Appendix A - Glossary (of OO software terms)

Abstract Class	A *class* that does not supply an *implementation* for its entire *interface*, and so consequently, cannot be *instantiated*.
ActiveX	Microsoft's *component* standard. ActiveX controls are in many ways similar to *JavaBeans*.
Actor	A role that someone or something can play in relation to a software system. Actors are used in *use case* modeling.
Aggregation	Represents is-part-of or is-composed-of relationships between *objects*. Also known as *composition*.
Analysis	The process of determining the detailed specifications for a system. Software analysis involves *use case* modeling and other *requirement* specifications, the creation of a *domain object model*, and a preliminary *architecture*.
API	Application Programming Interface – a collection of public *classes* and *methods* that comprise a reusable chunk of functionality.
Applet	A mobile set of *Java classes* that may be *instantiated* and run inside of any Internet browser which supports the *Java Virtual Machine*.
Architecture	The most important and/or long lived aspects of a software system's *design*, including: the choice of programming language and outside vendors, major *interfaces*, subsystems, and common infrastructure; the architecture balances economic and technical forces.
C++	A hybrid *object-oriented* and procedural programming language, based on C. C++ has excellent performance, but is relatively complex for the programmer. See *Java* & *Smalltalk*.
Cardinality	A property of a relationship between two *classes* (or other *entities*) specifying how many *instances* of one class may be involved in the relationship with one instance of the other class.
Class	Defines and *encapsulates* the *methods* and variables of its *instances / objects*, including the *inheritance* of *interface(s)* and/or *implementation*, but not including the object's *state*.
Class Diagram	Often referred to (erroneously) as an *object model*, the class diagram is a *UML* model which shows the attributes and *responsibilities* of *classes*, as well as the static relationships between them, such as *implementation* and/or *interface inheritance* and *aggregation*.
Client	The user of a *server*'s services, as an *object*; or as a single-user machine in a network environment which presents the *GUI* for a 2-*tier* software application.
Collaboration	One *class* calling on another to help fulfill a *responsibility*.
Collection Class	A *class* with the *responsibility* to contain references to and provide access to other classes, such as: List, Stack, HashMap, Vector, etc.
COM	*Component Object* Model – See *DCOM*.

Copyright © 1996-2002 Dave Leberknight & Ron LeMaster
All rights reserved.

Component	An *object* or subsystem designed to be configurable and reusable, that has an *interface* and *encapsulates* its *implementation*. *JavaBeans* and *ActiveX* controls are two industry standard component models.
Composition	See *Aggregation*.
Concrete Class	As opposed to an *abstract class*, a concrete class may be *instantiated*.
Concurrency	Parallel processing, allowing multiple users to share system resources, or allowing a program to have multiple threads of execution. A software system with concurrency typically requires management of shared resources using locks and/or *transactions*.
Container Class	See *Collection Class*.
CORBA	Common *Object* Request Broker *Architecture*, an *OMG* standard for *Object Request Brokers* (ORBS), the Internet Inter-Orb Protocol (IIOP), and the *Interface* Definition Language (IDL). Note that Microsoft has a competing architecture called *DCOM*.
Coupling	The degree to which one *object* depends upon the details of another.
CRC Card	Shows the *Responsibilities* & *Collaborations* for a *Class*.
Database	A persistent storage device. O*bject* and relational databases have very different characteristics. Both generally support *transactions*.
DBMS	*DataBase* Management System.
DCOM	Distributed *Component Object* Model, Microsoft's *architecture* for distributed applications. DCOM is an extension of *COM*, and is also compatible with *OLE* and *ActiveX*. Note that *CORBA* is an industry standard architecture that competes with DCOM.
Delegation	A *design* technique where one *class* delegates some of its *responsibilities* to another class. Seen another way, delegation provides a means for a class to extend and reuse another class as an alternative to *inheritance*.
Design	The process of creating a plan for the construction of a software system. This involves the creation of *UML* models that specify how the system will work. A well-designed system will be as simple as possible while still being robust and flexible. See *Architecture*.
Design Pattern	A generalized solution for a recurring pattern of *object-oriented* software *design* problems. These patterns promote reusable, robust and flexible software.
Domain Object Model	An *analysis* model of the *classes* and *objects* within a given domain (problem area), including their *responsibilities* and *collaborations*.
Dynamic Model	Represents the non-static aspects of a software system. The *UML* specifies several dynamic modeling notations, including *state*, sequence and activity diagrams.
Encapsulation	Hiding the complex details of the *implementation* of a *class* behind a simple *interface*.
Enterprise Java Bean (EJB)	A standard *framework* for 100% pure *Java*, portable, server-side reusable *components*.

Copyright © 1996-2002 Dave Leberknight & Ron LeMaster
All rights reserved.

