-mail address contains one or more characters followed by a `@` symbol followed by `.com`, `.net`, or

`.org` (of course, many other valid extensions exist). In turn, e-mail addresses could be matched against this regular expression to determine if they are properly formed.

A complete description of regular expression syntax is beyond the scope of this book. You can find a comprehensive tutorial on regular expressions at *http://www.regular-expressions.info*.

Class Comparison

Java and .NET provide classes that utilize regular expressions to determine if a specified string matches a given pattern. These classes can also use regular expressions to parse, find, and replace text. In .NET, all of these operations can be performed using the `System.Text.RegularExpressions.Regex` class. In Java, implementation of these tasks is divided between two classes: `java.util.regex.Pattern` and `java.util.regex.Matcher`.

Table 7.3 ***Match/Parse Regular Expression Types***

Type **/ Member Description**	**Type**	**[Class] / [Struct]ure / [C]onstructor / [M]ethod / [P]roperty / [I]ndexer / [O]perator / [F]ield**
Pattern Matching type	Class	`java.util.regex.`**`Pattern`**
	Class	`System.Text.RegularExpressions.`**`Regex`**
Precompile the given regular expression pattern	*M*	`Pattern` **`compile`**`(String` *`pattern[`*`, int` *`options]`*`)`
	C	`new` **`Regex`**`(string` *`pattern[`*`, int` *`options]`*`)`
Get options to use when matching	M	`int` **`flags`**`()`
	P	`RegexOptions` **`Options`**
Indicates if a string matches the regular expression	M	`boolean` **`matches`**`(String` *`pattern`*`, CharSequence` *`text`*`)`
	M	`bool` **`IsMatch`**`(string` *`text[`*`, int` *`startIndex]`*`)`
	M	`bool` **`IsMatch`**`(string` *`text`*`, string` *`pattern[`*`, RegexOptions` *`options]`*`)`
Get regular expression pattern	M	`String` **`pattern`**`()`
	M	`string` **`ToString`**`()`
Parse string using text that matches the regular expression as a delimiter	M	`String[]` **`split`**`(CharSequence` *`text[`*`, int` *`limit]`*`)`
	M	`string[]` **`Split`**`(string` *`text[`*`, int` *`length[`*`, int` *`startIndex]]`*`)`
	M	`string[]` **`Split`**`(string` *`text`*`, string` *`pattern[`*`, RegexOptions` *`options]`*`)`

Table 7.4 ***Find/Replace Regular Expression Types***

Type **/ Member Description**	**Type**	**[Class] / [Struct]ure / [C]onstructor / [M]ethod / [P]roperty / [I]ndexer / [O]perator / [F]ield**
Find/Replace type	Class	`java.util.regex.`**`Matcher`**
	Class	`System.Text.RegularExpressions.`**`Regex`**
Find one occurrence of text matching the given regular expression	M	`boolean` **`find`**`(`*`[`*`int` *`startIndex]`*`)`
	M	`Match` **`Match`**`(string` *`text[`*`, int` *`startIndex[`*`, int length`*`]]`*`)`
	M	`Match` **`Match`**`(string` *`text`*`, string` *`pattern[`*`, RegexOptions` *`options]`*`)`
Find all occurrences of text matching the given regular expression	M	`boolean` **`matches`**`()`
	M	`MatchCollection` **`Matches`**`(string` *`text[`*`, int` *`startIndex]`*`)`
	M	`MatchCollection` **`Matches`**`(string` *`text`*`, string` *`pattern[`*`, RegexOptions` *`options]`*`)`
Replace first occurrence	M	`String` **`replaceFirst`**`(String` *`newText`*`)`

of text matching the given regular expression with the specified text	M	string **Replace**(String *text*, String *newText*, 1*[*, int *startIndex]*)
	M	string **Replace**(String *text*, MatchEvaluator *evaluator*, 1*[*, int *startIndex]*)
Replace multiple occurrences of text matching the given regular expression with the specified text	M	String **replaceAll**(String *newText*)
	M	string **Replace**(String *text*, String *newText[*, int *length[*, int *startIndex]]*)
	M	string **Replace**(String *text*, MatchEvaluator *evaluator[*, int *length[*, int *startIndex]]*)
	M	string **Replace**(string *text*, string *pattern*, string *newText[*, RegexOptions *options]*)
	M	string **Replace**(string *text*, string *pattern*, MatchEvaluator *evaluator[*, RegexOptions *options]*)

Code Examples

Regular expressions can be utilized to perform a number of different functions. Code examples presented here demonstrate the use of regular expressions to perform the following operations:

- Matching
- Finding
- Parsing
- Replacing

Matching

Matching is the process of determining whether a given string matches a specific pattern. Listings 7.11 and 7.12 demonstrate regular expression matching in Java and C#.

Java

Listing 7.11 ***Matching with regular expression (Java)***

```
import java.util.regex.*;

public class RegularExpressionMatching
{
  public static void main(String[] args)
  {
    String invalidEmail = "sourcestream.net";
    String validEmail = "dustin@sourcestream.com";

    String pattern = "\\w+@\\w+.(com|net|org)";

    //static regular expression matcher
    System.out.println(invalidEmail + " is valid: " +
      Pattern.matches(pattern, invalidEmail));
    System.out.println(validEmail + " is valid: " +
      Pattern.matches(pattern, validEmail));

    //precompiled regular expression matcher
    Pattern p = Pattern.compile(pattern);

    Matcher m1 = p.matcher(invalidEmail);
    System.out.println(invalidEmail + " is valid: " +
      m1.matches());

    Matcher m2 = p.matcher(validEmail);
```

```
    System.out.println(validEmail + " is valid: " +
      m2.matches());
  }
}
```

As shown in Listing 7.11, the `Pattern` class's static `matches()` method indicates whether or not a specified string matches a given pattern. Additionally, for patterns that are to be matched multiple times, regular expressions can be precompiled using the `Pattern` class's `compile()` method. The `Matcher` class created by the `Pattern` class indicates whether or not the match was successful. The output from Listing 7.11 looks like this:

```
sourcestream.net is valid: false
dustin@sourcestream.com is valid: true
sourcestream.net is valid: false
dustin@sourcestream.com is valid: true
```

> **NOTE**
> In addition to the regular expression functionality provided by the `java.util.regex` classes, the Java `String` class implements a `matches(String` *`regExp`*`)` method that indicates whether or not the string matches a given regular expression.

C#

Listing 7.12 *Matching with regular expressions (C#)*

```
using System;
using System.Text.RegularExpressions;

public class RegularExpressionMatching
{
  public static void Main()
  {
    string invalidEmail = "sourcestream.net";
    string validEmail = "dustin@sourcestream.com";

    string pattern = "\\w+@\\w+.(com|net|org)";

    //static regular expression matcher
    Console.WriteLine(invalidEmail + " is valid: " +
      Regex.IsMatch(invalidEmail, pattern));
    Console.WriteLine(validEmail + " is valid: " +
      Regex.IsMatch(validEmail, pattern));

    //precompiled regular expression matcher
    Regex r = new Regex(pattern);
    Console.WriteLine(invalidEmail + " is valid: " +
      r.IsMatch(invalidEmail));
    Console.WriteLine(validEmail + " is valid: " +
      r.IsMatch(validEmail));
  }
}
```

Similar to Java's `Pattern` class, the .NET `Regex` class's static `IsMatch()` method indicates whether or not a given string matches a specified pattern. For patterns that are to be matched multiple times, it is more efficient to compile the regular expression by instantiating a new instance of `Regex`. The `Regex` instance's `IsMatch()` method indicates whether a given string matches the precompiled regular expression. The output produced by Listing 7.12 looks like this:

```
sourcestream.net is valid: False
dustin@sourcestream.com is valid: True
sourcestream.net is valid: False
dustin@sourcestream.com is valid: True
```

Finding

Finding refers to the process of locating portions of a string that match a given pattern. Listings 7.13 and 7.14 demonstrate how to find one or more parts of a string that match a given regular expression.

Java

Listing 7.13 ***Finding with regular expressions (Java)***

```
import java.util.regex.*;

public class RegularExpressionFinding
{
  public static void main(String[] args)
  {
    String s = "a b c 1def 2 ghi3jkl4mno 5  pqr 6 stu7vwx 8 yz";

    //search for any numbers surrounded by whitespace
    String pattern = "\\s+\\d+\\s+";

    Pattern p = Pattern.compile(pattern);
    Matcher m = p.matcher(s);

    //find single occurrence
    if (m.find())
    {
      String num = s.substring(m.start(), m.end()).trim();
      System.out.println(num);
    }

    //find all occurrences
    m.reset(); //reset to beginning of string
    while (m.find())
    {
      String num = s.substring(m.start(), m.end()).trim();
      System.out.print(num + " ");
    }
  }
}
```

As shown in Listing 7.13, each time its `find()` method is called, the `Matcher` class locates the next occurrence of text matching the given regular expression pattern. The beginning and ending location of the matching text can be retrieved using the `Matcher` class's `start()` and `end()` methods. The `reset()` method returns the `Matcher`'s search position to the beginning of the string. The output generated by Listing 7.13 looks like this:

```
2
2 5 6 8
```

C#

Listing 7.14 ***Finding with regular expressions (C#)***

```
using System;
using System.Text.RegularExpressions;

public class RegularExpressionFinding
{
  public static void Main()
  {
    String s = "a b c 1def 2 ghi3jkl4mno 5  pqr 6 stu7vwx 8 yz";

    //search for any numbers surrounded by whitespace
    String pattern = "\\s+\\d+\\s+";

    Regex r = new Regex(pattern);

    //find single occurrence
    Match m = r.Match(s);
    if (m.Success)
      Console.WriteLine(m.Value.Trim());

    //find multiple occurrences
    m = r.Match(s); //reset to beginning of string
    while (m.Success)
    {
      Console.Write("{0} ", m.Value.Trim());
      m = m.NextMatch();
    }

    Console.WriteLine();

    //find multiple occurrences
    MatchCollection mc = r.Matches(s);
    foreach(Match m2 in mc)
    {
      Console.Write("{0} ", m2.Value.Trim());
    }
  }
}
```

As shown in Listing 7.14, each time its `Match()` method is called, the `Regex` class locates the next occurrence of text matching the given regular expression pattern. This occurrence can be retrieved using the `Match` class's `Value` property. Notice that multiple occurrences can be located using the `Regex` class's `Match()` or `Matches()` methods. The output from Listing 7.14 looks like this:

```
2
2 5 6 8
2 5 6 8
```

Parsing

Parsing is the process of breaking a string into its component parts. Java and C# can use regular expressions to parse strings using complex parsing rules. Listings 7.15 and 7.16 show how to parse strings using regular expressions.

Java

Listing 7.15 ***Parsing with regular expressions (Java)***

```
import java.util.regex.*;

public class RegularExpressionParsing
{
  public static void main(String[] args)
  {
    String s = "apples,oranges plums+pears;bananas";

    //use comma, whitespace, +, and ; as delimiters
    Pattern p = Pattern.compile("[,\\s+;]");

    String[] fruits = p.split(s);

    for (int i = 0; i < fruits.length; i++)
    {
      System.out.println(fruits[i]);
    }
  }
}
```

C#

Listing 7.16 ***Parsing with regular expressions (C#)***

```
using System;
using System.Text.RegularExpressions;

public class RegularExpressionParsing
{
  public static void Main()
  {
    string s = "apples,oranges plums+pears;bananas";

    //use comma, whitespace, +, and ; as delimiters
    Regex r = new Regex("[,\\s+;]");

    string[] fruits = r.Split(s);

    foreach (String fruit in fruits)
    {
      Console.WriteLine(fruit);
    }
  }
}
```

Java and C# use the `split()` and `Split()` methods, respectively, to parse a string according to a given regular expression pattern.

Output

The output generated by Listings 7.15 and 7.16 looks like this:

```
apples
oranges
plums
pears
bananas
```

Replacing

Replacing is the process of finding a specified string and changing its value. Regular expressions can be used to find text within a string and alter its value. Listings 7.17 and 7.18 demonstrate how regular expressions can be used to replace text within strings.

Java

Listing 7.17 ***Replacing with regular expressions (Java)***

```
import java.util.regex.*;

public class RegularExpressionReplacing
{
  public static void main(String[] args)
  {
    String s = "a b c 1def 2 ghi3jkl4mno 5  pqr 6 stu7vwx 8 yz";

    //search for any numbers surrounded by whitespace
    String pattern = "\\s+\\d+\\s+";

    Pattern p = Pattern.compile(pattern);
    Matcher m = p.matcher(s);

    //replace first occurrence
    System.out.println(m.replaceFirst("*"));

    //replace all occurrences
    System.out.println(m.replaceAll("*"));
  }
}
```

C#

Listing 7.18 ***Replacing with regular expressions (C#)***

```
using System;
using System.Text.RegularExpressions;

public class RegularExpressionReplacing
{
  public static void Main()
  {
    string s = "a b c 1def 2 ghi3jkl4mno 5  pqr 6 stu7vwx 8 yz";

    //search for any numbers surrounded by whitespace
    string pattern = "\\s+\\d+\\s+";

    Regex r = new Regex(pattern);

    //replace first occurrence
    Console.WriteLine(r.Replace(s, "*", 1));

    //replace all occurrences
    Console.WriteLine(r.Replace(s, "*"));
  }
}
```

Java uses the `replaceFirst()` and `replaceAll()` methods while C# uses the `Replace()` method to parse strings using regular expressions.

Output

The output produced by Listings 7.17 and 7.18 looks like this:

```
a b c 1def*ghi3jkl4mno 5  pqr 6 stu7vwx 8 yz
a b c 1def*ghi3jkl4mno*pqr*stu7vwx*yz
```

Date and Time Functions

Java and .NET provide rich support for performing different types of date and time functions. These functions include creating, comparing, adding, subtracting, and formatting dates.

Class Comparison

In Java, most date functionality is encapsulated into three classes: `java.util.Date`, `java.util.GregorianCalendar`, and `java.text.SimpleDateFormat`. The `Date` class represents a specific moment in time. It also performs comparisons between it and other `Date` objects. The `GregorianCalendar` class is an implementation of the `java.util.Calendar` abstract class and is used to convert `Date` objects into meaningful measures of time on the Gregorian calendar such as weeks, months, and years. It also defines methods for performing many common date functions such as adding time to or subtracting time from a date. The `SimpleDateFormat` class provides date formatting and parsing functionality.

In contrast to Java, .NET encapsulates most of its date processing functionality into a single structure: `System.DateTime`. Not only does this structure represent a moment in time, it also exposes numerous methods for performing common date operations like comparison, arithmetic, formatting, and parsing. Note that the .NET `DateTime` object is not a class. It is an instance of a `struct`. Therefore, `DateTime` objects behave as value types rather than reference types.

Though not presented in the following comparison (since there is no Java equivalent), .NET provides another date-related structure that is commonly used for date calculations. The `System.TimeSpan` structure represents a time interval (e.g., 2 hours or 3 days). This structure is typically used to convey the difference between two dates or to specify an amount of time to add to or subtract from a date.

