Quick Revision Notes

I. Concepts and Common Architectures

1. Software Architecture and Design: The process of modeling a business such that all of its functional and non-functional service level requirements are adequately addressed.

   The Non-functional Quality of Service (QoS) requirements are:

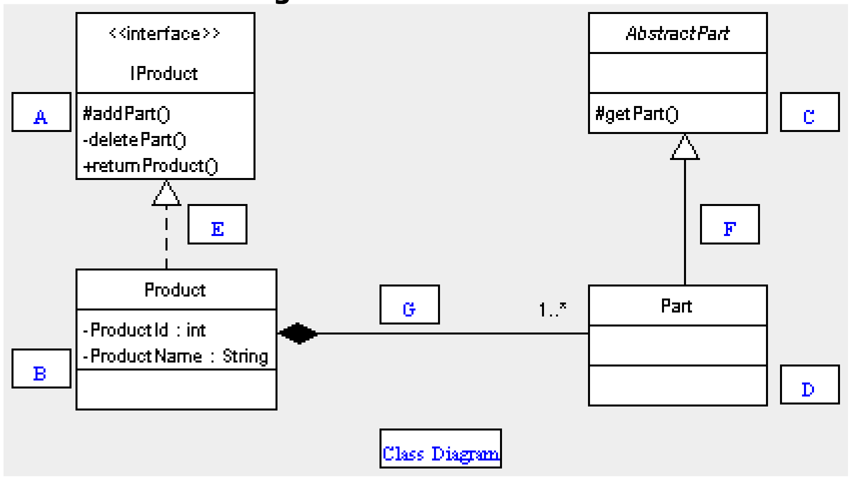
   1. **Performance** – A measure of the system in terms of response time or number of transactions per unit time. Load Distribution (e.g. DNS Round Robin) and Load Balancing are two techniques that aid in higher performance. Other development and deployment related tasks such as Application Tuning, Server Tuning, and Database Tuning also help the system perform better.
      DNS Round Robin: A process for distributing load in a system. If we have ten web servers that can service HTTP requests, the first request is directed to server 1, the second to server 2 and so on. When all ten servers have serviced one request each, the process starts all over again. Note that this is only a load distribution technique. DNS Round Robin does not balance the load.
      Reverse proxy load balancing: Reverse proxy load balancing is generally used when you have servers with different amounts of CPUs and Memory. You might have some really powerful servers just to be used for SSL sessions and others to handle static html. Using this will maximize the performance of your application
   2. **Scalability** – The ability of a system to perform and behave in a satisfactory manner with increases in load.
      Scalability can be achieved in two ways – Vertical (adding additional processors, memory or disks to existing hardware) and Horizontal (adding more machines to the system.)
      Vertical scalability is easier to implement than Horizontal scalability. Many J2EE vendors do however support Horizontal scaling as well.
   3. **Reliability** – The ability of a system to assure the integrity and consistency of the application and all its data as the load increases.
   4. **Availability** – The ability of a system to assure that all services and resources are always accessible. This can be achieved through fault tolerance (the ability to prevent system failures in the event of service(s) / component(s) failures, commonly implemented via redundancy) techniques such as Active and Passive Replication.
      Active Replication: This is comparable to 'hot backups.' All redundant machines / processes / servers / components are constantly updated. All replicas handle all requests. An interceptor mediates by sending only one response.
      Passive Replication: This is similar to 'warm backups.' The primary server handles all requests. It also synchronizes its state with the secondary replicas. Should a fail over occur, a secondary replica takes over.
   5. **Extensibility -** The ability to easily add new functionality to the existing system. This can be achieved by using best practices and well-defined architecture and design techniques.
   6. **Maintainability -** Ability to easily correct flaws in the existing system.
   7. **Security -** The ability to protect a system and all its components and services against potential attacks. Security attacks generally try to compromise confidentiality and integrity of the system. Sometimes they also take the form of 'Denial of Service' (DoS) attacks that bring down a system by flooding it with messages. Security can be addressed by the use of technologies (firewalls, DMZ, data encryption, Digital Certificates and so on) and methodologies (good security policies and procedures.)
   8. **Manageability -** The ability to monitor and perform preventive maintenance on a system.
2. **Common Architectures**