Entity	In a relational *database*, an entity is the same as a table.
Entity Relationship Diagram (ERD)	Also known as a *data model* or schema, the ERD is a model of *relational database* tables and the relationships between tables. ERDs do NOT support *inheritance*, *encapsulation* or *polymorphism*.
EXtreme (XP) Programming	An iterative software development methodology that advocates pair programming, *refactoring*, testing, planning, and simple design.
Framework	A set of reusable *classes* that provides rich functionality for a software system, but puts constraints on (or even dictates) the system's *design*. Frameworks often provide *abstract classes* that require specialized *concrete subclasses* for each new application.
Functional Decomposition	A method used by procedural programmers, which predates *object-orientation*. It involves organizing software around functions, sub-functions and the detailed sequencing of function invocations.
Garbage Collector	A memory management pattern, built into *Java* and *Smalltalk* but not *C++*, which eliminates memory leaks and simplifies programming.
GUI	Graphical User Interface (pronounced gooey).
IDL	*Interface* Definition Language, part of the *CORBA* standard.
IIOP	Internet Inter-*ORB* Protocol, part of the *CORBA* standard.
Implementation Inheritance	Refers to the kind of *inheritance* where a *subclass* inherits *implementation* (not just *interface*) from its *superclass*.
Inheritance	A relationship between two *classes* where one class is defined in terms of the other. The derived class inherits the parent class' *interface*, and possibly also some *implementation*.
Instance	An *object* is an instance of a *class*. Objects are created by the process of *instantiation*.
Instance Diagram	The *UML* model for diagramming *objects* (not *classes*). Also known as an *object model*.
Instantiation	The creation (construction) of a new *instance*.
Interface	Defines the set of *messages* to which an *object* can respond, *encapsulating* the implementation details. The sender of the message need not know the specific *class* of the responding object. See also: *Abstract class* & *polymorphism*
Introspection	The ability of a system to determine its own properties. For example, a *Java object* can dynamically discover its *class* and *methods*.
Java	The only *object-oriented* programming language whose programs can be executed inside of Internet browsers. Java is similar to *C++*, except that it is simpler, mobile, and portable (but it doesn't perform quite as well). See also: *Smalltalk*.
JavaBean	A *Java* class designed to be configurable and reusable, adhering to the JavaBean naming standards and design conventions.
JDBC	*Java* DataBase Connectivity. An *API* for connecting to a *DBMS* to issue *SQL* queries directly from Java.
JDOM	JDOM is a convenient *API* for manipulating *XML*, a future Java standard.

Copyright © 1996-2002 Dave Leberknight & Ron LeMaster
All rights reserved.

JMS	*Java* Messaging Service (*J2EE*). Java's *MOM*.
JNDI	*Java* Naming and Directory Interface (*J2EE*). A service for *clients* to find desired *services* across a distributed network, by name.
JSP	*Java* Server Pages (*J2EE*). An extension to *Servlets* for producing dynamic web content.
JTS	*Java Transaction* Service (*J2EE*). A transaction manager.
J2EE	*Java* 2 Enterprise Edition – Sun's application server *framework*, includes: *EJB, JMS, JNDI, JSP, JTS* & *RMI*.
Legacy System	An old software system that is still in use and must be considered when designing an extended or replacement system.
Life Cycle	The phases that a software project goes through, and in what order, typically including: definition, feasibility, planning, development, testing, deployment, and maintenance. Using the waterfall life cycle model, these phases are completed in the exact order given, whereas with an iterative life cycle model, the planning, development and testing phases may repeat periodically (see *eXtreme Programming*).
Message	*Objects* communicate with each other by sending messages, which invoke *methods* of the target object. Messages are also used in a wider context; they are used for communication in general.
MOM	*Message* Oriented *Middleware* – service to carry, route and deliver *messages* across networks (analogous to email).
Method	A function or procedure which is part of an *object*'s *interface*.
Method Signature	The detailed type information that is required to completely define a given *method*'s *interface*.
Middleware	Software which acts as the glue between the different *tiers* of *client - server* & multi-tier *architectures*. See *CORBA*, *EJB*, *J2EE*, and *MOM*.
Multiple Inheritance	Refers to a *class* that *inherits* from more than one *superclass* and/or *interface*. Multiple *implementation inheritance* is not allowed in *Java*.
Normalization	Elimination of redundancies in relational *data models*.
Object	An *instance* of a *class* with unique identity & values for the attribute(s) defined by the class. See *Object-Oriented*.
OLE	*Object* Linking and Embedding – from Microsoft – See *COM*.
OMG	The *Object* Management Group, which includes 800+ companies, is behind the *CORBA* standard.
Object Model	This term is often confused with *class diagram*; technically it is an *instance diagram*.
Object-Oriented	A paradigm for organizing a software system as a collection of *objects*. When an object-oriented *design* is done correctly, the resulting software is as simple as possible, while still being robust and flexible. *UML* models are often created in the process. Key concepts: *encapsulation, polymorphism, inheritance, delegation, interface*.

