**1**

**Packaging, Compiling, and interpreting Java Code**

**The Java Platform**

**Platform Independence**

When the Java language is compiled, it is targeted for execution on the Java virtual  
machine, or JVM, instead of a specific hardware architecture. The compiled Java code is called bytecode. The only requirement for the code to work on any computer is the presence of a compatible JVM.

JVMs share a common Java core, but platform independence is limited to compatible versions.

**Java’s Object Oriented Philosophy**

Java was conceived as an object-oriented language, in contrast to the C language, which is procedural. An object-oriented language organizes related data and code together—a process called encapsulation. A properly encapsulated object uses data protection and exposes only some of its data and methods.

Object-oriented design also encourages abstraction, the ability to generalize algorithms. Abstraction facilitates code reuse and flexibility. Inheritance and polymorphism are key concepts in creating reusable code.

**Robust and Secure**

Security and robustness were major design goals when Java was created.

With Java, the JVM periodically runs the garbage collector, which looks for any objects that have gone out of scope or that are no longer referenced, and it automatically deallocates their memory. This frees the developer from this manual, error-prone task and increases robustness by ensuring that memory is properly managed.

**Understand Packages**

**Package Design**

Packaging is encouraged by Java coding standards to decrease the likelihood of classes colliding in the same namespace. The package name plus the class names creates the fully qualified class name. Packaging your classes also promotes code reuse, maintainability, and the object-oriented principle of encapsulation and modularity.

**package and import Statements**

To place a source file into a package, you use the package statement at the beginning of that file. You may use zero or one package statements per source file. To import classes from other packages into your source file, you may use the import statement or you may precede each class name with its package name. The java.lang package that houses the core language classes is imported by default.

**The package Statement**

The package statement includes the package keyword, followed by the package path delimited with periods. Package statements have the following attributes:

* They are optional.
* They are limited to one per source file.
* Standard coding convention for package statements reverses the domain name of the organization or group creating the package. For example, the owners of the domain name ocajexam.com may use the following package name for a utilities package: com.ocajexam.utilities.
* Package names equate to directory structures. The package name com .ocajexam.utils would equate to the directory com/ocajexam/utils. If a class includes a package statement that does not map to the relative directory structure, the class will not be usable.
* The package names beginning with java.\* and javax.\* are reserved.
* Package names should be lowercase. Individual words within the package name should be separated by underscores.

**The import Statement**

An import statement enables you to include source code from other classes into a source file at compile time. The import statement includes the import keyword followed by the package path delimited with periods and ending with a class name or an asterisk. These import statements occur after the optional package statement and before the class definition. Each import statement can relate to one package only.

***For maintenance purposes, it is better that you import your classes explicitly. This will allow the programmer to determine quickly which external classes are used throughout the class. For example, rather than using import java.util.\*, use import java.util.Vector. In this real-world example, the coder would quickly see (with the latter approach) that the class imports  
only one class and it is a collection type. In this case, it is a legacy type and  
the determination to update the class with a newer collection type could be  
done quickly.***

**The static import Statement**

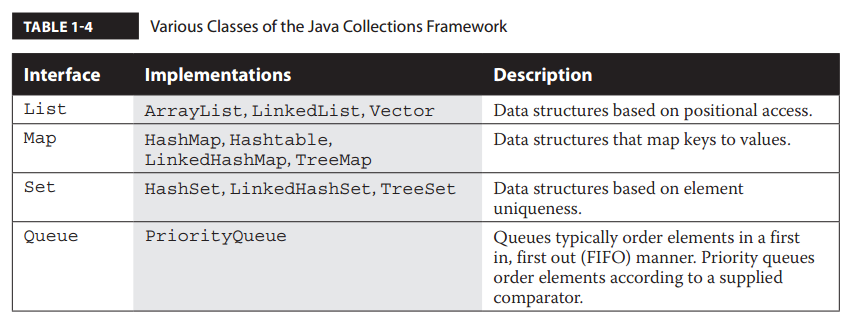
Simply put, static import statements allow you to import static members.