Table 7.5 ***Date/Time Types***

Type **/ Member Description**	**Type**	**[Class] / [Struct]ure / [C]onstructor / [M]ethod / [P]roperty / [I]ndexer / [O]perator / [F]ield**
Date/Time type	Class	`java.util.`**`Date`**
	Struct	`System.`**`DateTime`**
After comparison	M	`boolean` **`after`**`(Date` *`anotherDate`*`)`
	O	*`thisDate`* **`>`** *`anotherDate`*
Before comparison	M	`boolean` **`before`**`(Date` *`anotherDate`*`)`
	O	*`thisDate`* **`<`** *`anotherDate`*
Create current date and time	C	`new` **`Date`**`()`
	P	`DateTime` **`Now`**
	P	`DateTime` **`Today`** *(current date without time)*
Create date from long	C	`new` **`Date`**`(long` *`milliseconds`*`)`
	C	`new` **`DateTime`**`(long` *`ticks`*`)`
Relative position date	M	`int` **`compareTo`**`(Date` *`anotherDate`*`)`

comparison	*M*	`int` **`Compare`**`(DateTime` *`thisDate`*`, DateTime` *`anotherDate`*`)`
Relative position object comparison	M	`int` **`compareTo`**`(Object` *`anotherDate`*`)`
	M	`int` **`CompareTo`**`(object` *`anotherDate`*`)`
Equals comparison	M	`boolean` **`equals`**`(Object` *`obj`*`)`
	O	`thisDate` **`==`** *`obj`*
Inequality comparison	M	`boolean` **`!equals`**`(Object` *`obj`*`)`
	O	`thisDate` **`!=`** *`obj`*
Get time as long integer	M	`long` **`getTime`**`()`
	P	`long` **`Ticks`**
Get unique hash code for date	M	`int` **`hashCode`**`()`
	M	`int` **`GetHashCode`**`()`

Table 7.6 ***Date Operations Types***

Type **/ Member Description**	**Type**	**[Class] / [Struct]ure / [C]onstructor / [M]ethod / [P]roperty / [I]ndexer / [O]perator / [F]ield**
Date operations type	Class	`java.util.`**`GregorianCalendar`**
	Struct	`System.`**`DateTime`**
Create current date and time	C	`new` **`GregorianCalendar`**`()`
	P	`DateTime` **`Now`**
	P	`DateTime` **`Today`** *(current date without time)*
Create specific date	C	`new` **`GregorianCalendar`**`(int` *`year`*`, int` *`month`*`, int` *`day`*`)`
	C	`new` **`GregorianCalendar`**`(int` *`year`*`, int` *`month`*`, int` *`day`*`, int` *`hour`*`, int` *`minute`*`)`
	C	`new` **`GregorianCalendar`**`(int` *`year`*`, int` *`month`*`, int` *`day`*`, int` *`hour`*`, int` *`minute`*`, int` *`second`*`)`
	C	`new` **`GregorianCalendar`**`(Locale` *`locale`*`)`
	C	`new` **`GregorianCalendar`**`(TimeZone` *`timeZone`*`)`
	C	`new` **`GregorianCalendar`**`(TimeZone` *`timeZone`*`, Locale` *`locale`*`)`
	C	`new` **`DateTime`**`(long` *`ticks`*`)`
	C	`new` **`DateTime`**`(int` *`year`*`, int` *`month`*`, int` *`day`*`)`
	C	`new` **`DateTime`**`(int` *`year`*`, int` *`month`*`, int` *`day`*`, Calendar` *`calendar`*`)`
	C	`new` **`DateTime`**`(int` *`year`*`, int` *`month`*`, int` *`day`*`, int` *`hour`*`, int` *`minute`*`, int` *`second`*`)`
	C	`new` **`DateTime`**`(int` *`year`*`, int` *`month`*`, int` *`day`*`, int` *`hour`*`, int` *`minute`*`, int` *`second`*`, Calendar` *`calendar`*`)`
	C	`new` **`DateTime`**`(int` *`year`*`, int` *`month`*`, int` *`day`*`, int` *`hour`*`, int` *`minute`*`, int` *`second`*`, int` *`millisecond`*`)`
	C	`new` **`DateTime`**`(int` *`year`*`, int` *`month`*`, int` *`day`*`, int` *`hour`*`, int` *`minute`*`, int` *`second`*`, int` *`millisecond`*`, Calendar` *`calendar`*`)`
Generic date addition	M	`void` **`add`**`(int` *`field`*`, int` *`amount`*`)`
	M	`DateTime` **`Add`**`(TimeSpan` *`interval`*`)`
	O	*`thisDate`* **`+`** *`timeSpan`*
Add days	M	`void` **`add`**`(Calendar.DATE, int` *`days`*`)`
	M	`void` **`add`**`(Calendar.DAY_OF_MONTH, int` *`days`*`)`

	M	`void` **`add`**`(Calendar.DAY_OF_WEEK, int` *`days`*`)`
	M	`void` **`add`**`(Calendar.DAY_OF_WEEK_IN_MONTH, int` *`days`*`)`
	M	`void` **`add`**`(Calendar.DAY_OF_YEAR, int` *`days`*`)`
	M	`DateTime` **`AddDays`**`(double` *`days`*`)`
Add hours	M	`void` **`add`**`(Calendar.HOUR, int` *`hours`*`)`
	M	`void` **`add`**`(Calendar.HOUR_OF_DAY, int` *`hours`*`)`
	M	`DateTime` **`AddHours`**`(double` *`hours`*`)`
Add milliseconds	M	`void` **`add`**`(Calendar.MILLISECOND, int` *`milliseconds`*`)`
	M	`DateTime` **`AddMilliseconds`**`(double` *`milliseconds`*`)`
Add minutes	M	`void` **`add`**`(Calendar.MINUTE, int` *`minutes`*`)`
	M	`DateTime` **`AddMinutes`**`(double` *`minutes`*`)`
Add months	M	`void` **`add`**`(Calendar.MONTH, int` *`months`*`)`
	M	`DateTime` **`AddMonths`**`(double` *`months`*`)`
Add seconds	M	`void` **`add`**`(Calendar.SECOND, int` *`seconds`*`)`
	M	`DateTime` **`AddSeconds`**`(double` *`seconds`*`)`
Add years	M	`void` **`add`**`(Calendar.YEAR, int` *`years`*`)`
	M	`DateTime` **`AddYears`**`(double` *`years`*`)`
Generic date subtraction	M	`void` **`add`**`(int field, int -`*`amount`*`)`
	M	`DateTime` **`Subtract`**`(DateTime` *`anotherDate`*`)`
	M	`DateTime` **`Subtract`**`(TimeSpan` *`interval`*`)`
	O	*`thisDate`* **`-`** *`anotherDate`*
	O	*`thisDate`* **`-`** *`timeSpan`*
Equals comparison	M	`boolean` **`equals`**`(Object` *`obj`*`)`
	O	*`thisDate`* **`==`** *`obj`*
Inequality comparison	M	`boolean` **`!equals`**`(Object` *`obj`*`)`
	O	*`thisDate`* **`!=`** *`obj`*
Get day of month	M	`int` **`get`**`(Calendar.DATE)`
	M	`int` **`get`**`(Calendar.DAY_OF_MONTH)`
	P	`int` **`Day`**
Get day of week	M	`int` **`get`**`(Calendar.DAY_OF_WEEK)`
	P	`int` **`DayOfWeek`**
Get day of year	M	`int` **`get`**`(Calendar.DAY_OF_YEAR)`
	P	`int` **`DayOfYear`**
Get hour of day	M	`int` **`get`**`(Calendar.HOUR_OF_DAY)`
	P	`int` **`Hour`**
Get milliseconds component of date	M	`int` **`get`**`(Calendar.MILLISECOND)`
	P	`int` **`Millisecond`**
Get minutes component of date	M	`int` **`get`**`(Calendar.MINUTE)`
	P	`int` **`Minute`**
Get month	M	`int` **`get`**`(Calendar.MONTH)`
	P	`int` **`Month`**
Get seconds component of date	M	`int` **`get`**`(Calendar.SECOND)`
	P	`int` **`Second`**

Get year	M	`int` **`get`**`(Calendar.YEAR)`
	P	`int` **`Year`**
Get unique hash code for date	M	`int` **`hashCode`**`()`
	M	`int` **`GetHashCode`**`()`
Determine leap year	M	`boolean` **`isLeapYear`**`(int` *`year`*`)`
	M	`bool` **`IsLeapYear`**`(int year)`
Get minimum value	M	`int` **`getMinimum`**`(int` *`field`*`)`
	F	`DateTime` **`MinValue`**
Get maximum value	M	`int` **`getMaximum`**`(int` *`field`*`)`
	F	`DateTime` **`MaxValue`**
Get number of days in month	M	`int` **`getActualMaximum`**`(Calendar.DAY_OF_MONTH)`
	M	`int` **`DaysInMonth`**`(int` *`year`*`, int` *`month`*`)`

Table 7.7 ***Date Formatting/Parsing Types***

Type **/ Member Description**	**Type**	**[Class] / [Struct]ure / [C]onstructor / [M]ethod / [P]roperty / [I]ndexer / [O]perator / [F]ield**
Date formatting/parsing type	Class	`java.text.`**`SimpleDateFormat`**
	Struct	`System.`**`DateTime`**
Default format	C,M	`String new` **`SimpleDateFormat`**`()`**`.format`**`(Date` *`date`*`)`
	M	`string` **`ToString`**`()`
Short date format	*M,M*	`String` **`getDateInstance`**`(DateFormat.SHORT)`**`.format`**`(Date` *`date`*`)`
	M	`string` **`ToShortDateString`**`()`
Short time format	*M,M*	`String` **`getTimeInstance`**`(DateFormat.SHORT)`**`.format`**`(Date` *`time`*`)`
	M	`string` **`ToShortTimeString`**`()`
Long date format	*M,M*	`String` **`getDateInstance`**`(DateFormat.LONG)`**`.format`**`(Date` *`date`*`)`
	M	`string` **`ToLongDateString`**`()`
Long time format	*M,M*	`String` **`getTimeInstance`**`(DateFormat.LONG)`**`.format`**`(Date` *`time`*`)`
	M	`string` **`ToLongTimeString`**`()`
Custom format	C,M	`String new` **`SimpleDateFormat`**`(String` *`format`*`)`**`.format`**`(Date` *`date[`*`, StringBuffer` *`toAppendTo`*`, FieldPosition` *`position]`*`)`
	M	`string` **`ToString`**`(string` *`format[`*`, IFormatProvider` *`cultureInfo]`*`)`
	M	`string` **`ToString`**`(IFormatProvider` *`cultureInfo`*`)`
Parse date	C,M	`Date new` **`SimpleDateFormat`**`(String` *`format`*`)`**`.parse`**`(String` *`text[`*`, ParsePosition` *`position]`*`)`
	M	`DateTime` **`Parse`**`(string` *`text[`*`, IFormatProvider` *`cultureInfo[`*`, DateTimeStyles` *`formatOptions]]`*`)`
	M	`DateTime` **`ParseExact`**`(string` *`text`*`, string` *`format`*`, IFormatProvider` *`cultureInfo[`*`, DateTimeStyles` *`formatOptions]`*`)`
	M	`DateTime` **`ParseExact`**`(string` *`text`*`, string[]` *`formats`*`, IFormatProvider` *`cultureInfo[`*`, DateTimeStyles` *`formatOptions]`*`)`

Code Examples

The code examples presented in this section highlight many of the most common tasks relating to dates. These tasks include the following:

- Creating/Parsing
- Comparing
- Adding/Subtracting
- Formatting

Creating/Parsing

In both Java and .NET, there are two ways to create date types. First, a date type can be created through one of the date object's constructors. Second, a parse method can be used to construct a date type from a string. Listings 7.19 and 7.20 demonstrate how date objects can be created in Java and C#.

Java

Listing 7.19 ***Creating date objects (Java)***

```
import java.util.Date;
import java.util.Calendar;
import java.util.GregorianCalendar;
import java.text.DateFormat;
import java.text.SimpleDateFormat;

public class CreateDate
{
  public static void main(String[] args)
  {
    //create Date object representing current date and time
    Date date = new Date();

    //create Calendar object representing current date and time
    Calendar cal = new GregorianCalendar();

    //create Calendar object representing January 1, 2000
    Calendar cal2 = new GregorianCalendar(2000, 0, 1);

    //create Date object from Calendar object
    Date date2 = cal2.getTime();

    //create Date object from long representing January 1, 2000
    Date date3 = new Date(date2.getTime());

    //create Calendar object representing 9am on January 1, 2000
    Calendar cal3 = new GregorianCalendar(2000, 0, 1, 9, 0, 0);

    try
    {
      //parse date from string using SimpleDateFormat
      DateFormat df = new SimpleDateFormat("yyyy/M/d");
      Date date4 = df.parse("2000/1/1");
      System.out.println(date4);

      //parse date and time from string using SimpleDateFormat
      DateFormat df2 = new SimpleDateFormat("yyyy/M/d H:mm");
      Date date5 = df2.parse("2000/1/1 9:00");
    }
    catch (java.text.ParseException ignored) {}
```

```
    //print dates to standard out
    System.out.println(date);
    System.out.println(cal.getTime());
    System.out.println(cal2.getTime());
    System.out.println(date2);
    System.out.println(date3);
    System.out.println(cal3.getTime());
    System.out.println(date4);
    System.out.println(date5);
  }
}
```

Notice that months in Java are zero-based. That is, January is represented by zero rather than one. This is a common source of errors when programming dates in Java.

For a complete list of format strings supported by the `SimpleDateFormat` date parser, see the JavaDoc for the `java.text.SimpleDateFormat` class that ships with the Java SDK. The output from Listing 7.19 looks like this:

```
Mon Jan 19 21:50:09 MST 2004
Mon Jan 19 21:50:09 MST 2004
Sat Jan 01 00:00:00 MST 2000
Sat Jan 01 00:00:00 MST 2000
Sat Jan 01 00:00:00 MST 2000
Sat Jan 01 09:00:00 MST 2000
Sat Jan 01 00:00:00 MST 2000
Sat Jan 01 09:00:00 MST 2000
```

C#

Listing 7.20 *Creating date objects (C#)*

```
using System;

public class CreateDate
{
  public static void Main()
  {
    //create DateTime object representing current date and time
    DateTime date = DateTime.Now;

    //create DateTime object representing current date (no time)
    DateTime date2 = DateTime.Today;

    //create DateTime object representing January 1, 2000
    DateTime date3 = new DateTime(2000, 1, 1);

    //create DateTime object from long representing Jan. 1, 2000
    DateTime date4 = new DateTime(date3.Ticks);

    //create DateTime object representing 9am on January 1, 2000
    DateTime date5 = new DateTime(2000, 1, 1, 9, 0, 0);

    //parse date from string
    DateTime date6 = DateTime.ParseExact("2000/1/1",
      "yyyy/M/d", null);

    //parse date and time from string
    DateTime date7 = DateTime.ParseExact("2000/1/1 9:00",
```

```
      "yyyy/M/d H:mm", null);

    //print dates to standard out
    Console.WriteLine(date);
    Console.WriteLine(date2);
    Console.WriteLine(date3);
    Console.WriteLine(date4);
    Console.WriteLine(date5);
    Console.WriteLine(date6);
    Console.WriteLine(date7);
  }
}
```

Notice that, unlike Java, months in .NET are one-based. This convention is more intuitive than Java's zero-based approach.