1. 1-Tier Architecture: Legacy systems such as those based on Mainframe technology. Dumb terminal clients directly connecting to the mainframe characterize these systems.
   1-Tier systems are not very scalable or extensible. They typically involve single points of failure. They are also difficult to maintain.
   1-Tier systems may be easy to manage because of the limited distribution of components and services.
2. 2-Tier Architecture: Client / Server based models where both clients and servers are intelligent machines and the workload is distributed among them.
   Typically clients maintain individual connections to the server and are responsible for data presentation and the processing of validations / business logic.
   Servers are generally responsible for managing data but may perform some business logic via database stored procedures and triggers.
   In the strictest sense, 2-Tier models may not always represent a single point of failure, though that is most often the case. For example, if servers in a server farm of databases hold 1 million accounts each, the failure of one of these servers need not constitute a system failure. However if the client uses a catalog server, which re-routes requests appropriately via a Database Link, the failure of the catalog server would constitute a total system failure.
   2-Tier systems are also not very extensible, maintainable, manageable, or secure. Any client changes involve deployment to all client workstations. Since client machines reside at various locations, they also constitute a management nightmare. Most clients also use initialization files that may contain sensitive information such as user id, password and so on.
   3. N-Tier Architecture: Distributed architecture involving multiple tiers, each performing a specialized function.
   J2EE based architectures are good examples. Typically made of:
   a. **Client Tier:** The tier with which the end user interacts. Clients can be 'thin clients' as in the case of Browser based applications or fat clients as in the case of client Java applications.
   b. **Web Tier:** Decouples the client tier from the Business tier. Java Servlets and JSPs reside in this tier. Servlets act as Controllers, they translate incoming requests, and dispatch them to components that can invoke the necessary business events in the Business Tier. JSP combine static templates with dynamic data to create dynamic output that the client tier uses for presentation to the user.
   c. **Business / Application Logic Tier:** Generally implemented using Enterprise Java Beans (EJB) that act as business process objects and business domain objects. EJB containers provide various services such as Object Distribution, Persistence, Transaction, Resource Management, Security and so on.
   d. **Enterprise Information System Integration Tier:** The EIS Integration tier interfaces between the Business (and sometimes Web tier) objects and Enterprise Information Systems. Example, Data Access Objects (DAO) decouple Enterprise beans (typically Session Beans or BMP Entity Beans) with Enterprise Data.
   e. **Enterprise Information System Tier:** This tier represents all the Enterprise data. This could be in many forms including Relational Databases, XML Databases, ERP Systems and so on.

   N-Tier Systems (if well architected and designed) can help achieve all non-functional service level requirements of the system. Note: Due to a highly distributed model, manageability may suffer a little in N-Tier systems, but since all systems are highly modular, the issue is generally fairly easy to address. It is also notable that Architects have to sometimes compromise a little on one service level requirement, to achieve the desired effect on another. For example Performance and Security show an inverse proportional relationship.

3. Unified Modeling Language (UML) – Important Diagrams

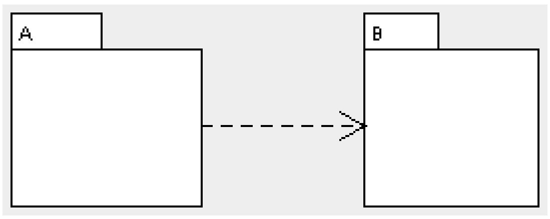
1. Class Diagram



A Class Diagram shows the classes and Interfaces in a system and their relationships.