Copyright © 1996-2002 Dave Leberknight & Ron LeMaster
All rights reserved.

Object Request Broker (ORB)	A *middleware* product supporting the *CORBA* standard that facilitates distributed applications development by providing location, platform and programming language transparencies (the programmer does not have to know or care about the location, programming environment or operating system of network services).
Persistent Object	An *object* that can outlive the program that creates it, made possible by having its *state* stored in some kind of file or *database*.
Polymorphism	The ability to use any *object* which implements a given *interface*, where the specific *class* of the object need not be specified. This allows different concrete *implementations* of the interface to each respond to the same *message* in its own way.
Proxy	A Common *Design Pattern*. A proxy *object* receives *messages* on behalf of, and *delegates* to, the real object, at its sole discretion.
Refactoring	The process by which existing portions of a model or code are carefully changed, to become simpler, more general, more flexible, more reusable, easier to understand, easier to maintain, faster ...
Referential Integrity	Constraints in a *relational database*, applicable to all insert, update and delete operations, for all relationships. For example, a database might not allow the deletion of a particular record if some other record in some other table refers to it as a foreign key; such a deletion would leave that other table with an invalid foreign key reference. Alternatively the deletion could cascade, deleting not only the original record, but also all records which refer to it.
Relational Database	A kind of *database*, whose *design* has a mathematical basis in set theory. See *DBMS* & *Entity Relationship Diagram*.
Requirements	There are functional and non-functional requirements. *Use cases* are excellent for describing the functional requirements. Non-functional properties include: robustness, flexibility, reusability, reliability, performance, simplicity, documentation, quality, maintenance, etc...
Responsibility	The services provided by a *class*. A class can be responsible for performing actions, maintaining knowledge, enforcing constraints, etc.
RMI	Remote *Method* Invocation. A *Java* extension which provides similar functionality as an *ORB*, but only for pure Java applications, allowing a *client* to call a *method* on a remote *server object*.
Server	An *object* or system which provides services. See *client* and *tier*.
Servlet	Stateless *Java* code (*J2EE*) that runs on a *server*, analogous to an *applet*. Can be accessed from applets or from HTML web pages. See *Tomcat*.
Smalltalk	Perhaps the purest *object-oriented* programming language.
SQL	Structured Query Language is an ad-hoc query language for reading and manipulating data in a *relational database*.
State	The values of an *object*'s attributes, lumped together in such a way that no two states of an object have identical responses to all *messages*.
Subclass	*Class* that *inherits implementation* and/or *interface* from a *superclass*.

Copyright © 1996-2002 Dave Leberknight & Ron LeMaster
All rights reserved.

Superclass	*Class*, often *abstract*, with *subclasses*.
Tier	A machine which runs software as part of a distributed application in a network environment. The well-known *client* / *server architecture* is an example of a 2-tier system. Nowadays, N-tier architectures are becoming increasingly common due to their increased flexibility, scalability, ease of maintenance, and other benefits.
Tomcat	The servlet container used as the Official Reference Implementation for the Java *Servlet* and *Java Server Pages (JSP)* technologies.
Transaction	A single unit of work to be accomplished by a software system with the properties that it 1) is an atomic, indivisible unit of work, such that either all or none of it will execute; 2) leaves the system in a consistent, stable state, or else it will abort; 3) is not affected by other transactions that execute concurrently (implying a locking mechanism); 4) is durable and permanent (implying the need for a persistence mechanism, such as a *database*).
Unified Modeling Language (UML)	The UML is a robust and standard modeling language for designing *object-oriented* software systems. It provides *class*, *instance*, *state*, activity, interaction (sequence, collaboration), package, deployment and *use case* diagrams. Refer to: *object-oriented design*.
Use Case	A *UML* model that describes system processes, workflows, or scenarios, and is designed to ensure that the system behavior (functional *requirements*) is what the users (*actors*) require.
Virtual Machine	The *Java* platform, the JVM can be implemented on many other real platforms. This allows Java programs to not have to know or care about the real platform they are running on, achieving portability.
Virtual Method	*C++* term for a method whose *implementation* can be overridden by a *subclass*, with dynamic (run-time) binding. *Polymorphism* requires a virtual method mechanism.
XML	Standard *eXtensible Markup Language* – structured data as a text file.

Copyright © 1996-2002 Dave Leberknight & Ron LeMaster
All rights reserved.