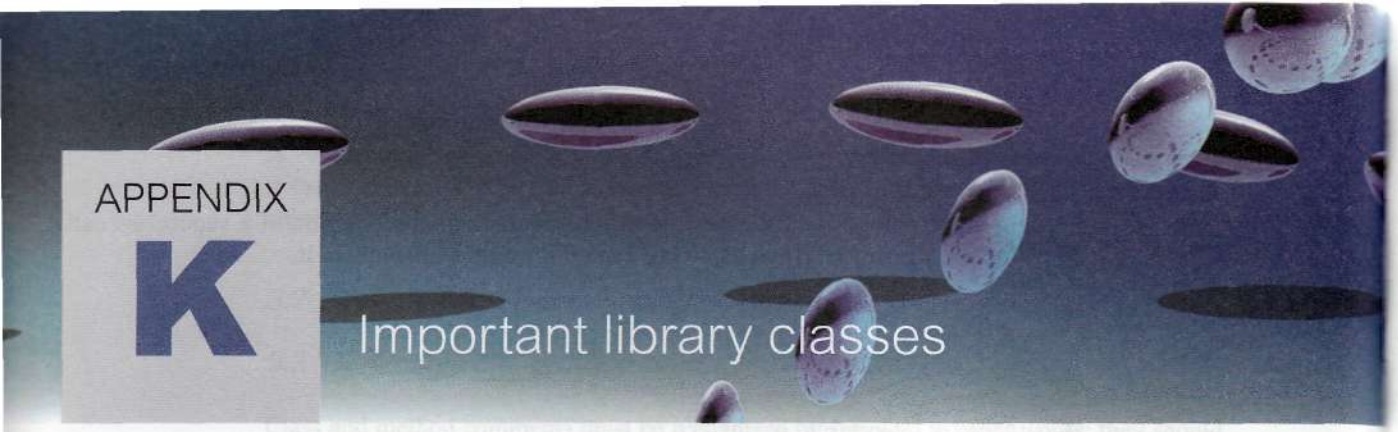

APPENDIX

K

Important library classes

The Java 2 Platform includes a rich set of libraries that support a wide variety of programming tasks.

In this appendix we briefly summarize details of some classes and interfaces from the most important packages of the Java 2 Platform API. A competent Java programmer should be familiar with most of these. This appendix is only a summary, and it should be read in conjunction with the full API documentation.

K.1 The **java.lang** package

Classes and interfaces in the **java.lang** package are fundamental to the Java language. As such, this package is automatically imported implicitly into any class definition.

package **java.lang** - Summary of the most important classes

class **Math**	**Math** is a class containing only static fields and methods. Values for the mathematical constants **e** and it are defined here, along with trigonometric functions, and others such as **abs, min, max,** and **sqrt.**
class **Object**	All classes have **Object** as a superclass at the root of their class hierarchy. From it all objects inherit default implementations for important methods such as **equals** and **toString.** Other significant methods defined by this class are **clone** and **hashCode.**
class **String**	Strings are an important feature of many applications, and they receive special treatment in Java. Key methods of the **String** class are **charAt, equals, indexOf, length, split,** and **substring.** Strings are immutable objects, so methods such as **trim** that appear to be mutators actually return a new **String** object representing the result of the operation.
class **StringBuffer**	The **StringBuffer** class offers an efficient alternative to **String** when it is required to build up a string from a number of components: e.g. via concatenation. Its key methods are **append, insert,** and **toString.**

K.2 The **java.util** package

The **java.util** package is a relatively incoherent collection of useful classes and interfaces.

package **java.util** - Summary of the most important classes and interfaces	
interface **Collection**	This interface provides the core set of methods for most of the collection-based classes defined in the **java.util** package, such as **ArrayList, HashSet,** and **LinkedList.** It defines signatures for the **add, clear, iterator, remove,** and **size** methods.
interface **Iterator**	**Iterator** defines a simple and consistent interface for iterating over the contents of a collection. Its three methods are **hasNext, next,** and **remove.**
interface **List**	**List** is an extension of the **Collection** interface, and provides a means to impose a sequence on the selection. As such, many of its methods take an index parameter: for instance, **add, get,** remove, and **set.** Classes such as **ArrayList** and **LinkedList** implement **List.**
interface **Map**	The **Map** interface offers an alternative to list-based collections by supporting the idea of associating each object in a collection with a *key* value. Objects are added and retrieved via its **put** and **get** methods. Note that a **Map** does not return an **Iterator,** but its **keyset** method returns a **Set** of the keys, and its **values** method returns a **Collection** of the objects in the map.
interface **Set**	**Set** extends the **Collection** interface with the intention of mandating that a collection contains no duplicate elements. It is worth pointing out that, because it is an interface, **Set** has no actual implication to enforce this restriction. This means that **Set** is actually provided as a marker interface to enable collection implementers to indicate that their classes fulfill this particular restriction.
class **ArrayList**	An implementation of the **List** interface that uses an array to provide efficient direct access via integer indices to the objects it stores. If objects are added or removed from anywhere other than the last position in the list, then following items have to be moved to make space or close the gap. Key methods are **add, get, iterator, remove,** and **size.**
class **Collections**	**Collections** is a collecting point for static methods that are used to manipulate collections. Key methods are **binarySearch, fill,** and **sort.**
class **HashMap**	**HashMap** is an implementation of the **Map** interface. Key methods are **get, put, remove,** and **size.** Iteration over a **HashMap** is usually a two-stage process: obtain the set of keys via its **keySet** method, and then iterate over the keys.
class **HashSet**	**HashSet** is a hash-based implementation of the **Set** interface. It is closer in usage to a **Collection** than to a **HashMap.** Key methods are **add, remove,** and **size.**
class **LinkedList**	**LinkedList** is an implementation of the **List** interface that uses an internal linked structure to store objects. Direct access to the ends of the list is efficient, but access to individual objects via an index is less efficient than with an **ArrayList.** On the other hand, adding objects or removing them from within the list requires no shifting of existing objects. Key methods are **add, getFirst, getLast, iterator, removeFirst, removeLast,** and **size.**