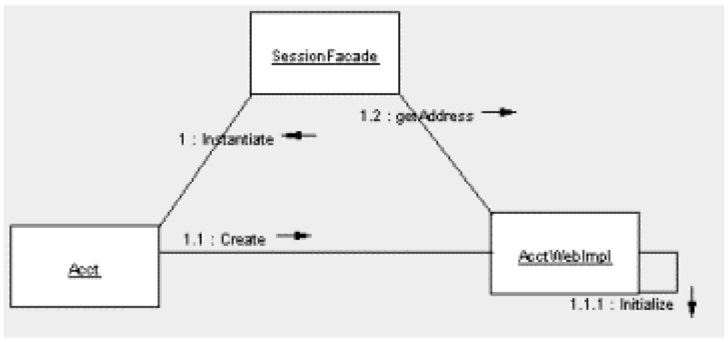
A: Interface, indicated by the stereotype <<interface>>. It has three operations:
#addPart () (**#**: Protected)
-deletePart () (**-**: Private)
+returnProduct () (**+**: Public)
B: Class Product implements interface A. It has two private attributes:
-ProductId: int
-ProductName: string
C: Abstract Class *Part* (note that the class name is in *Italics* indicating that it's an abstract class.) It has a protected method called getPart ()
D: Concrete class Part extends *AbstractPart*
E: Realization (B implements A)
F: Generalization D extends C
G: Composition Relationship between B and D. Note: A filled diamond represents Composition. A hollow diamond (not shown in the diagram) shows an aggregation relationship. No diamond would imply an association relationship. '1..*' Represents the multiplicity. It implies that each instance of B can have one or more instances of D associated with it.

2. Package Diagram



The diagram denotes a package diagram. A package diagram (Fowler 108) shows packages of classes and the dependencies among them. In the diagram Package A has a dependency to Package B (denoted by the dotted line with an arrow on one end.)
"A dependency exists between two elements if changes to the definition of one element may cause changes to the other…A dependency between two packages exists if any dependency exists between any two classes in the packages…With packages dependencies are non-transitive."

3. Collaboration Diagram

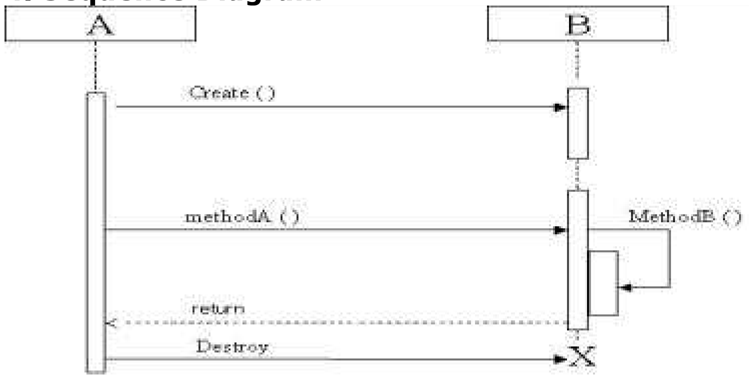


The above diagram is a collaboration diagram.

Interaction diagrams (Fowler 67) are models that describe how groups of objects collaborate in some behavior. There are two kinds of Interaction diagrams – Sequence diagrams and Collaboration diagrams.

(Cade 46) Interaction diagrams address the dynamic view of a system. A sequence diagram emphasizes on the time ordering of messages whereas a collaboration diagram emphasizes the structural organization of the objects that send or receive messages. Sequence diagrams and Collaboration diagrams are also isomorphic (you can take one and transform it to the other.)

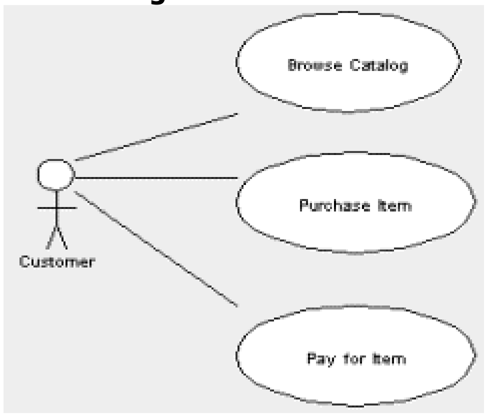
4. Sequence Diagram



The above diagram is a sequence diagram.

