**Simple**

 Looks familiar to existing programmers: related to C and C++:

 Omits many rarely used, poorly understood, confusing features of C++, like operator overloading, multiple inheritance, automatic coercions, etc.

 Contains no *goto* statement, but *break* and *continue*

 Has no header files and eliminated C preprocessor

 Eliminates much redundancy (e.g. no structs, unions, or functions)

 has no pointers

Added features to simplify:

 Garbage collection, so the programmer won't have to worry about storage management, which leads to fewer bugs.

 A rich predefined class library

**Object-Oriented**

* Java is an object-oriented language, which means that you focus on the *data* in your application and *methods* that manipulate that data, rather than thinking strictly in terms of procedures.
* In an object-oriented system, a *class* is a collection of data and methods that operate on that data. Taken together, the data and methods describe the state and behavior of an *object*. Classes are arranged in a hierarchy, so that a subclass can inherit behavior from its superclass.
* Java comes with an extensive set of classes, arranged in *packages*, that you can use in your programs.

**Distributed**

 It has a spring-like transparent RPC system

 Now uses mostly tcp-ip based protocols like ftp & http

Java supports various levels of network connectivity through classes in the java.net package (e.g. the URL class allows a Java application to open and access remote objects on the internet).

**Interpreted**

The Java compiler generates *byte-codes*, rather than native machine code. To actually run a Java program, you use the Java interpreter to execute the compiled byte-codes. Java byte-codes provide an architecture-neutral object file format. The code is designed to transport programs efficiently to multiple platforms.

 rapid turn-around development

 Software author is protected, since binary byte streams are downloaded and not the source code

**Robust**

Java has been designed for writing highly reliable or robust software:

 language restrictions (e.g. no pointer arithmetic and real arrays) to make it impossible for applications to smash memory (e.g overwriting memory and corrupting data)

 Java does **automatic garbage collection**, which prevents memory leaks

 extensive compile-time checking so bugs can be found early; this is repeated at runtime for flexibilty and to check consistency

**Secure**

* Security is an important concern, since Java is meant to be used in networked environments.
* Without some assurance of security, you certainly wouldn't want to download an applet from a random site on the net and let it run on your computer.
* Java's memory allocation model is one of its main defenses against malicious code (e.g can't cast integers to pointers, so can't forge access). Furthermore:

 access restrictions are enforced (public, private)

 byte codes are verified, which copes with the threat of a hostile compiler

**Architecture-Neutral**

 compiler generates bytecodes, which have nothing to do with a particular computer architecture

 easy to interpret on any machine

**Portable**

Java goes further than just being architecture-neutral:

 no "implementation dependent" notes in the spec (arithmetic and evaluation order)

 standard libraries hide system differences

 the Java environment itself is also portable: the portability boundary is POSIX compliant

**High-Performance**

Java is an interpreted language, so it will never be as fast as a compiled language as C or C++.

In fact, it is about 20 times as slow as C.

However, this speed is more than enough to run interactive, GUI and network-based applications, where the application is often idle, waiting for the user to do something, or waiting for data from the network.

**Multithreaded**

Java allows multiple concurrent threads of execution to be active at once.

This means that you could be listening to an audio clip while scrolling the page and in the background downloading an image.

Java contains sophisticated synchronization primitives (monitors and condition variables), that are integrated into the language to make them easy to use and robust. The *java.lang* package provides a *Thread* class that supports methods to start, run, and stop a thread, and check on its status.

**Dynamic**

Java was designed to adapt to an evolving environment:

 Even after binaries have been released, they can adapt to a changing environment

 Java loads in classes as they are needed, even from across the network

 It defers many decisions (like object layout) to runtime, which solves many of the version problems that C++ has

 Dynamic linking is the only kind there is