package **java.util** - Summary of the most important classes and interfaces	
class **Random**	The **Random** class supports generation of pseudo-random values - typically random numbers. The sequence of numbers generated is determined by a seed value, which may be passed to a constructor or set via a call to **setSeed.** Two **Random** objects starting from the same seed will return the same sequence of values to identical calls. Key methods are **next-Boolean, nextDouble, nextInt,** and **setSeed.**
class **StringTokenizer**	The **StringTokenizer** class provides an alternative to the **split** method of **String** for breaking up strings. It uses a set of delimiters to identify the boundaries between tokens. Key methods are **countTokens, hasMoreTokens,** and **next Token.**

K.3 The **java.io** package

The **java.io** package contains classes that support input and output. Many of the input/output classes are distinguished by whether they are stream-based - operating on binary data - or **readers** and **writers** - operating on characters.

package **java.io** - Summary of the most important classes and interfaces	
interface **Serializable**	The **Serializable** interface is an empty interface requiring no code to be written in an implementing class. Classes implement this interface in order to be able to participate in the serialization process. **Serializable** objects may be written and read as a whole to and from sources of output and input. This makes storage and retrieval of persistent data a relatively simple process in Java. See the **ObjectInputStream** and **ObjectOutputStream** classes for further information.
class **BufferedReader**	**BufferedReader** is a class that provides buffered character-based access to a source of input. Buffered input is often more efficient than unbuffered, particularly if the source of input is in the external file system. Because it buffers input, it is able to offer a **readLine** method that is not available in most other input classes. Key methods are **close, read,** and **readLine.**
class **BufferedWriter**	**Buf f eredWriter** is a class that provides buffered character-based output. Buffered output is often more efficient than unbuffered, particularly if the destination of the output is in the external file system. Key methods are **close, flush,** and **write.**
class **File**	The **File** class provides an object representation for files and folders (directories) in an external file system. Methods exist to indicate whether a file is readable and/or writeable, and whether it is a file or a folder. A **File** object can be created for a non-existent file, which may be a first step in creating a physical file on the file system. Key methods are **canRead, canWrite, createNewFile, createTempFile, getName, getParent, getPath, isDirectory, isFile, and listFiles.**

package **java.io** - Summary of the most important classes and interfaces	
class **FileReader**	The **FileReader** class is used to open an external file ready for reading its contents as characters. A **FileReader** object is often passed to the constructor of another reader class (such as a **BuffenedReader)** rather than being used directly. Key methods are **close** and **read.**
class **FileWriter**	The **FileWriter** class is used to open an external file ready for writing character-based data. Pairs of constructors determine whether an existing file will be appended or its existing contents discarded. A **FileWriter** object is often passed to the constructor of another writer class (such as a **BufferedWriter)** rather than being used directly. Key methods are **close, flush,** and **write.**
class **IOException**	**IOException** is a checked exception class that is at the root of the exception hierarchy of most input/output exceptions.

K.4 The **java.net** package

The **java.net** package contains classes and interfaces supporting networked applications. Most of these are outside the scope of this book.

package **java.net** - Summary of the most important classes	
class **URL**	The **URL** class represents a Uniform Resource Locator: in other words, it provides a way to describe the location of something on the Internet. In fact, it can also be used to describe the location of something on a local file system. We have included it here because classes from the **java. io** and **javax.swing** packages often use **URL** objects. Key methods are **getContent, getFile, getHost, getPath,** and **openStream.**

K.5 Other important packages

Other important packages are

java.awt
java.awt.event
javax.swing
javax.swing.event

These are used extensively when writing graphical user interfaces (GUIs), and they contain many useful classes that a GUI programmer should become familiar with.