For a complete list of format strings supported by the `DateTime` date parser, see the API documentation for the `System.Globalization.DateTimeFormatInfo` class that ships with the .NET Framework SDK. The output from Listing 7.20 looks like this:

```
1/19/2004 9:53:06 PM
1/19/2004 12:00:00 AM
1/1/2000 12:00:00 AM
1/1/2000 12:00:00 AM
1/1/2000 9:00:00 AM
1/1/2000 12:00:00 AM
1/1/2000 9:00:00 AM
```

Comparing

When comparing dates, the most obvious difference between Java and .NET is that, with Java, dates are compared using object methods while in .NET overloaded operators are used for date comparison. Listings 7.21 and 7.22 illustrate how dates are compared on both platforms.

Java

Listing 7.21 ***Comparing date objects (Java)***

```
import java.util.Date;
import java.util.GregorianCalendar;

public class DateComparison
{
  public static void main(String[] args)
  {
    //create calendar and date representing January 1, 2000
    Calendar cal1 = new GregorianCalendar(2000, 0, 1);
    Date sooner = cal1.getTime();

    //create calendar and date representing January 26, 2000
    Calendar cal2 = new GregorianCalendar(2000, 0, 26);
    Date later = cal2.getTime();

    if (sooner.before(later))
      System.out.println("sooner is before later");

    if (sooner.after(later))
      System.out.println("sooner is after later");

    if (sooner.equals(later))
      System.out.println("sooner is equal to later");
```

```
    if (!sooner.equals(later))
      System.out.println("sooner is not equal to later");

    //month comparison
    if (cal1.get(Calendar.MONTH) == cal2.get(Calendar.MONTH));
      System.out.println("sooner and later months are equal");

    //year comparison
    if (cal1.get(Calendar.YEAR) == cal2.get(Calendar.YEAR))
      System.out.println("sooner and later years are equal");
  }
}
```

C#

Listing 7.22 ***Comparing date objects (C#)***

```
using System;

public class DateComparison
{
  public static void Main()
  {
    //create date representing January 1, 2000
    DateTime sooner = new DateTime(2000, 1, 1);

    //create date representing January 26, 2000
    DateTime later = new DateTime(2000, 1, 26);

    if (sooner < later)
      Console.WriteLine("sooner is before later");

    if (sooner > later)
      Console.WriteLine("sooner is after later");

    if (sooner == later)
      Console.WriteLine("sooner is equal to later");

    if (sooner != later)
      Console.WriteLine("sooner is not equal to later");

    //month comparison
    if (sooner.Month == later.Month)
      Console.WriteLine("sooner and later months are equal");

    //year comparison
    if (sooner.Year == later.Year)
      Console.WriteLine("sooner and later years are equal");

    Console.ReadLine();
  }
}
```

Output

The output from listings 7.21 and 7.22 looks like this:

```
sooner is before later
sooner is not equal to later
sooner and later months are equal
```

```
sooner and later years are equal
```

Adding/Subtracting

The primary difference between adding and subtracting dates in Java and .NET is that Java uses a generic `add()` method for all arithmetic operations. .NET, on the other hand, employs specific methods for adding or subtracting different measures of time such as `AddDays()` or `AddMonths()`. Additionally, in conjunction with a `System.TimeSpan` object, .NET date arithmetic can be performed using the overloaded + and – operators. Listings 7.23 and 7.24 demonstrate how date arithmetic is performed on both platforms.

Java

Listing 7.23 ***Adding/Subtracting date objects (Java)***

```
import java.util.Calendar;
import java.util.GregorianCalendar;

public class DateArithmetic
{
  public static void main(String[] args)
  {
    //create calendar representing January 1, 2000 9:00 am
    Calendar cal = new GregorianCalendar(2000, 0, 1, 9, 0, 0);

    System.out.println(cal.getTime());

    //add one hour
    cal.add(Calendar.HOUR, 1);

    System.out.println(cal.getTime());

    //subtract two days
    cal.add(Calendar.DAY_OF_MONTH, -2);

    System.out.println(cal.getTime());

    //add three weeks
    cal.add(Calendar.WEEK_OF_MONTH, 3);

    System.out.println(cal.getTime());

    //subtract four months
    cal.add(Calendar.MONTH, -4);

    System.out.println(cal.getTime());

    //add five years
    cal.add(Calendar.YEAR, 5);

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

The output from Listing 7.23 looks like this:

```
Sat Jan 01 09:00:00 MST 2000
Sat Jan 01 10:00:00 MST 2000
Thu Dec 30 10:00:00 MST 1999
Thu Jan 20 10:00:00 MST 2000
```

```
Mon Sep 20 10:00:00 MDT 1999
Mon Sep 20 10:00:00 MDT 2004
```

C#

Listing 7.24 ***Adding/Subtracting date objects (C#)***

```
using System;

public class DateArithmetic
{
  public static void Main()
  {
    //create date representing January 1, 2000 9:00 am
    DateTime date = new DateTime(2000, 1, 1, 9, 0, 0);

    Console.WriteLine(date);

    //add one hour
    date = date.AddHours(1);

    Console.WriteLine(date);

    //subtract two days
    date = date.AddDays(-2);

    Console.WriteLine(date);

    //add three weeks
    date = date.AddDays(21);

    Console.WriteLine(date);

    //subtract four months
    date = date.AddMonths(-4);

    Console.WriteLine(date);

    //add five years
    date = date.AddYears(5);

    Console.WriteLine(date);

    //add one hour using TimeSpan
    TimeSpan span = new TimeSpan(1, 0, 0);
    date = date + span;

    Console.WriteLine(date);

    //subtract one day using TimeSpan
    span = new TimeSpan(1, 0, 0, 0);
    date = date - span;

    Console.WriteLine(date);
  }
}
```

The output from Listing 7.24 looks like this:

```
1/1/2000 9:00:00 AM
```

```
1/1/2000 10:00:00 AM
12/30/1999 10:00:00 AM
1/20/2000 10:00:00 AM
9/20/1999 10:00:00 AM
9/20/2004 10:00:00 AM
9/20/2004 11:00:00 AM
9/19/2004 11:00:00 AM
```

Formatting

The primary difference between formatting dates in Java and .NET is that Java utilizes a special helper class called `java.text.SimpleDateFormat` to perform formatting tasks. On the other hand, formatting is built into .NET's standard `System.DateTime` class. Listings 7.25 and 7.26 demonstrate how dates are formatted in Java and .NET.

Java

Listing 7.25 ***Formatting dates (Java)***

```
import java.util.Date;
import java.util.GregorianCalendar;
import java.util.Calendar;
import java.text.DateFormat;
import java.text.SimpleDateFormat;

public class DateFormatting
{
  public static void main(String[] args)
  {
    //create date representing January 1, 2000 9:00 am
    Calendar cal = new GregorianCalendar(2000, 0, 1, 9, 0, 0);
    Date date = cal.getTime();

    //format date in year/month/day format with SimpleDateFormat
    DateFormat df = new SimpleDateFormat("yyyy/MM/dd");
    String formattedDate = df.format(date);

    System.out.println(formattedDate);

    //format date for 24-hour time using SimpleDateFormat
    df = new SimpleDateFormat("yyyy/MM/dd HH:mm:ss");
    formattedDate = df.format(date);

    System.out.println(formattedDate);

    //format date for 12-hour time using SimpleDateFormat
    df = new SimpleDateFormat("yyyy/MM/dd hh:mm:ss a");
    formattedDate = df.format(date);

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

C#

Listing 7.26 ***Formatting dates (C#)***

```
using System;

public class Temp
{
```

```
  public static void Main()
  {
    //create date representing January 1, 2000
    DateTime date = new DateTime(2000, 1, 1, 9, 0, 0);

    //format date in year/month/day format
    String formattedDate = date.ToString("yyyy/MM/dd");

    Console.WriteLine(formattedDate);

    //format date for 24-hour time
    formattedDate = date.ToString("yyyy/MM/dd HH:mm:ss");

    Console.WriteLine(formattedDate);

    //format date for 12-hour time using SimpleDateFormat
    formattedDate = date.ToString("yyyy/MM/dd hh:mm:ss tt");

    Console.WriteLine(formattedDate);
  }
}
```

Output

The output produced by Listings 7.25 and 7.26 looks like this:

```
2000/01/01
2000/01/01 09:00:00
2000/01/01 09:00:00 AM
```

Random Numbers

The process of generating pseudo-random numbers is almost identical in Java and .NET. On both platforms, this process consists of providing a numeric seed to the `Random` class and then invoking the appropriate method to return a pseudo-random integer or floating point value.

Class Comparison

Java and .NET support the generation of pseudo-random numbers through the `java.util.Random` and `System.Random` classes, respectively. Table 7.8 compares the random number generating classes provided by Java and .NET.

Table 7.8 *Random number generators*

Type **/ Member Description**	**Type**	**[Class] / [Struct]ure / [C]onstructor / [M]ethod / [P]roperty / [I]ndexer / [O]perator / [F]ield**
Random number generator	Class	`java.util.`**`Random`**
	Class	`System.`**`Random`**
Instantiate using default seed (based on time)	C	`new `**`Random`**`()`
	C	`new `**`Random`**`()`
Instantiate using custom seed	C	`new `**`Random`**`(long `*`seed`*`)`
	C	`new `**`Random`**`(int `*`seed`*`)`
Generate random bytes	M	`void `**`nextBytes`**`(byte[] bytes)`

	M	`void` **`NextBytes`**`(byte[] bytes)`
Generate random double value between 0.0 and 1.0	M	`double` **`nextDouble`**`()`
	M	`double` **`NextDouble`**`()`
Generate random integer value	M	`int` **`nextInt`**`(`*`[int maxValue]`*`)`
	M	`int` **`Next`**`(`*`[int maxValue]`*`)`
	M	`int` **`Next`**`(int `*`minValue`*`, int `*`maxValue`*`)`

Random Number Code Examples

Generating random numbers in Java and .NET is nearly identical. Listings 7.27 and 7.28 demonstrate these operations:

Java

Listing 7.27 *Generating random numbers (Java)*

```
import java.util.Random;

public class GenerateRandomNumbers
{
  public static void main(String[] args)
  {
    //use default seed
    Random rand = new Random();

    //generate random number across entire integer range
    System.out.println(rand.nextInt());

    //generate random number between 0 and 9
    System.out.println(rand.nextInt(10));

    //generate random number between 1 and 10
    System.out.println(rand.nextInt(10) + 1);

    //generate 10 random bytes
    byte[] bytes = new byte[10];
    rand.nextBytes(bytes);
    System.out.println(new String(bytes));

    //generate a random number between 0.0 and 1.0
    System.out.println(rand.nextDouble());

    //generate random integer between 1 and 50 using custom seed
    long seed = System.currentTimeMillis();
    rand = new Random(seed);
    System.out.println(rand.nextInt(50) + 1);
  }
}
```

The output generated by Listing 7.27 looks like this:

```
1080764717
2
1
k§¿Ù□U(¼nù
0.6632425749577202
49
```

Of course, since random numbers are being generated, this output varies with each execution.

C#

Listing 7.28 ***Generating random numbers (C#)***

```
using System;

public class GenerateRandomNumbers
{
  public static void Main()
  {
    //use default seed
    Random rand = new Random();

    //generate random number across entire integer range
    Console.WriteLine(rand.Next());

    //generate random number between 0 and 9
    Console.WriteLine(rand.Next(10));

    //generate random number between 1 and 10
    Console.WriteLine(rand.Next(1, 11));

    //generate 10 random bytes
    byte[] bytes = new byte[10];
    rand.NextBytes(bytes);
    string text = System.Text.Encoding.ASCII.GetString(bytes);
    Console.WriteLine(text);

    //generate a random number between 0.0 and 1.0
    Console.WriteLine(rand.NextDouble());

    //generate random integer between 1 and 50 using custom seed
    int seed = (int)DateTime.Now.Ticks;
    rand = new Random(seed);
    Console.WriteLine(rand.Next(1, 51));
  }
}
```

Notice that in both Java and C#, the lower bound of the random number range is inclusive while the upper bound is exclusive. For example, a lower bound of 0 and an upper bound of 10 will generate a random number between 0 and 9. The output generated by Listing 7.28 looks like this:

```
1669848693
9
3
17\CsM↓IX?
0.660078493254296
15
```

Timers

Most software platforms provide a means of executing code at a future time or at specified intervals. Java and .NET are no exceptions. Though the timing functionality provided by both platforms is very similar, the manner in which this functionality is implemented differs significantly.

Class Comparison

The Java and .NET platforms provide timing functionality through the `java.util.Timer` and `System.Threading.Timer` classes, respectively. Additionally, timer events invoke methods defined by Java's `java.util.TimerTask` abstract class or .NET's `System.Threading.TimerCallback` delegate. Tables 7.9 and 7.10 compare these classes.

Table 7.9 ***Timer classes***

Type **/ Member Description**	**Type**	**[Class] / [Struct]ure / [C]onstructor / [M]ethod / [P]roperty / [I]ndexer / [O]perator / [F]ield**
Timer	Class	java.util.**Timer**
	Class	System.Threading.**Timer**
Create timer, set start time	C,M	void new **Timer**().**schedule**(long *delayInMillis*)
	C,M	void new **Timer**().**schedule**(Date *startTime*)
	C	new **Timer**(TimerCallback *callback*, object *state*, int *delayInMillis*, 0)
	C	new **Timer**(TimerCallback *callback*, object *state*, long *delayInMillis*, 0)
	C	new **Timer**(TimerCallback *callback*, object *state*, TimeSpan *delay*, new TimeSpan(0))
	C	new **Timer**(TimerCallback *callback*, object *state*, uint *delayInMillis*, 0)
Create timer, set start time and interval	C,M	void new **Timer**().**schedule**(TimerTask *task*, long *initialDelayInMillis*, long *intervalInMillis*)
	C,M	void new **Timer**().**schedule**(TimerTask *task*, Date *firstTime*, long *intervalInMillis*)
	C,M	void new **Timer**().**scheduleAtFixedRate**(TimerTask *task*, long *initialDelayInMillis*, long *intervalInMillis*)
	C,M	void new **Timer**().**scheduleAtFixedRate**(TimerTask *task*, Date *firstTime*, long *intervalInMillis*)
	C	new **Timer**(TimerCallback *callback*, object *state*, int *initialDelayInMillis*, int *intervalInMillis*)
	C	new **Timer**(TimerCallback *callback*, object *state*, long *initialDelayInMillis*, long *intervalInMillis*)
	C	new **Timer**(TimerCallback *callback*, object *state*, TimeSpan *initialDelay*, TimeSpan *interval*)
	C	new **Timer**(TimerCallback *callback*, object *state*, uint *initialDelayInMillis*, uint *intervalInMillis*)

Table 7.10 ***Timer event objects***

Type **/ Member Description**	**Type**	**[Class] / [Struct]ure / [Del]egate / [C]onstructor / [M]ethod / [P]roperty / [I]ndexer / [O]perator / [F]ield**
Timer event objects	Class	java.util.**TimerTask**
	Del	System.Threading.**TimerCallback**
Timer event method	M	void **run**()
	C	new **TimerCallback**(String *methodName*)

Code Examples

The following examples demonstrate these common timing functions:

- Single delayed event
- Fixed rate recurring events
- Fixed delay recurring events

Single Delayed Event

A *single delayed event* is an action that takes place once following a specific amount of time. Java and .NET support single delayed events by specifying an initial delay without stipulating a recurring interval. Listings 7.29 and 7.30 demonstrate single delayed events.