Sequence Diagrams (Cade 46) are "interaction diagrams that emphasize the time ordering of messages." Interaction diagrams address the dynamic view of a system and are frequency used by designers and developers.

5. Use Case Diagram

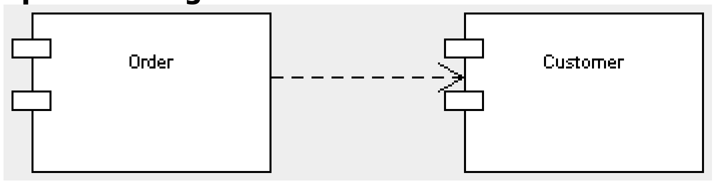


The above diagram is a Use Case Diagram.

Use Case diagrams (Cade 43) "show a set of use cases and actors and their relationships. Use Case diagrams show the static view of a system. These diagrams are especially important in organizing and modeling the behaviors of a system." Use case diagrams are frequently used by Business Analysts to capture business requirements of a system.

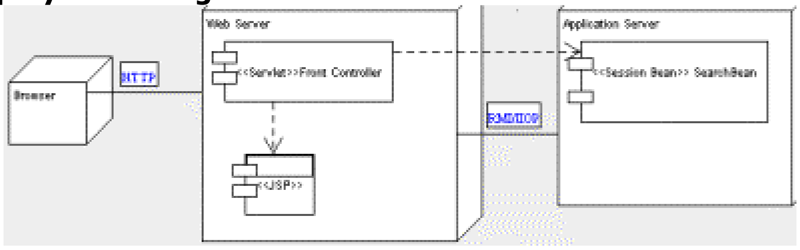
In the above diagram, Customer is the actor and 'Browse Catalog', 'Purchase Item' and 'Pay for Item' are the Use cases.

6. **Component Diagram**



(Fowler 141) "A component diagram shows the various components in a system and their dependencies. A component represents a physical module of code."
Component diagrams address the static view of a system.

7. **Deployment Diagram**



(Cade 50) "A deployment diagram shows the configuration of run-time processing nodes and the components that live on these nodes."
Deployment diagrams also represent a static view of the system.

II. Legacy Connectivity

Screen Scraper: A screen scraper emulates a mainframe terminal. Basically the screen scraper logs on to the mainframe like a normal user and sends requests to the mainframe and then reads the response. The problem with a screen scraper is that if you change any of the mainframes code there is always the possibility that the screen scraper will stop working.

Java Native Interface: JNI is used to allow Java to communicate with programs written in languages like C++. In effect you are wrapping the C++ code to make it available to Java. For example you will wrap a C++ method called debitAccount (int amount) with a similar Java method, the Java method will just call the C method. This means you can now make the method accessible via RMI

JAVA IDL: Java IDL adds CORBA (Common Object Request Broker Architecture) capability to the Java platform, providing standards-based interoperability and connectivity. Java IDL enables distributed Web-enabled Java applications to transparently invoke operations on remote network services using the industry standard IDL (Object Management Group Interface Definition Language) and IIOP (Internet Inter-ORB Protocol) defined by the Object Management Group. Runtime components include Java ORB for distributed computing using IIOP communication.

Go to: http://java.sun.com/j2se/1.3/docs/guide/idl/index.html for more information.

Java Connector Architecture:

The following is taken from:
http://java.sun.com/j2ee/connector/

"The J2EE Connector architecture provides a Java solution to the problem of connectivity between the many application servers and EISs already in existence. By using the J2EE Connector architecture, EIS vendors no longer need to customize their product for each application server. Application server vendors who conform to the J2EE Connector architecture do not need to add custom code whenever they want to add connectivity to a new EIS."