**Understand Package-Derived Classes**

The following sections address these APIs:

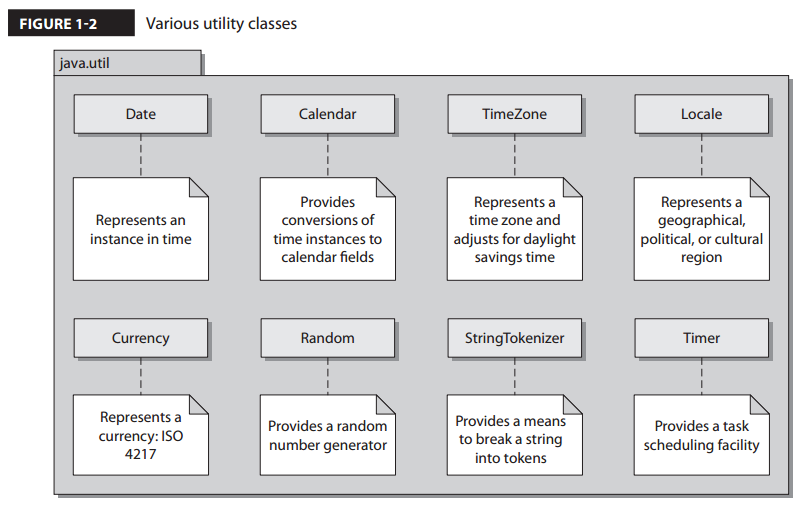
1. **Java Utilities API:**

* The Java Utilities API is contained in the package java.util.
* Categories of classes:
* Java Collections Framework;
* date and time facilities;
* internationalization;
* miscellaneous utility classes



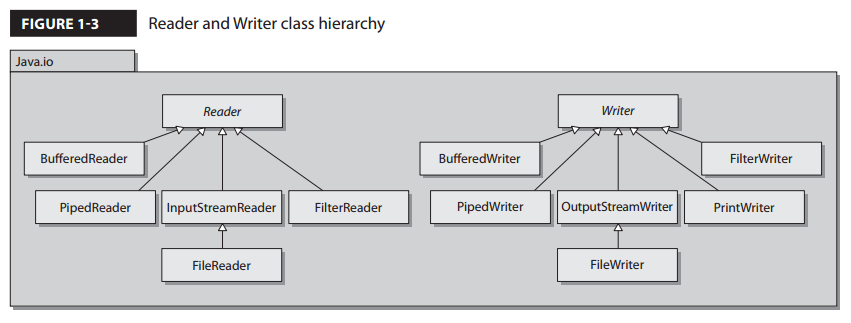
* Legacy date and time facilities are represented by the Date, Calendar, and TimeZone classes.
* Geographical regions are represented by the Locale class.
* The Currency class represents currencies per the ISO 4217 standard.
* A random-number generator is provided by the Random class.
* And StringTokenizer breaks strings into tokens.

***Many packages have related classes and interfaces with unique functionality, so they are included in their own subpackages. For example, regular expressions are stored in a subpackage of the Java utilities (java.util) package. The subpackage is named java.util.regex and houses the Matcher and Pattern classes. Where needed, consider creating subpackages for your own projects.***



1. **Java Basic Input/ Output API:**

* The Java Basic Input/Output API is contained in the package java.io.
* This API provides functionality for general system input and output in relation to data streams, serialization, and the file system.
* Data-stream classes include byte-stream subclasses of the InputStream and OutputStream classes.
* Data-stream classes also include character-stream subclasses of the Reader and Writer classes.
* Other important java.io classes and interfaces include File, FileDescriptor, FilenameFilter, and RandomAccessFile.
* The File class provides a representation of file and directory pathnames.
* The FileDescriptor class provides a means to function as a handle for opening files and sockets.
* The FilenameFilter interface, as its name implies, defines the functionality to filter filenames.
* The RandomAccessFile class allows for the reading and writing of files to specified locations.
* In JDK 7, the NIO.2 API was introduced in the package java.nio. This included the useful Paths interface, the Path class, and the Files class. The Files class has lines, list, walk, and find methods that work hand-in-hand with the Stream API.



1. **The Java Networking API**