Java

Listing 7.29 ***Single delayed event (Java)***

```
import java.util.Timer;
import java.util.TimerTask;
import java.io.IOException;

public class SingleDelayedEvent
{
  static java.util.Date now = null;

  public static void main(String[] args) throws IOException
  {
    Timer timer = new Timer(true);
    TimerTask task = new MyTimerTask();

    now = new java.util.Date();
    System.out.println("Timer scheduled at: " + now);

    //schedule timer to fire event after 3 seconds
    timer.schedule(task, 3000);

    System.in.read(); //pause until a key is pressed
  }

  static class MyTimerTask extends TimerTask
  {
    public void run()
    {
      now = new java.util.Date();
      System.out.println("Timer fired at: " + now);
    }
  }
}
```

Listing 7.29 schedules a timer to invoke the `run()` method of `MyTimerTask` after a 3 second delay. The output from Listing 7.29 looks like this:

```
Timer scheduled at: Mon Feb 02 22:06:30 MST 2004
Timer fired at: Mon Feb 02 22:06:33 MST 2004
```

Notice that a boolean value of `true` was passed to the `Timer` class's constructor. This value indicates that the timer should run as a "daemon" thread that is automatically killed when its parent program terminates (i.e., the program that spawned it). Passing no parameters or a boolean `false` to the `Timer` constructor creates a timer that continues to execute in the background (firing the `TimerTask` after the `main()` method has exited). The true parameter was passed to the constructor in order to mimic the behavior of .NET's `Timer` class (which always terminates the `Timer` thread when the program ends).

C#

Listing 7.30 ***Single delayed event (C#)***

```
using System;
using System.Threading;

public class SingleDelayedEvent
{
  public static void Main()
  {
    TimerCallback method = new TimerCallback(TimerMethod);

    Console.WriteLine("Timer scheduled at: " + DateTime.Now);

    //schedule the timer to fire event after 3 seconds
    Timer timer = new Timer(method, null, 3000, 0);

    Console.Read(); //pause until a key is pressed
  }

  static void TimerMethod(object o)
  {
    Console.WriteLine("Timer fired at: " + DateTime.Now);
  }
}
```

Notice that the C# program invokes the `Timer` event using a method delegate rather than a class like Java. The output from Listing 7.30 looks like this:

```
Timer scheduled at: 2/2/2004 10:26:47 PM
Timer fired at: 2/2/2004 10:26:50 PM
```

Fixed Rate Recurring Events

A *fixed rate recurring event* is an action that repeats in scheduled intervals relative to the initial execution time. Though Java and .NET timers support fixed rate recurring events, their behavior differs slightly. In .NET, the rate is fixed because each recurring event is executed on a separate thread. Because the .NET timer is multithreaded, subsequent events can begin on schedule even if execution of the previous event has not completed. This allows events to overlap (i.e., occur concurrently). On the other hand, the Java timer is not multithreaded. In Java, if execution of an action takes longer than the specified interval, subsequent actions are delayed. Upon completion, delayed actions are executed in rapid succession in order to "catch up." Actions may be performed less than one interval apart in order to maintain a constant rate. Listings 7.31 and 7.32 demonstrate how to schedule fixed rate recurring events in Java and .NET.

Java

Listing 7.31 ***Fixed Rate Recurring Events (Java)***

```
import java.util.Timer;
import java.util.TimerTask;
import java.io.IOException;

public class FixedRateRecurringEvents
{
```

```
  static java.util.Date now = null;

  public static void main(String[] args) throws IOException
  {
    Timer timer = new Timer(true);
    TimerTask task = new MyTimerTask();

    now = new java.util.Date();
    System.out.println("Scheduling timer at: " + now);

    //recur every 5 seconds after initial 3 second delay
    timer.scheduleAtFixedRate(task, 3000, 5000);

    System.in.read();
  }

  static class MyTimerTask extends TimerTask
  {
    public void run()
    {
      now = new java.util.Date();
      System.out.println("Timer fired at: " + now);

      try
      {
        Thread.sleep(2000); //sleep to demonstrate fixed rate
      }
      catch (InterruptedException ignored)
      {
      }
    }
  }
}
```

Listing 7.31 schedules a timer to invoke the `run()` method of `MyTimerTask` every 5 seconds following an initial 3 second delay. The output from Listing 7.31 looks like this:

```
Scheduling timer at: Wed Feb 04 22:32:33 MST 2004
Timer fired at: Wed Feb 04 22:32:36 MST 2004
Timer fired at: Wed Feb 04 22:32:41 MST 2004
Timer fired at: Wed Feb 04 22:32:46 MST 2004
Timer fired at: Wed Feb 04 22:32:51 MST 2004
Timer fired at: Wed Feb 04 22:32:56 MST 2004
```

C#

Listing 7.32 *Fixed Rate Recurring Events (C#)*

```
using System;
using System.Threading;

public class FixedRateRecurringEvents
{
  public static void Main()
  {
    TimerCallback method = new TimerCallback(TimerMethod);

    Console.WriteLine("Timer scheduled at: " + DateTime.Now);

    //recur every 5 seconds after initial 3 second delay
```

```
    Timer timer = new Timer(method, null, 3000, 5000);

    Console.Read(); //pause until a key is pressed
  }

  static void TimerMethod(object o)
  {
    Console.WriteLine("Timer fired at: " + DateTime.Now);

    Thread.Sleep(2000); //sleep to demonstrate fixed rate
  }
}
```

Listing 7.32 schedules a timer to invoke the `TimerMethod()` method every 5 seconds following an initial 3 second delay. The output from Listing 7.32 looks like this:

```
Timer scheduled at: 2/4/2004 10:38:34 PM
Timer fired at: 2/4/2004 10:38:37 PM
Timer fired at: 2/4/2004 10:38:42 PM
Timer fired at: 2/4/2004 10:38:47 PM
Timer fired at: 2/4/2004 10:38:52 PM
Timer fired at: 2/4/2004 10:38:57 PM
```

Fixed Delay Recurring Events

A *fixed delay recurring event* is an action that repeats in intervals relative to the time required to execute the previous action. As long as each action's execution time does not exceed that of the scheduled interval, each subsequent action will be performed on schedule. If execution of an action takes longer than the scheduled interval, subsequent actions will be delayed proportionately. In other words, the subsequent action will wait one interval following the completion of the previous action. In order to maintain a constant delay, actions are never performed less than one interval apart. Listings 7.33 and 7.34 demonstrate how to schedule fixed delay recurring events.

Java

Listing 7.33 ***Fixed Delay Recurring Events (Java)***

```
import java.util.Timer;
import java.util.TimerTask;
import java.io.IOException;

public class FixedDelayRecurringEvents
{
  static java.util.Date now = null;
  static int numInterval = 1;

  public static void main(String[] args) throws IOException
  {
    Timer timer = new Timer(true);
    TimerTask task = new MyTimerTask();

    now = new java.util.Date();
    System.out.println("Scheduling timer at: " + now);

    //recur every 5 seconds after initial 3 second delay
    timer.schedule(task, 3000, 5000);

    System.in.read();
```

```
  }

  static class MyTimerTask extends TimerTask
  {
    public void run()
    {
      now = new java.util.Date();
      System.out.println("Timer fired at: " + now);

      try
      {
        //sleep 7 seconds on first 2 calls, then 2 seconds
        if (numInterval++ <= 2)
          Thread.sleep(7000);
        else
          Thread.sleep(2000);
      }
      catch (InterruptedException ignored)
      {
      }
    }
  }
}
```

Listing 7.33 schedules a timer to invoke the `run()` method of `MyTimerTask` every 5 seconds following an initial 3 second delay. The output from Listing 7.33 looks like this:

```
Scheduling timer at: Thu Feb 05 22:38:11 MST 2004
Timer fired at: Thu Feb 05 22:38:14 MST 2004
Timer fired at: Thu Feb 05 22:38:21 MST 2004
Timer fired at: Thu Feb 05 22:38:28 MST 2004
Timer fired at: Thu Feb 05 22:38:33 MST 2004
Timer fired at: Thu Feb 05 22:38:38 MST 2004
Timer fired at: Thu Feb 05 22:38:43 MST 2004
```

Notice that since the first 2 calls to the `TimerTask`'s `run()` method required 7 seconds of execution time, those events are 7 seconds apart while the following events fire at the normally scheduled interval of 5 seconds.

C#

Listing 7.34 *Fixed Delay Recurring Events (C#)*

```
using System;
using System.Threading;

public class FixedDelayRecurringEvents
{
  const int INTERVAL = 5000;
  static int numInterval = 1;
  static Timer timer = null;
  static bool timerMethodRunning = false;

  public static void Main()
  {
    TimerCallback method = new TimerCallback(TimerMethod);

    Console.WriteLine("Timer scheduled at: " + DateTime.Now);

    timer = new Timer(method, null, 3000, INTERVAL);
```

```
    Console.Read(); //pause until a key is pressed
  }

  static void TimerMethod(object o)
  {
    //determine if previous timer execution is still running
    if (timerMethodRunning)
    {
      //still running so cancel timer, reset timer flag, and exit
      timer.Change(-1, -1);
      timerMethodRunning = false;
      return;
    }
    else //set flag indicating that we are in the timer method
      timerMethodRunning = true;

    Console.WriteLine("Timer fired at: " + DateTime.Now);

    //sleep 7 seconds on first 2 calls, then 2 seconds after that
    if (numInterval++ <= 2)
      Thread.Sleep(7000);
    else
      Thread.Sleep(2000);

    //if timer flag has been reset, the timer must be restarted
    if (!timerMethodRunning)
      timer.Change(0, INTERVAL); //fire now then wait interval
    else
      timerMethodRunning = false; //reset timer flag
  }
}
```

Listing 7.34 schedules a timer to invoke the `TimerMethod()` method every 5 seconds following an initial 3 second delay. The output from Listing 7.34 looks like this:

```
Timer scheduled at: 2/5/2004 10:49:05 PM
Timer fired at: 2/5/2004 10:49:08 PM
Timer fired at: 2/5/2004 10:49:15 PM
Timer fired at: 2/5/2004 10:49:22 PM
Timer fired at: 2/5/2004 10:49:27 PM
Timer fired at: 2/5/2004 10:49:32 PM
Timer fired at: 2/5/2004 10:49:37 PM
```

Notice that, unlike Java, scheduling fixed delay events in .NET requires some tricky programming. Obviously, the .NET timer is primarily meant for fixed rate timing applications. However, with a little extra work, this timer can be tailored to execute fixed delay events as well.

Threads

It is often desirable for a program to perform multiple tasks simultaneously (such as monitoring user input while performing a lengthy operation). However, it is not possible for a single-CPU computer to execute more than one operation at a time. Fortunately, using a strategy called threading, program tasks (or threads) can be executed *virtually* concurrently. A *thread* represents a single path of execution. *Multithreading* allows multiple threads to execute simultaneously through a process known as time slicing. *Time slicing* is a method of switching back and forth between multiple threads in rapid succession. By constantly switching between threads, a little work can be performed on each thread at regular intervals. In effect, this rapid switching allows multiple threads to execute in parallel.

Class Comparison

In both Java and .NET, the primary threading class is named `Thread`. The `java.lang.Thread` Java class and the `System.Threading.Thread` .NET class represent single threads of execution. In addition, .NET provides the `System.Threading.Monitor` class for inter-thread communication. In Java, equivalent communication methods are implemented by `java.lang.Object` which is the base class from which all objects are derived. Tables 7.11 and 7.12 compare these types.

Table 7.11 *Thread classes*

***Type* / Member Description**	**Type**	**[Class] / [Struct]ure / [C]onstructor / [M]ethod / [P]roperty / [I]ndexer / [O]perator / [F]ield**
Thread classes	Class	`java.lang.`**`Thread`**
	Class	`System.Threading.`**`Thread`**
Get current thread	*M*	`Thread` **`currentThread`**`()`
	P	`Thread` **`CurrentThread`**
Kill thread	M	`void` **`destroy`**`()`
	M	`void` **`Abort`**`()`
Get name of thread	M	`String` **`getName`**`()`
	P	`string` **`Name`**
Get priority of thread	M	`int` **`getPriority`**`()`
	P	`ThreadPriority` **`Priority`**
Interrupt thread	M	`void` **`interrupt`**`()`
	M	`void` **`Interrupt`**`()`
Check if thread is alive	M	`boolean` **`isAlive`**`()`
	P	`bool` **`IsAlive`**
Determine thread state	M	`boolean` **`isInterrupted`**`()`
	P	`ThreadState` **`ThreadState`**
Determine if thread runs in background only	M	`boolean` **`isDaemon`**`()`
	P	`bool` **`IsBackground`**
Block calling thread until this thread dies	M	`void` **`join`**`(`*`[`*`long` *`millisecTimeout[`*`,` `int` *`nanosecTimeout]]`*`)`
	M	`void` **`Join`**`(`*`[`*`int` *`millisecTimeout]`*`)`
	M	`void` **`Join`**`(`*`[`*`TimeSpan` *`timeout]`*`)`
Mark thread as a background thread	M	`void` **`setDaemon`**`(boolean` *`isDaemon`*`)`
	P	`bool` **`IsBackground`**
Set name of thread	M	`void` **`setName`**`(String` *`name`*`)`
	P	`string` **`Name`**
Set priority of thread	M	`void` **`setPriority`**`(int` *`newPriority`*`)`
	P	`ThreadPriority` **`Priority`**
Put thread to sleep	*M*	`void` **`sleep`**`(long` *`millisToSleep[`*`,` `int` *`nanosToSleep]`*`)`
	M	`void` **`Sleep`**`(int` *`millisToSleep`*`)`

	M	`void Sleep(TimeSpan timeToSleep)`
Start execution of thread	M	`void start()`
	M	`void Start()`
Suspend the thread	M	`void suspend()` *(deprecated)*
	M	`void Suspend()`
Resume a suspended thread	M	`void resume()` *(deprecated)*
	M	`void Resume()`
Yield to any competing thread	*M*	`void yield()`
	M	`void Sleep(0)`

Table 7.12 ***Thread communication classes***

Type **/ Member Description**	**Type**	**[Class] / [Struct]ure / [C]onstructor / [M]ethod / [P]roperty / [I]ndexer / [O]perator / [F]ield**
Thread communication classes	Class	`java.lang.Object`
	Class	`System.Threading.Monitor`
Wake up a single waiting thread	M	`void notify()`
	M	`void Pulse(object lockedObject)`
Wake up all waiting threads	M	`void notifyAll()`
	M	`void PulseAll(object lockedObject)`
Cause thread to wait until awoken	M	`void wait([long millisTimeout[, int nanosTimeout]])`
	M	`bool Wait(object lockedObject[, int millisTimeout[, bool exitContext]])`
	M	`bool Wait(object lockedObject[, TimeSpan timeout[, bool exitContext]])`

Code Examples

This section includes the following code examples:

- Basic Threads
- Thread Safety
- Thread Communication

Basic Multithreading

In Java and .NET, multiple threads of execution can be spawned using each platform's respective `Thread` class. Listings 7.35 and 7.36 demonstrate how to execute two threads concurrently.