III. Enterprise Java Beans

Types of Enterprise Beans:

In EJB 1.1 there are two types of Enterprise Beans – **Session Beans** and **Entity Beans**. Session Beans manage workflow in a system and Entity Beans manage enterprise data.
Note that a new Enterprise Bean called the **Message Driven Bean** has been introduced in EJB 2.0.
Session Beans can be of two types – Stateful Session Beans and Stateless Session Beans. Stateful Session beans can maintain conversational state with the client and are therefore dedicated to the same client for the duration of their lives. Stateless Session Beans do not maintain conversational state. Hence they are lightweight objects that yield the highest performance. The container maintains a bean pool for Stateless Session Beans and swaps instances in and out of the pool to service client requests. This is an illustration of the Flyweight Design Pattern, where a few instances of the object are used to service a large number of client requests.

Classes and Interfaces required for Enterprise Beans

Home Interface: Defines the Bean's life cycle methods – creation, location and removal.
The Home Interface extends javax.ejb.EJBHome. Note that the create method is optional for Entity Beans. This is useful when you do not want to clients to be able to insert data into the database.

Remote Interface (Component Interface in EJB 2.0): Defines the Bean's business methods. The Remote Interface extends javax.ejb.EJBObject.

Bean Class: Implements the Bean's business methods. Note that the Bean class does not implement either the Home Interface or the Remote Interface.
The Bean class extends javax.ejb.EntityBean for Entity Beans and javax.ejb.SessionBean for Session Beans.
The Bean class provides implementations for callback methods and the methods defined in the Bean's Home and Remote Interfaces.

Primary Key: Is an object that uniquely identifies an Entity Bean according to its Bean Type, Home Interface and Container Context.
The Primary Key implements java.io.Serializable and it contains one or more public fields whose names and types match a subset of Container Managed fields in the Bean Class.
Features:

- The Primary Key must override the equals () and hashCode () methods.
- Can be compound or single field (Primitive Wrapper)
- Can remain undefined until deployment
- All fields to be declared public.
- When using Container Managed Persistence (CMP), a no argument constructor is required.

EJB and Transactions

The following is taken from:
http://java.sun.com/blueprints/guidelines/designing_enterprise_applications_2e/transactions/transactions7.html
"Transaction Attributes: A transaction attribute is a value associated with a method of an enterprise bean that uses container-managed transaction demarcation. A transaction attribute is defined for an enterprise bean method in the bean's deployment descriptor, usually by an application component provider or application assembler. The transaction attribute controls how the EJB container demarcates transactions of enterprise bean methods. In most cases, all methods of an enterprise bean will have the same transaction attribute. For optimization purposes, it is possible to have different attributes for different methods. For example, an enterprise bean may have methods that do not need to be transactional.
A transaction attribute must be specified for the methods in the component interface of a session bean and for the methods in the component and home interfaces of an entity bean.
Required - If the transaction attribute is Required, the container ensures that the enterprise bean's method will always be invoked with a JTA transaction. If the calling client is associated with a JTA transaction, the enterprise bean method will be invoked in the same transaction context. However, if a client is not associated with a transaction,