Java

Listing 7.35 ***Basic Multithreading Example (Java)***

```
public class BasicMultithreading implements Runnable
{
  public static void main(String[] args)
  {
    BasicMultithreading bm = new BasicMultithreading();

    Thread thread1 = new Thread(bm);
    Thread thread2 = new Thread(bm);

    thread1.start();
    thread2.start();
  }
```

```
  public void run()
  {
    for (char c = 'A'; c <= 'Z'; c++)
    {
      System.out.print(c);
      Thread.yield(); //yield to competing thread
    }
  }
}
```

In Java, a class can be executed in a separate thread if it extends the `java.lang.Thread` class or implements the `java.lang.Runnable` interface. Since Java supports only single inheritance, it is often preferable to implement the `Runnable` interface rather than extending `Thread` (thus allowing the threaded class the opportunity to inherit functionality from another class). The `Runnable` interface defines a single method having the following signature:

```
void run()
```

Therefore, any class that implements the `Runnable` interface must implement the `run()` method. Likewise, any class that extends `Thread` should also implement the `run()` method (though the `Thread` class does provide a default implementation of `run()`, this implementation does nothing).

A new `Thread` is instantiated by passing an instance of a `Runnable` class to the `Thread`'s constructor. Keep in mind that creating a `Thread` object is not the same as spawning a new thread of execution. On the contrary, a new thread is not spawned until the `Thread` object's `start()` method is called. When the `start()` method is invoked, the `Runnable` object's `run()` method is executed in a separate thread. When the method returns, the new thread is terminated.

In contrast to the `Runnable` approach, classes that extend `Thread` can be started like this:

```
new MyThreadedClass().start();
```

Since the behavior regarding how CPU cycles are shared between competing threads can vary between virtual machines, the sample code includes a call to `yield()` in order to allow the competing thread time to execute each time a letter is output. Without yielding, the first thread may be allowed to run to completion before the second thread is given any time (thus frustrating the demonstration of concurrently executing threads). Of course, yielding is not necessary for threads that execute lengthier operations (depending on priority, the JVM will eventually switch between competing threads).

C#

Listing 7.36 *Basic Multithreading Example (C#)*

```
using System;
using System.Threading;

public class BasicMultithreading
{
  public static void Main()
  {
    BasicMultithreading bm = new BasicMultithreading();

    Thread thread1 = new Thread(new ThreadStart(
      bm.PrintAlphabet));
```

```
    Thread thread2 = new Thread(new ThreadStart(
      bm.PrintAlphabet));

    thread1.Start();
    thread2.Start();
  }

  public void PrintAlphabet()
  {
    for (char c = 'A'; c <= 'Z'; c++)
    {
      Console.Write(c);
      Thread.Sleep(0); //yield to competing thread
    }
  }
}
```

A .NET `Thread` is instantiated by passing a `ThreadStart` delegate to its constructor. The `ThreadStart` delegate indicates the method that should be invoked when the `Thread` is started. Similar to Java, calling the `Thread`'s `Start()` method is what actually spawns a new thread in which the specified method begins to execute. When the method returns, the new thread is terminated. Serving the same purpose as Java's `yield()` method, the `Sleep()` method is invoked in order to yield control to the competing thread. This ensures that the first thread will not complete execution prior to the CLR switching to the second thread.

Output

The output produced by Listings 7.35 and 7.36 looks like this:

```
AABBCCDDEEFFGGHHIIJJKKLLMMNNOOPPQQRRSSTTUUVVWWXXYYZZ
```

Thread Safety

Thread safety is a primary concern in modern programming. An application can be considered *thread safe* if it always produces consistent results regardless of the number of threads running concurrently in its process space. To achieve this deterministic behavior, the programmer must ensure that no thread can access a shared resource while another thread is in the process of changing it. To illustrate, consider the following example:

1. Having an available balance of $1000, a bank customer withdraws $800 from an ATM. To process this transaction, the ATM first checks the balance to verify that sufficient funds exist and then decrements the balance by $800 as the funds are dispensed.
2. Simultaneously, the customer's spouse attempts to withdraw $500 from another ATM across town. Like before, this ATM first checks the balance to verify that sufficient funds exist. Typically, you would expect that this second transaction would fail due to insufficient funds. However, since these ATMs are not thread safe, the second ATM is able to verify the balance between the time the first ATM checks the balance and the instant it actually debits the account.
3. Since both ATMs retrieved a balance of $1000, both transactions are approved and the account is left overdrawn.

This example contains a thread safety issue known as a race condition. A *race condition* occurs when the correct behavior of a program relies on one thread completing a process before another (e.g., checking the balance *and* debiting an account before another thread checks the balance). Race conditions, among other thread safety errors, can produce sporadic results that can be exceptionally difficult to debug. Fortunately, Java and .NET both include programming mechanisms for ensuring thread safety. These mechanisms guarantee that thread sensitive portions of code can only be executed by a single thread at a time (resulting in one indivisible, or *atomic*, operation), thus eliminating race conditions and other common thread safety concerns. Listings 7.37 and 7.38 demonstrate how blocks of code can be restricted to a single thread at a time.

Java

Listing 7.37 ***Thread Safety Example (Java)***

```
public class ThreadSafety implements Runnable
{
  public static void main(String[] args)
  {
    ThreadSafety ts = new ThreadSafety();

    Thread thread1 = new Thread(ts);
    Thread thread2 = new Thread(ts);

    thread1.start();
    thread2.start();
  }

  public void run()
  {
    synchronized (this)
    {
      for (char c = 'A'; c <= 'Z'; c++)
      {
        System.out.print(c);

        try
        {
          Thread.sleep(1);
        }
        catch (InterruptedException ignored) {}
      }
    }
  }
}
```

The `synchronized` keyword allows a thread to acquire an exclusive lock (or monitor) on a specified object. Code contained within a `synchronized` block can only be executed by a thread that owns the associated lock. All competing threads are blocked until the first thread exits the `synchronized` section and releases the lock, at which point only one of the waiting threads is allowed to acquire the lock and continue execution (while the others continue to wait). By synchronizing the code in the `run()` method, the alphabet loop becomes a single indivisible operation notwithstanding the fact that the thread sleeps for a short time during each iteration.

> **NOTE**
> In addition to a block of code, an entire method can be marked `synchronized` like this: `public synchronized void run()`

C#

Listing 7.38 ***Thread Safety Example (C#)***

```
using System;
using System.Threading;

public class ThreadSafety
```

```
{
  public static void Main()
  {
    ThreadSafety ts = new ThreadSafety();

    Thread thread1 = new Thread(new ThreadStart(
      ts.PrintAlphabet));
    Thread thread2 = new Thread(new ThreadStart(
      ts.PrintAlphabet));

    thread1.Start();
    thread2.Start();
  }

  public void PrintAlphabet()
  {
    lock (this)
    {
      for (char c = 'A'; c <= 'Z'; c++)
      {
        Console.Write(c);
        Thread.Sleep(1);
      }
    }
  }
}
```

C# uses the `lock` keyword to acquire an object's exclusive lock. Like Java's `synchronized` keyword, code contained in a `lock` block can only be executed by a single thread at a time. The `lock` keyword is a syntactic shortcut to calling the `Monitor` object's `Enter()` and `Exit()` methods like this:

```
Monitor.Enter(this);
try
{
  for (char c = 'A'; c <= 'Z'; c++)
  {
    Console.Write(c);
    Thread.Sleep(1);
  }
}
finally
{
  Monitor.Exit(this);
}
```

Like the Java example, by locking the code in the `PrintAlphabet()` method, the alphabet loop executes as a single indivisible operation regardless of the fact that the thread sleeps for a short time during each iteration.

Output

The output produced by Listings 7.37 and 7.38 looks like this:

```
ABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZ
```

Thread Communication

At times, it may be necessary for threads to cooperate in order to complete a task. To facilitate cooperation, Java and .NET both provide a means for threads to communicate. Java uses the `notify()` method while .NET uses `Pulse()` to wake up sleeping threads. These methods wake up the next thread that is waiting on the current lock as soon as the lock is released by the current thread. Java and .NET use the `wait()` and `Wait()` methods, respectively, to force a thread to surrender control to a waiting thread. Listings 7.39 and 7.40 demonstrate thread communication.

Java

Listing 7.39 ***Thread Communication Example (Java)***

```
public class ThreadCommunication implements Runnable
{
  public static void main(String[] args)
  {
    ThreadCommunication tc = new ThreadCommunication();

    Thread thread1 = new Thread(tc);
    Thread thread2 = new Thread(tc);

    thread1.start();
    thread2.start();
  }

  public void run()
  {
    synchronized (this)
    {
      for (char c = 'A'; c <= 'Z'; c++)
      {
        System.out.print(c);

        this.notify(); //notify waiting thread

        try
        {
          if (c < 'Z') //only enter wait state if still looping
            this.wait(); //release lock to waiting thread
        }
        catch (InterruptedException ignored)
        {
        }
      }
    }
  }
}
```

Since the code in the `run()` method is synchronized, the first thread would normally complete execution before the second thread can enter the `synchronized` block. However, by calling `notify()` to notify the waiting thread and then calling `wait()`, the first thread is forced to relinquish control to the second. This “notify then wait” process is repeated in both threads for each iteration of the `for` loop. To avoid a deadlock situation where each thread is stuck waiting for the other, the `wait()` method is only invoked until the end of the loop. That is, once a thread’s loop reaches “Z”, `wait()` is not called and the thread is allowed to terminate normally by exiting the `run()` method.

C#

Listing 7.40 ***Thread Communication Example (C#)***

```
public class ThreadCommunication
{
  public static void Main()
  {
    ThreadCommunication tc = new ThreadCommunication();

    Thread thread1 = new Thread(new ThreadStart(
      tc.PrintAlphabet));
    Thread thread2 = new Thread(new ThreadStart(
      tc.PrintAlphabet));

    thread1.Start();
    thread2.Start();
  }

  public void PrintAlphabet()
  {
    lock (this)
    {
      for (char c = 'A'; c <= 'Z'; c++)
      {
        Console.Write(c);

        Monitor.Pulse(this); //notify waiting thread

        if (c < 'Z') //only enter wait state if still looping
          Monitor.Wait(this); //release lock to waiting thread
      }
    }
  }
}
```

Like the Java example, since the code in `PrintAlphabet()` is locked, the first thread would normally complete its loop before the second thread could enter the `lock` block. However, since the `Pulse()` and `Wait()` methods are called on each iteration of the `for` loop, control is transferred back and forth between both threads. In effect, by pulsing and then waiting, the two threads are able to execute the locked code concurrently.

Output

The output generated by Listings 7.39 and 7.40 looks like this:

```
AABBCCDDEEFFGGHHIIJJKKLLMMNNOOPPQQRRSSTTUUVVWWXXYYZZ
```

Collections

A *collection* is a data structure that can store a group of objects and perform operations on them. Collections are among the most common and useful programming structures. Java and .NET support various types of collections. Each type has unique characteristics that make it best suited for particular tasks. For instance, some collection types are optimized for addition or insertion speed while others are optimized for searching and sorting performance. In addition to performance discrepancies between functions, collection types may behave very differently.

Some collections may use a first-in/first-out (FIFO) policy while others employ a last-in/first-out (LIFO) strategy. Some may maintain the order of the collection or allow duplicate elements while others may not. It is up to the programmer to determine which collection type is best suited to a particular application.

Class Comparison

For almost every collection-related type in Java, there is an equivalent type in .NET and vice-versa. These types consist of interfaces that define collection behavior and classes implement it. For example, collection interfaces may define the methods used to iterate through a collection or add and remove elements from it. On the other hand, collection classes provide a concrete implementation of these methods. The following tables compare many of the Java and .NET platforms' most common interfaces and classes pertaining to collections.

Table 7.13 ***Iteration interface***

Type **/ Member Description**	**Type**	**[Class] / [Struct]ure / [Inter]face / [C]onstructor / [M]ethod / [P]roperty / [I]ndexer / [O]perator / [F]ield**
Iteration interface	Inter	`java.util.`**`Iterator`** *(also see* `Enumeration` *and* `ListIterator`*)*
	Inter	`System.Collections.`**`IEnumerator`**
Get current element	--	*no direct equivalent, use* `next()`
	P	`object` **`Current`**
Determine if next element exists	M	`boolean` **`hasNext`**`()`
	--	*no direct equivalent, use* `MoveNext()`
Move to next element	M	`object` **`next`**`()`
	M	`bool` **`MoveNext`**`()`
Remove current element	M	`void` **`remove`**`()`
	--	*no equivalent*
Reset pointer to initial position	--	*no equivalent*
	M	`void` **`Reset`**`()`

Table 7.14 ***Compare to interface***

Type **/ Member Description**	**Type**	**[Class] / [Struct]ure / [Inter]face / [C]onstructor / [M]ethod / [P]roperty / [I]ndexer / [O]perator / [F]ield**
Compare to interface	Inter	`java.lang.`**`Comparable`**
	Inter	`System.Collections.`**`IComparable`**
Compare current object to given object	M	`int` **`compareTo`**`(Object` *`o`*`)`
	M	`int` **`CompareTo`**`(object` *`o`*`)`

Table 7.15 ***Compare interface***

Type **/ Member Description**	**Type**	**[Class] / [Struct]ure / [C]onstructor / [M]ethod / [P]roperty / [I]ndexer / [O]perator / [F]ield / [Inter]face**
Compare interface	Inter	`java.util.`**`Comparator`**
	Inter	`System.Collections.`**`Comparer`**
Compare two objects	M	`int` **`compare`**`(Object` *`a`*`, Object` *`b`*`)`
	M	`int` **`Compare`**`(object` *`a`*`, object` *`b`*`)`

Table 7.16 ***List class***

Type **/ Member Description**	**Type**	**[Class] / [Struct]ure / [Inter]face / [C]onstructor / [M]ethod / [P]roperty / [I]ndexer / [O]perator / [F]ield**
List class	Class	`java.util.`**`ArrayList`** *(also see* `Vector`*)*