the container will automatically begin a new transaction and try to commit the transaction when the method completes.
RequiresNew - If the transaction attribute is RequiresNew, the container always creates a new transaction before invoking the enterprise bean method and commits the transaction when the method returns. If the calling client is associated with a transaction context, the container suspends the association of the transaction context with the current thread before starting the new transaction. When the method and the transaction complete, the container resumes the suspended transaction.
NotSupported - If the transaction attribute is NotSupported, the transactional context of the calling client is not propagated to the enterprise bean. If a client calls with a transaction context, the container suspends the client's transaction association before invoking the enterprise bean's method. After the method completes, the container resumes the suspended transaction association.
Supports - It the transaction attribute is Supports and the client is associated with a transaction context, the context is propagated to the enterprise bean method, similar to the way the container treats the Required case. If the client call is not associated with any transaction context, the container behaves similarly to the NotSupported case. The transaction context is not propagated to the enterprise bean method.
Mandatory - The transaction attribute Mandatory requires the container to invoke a bean's method in a client's transaction context. If the client is not associated with a transaction context when calling this method, the container throws javax.transaction.TransactionRequiredException if the client is a remote client or javax.ejb.TransactionRequiredLocalException if the client is a local client. If the calling client has a transaction context, the case is treated as Required by the container.
Never - The transaction attribute Never requires that the enterprise bean method explicitly not be called within a transaction context. If the client calls with a transaction context, the container throws java.rmi.RemoteException if the client is a remote client or javax.ejb.EJBException if the client is a local client. If the client is not associated with any transaction context, the container invokes the method without initiating a transaction."

IV. EJB Container Model

The EJB Container sits between an EJB Server and EJBs. An EJB Server can have one or more EJB Containers and each container can manage one or more components.
The EJB Container manages the EJBHome and EJBObject implementations. Via these objects, the container decorates Enterprise Bean classes and provides various services such as:

- Life cycle management
- Naming
- Object Distribution
- Persistence
- Security
- Transactions and
- Concurrency

The EJB Container also does resource management through Instance Pooling and swapping (in the case of Stateless Session Beans) and Passivation / Activation, in the case of Stateful Session Beans and Entity Beans.
Note: Even while the Bean is passivated, the client’s connection to the EJBObject is maintained. Before Passivation, all non-transient non-serializable fields must be set to NULL.

EJB 2.0
Persistence
The following is taken from:
http://developer.java.sun.com/developer/technicalArticles/ebeans/EJB20CMP/
Prior to the EJB 2.0 specification, a client stored and accessed persistent data via an entity bean's instance variables. With the introduction of the 2.0 specification, you can designate instance variables to be container-managed persistence fields (`CMP` fields) or

container-managed relationship fields (`CMR` fields). You define these `CMP` and `CMR` fields in the deployment descriptor.
You retrieve and set the values of these `CMP` and `CMR` fields using public get and set methods defined in an entity bean. Similar to the JavaBeans model, you do not access the instance variables directly, but instead use the entity bean's get and set methods to retrieve and set these instance variables. (An enterprise bean does not declare these instance variables.)
Furthermore, you use the deployment descriptor to specify the relationships between entity beans. These relationship specifications serve as the schema definition, so that when the bean is deployed, the bean relationships may be captured in a relational database. For example, a relationship between two beans specified in the deployment descriptor may appear as a foreign key relationship in a relational database.

Message-Driven Beans

The following is taken from:
http://java.sun.com/j2ee/tutorial/1_3-fcs/doc/EJBConcepts5.html
A *message-driven bean* is an enterprise bean that allows J2EE applications to process messages asynchronously. It acts as a JMS message listener, which is similar to an event listener except that it receives messages instead of events. The messages may be sent by any J2EE component--an application client, another enterprise bean, or a Web component--or by a JMS application or system that does not use J2EE technology.
Message-driven beans currently process only JMS messages, but in the future they may be used to process other kinds of messages.
The most visible difference between message-driven beans and session and entity beans is that clients do not access message-driven beans through interfaces. Unlike a session or entity bean, a message-driven bean has only a bean class.
Message-Driven beans don't implement any interfaces in EJB 2.0.

Local Clients

Prior to the Enterprise JavaBean 2.0 specification all Bean clients were seen as remote clients. This meant that if a Session Bean needed to talk to an Entity Bean inside the same Virtual Machine it would still need to make a remote call. This obviously had a direct impact on performance. In EJB 2.0 Enterprise Beans can now treat other beans in the same VM as local clients. Local Clients can access the beans through its local and local home interfaces.

Component Interface

The Component interface is the new name for the Remote interface as introduced in the EJB 2.0 specification.
Remote Interface (Component Interface in EJB 2.0): Defines the Bean's business methods. The Remote Interface extends javax.ejb.EJBObject.
Home Interface: Defines the Bean's life cycle methods – creation, location and removal. The Home Interface extends javax.ejb.EJBHome. Note that the create method is optional for Entity Beans. This is useful when you do not want to clients to be able to insert data into the database.

BMP/CMP

When you use Bean Managed Persistence you are writing all the SQL needed to persist the bean yourself. This means that the SQL would have been tailored to the data store you are using and the same SQL might not work with a different database vendor. You can cancel this out by using a Data Access Object. The Data Access Object pattern (DAO) is used to reduce the dependency between Enterprise Beans and the underlying database. This means that the data object manages the connection to the data source and if the data source changes you only need update this one object, the change doesn't affect the rest of your application.
Please note that this is referencing EJB1.1
This is a time dependent question as technologies have changed. When Sun wrote the EJB specification 1.1 Application Servers weren't very efficient at generating SQL used for persistence and it used to be recommended that you use BMP instead. However App Servers have improved and some would argue that CMP is now more efficient than BMP. In the SCEA guidebook by Cade it says BMP will out perform CMP, When the exam is

updated to EJB 2.0 the answer to this correct answer to this question would probably change.

BMT/CMT

The following is taken from:
http://java.sun.com/j2ee/tutorial/1_3-fcs/doc/Transaction3.html
In an enterprise bean with container-managed transactions, the EJB container sets the boundaries of the transactions. You can use container-managed transactions with any type of enterprise bean: session, entity, or message-driven. Container-managed transactions simplify development because the enterprise bean code does not explicitly mark the transaction's boundaries. The code does not include statements that begin and end the transaction.
http://java.sun.com/j2ee/tutorial/1_3-fcs/doc/Transaction4.html
In a bean-managed transaction, the code in the session or message-driven bean explicitly marks the boundaries of the transaction. An entity bean cannot have bean-managed transactions; it must use container-managed transactions instead. Although beans with container-managed transactions require less coding, they have one limitation: When a method is executing, it can be associated with either a single transaction or no transaction at all. If this limitation will make coding your bean difficult, you should consider using bean-managed transactions.

V. Protocols

Port numbers for some basic protocols:

The following is a list of Protocols and their default Port numbers:

- €HTTP – 80
- €HTTPS - 443
- €FTP - 21
- €JRMP - 1099
- €IIOP - 535
- €Telnet - 23

Important information about HTTP

HTTP (HyperText Transfer Protocol) is a transport mechanism for MIME (Multipurpose Internet Mail Extensions) documents. MIME documents often contain HTML (HyperText Markup Language) code for display in browser windows. HTTP consists of a request sent by the client to the server, followed by a response sent from the server back to the client. HTTP uses TCP/IP as the underlying transport and network protocols.
The following is taken from:
http://www.w3.org/Protocols/Activity.html
"Whenever a client accesses a document, an image, a sound bite etc. HTTP/1.0 creates a new TCP connection and as soon as it is done, it is immediately dismissed and never reused."
"HTTP/1.1 fixes this in two ways. First, it allows the client to reuse the same TCP connection (**persistent connections**) again and again when talking to the same server. Second, it makes sure that the courier carries as much information as possible (**pipelining**) so that it doesn't have to run back and forth as much. That is, not only does HTTP/1.1 use less TCP connections, it also makes sure that they are better used. The result is less traffic jam and faster delivery."
The following is taken from:
http://www.w3.org/Protocols/HTTP/HTTP2.html
"HTTP is a protocol with the lightness and speed necessary for a distributed collaborative hypermedia information system. It is a generic stateless object-oriented protocol, which may be used for many similar tasks such as name servers, and distributed object-oriented systems, by extending the commands, or 'methods', used."