	Class	`System.Collections.`**`ArrayList`**
Add element to list	M	`boolean` **`add`**`(Object` *`o`*`)`
	M	`boolean` **`add`**`(int` *`index`*`, Object` *`o`*`)`
	M	`int` **`Add`**`(object` *`o`*`)`
Add collection of elements to list	M	`boolean` **`addAll`**`(Collection` *`c`*`)`
	M	`boolean` **`addAll`**`(int` *`index`*`, Collection` *`c`*`)`
	M	`void` **`AddRange`**`(ICollection` *`c`*`)`
Remove all elements from list	M	`void` **`clear`**`()`
	M	`void` **`Clear`**`()`
Create shallow copy of list	M	`Object` **`clone`**`()`
	M	`object` **`Clone`**`()`
Determine if list contains a specific element	M	`boolean` **`contains`**`(Object` *`o`*`)`
	M	`bool` **`Contains`**`(object` *`o`*`)`
Specify capactiy	P	`int` **`Capacity`**
	M	`void` **`ensureCapacity`**`(int` *`minCapacity`*`)`
Get specified element	M	`Object` **`get`**`(int` *`index`*`)`
	I	`object` **`this[`**`int` *`index`***`]`**
Locate specified element	M	`int` **`indexOf`**`(Object` *`element`*`)`
	M	`int` **`IndexOf`**`(object` *`element`*`)`
	M	`int` **`IndexOf`**`(object` *`element`*`, int` *`startIndex`*`)`
	M	`int` **`IndexOf`**`(object` *`element`*`, int` *`startIndex`*`, int` *`length`*`)`
Determine if list is empty	M	`boolean` **`isEmpty`**`()`
	P	`int` **`Count == 0`**
Locate last occurrence of specified element	M	`int` **`lastIndexOf`**`(Object` *`element`*`)`
	M	`int` **`LastIndexOf`**`(object` *`element`*`)`
	M	`int` **`LastIndexOf`**`(object` *`element`*`, int` *`startIndex`*`)`
	M	`int` **`LastIndexOf`**`(object` *`element`*`, int` *`startIndex`*`, int` *`length`*`)`
Remove specified element	M	`boolean` **`remove`**`(Object` *`element`*`)`
	M	`void` **`Remove`**`(object` *`element`*`)`
Remove element at specified location	M	`Object` **`remove`**`(int` *`index`*`)`
	M	`void` **`RemoveAt`**`(int` *`index`*`)`
Remove range of elements from list	M	`void` **`removeRange`**`(int` *`startIndex`*`, int` *`endIndex`*`)`
	M	`void` **`RemoveRange`**`(int` *`startIndex`*`, int` *`length`*`)`
Store element at specified index	M	`Object` **`set`**`(int` *`index`*`, Object` *`element`*`)`
	I	**`this[`**`int` *`index`***`]`**` = (object` *`element`*`)`
Get size of list	M	`int` **`size`**`()`
	P	`int` **`Count`**
Populate array with elements in list	M	`Object[]` **`toArray`**`()`
	M	`Object[]` **`toArray`**`(Object[]` *`array`*`)`
	M	`object[]` **`ToArray`**`()`
	M	`Array` **`ToArray`**`(Type` *`type`*`)`
	M	`void` **`CopyTo`**`(Array` *`array`*`)`

	M	`void` **`CopyTo`**`(Array` *`array`*`, int` *`startIndex`*`)`
	M	`void` **`CopyTo`**`(Array` *`array`*`, int` *`startIndex`*`, int` *`length`*`)`
Trim list capacity to current size	M	`void` **`trimToSize`**`()`
	M	`void` **`TrimToSize`**`()`

Table 7.17 ***Key/Value Pair Class***

Type **/ Member Description**	**Type**	**[Class] / [Struct]ure / [Inter]face / [C]onstructor / [M]ethod / [P]roperty / [I]ndexer / [O]perator / [F]ield**
Key/Value pair class	Class	`java.util.`**`HashMap`** *(also see* `Hashtable`*)*
	Class	`System.Collections.`**`Hashtable`**
Removes all elements from collection	M	`void` **`clear`**`()`
	M	`void` **`Clear`**`()`
Create a shallow copy of the collection	M	`Object` **`clone`**`()`
	M	`object` **`Clone`**`()`
Determine if collection contains specified key	M	`boolean` **`containsKey`**`(Object` *`key`*`)`
	M	`bool` **`ContainsKey`**`(object` *`key`*`)`
	M	`bool` **`Contains`**`(object` *`key`*`)`
Determine if collection contains specified value	M	`boolean` **`containsValue`**`(Object` *`value`*`)`
	M	`bool` **`ContainsValue`**`(object` *`value`*`)`
Get value associated with specified key	M	`Object` **`get`**`(Object` *`key`*`)`
	I	`object` **`this[`**`object` *`key`***`]`**
Determine if collection is empty	M	`boolean` **`isEmpty`**`()`
	P	**`Count == 0`**
Get all keys contained in collection	M	`Set` **`keySet`**`()`
	P	`ICollection` **`Keys`**
Add key/value pair to collection	M	`Object` **`put`**`(Object` *`key`*`, Object` *`value`*`)`
	M	`void` **`Add`**`(object` *`key`*`, object` *`value`*`)`
Remove a key/value pair from the collection	M	`Object` **`remove`**`(Object` *`key`*`)`
	M	`void` **`Remove`**`(object` *`key`*`)`
Get number of key/value pairs in collection	M	`int` **`size`**`()`
	P	`int` **`Count`**
Get all values contained in collection	M	`Collection` **`values`**`()`
	P	`ICollection` **`Keys`**

Table 7.18 ***Key/Value Entry Type***

Type **/ Member Description**	**Type**	**[Class] / [Struct]ure / [Inter]face / [C]onstructor / [M]ethod / [P]roperty / [I]ndexer / [O]perator / [F]ield**
Key/Value entry type	Inter	`java.util.`**`Map.Entry`** *(also see* `Hashtable`*)*
	Struct	`System.Collections.`**`DictionaryEntry`**
Get key from entry	M	`Object` **`getKey`**`()`
	P	`object` **`Key`**
Get value from entry	M	`Object` **`getValue`**`()`
	P	`object` **`Value`**
Set value in entry	M	`Object` **`setValue`**`(Object` *`value`*`)`
	P	`object` **`Value`**

Table 7.19 *Queue Class*

Type / Member Description	Type	[Class] / [Struct]ure / [Inter]face / [C]onstructor / [M]ethod / [P]roperty / [I]ndexer / [O]perator / [F]ield
Queue class	Class	`java.util.`**`ConcurrentLinkedQueue`**
	Class	`System.Collections.`**`Queue`**
Add an element to the end of the queue	M	`boolean` **`add`**`(Object` *`element`*`)`
	M	`boolean` **`offer`**`(Object` *`element`*`)`
	M	`void` **`Enqueue`**`(object` *`element`*`)`
Remove all elements from queue	M	`void` **`clear`**`()`
	M	`void` **`Clear`**`()`
Determine if queue contains the specified element	M	`boolean` **`contains`**`(Object` *`element`*`)`
	M	`bool` **`Contains`**`(object` *`element`*`)`
Determine if queue is empty	M	`boolean` **`isEmpty`**`()`
	P	**`Count == 0`**
Get a class for iterating over each element in the queue	M	`Iterator` **`iterator`**`()`
	M	`IEnumerator` **`GetEnumerator`**`()`
Get element at beginning of the queue without removing it	M	`Object` **`peek`**`()`
	M	`Object` **`element`**`()`
	M	`object` **`Peek`**`()`
Get element at beginning of queue and remove it from queue	M	`Object` **`poll`**`()`
	M	`Object` **`remove`**`()`
	M	`object` **`Dequeue`**`()`
Get number of elements in queue	M	`int` **`size`**`()`
	P	`int` **`Count`**
Copy elements in queue to an array	M	`Object[]` **`toArray`**`()`
	M	`Object[]` **`toArray`**`(Object[]` *`array`*`)`
	M	`void` **`CopyTo`**`(Array` *`array`*`, int` *`index`*`)`

Table 7.20 *Stack Class*

Type / Member Description	Type	[Class] / [Struct]ure / [Inter]face / [C]onstructor / [M]ethod / [P]roperty / [I]ndexer / [O]perator / [F]ield
Stack class	Class	`java.util.`**`Stack`**
	Class	`System.Collections.`**`Stack`**
Determine if stack is empty	M	`boolean` **`empty`**`()`
	P	**`Count == 0`**
Get element at top of stack without removing it	M	`Object` **`peek`**`()`
	M	`object` **`Peek`**`()`
Get element at top of stack and remove from stack	M	`Object` **`pop`**`()`
	M	`object` **`Pop`**`()`
Add an element to the top of the stack	M	`Object` **`push`**`(Object` *`element`*`)`
	M	`void` **`Push`**`(object` *`element`*`)`
Determine if specified element is in stack	M	`int` **`search`**`(Object` *`element`*`)`
	M	`bool` **`Contains`**`(object` *`element`*`)`

Table 7.21 ***Bit Collection Class***

Type **/ Member Description**	**Type**	**[Class] / [Struct]ure / [Inter]face / [C]onstructor / [M]ethod / [P]roperty / [I]ndexer / [O]perator / [F]ield**
Bit collection class	Class	`java.util.`**`BitSet`**
	Class	`System.Collections.`**`BitArray`**
Perform a logical AND operation	M	`void` **`and`**`(BitSet` *`bits`*`)`
	M	`BitArray` **`And`**`(BitArray` *`bits`*`)`
Set all bits to false	M	`void` **`clear`**`()`
	M	`void` **`SetAll`**`(`**`false`**`)`
Set specified bit to false	M	`void` **`clear`**`(int` *`index`*`)`
	M	`void` **`Set`**`(int` *`index`*, **`false`**`)`
	I	**`this`**`[int` *`index`*`] =` **`false`**
Make a shallow copy of the bit collection	M	`Object` **`clone`**`()`
	M	`object` **`Clone`**`()`
Invert all bits	M	`void` **`flip`**`(`**`0, this.size()-1`**`)`
	M	`BitArray` **`Not`**`()`
Get bit at specified location	M	`boolean` **`get`**`(int` *`index`*`)`
	M	`bool` **`Get`**`(int` *`index`*`)`
	I	`bool` **`this[`**`int` *`index`***`]`**
Determine if collection is empty	M	`boolean` **`isEmpty`**`()`
	P	**`Count == 0`**
Perform a logical OR operation	M	`void` **`or`**`(BitSet` *`bits`*`)`
	M	`BitArray` **`Or`**`(BitArray` *`bits`*`)`
Set bit at specified location	M	`void` **`set`**`(int` *`index`*`)`
	M	`void` **`Set`**`(int` *`index`*`, bool` *`value`*`)`
	I	**`this[`**`int` *`index`***`]`** `= (bool` *`value`*`)`
Get number of bits in collection	M	`int` **`size`**`()`
	P	`int` **`Count`**
	P	`int` **`Length`**
Perform a logical XOR operation	M	`void` **`xor`**`(BitSet` *`bits`*`)`
	M	`BitArray` **`Xor`**`(BitArray` *`bits`*`)`

Table 7.22 ***Other types***

Type **/ Member Description**	**Type**	**[Class] / [Struct]ure / [Inter]face / [C]onstructor / [M]ethod / [P]roperty / [I]ndexer / [O]perator / [F]ield**
List collection interface	Inter	`java.util.`**`List`** *(implemented by `ArrayList`)*
	Inter	`System.Collections.`**`IList`** *(implemented by `ArrayList`)*
Key/Value pair collection interface	Inter	`java.util.`**`Map`** *(implemented by `HashMap`)*
	Inter	`System.Collections.`**`IDictionary`** *(implemented by `Hashtable`)*
Sorted key/value collection class	Class	`java.util.`**`TreeMap`**
	Class	`System.Collections.`**`SortedList`**

Code Examples

This section includes code examples that demonstrate the following types:

- Lists
- Key/Value Pair Lists
- Queues
- Stacks

Lists

List data types are dynamically sized arrays that store objects in a well-defined order. By default, objects are stored in the order in which they were added to the collection. Of course, the order can change when the list is sorted. In both Java and .NET, the list data type is called `ArrayList`.

Java

Listing 7.41 *ArrayList Example (Java)*

```
import java.util.ArrayList;
import java.util.Iterator;
import java.util.Collections;

public class ArrayListExample
{
  public static void main(String[] args)
  {
    //create list
    ArrayList colors = new ArrayList();

    //add colors to list
    colors.add("red");
    colors.add("white");
    colors.add("blue");

    //show size of list
    System.out.printf("Initial size: %d\n", colors.size());

    //determine if list contains color "white"
    boolean containsWhite = colors.contains("white");
    System.out.printf("Contains white: %s\n", containsWhite);

    //get first color from list
    String firstColor = (String)colors.get(0);
    System.out.printf("Initial first color: %s\n", firstColor);

    //remove first color from list
    colors.remove(0);

    //show size after remove
    System.out.printf("Size after remove: %d\n", colors.size());

    //get new first color from list
    firstColor = (String)colors.get(0);
    System.out.printf("First color after remove: %s\n", firstColor);

    //add color
    colors.add("red");

    //iterate through list
    for (Object color : colors)
    {
      System.out.printf("Color: %s\n", color);
    }
```

```
    //sort collection
    Collections.sort(colors);

    //iterate through sorted list
    for (Object color : colors)
    {
      System.out.printf("Sorted color: %s\n", color);
    }
  }
}
```

Notice that the `Collections.sort()` static method is used to sort the items in the `ArrayList`. The status methods provided by the `Collections` class can be used to sort any class that implements the `List` interface.

C#

Listing 7.42 ***ArrayList Example (C#)***

```
using System;
using System.Collections;

public class ArrayListExample
{
  public static void Main()
  {
    //create list
    ArrayList colors = new ArrayList();

    //add colors to list
    colors.Add("red");
    colors.Add("white");
    colors.Add("blue");

    //show size of list
    Console.WriteLine("Initial Size: {0}", colors.Count);

    //determine if list contains color "white"
    bool containsWhite = colors.Contains("white");
    Console.WriteLine("Contains white: {0}", containsWhite);

    //get first color from list
    string firstColor = (string)colors[0];
    Console.WriteLine("Initial first color: {0}", firstColor);

    //remove first color from list
    colors.RemoveAt(0);

    //show size after remove
    Console.WriteLine("Size after remove: {0}", colors.Count);

    //get new first color from list
    firstColor = (string)colors[0];
    Console.WriteLine("First color after remove: {0}",
      firstColor);

    //add color
    colors.Add("red");

    //iterate through list
    foreach (string color in colors)
```

```
    {
      Console.WriteLine("Color: {0}", color);
    }

    //sort collection
    colors.Sort();

    //iterate through sorted list
    foreach (string color in colors)
    {
      Console.WriteLine("Sorted color: {0}", color);
    }
  }
}
```

Notice that, unlike Java, the `Sort()` method is implemented by the `ArrayList` itself. Simply call the `ArrayList` object's `Sort()` method to sort the list.

Output

The output produced by Listings 7.41 and 7.42 looks like this:

```
Initial size: 3
Contains white: true
Initial first color: red
Size after remove: 2
First color after remove: white
Color: white
Color: blue
Color: red
Sorted color: blue
Sorted color: red
Sorted color: white
```

Key/Value Pair Lists

Using a key/value pair list, an object (i.e., value) can be bound to a name (i.e., key) when it is added to the list. The key name provides quick access to its associated object. The key/value pair list data type is named `HashMap` in Java and `Hashtable` in .NET (though Java also provides a `Hashtable` class, `HashMap` is more equivalent to .NET's `Hashtable`).

Java

Listing 7.43 ***HashMap Example (Java)***

```
import java.util.HashMap;

public class HashMapExample
{
  public static void main(String[] args)
  {
    //create key/value pair list
    HashMap customers = new HashMap();

    //create three customers
    Customer c1 = new Customer("1", "Larry");
    Customer c2 = new Customer("2", "Curly");
    Customer c3 = new Customer("3", "Moe");

    //add customers to list
    customers.put(c1.id, c1);
    customers.put(c2.id, c2);
```

```
    customers.put(c3.id, c3);

    //show size of list
    System.out.printf("Initial size: %d\n", customers.size());

    //determine if list contains customer with ID of "1"
    boolean containsKey = customers.containsKey("1");
    System.out.printf("Contains customer 1: %s\n", containsKey);

    //determine if list contains customer object c2
    boolean containsValue = customers.containsValue(c2);
    System.out.printf("Contains object c2: %s\n", containsValue);

    //get customer with ID of "3" from list
    Customer c = (Customer)customers.get("3");
    System.out.printf("Customer 3 name: %s\n", c.name);

    //iterate through all of the keys in the list
    for (Object key : customers.keySet())
    {
      System.out.printf("Key: %s\n", key);
    }

    //iterate through all of the values in the list
    for (Object value : customers.values())
    {
      System.out.printf("Value: %s\n", ((Customer)value).name);
    }

    //remove customer with ID of "2"
    customers.remove("2");

    //iterate through all of the values in the list
    for (Object value : customers.values())
    {
      System.out.printf("Value after remove: %s\n",
        ((Customer)value).name);
    }
  }


  static class Customer
  {
    public String id;
    public String name;

    public Customer(String id, String name)
    {
      this.id = id;
      this.name = name;
    }
  }
}
```

Notice that this class uses an inner class named `Customer`. Multiple `Customer` objects are instantiated and added to the `HashMap` using the customer ID as the key.

C#

Listing 7.44 ***Hashtable Example (C#)***

```
using System;
using System.Collections;

public class HashtableExample
{
  public static void Main()
  {
    //create key/value pair list
    Hashtable customers = new Hashtable();

    //create three customers
    Customer c1 = new Customer("1", "Larry");
    Customer c2 = new Customer("2", "Curly");
    Customer c3 = new Customer("3", "Moe");

    //add customers to list
    customers.Add(c1.id, c1);
    customers.Add(c2.id, c2);
    customers.Add(c3.id, c3);

    //show size of list
    Console.WriteLine("Initial size: {0}", customers.Count);

    //determine if list contains customer with ID of "1"
    bool containsKey = customers.ContainsKey("1");
    Console.WriteLine("Contains customer 1: {0}", containsKey);

    //determine if list contains customer object c2
    bool containsValue = customers.ContainsValue(c2);
    Console.WriteLine("Contains object c2: {0}", containsValue);

    //get customer with ID of "3" from list
    Customer c = (Customer)customers["3"];
    Console.WriteLine("Customer 3 name: {0}", c.name);

    //iterate through all of the keys in the list
    foreach (string key in customers.Keys)
    {
      Console.WriteLine("Key: {0}", key);
    }

    //iterate through all of the values in the list
    foreach (Customer value in customers.Values)
    {
      Console.WriteLine("Value: {0}", value.name);
    }

    //remove customer with ID of "2"
    customers.Remove("2");

    //iterate through all of the values in the list
    foreach (Customer value in customers.Values)
    {
      Console.WriteLine("Value after remove: {0}", value.name);
    }
  }
```

```
  class Customer
  {
    public String id;
    public String name;

    public Customer(String id, String name)
    {
      this.id = id;
      this.name = name;
    }
  }
}
```

Output

The output generated by Listings 7.43 and 7.44 looks like this:

```
Initial size: 3
Contains customer 1: true
Contains object c2: true
Customer 3 name: Moe
Key: 3
Key: 2
Key: 1
Value: Moe
Value: Curly
Value: Larry
Value after remove: Moe
Value after remove: Larry
```

Queues

A *queue* is a first-in/first-out (FIFO) data structure. That is, the first object added to the queue is the first object that will be removed. Java provides a number of different queue implementations. However, of these implementations, Java's `ConcurrentLinkedQueue` class is the most equivalent to the .NET `Queue` class. Listings 7.45 and 7.46 demonstrate these two classes.

Java

Listing 7.45 *Queue Example (Java)*

```
import java.util.concurrent.ConcurrentLinkedQueue;

public class QueueExample
{
  public static void main(String[] args)
  {
    //create queue
    ConcurrentLinkedQueue letters = new ConcurrentLinkedQueue();

    //add three items to queue
    letters.add("a");
    letters.add("b");
    letters.add("c");

    //show size of queue
    System.out.printf("Initial size: %d\n", letters.size());

    //determine if queue contains letter "b"
    boolean containsLetter = letters.contains("b");
    System.out.printf("Contains letter b: %s\n", containsLetter);
```

```
    //get first item in queue without removing it
    String letter = (String)letters.peek();
    System.out.printf("First item: %s\n", letter);

    //show size of queue after peek
    System.out.printf("Size after peek: %d\n", letters.size());

    //get first item and remove it from queue
    letter = (String)letters.poll();
    System.out.printf("First item: %s\n", letter);

    //show size of queue after poll
    System.out.printf("Size after poll: %d\n", letters.size());

    //get second item and remove it from queue
    letter = (String)letters.poll();
    System.out.printf("Second item: %s\n", letter);

    //show size of queue after second poll
    System.out.printf("Size after poll: %d\n", letters.size());
  }
}
```

The output generated by Listing 7.45 looks like this:

```
Initial size: 3
Contains letter b: true
First item: a
Size after peek: 3
First item: a
Size after poll: 2
Second item: b
Size after poll: 1
```

C#

Listing 7.46 ***Queue Example (C#)***

```
using System;
using System.Collections;

public class QueueExample
{
  public static void Main()
  {
    //create queue
    Queue letters = new Queue();

    //add three items to queue
    letters.Enqueue("a");
    letters.Enqueue("b");
    letters.Enqueue("c");

    //show size of queue
    Console.WriteLine("Initial size: {0}", letters.Count);

    //determine if queue contains letter "b"
    bool containsLetter = letters.Contains("b");
    Console.WriteLine("Contains letter b: {0}", containsLetter);
```

```
    //get first item in queue without removing it
    string letter = (string)letters.Peek();
    Console.WriteLine("First item: {0}", letter);

    //show size of queue after peek
    Console.WriteLine("Size after peek: {0}", letters.Count);

    //get first item and remove it from queue
    letter = (string)letters.Dequeue();
    Console.WriteLine("First item: {0}", letter);

    //show size of queue after dequeue
    Console.WriteLine("Size after dequeue: {0}", letters.Count);

    //get second item and remove it from queue
    letter = (string)letters.Dequeue();
    Console.WriteLine("Second item: {0}", letter);

    //show size of queue after second dequeue
    Console.WriteLine("Size after dequeue: {0}", letters.Count);
  }
}
```

The output produced by Listing 7.46 looks like this:

```
Initial size: 3
Contains letter b: True
First item: a
Size after peek: 3
First item: a
Size after dequeue: 2
Second item: b
Size after dequeue: 1
```

Stacks

A *stack* is a last-in/first-out (LIFO) data structure. That is, the first object added to the stack is the last object that will be removed. Java and .NET both provide a `Stack` class that implements LIFO behavior. Listings 7.47 and 7.48 demonstrates the `Stack` classes.

Java

Listing 7.47 *Stack Example (Java)*

```
import java.util.Stack;

public class StackExample
{
  public static void main(String[] args)
  {
    //create stack
    Stack letters  = new Stack();

    //add three items to stack
    letters.push("a");
    letters.push("b");
    letters.push("c");

    //show size of stack
    System.out.printf("Initial size: %d\n", letters.size());
```

```
    //determine if stack contains letter "a"
    boolean containsLetter = letters.contains("a");
    System.out.printf("Contains letter a: %s\n", containsLetter);

    //get first item in stack without removing it
    String letter = (String)letters.peek();
    System.out.printf("First item: %s\n", letter);

    //show size of stack after peek
    System.out.printf("Size after peek: %d\n", letters.size());

    //get first item and remove it from stack
    letter = (String)letters.pop();
    System.out.printf("First item: %s\n", letter);

    //show size of stack after pop
    System.out.printf("Size after pop: %d\n", letters.size());

    //get second item and remove it from stack
    letter = (String)letters.pop();
    System.out.printf("Second item: %s\n", letter);

    //show size of stack after second pop
    System.out.printf("Size after pop: %d\n", letters.size());
  }
}
```

C#

Listing 7.48 *Stack Example (C#)*

```
using System;
using System.Collections;

public class StackExample
{
  public static void Main()
  {
    //create stack
    Stack letters = new Stack();

    //add three items to stack
    letters.Push("a");
    letters.Push("b");
    letters.Push("c");

    //show size of stack
    Console.WriteLine("Initial size: {0}", letters.Count);

    //determine if stack contains letter "a"
    bool containsLetter = letters.Contains("a");
    Console.WriteLine("Contains letter a: {0}", containsLetter);

    //get first item in stack without removing it
    string letter = (string)letters.Peek();
    Console.WriteLine("First item: {0}", letter);

    //show size of stack after peek
    Console.WriteLine("Size after peek: {0}", letters.Count);
```

```
    //get first item and remove it from stack
    letter = (string)letters.Pop();
    Console.WriteLine("First item: {0}", letter);

    //show size of stack after pop
    Console.WriteLine("Size after pop: {0}", letters.Count);

    //get second item and remove it from stack
    letter = (string)letters.Pop();
    Console.WriteLine("Second item: {0}", letter);

    //show size of stack after second pop
    Console.WriteLine("Size after pop: {0}", letters.Count);
  }
}
```

Output

The output generated by Listings 7.47 and 7.48 looks like this:

```
Initial size: 3
Contains letter a: true
First item: c
Size after peek: 3
First item: c
Size after pop: 2
Second item: b
Size after pop: 1
```

File Input/Output

To one degree or another, almost all applications read from or write to files stored on the local hard drive or on the network. As you might expect, Java and .NET both provide classes that facilitate these common I/O operations. In Java, the file I/O classes include `File`, `FileReader`, `BufferedReader`, `FileWriter`, and `PrintWriter`. In .NET, the `FileInfo`, `DirectoryInfo`, `StreamReader`, and `StreamWriter` classes provide equivalent functionality.

Class Comparison

While both platforms provide equivalent functionality, their respective object libraries are architected differently. Therefore, there is not always a one-to-one mapping between Java and .NET classes. Rather, there are times when multiple Java classes are required in order to perform the functionality provided by a single .NET class and vice-versa. For example, both the `FileReader` and `BufferedReader` Java classes are required to perform the basic functions that are inherent within .NET's `StreamReader` class.

Table 7.23 ***File Class***

Type **/ Member Description**	**Type**	**[Class] / [Struct]ure / [Inter]face / [C]onstructor / [M]ethod / [P]roperty / [I]ndexer / [O]perator / [F]ield**
File class	Class	`java.io.`**`File`**
	Class	`System.IO.`**`FileInfo`**
Create a file	M	`boolean` **`createNewFile`**`()`

	M	FileStream **Create**()
	M	StreamWriter **CreateText**()
Delete a file	M	boolean **delete**()
	M	void **deleteOnExit**()
	M	void **Delete**()
Determine if a file exists	M	boolean **exists**()
	M	bool **Exists**()
Get file attributes	M	boolean **canRead**()
	M	boolean **canWrite**()
	M	boolean **isDirectory**()
	M	boolean **isHidden**()
	M	long **lastModified**()
	P	FileAttributes **Attributes**
	P	DateTime **CreationTime**
	P	DateTime **CreationTimeUtc**
	P	DateTime **LastAccessTime**
	P	DateTime **LastAccessTimeUtc**
	P	DateTime **LastWriteTime**
	P	DateTime **LastWriteTimeUtc**
Get file length	M	long **length**()
	P	long **Length**
Move a file	M	boolean **renameTo**(File *destination*)
	M	void **MoveTo**(string *destination*)
Get file name	M	String **getName**()
	P	string **Name**
Get full path	M	String **getPath**()
	P	string **FulleName**
Get parent directory	M	File **getParentFile**()
	P	DirectoryInfo **Directory**
Set file attributes	M	boolean **setLastModified**(long *time*)
	M	boolean **setReadOnly**()
	P	FileAttributes **Attributes**
	P	DateTime **CreationTime**
	P	DateTime **CreationTimeUtc**
	P	DateTime **LastAccessTime**
	P	DateTime **LastAccessTimeUtc**
	P	DateTime **LastWriteTime**
	P	DateTime **LastWriteTimeUtc**

Table 7.24 *Directory Class*

***Type* / Member Description**	**Type**	**[Class] / [Struct]ure / [Inter]face / [C]onstructor / [M]ethod / [P]roperty / [I]ndexer / [O]perator / [F]ield**
Directory class	Class	java.io.**File**
	Class	System.IO.**DirectoryInfo**

Create directory	M	boolean **mkdir**()
	M	boolean **mkdirs**()
	M	void **Create**()
Get directory attributes	M	boolean **canRead**()
	M	boolean **canWrite**()
	M	boolean **isDirectory**()
	M	boolean **isHidden**()
	M	long **lastModified**()
	P	FileAttributes **Attributes**
	P	DateTime **CreationTime**
	P	DateTime **CreationTimeUtc**
	P	DateTime **LastAccessTime**
	P	DateTime **LastAccessTimeUtc**
	P	DateTime **LastWriteTime**
	P	DateTime **LastWriteTimeUtc**
Delete directory	M	boolean **delete**()
	M	void **deleteOnExit**()
	M	void **Delete**()
Determine if directory exists	M	boolean **exists**()
	M	bool **Exists**()
Get directory name	M	String **getName**()
	P	string **Name**
Get parent directory	M	File **getParentFile**()
	P	DirectoryInfo **Parent**
Get root directory	M	File[] **listRoots**()
	M	DirectoryInfo **Root**
Get list of files in directory	M	String[] **list**(*[*FilenameFilter *filter]*)
	M	File[] **listFiles**()
	M	File[] **listFiles**(FileFilter *filter*)
	M	File[] **listFiles**(FilenameFilter *filter*)
	M	FileInfo[] **GetFiles**(*[*string *searchPattern]*)
	M	FileSystemInfo[] **GetFileSystemInfos**(*[*string *searchPattern]*)
Move a directory	M	boolean **renameTo**(File *destDir*)
	M	void **MoveTo**(string *destDirName*)
Set directory attributes	M	boolean **setLastModified**(long *time*)
	M	boolean **setReadOnly**()
	P	FileAttributes **Attributes**
	P	DateTime **CreationTime**
	P	DateTime **CreationTimeUtc**
	P	DateTime **LastAccessTime**
	P	DateTime **LastAccessTimeUtc**
	P	DateTime **LastWriteTime**

	P	DateTime **LastWriteTimeUtc**

Table 7.25 ***File Reader Class***

Type / **Member Description**	**Type**	**[Class] / [Struct]ure / [Inter]face / [C]onstructor / [M]ethod / [P]roperty / [I]ndexer / [O]perator / [F]ield**
File reader class	Class	`java.io.`**`FileReader`**
	Class	`System.IO.`**`StreamReader`**
Close the reader	M	`void `**`close`**`()`
	M	`void `**`Close`**`()`
Get character encoding	M	`String `**`getEncoding`**`()`
	P	`Encoding `**`CurrentEncoding`**
Read a single character	M	`int `**`read`**`()`
	M	`int `**`Read`**`()`
Read a block of characters	M	`int `**`read`**`(char[] `*`buffer[`*`, int `*`length`*`, int `*`length]`*`)`
	M	`int `**`read`**`(CharBuffer `*`buffer`*`)`
	M	`int `**`Read`**`(char[] `*`buffer`*`, int `*`length`*`, int `*`length`*`)`
	M	`int `**`ReadBlock`**`(char[] `*`buffer`*`, int `*`length`*`, int `*`length`*`)`

Table 7.26 ***Line Reader Class***

Type / **Member Description**	**Type**	**[Class] / [Struct]ure / [Inter]face / [C]onstructor / [M]ethod / [P]roperty / [I]ndexer / [O]perator / [F]ield**
Line reader class	Class	`java.io.`**`BufferedReader`**
	Class	`System.IO.`**`StreamReader`**
Read a line of text	M	`String `**`readLine`**`()`
	M	`string `**`ReadLine`**`()`

Table 7.27 ***File Writer Class***

Type / **Member Description**	**Type**	**[Class] / [Struct]ure / [Inter]face / [C]onstructor / [M]ethod / [P]roperty / [I]ndexer / [O]perator / [F]ield**
File writer class	Class	`java.io.`**`FileWriter`**
	Class	`System.IO.`**`StreamWriter`**
Close the output stream	M	`void `**`close`**`()`
	M	`void `**`Close`**`()`
Flush the output stream	M	`void `**`flush`**`()`
	M	`void `**`Flush`**`()`
Get character encoding	M	`String `**`getEncoding`**`()`
	P	`Encoding `**`Encoding`**
Get underlying output stream	F	`Writer `**`out`**
	P	`Stream `**`BaseStream`**
Write to the file	M	`void `**`write`**`(char[] `*`buffer[`*`, int `*`index`*`, int `*`length]`*`)`
	M	`void `**`write`**`(int `*`value`*`)`
	M	`void `**`write`**`(String `*`value[`*`, int `*`index`*`, int `*`length]`*`)`
	M	`void `**`Write`**`(char `*`value`*`)`
	M	`void `**`Write`**`(char[] `*`buffer[`*`, int `*`index`*`, int `*`length]`*`)`
	M	`void `**`Write`**`(string `*`value`*`)`
	M	`void `**`Write`**`(bool `*`value`*`)`
	M	`void `**`Write`**`(decimal `*`value`*`)`

	M	void **Write**(double *value*)
	M	void **Write**(int *value*)
	M	void **Write**(long *value*)
	M	void **Write**(float *value*)
	M	void **Write**(uint *value*)
	M	void **Write**(ulong *value*)
	M	void **Write**(object *value*)
	M	void **Write**(string *format*, object *arg*)
	M	void **Write**(string *format*, params object[] *args*)
	M	void **Write**(string *format*, object *arg0*, object *arg1*)
	M	void **Write**(string *format*, object *arg0*, object *arg1*, object *arg3*)

Table 7.28 ***Line Writer Class***

Type **/ Member Description**	**Type**	**[Class] / [Struct]ure / [Inter]face / [C]onstructor / [M]ethod / [P]roperty / [I]ndexer / [O]perator / [F]ield**
Line writer class	Class	java.io.**PrintWriter**
	Class	System.IO.**StreamWriter**
Write a line of text terminated by a new line character	M	void **println**()
	M	void **println**(boolean *value*)
	M	void **println**(char[] *value*)
	M	void **println**(double *value*)
	M	void **println**(float *value*)
	M	void **println**(int *value*)
	M	void **println**(long *value*)
	M	void **println**(Object *value*)
	M	void **println**(String *value*)
	M	void **WriteLine**()
	M	void **WriteLine**(bool *value*)
	M	void **WriteLine**(char *value*)
	M	void **WriteLine**(char[] *value[*, int *index*, int *length]*)
	M	void **WriteLine**(decimal *value*)
	M	void **WriteLine**(double *value*)
	M	void **WriteLine**(int *value*)
	M	void **WriteLine**(long *value*)
	M	void **WriteLine**(object *value*)
	M	void **WriteLine**(float *value*)
	M	void **WriteLine**(string *value*)
	M	void **WriteLine**(uint *value*)
	M	void **WriteLine**(ulong *value*)
	M	void **WriteLine**(string *value*)
	M	void **WriteLine**(string *format*, object *arg*)
	M	void **WriteLine**(string *format*, params object[] *args*)
	M	void **WriteLine**(string *format*, object *arg0*, object *arg1*)

	M	`void **WriteLine**(string *format*, object *arg0*, object *arg1*, object *arg3*)`

Code Examples

This section includes code examples that demonstrate the following file operations:

- Create, write, and append to a text file
- Read from a text file
- Create and examine a directory
- Rename, move, copy, and delete a file
- Get file attributes

Create, Write, and Append to a Text File

Java and .NET provide simple APIs for creating and writing to files. Java uses the `FileWriter` and `PrintWriter` classes to accomplish these tasks. .NET employs the `StreamWriter` class to perform equivalent functionality.

Java

Listing 7.49 ***Text File Creation Example (Java)***

```
import java.io.File;
import java.io.FileWriter;
import java.io.PrintWriter;
import java.io.IOException;

public class CreateTextFile
{
  public static void main(String[] args)
  {
    File newFile = null;
    FileWriter fw = null;

    try
    {
      newFile = new File("file.txt"); //create file object

      fw = new FileWriter(newFile); //open output stream

      fw.write("The colors of the rainbow are:"); //write file

      fw.close(); //close to flush output to file

      //open a new file writer in append mode (true means append)
      fw = new FileWriter(newFile, true);

      //chain to print writer to get println() functionality
      PrintWriter pw = new PrintWriter(fw);

      //append colors to the file
      pw.println();
      pw.println("Red");
      pw.println("Orange");
      pw.println("Yellow");
      pw.println("Green");
      pw.println("Blue");
      pw.println("Indigo");
      pw.println("Violet");
```

```
    }
    catch (IOException ioe)
    {
      System.out.println("Error: " + ioe);
      return;
    }
    finally
    {
      try
      {
        fw.close(); //close file writer in finally block
      }
      catch (IOException ignored) {}
    }

    System.out.println("File created at: " +
      newFile.getAbsolutePath());
  }
}
```

C#

Listing 7.50 ***Text File Creation Example (C#)***

```
using System;
using System.IO;

public class CreateTextFile
{
  public static void Main()
  {
    StreamWriter sw = null;

    try
    {
      sw = new StreamWriter("file.txt");

      sw.Write("The colors of the rainbow are:");

      sw.Close();

      sw = new StreamWriter("file.txt", true);

      sw.WriteLine();
      sw.WriteLine("Red");
      sw.WriteLine("Orange");
      sw.WriteLine("Yellow");
      sw.WriteLine("Green");
      sw.WriteLine("Blue");
      sw.WriteLine("Indigo");
      sw.WriteLine("Violet");
    }
    finally
    {
      sw.Close();
    }

    FileInfo fi = new FileInfo("file.txt");
    Console.WriteLine("File created at: {0}", fi.FullName);
  }
}
```

Read From a Text File

Reading from a text file in Java and .NET is similar to writing to one. Java employs the `FileReader` and `BufferedReader` classes to read character data from a file. .NET provides the `StreamReader` class for this purpose.

Java

Listing 7.51 ***Read Text File Example (Java)***

```
import java.io.File;
import java.io.FileReader;
import java.io.BufferedReader;
import java.io.IOException;

public class ReadTextFile
{
  public static void main(String[] args)
  {
    FileReader fr = null;

    try
    {
      File newFile = new File("file.txt"); //create file object

      fr = new FileReader(newFile);

      //chain to BufferedReader to get readLine() functionality
      BufferedReader br = new BufferedReader(fr);

      String line = br.readLine();

      while (line != null)
      {
        System.out.println(line);
        line = br.readLine();
      }
    }
    catch (IOException ioe)
    {
      System.out.println("Error: " + ioe);
      return;
    }
    finally
    {
      try
      {
        fr.close();
      }
      catch (IOException ignored) {}
    }
  }
}
```

C#

Listing 7.52 ***Read Text File Example (C#)***

```
using System;
using System.IO;

public class Temp
```

```
{
  public static void Main()
  {
    StreamReader sr = null;

    try
    {
      sr = new StreamReader("file.txt");

      string line = sr.ReadLine();

      while (line != null)
      {
        Console.WriteLine(line);
        line = sr.ReadLine();
      }
    }
    finally
    {
      sr.Close();
    }
  }
}
```

Output

The output generated by Listings 7.51 and 7.52 looks like this:

```
The colors of the rainbow are:
Red
Orange
Yellow
Green
Blue
Indigo
Violet
```

Create and Examine a Directory

Java employs a single class for manipulating both files and directories: `java.io.File`. This design decision stems from the fact that Java considers a directory to be a special type of file. .NET, on the other hand, treats files and directories differently and providse separate classes for working each: `System.IO.FileInfo` and `System.IO.DirectoryInfo`. In this section, we will demonstrate how the `File` and `DirectoryInfo` classes can be used to create and examine directories in Java and .NET, respectively.

Java

Listing 7.53 ***Directory Example (Java)***

```
import java.io.File;

public class DirectoryExample
{
  public static void main(String[] args)
  {
    File newDir = new File("new_dir");

    //if directory does not exist, create it
    if (!newDir.exists())
    {
```

```
      newDir.mkdir();
    }

    File existingDir = new File("existing_dir");

    //if directory exists, display its contents
    if (existingDir.exists() && existingDir.isDirectory())
    {
      File[] files = existingDir.listFiles();

      for (int x=0; x < files.length; x++)
      {
        File file = files[x];

        //indicate if the item is a file or directory
        if (file.isDirectory())
        {
          System.out.println("<dir>  " + file.getName());
        }
        else
        {
          System.out.println("<file> " + file.getName());
        }
      }
    }
  }
}
```

C#

Listing 7.54 *Directory Example (C#)*

```
using System;
using System.IO;

public class DirectoryExample
{
  public static void Main()
  {
    DirectoryInfo newDir = new DirectoryInfo("new_dir");

    //if directory does not exist, create it
    if (!newDir.Exists)
    {
      newDir.Create();
    }

    DirectoryInfo existDir = new DirectoryInfo("existing_dir");

    //if directory exists, display its contents
    if (existDir.Exists)
    {
      FileSystemInfo[] files = existDir.GetFileSystemInfos();

      foreach(FileSystemInfo fsi in files)
      {
        //indicate if the item is a file or directory
        if ((fsi.Attributes & FileAttributes.Directory) > 0)
        {
          Console.WriteLine("<dir>  {0}", fsi.Name);
        }
```

```
        else
        {
          Console.WriteLine("<file> {0}", fsi.Name);
        }
      }
    }
  }
}
```

Rename, Move, Copy, Delete

Java and .NET provide functionality to rename, move, copy, and delete files. As with most file I/O operations, this functionality is implemented by Java's `File` class and .NET's `FileInfo` and `DirectoryInfo` classes. However, unlike .NET, Java does not include a method to copy files. Therefore, Listing 7.55 implements a simple `copy()` method to make up for this deficiency.

Java

Listing 7.55 ***Rename, Move, Copy, Delete File Example (Java)***

```
import java.io.*;

public class RenameMoveCopyDelete
{
  public static void main(String[] args)
  {
    File file = new File("file.txt"); //create file object

    if (file.exists())
    {
      //rename file to "file2.txt"
      File newFile = new File("file2.txt");
      file.renameTo(newFile);

      //move file to temp directory
      File directory = new File("temp");
      if (!directory.exists())
      {
        directory.mkdir(); //create directory if it deosn't exist
      }
      File movedFile = new File(directory, newFile.getName());
      newFile.renameTo(movedFile);

      //copy file to "file3.txt" and "file4.txt"
      File copyFile1 = new File("file3.txt");
      copy(movedFile, copyFile1);
      File copyFile2 = new File("file4.txt");
      copy(movedFile, copyFile2);

      //delete "file4.txt" file
      copyFile2.delete();
    }
  }

  static void copy(File src, File dst)
  {
    FileReader fr = null;
    FileWriter fw = null;

    try
```

```
    {
      fr = new FileReader(src);
    }
    catch (FileNotFoundException fnfe)
    {
      System.out.println("File not found: " + src.getName());
      return;
    }

    BufferedReader br = new BufferedReader(fr);

    try
    {
      fw = new FileWriter(dst);
      PrintWriter pw = new PrintWriter(fw);

      String line = br.readLine();

      while (line != null)
      {
        pw.println(line);
        line = br.readLine();
      }
    }
    catch (IOException ioe)
    {
      System.out.println("Error: " + ioe);
    }
    finally
    {
      try
      {
        fr.close();
      }
      catch (IOException ignored) {}

      try
      {
        fw.close();
      }
      catch (IOException ignored) {}
    }
  }
}
```

C#

Listing 7.56 ***Rename, Move, Copy, Delete File Example (C#)***

```
using System;
using System.IO;

public class Temp
{
  public static void Main()
  {
    FileInfo file = new FileInfo("file.txt"); //create object

    if (file.Exists)
    {
      //rename file to "file2.txt"
```

```
      file.MoveTo("file2.txt");

      //move file to temp directory
      DirectoryInfo directory = new DirectoryInfo("temp");
      if (!directory.Exists)
      {
        directory.Create();
      }
      file.MoveTo("temp\\file2.txt");

      //copy file twice in original directory
      file.CopyTo("file3.txt");
      file.CopyTo("file4.txt");

      //delete "file4.txt" file
      FileInfo deleteFile = new FileInfo("file4.txt");
      deleteFile.Delete();
    }
  }
}
```

Get File Attributes

In Java, file attributes are accessible from the `File` class. In .NET, file attributes can be retrieved using a combination of the `FileInfo` and `FileAttributes` classes.

Java

Listing 7.57 ***Get File Attributes Example (Java)***

```
import java.io.File;

public class GetFileAttributes
{
  public static void main(String[] args)
  {
    File file = new File("file.txt"); //create file object

    if (file.exists())
    {
      System.out.printf("File length: %s\n", file.length());

      System.out.printf("Last modified: %s\n",
        new java.util.Date(file.lastModified()));

      System.out.printf("Path to file: %s\n",
        file.getAbsolutePath());

      System.out.printf("File is hidden: %s\n", file.isHidden());

      System.out.printf("File is directory: %s\n",
        file.isDirectory());

      System.out.printf("File is read-only: %s\n",
        !file.canWrite());
    }
  }
}
```

The output produced by Listing 7.57 looks like this:

```
File length: 82
Last modified: Sat Jul 24 18:34:56 MDT 2004
Path to file: C:\projects\Book\file.txt
File is hidden: false
File is directory: false
File is read-only: false
```

C#

Listing 7.58 ***Get File Attributes Example (C#)***

```
using System;
using System.IO;

public class GetFileAttributes
{
  public static void Main()
  {
    FileInfo file = new FileInfo("file.txt"); //create object

    if (file.Exists)
    {
      Console.WriteLine("File length: {0}", file.Length);

      Console.WriteLine("Last modified: {0}",
        file.LastWriteTime);

      Console.WriteLine("Path to file: {0}", file.FullName);

      Console.WriteLine("File is hidden: {0}",
        (file.Attributes & FileAttributes.Hidden) > 0);

      Console.WriteLine("File is directory: {0}",
        (file.Attributes & FileAttributes.Directory) > 0);

      Console.WriteLine("File is read-only: {0}",
        (file.Attributes & FileAttributes.ReadOnly) > 0);
    }
  }
}
```

The output from Listing 7.58 looks like this:

```
File length: 82
Last modified: 7/21/2004 9:13:56 PM
Path to file: C:\projects\Book\file.txt
ts\Book\bin\Debug\file.txt
File is hidden: False
File is directory: False
File is read-only: False
```