Important Information about SSL

The following is taken from:
http://developer.netscape.com/docs/manuals/security/sslin/contents.htm
"The Transmission Control Protocol/Internet Protocol (TCP/IP) governs the transport and routing of data over the Internet. Other protocols, such as the HyperText Transport

Protocol (HTTP), Lightweight Directory Access Protocol (LDAP), or Internet Messaging Access Protocol (IMAP), run "on top of" TCP/IP in the sense that they all use TCP/IP to support typical application tasks such as displaying web pages or running email servers.
The SSL protocol runs above TCP/IP and below higher-level protocols such as HTTP or IMAP. It uses TCP/IP on behalf of the higher-level protocols, and in the process allows an SSL-enabled server to authenticate itself to an SSL-enabled client, allows the client to authenticate itself to the server, and allows both machines to establish an encrypted connection."

VI. Applicability of J2EE

Deciding between Java IDL RMI-JRMP and RMI-IIOP:

Java IDL - If you have been developing CORBA applications using IDL for some time, you will probably want to stay in this environment. Create the interfaces using IDL, and define the client and server applications using the Java programming language to take advantage of its "Write Once, Run Anywhere™" portability, its highly productive implementation environment, and its very robust platform.

RMI-JRMP - If all of your applications are written in the Java programming language, you will probably want to use Java RMI to enable communication between Java objects on different virtual machines and different physical machines. Using Java RMI without its IIOP option leverages its strengths of code portability, security, and garbage collection.

RMI-IIOP - If you are writing most of your new applications using the Java programming language, but need to maintain legacy applications written in other programming languages as well, you will probably want to use Java RMI with its IIOP compiler option.

When to use Enterprise Javabeans?

It is recommended that you use Enterprise Javabeans if Transactions are involved in the application. See below for more details.
The following is taken from:
http://java.sun.com/j2ee/tutorial/1_3-fcs/doc/EJBConcepts2.html
The application must be scalable. To accommodate a growing number of users, you may need to distribute an application's components across multiple machines. Not only can the enterprise beans of an application run on different machines, but also their location will remain transparent to the clients.
Transactions are required to ensure data integrity. Enterprise beans support transactions, the mechanisms that manage the concurrent access of shared objects.
The application will have a variety of clients. With just a few lines of code, remote clients can easily locate enterprise beans. These clients can be thin, various, and numerous.

VII. Design Patterns

Creational Patterns

Abstract Factory - (GOF 87): "Provide an interface for creating families of related or dependent objects without specifying their concrete classes."
Builder - (GOF 97): "Separate the construction of a complex object from its representation so that the same construction process can create different representations."
Factory Method – (GOF 107): "Define an interface for creating an object, but let subclasses decide, which class to instantiate. Factory Method lets a class defer instantiation to subclasses."
Prototype – (GOF 117): "Specify the kinds of objects to create using a prototypical instance, and create new objects by copying this prototype."
Singleton – (GOF 127): "Ensure a class has only one instance, and provide a global point of access to it."

Structural Patterns

Adapter – (GOF 139): "Convert the interface of a class into another interface clients expect. Adapter lets classes work together that couldn't otherwise because of incompatible interfaces."

Bridge – (GOF 151): "Decouple an abstraction from its implementation so that the two can vary independently."

Composite – (GOF 163): "Compose objects into tree structures to represent part-whole hierarchies. Composite lets clients treat individual objects and compositions of objects uniformly."

Decorator – (GOF 175): "Attach additional responsibilities to an object dynamically. Decorators provide a flexible alternative to subclassing for extending functionality." A request intended for a component is routed to the decorator instead. The decorator forwards the request to the component. It may perform pre or post processing tasks before or after forwarding the request. J2EE is filled with examples of the use of design patterns. The container provided implementations of EJBHome and EJBObject decorate bean classes by providing transactional and security functionalities.

Façade – (GOF 185): "Provide a unified interface to a set of interfaces in a subsystem. Façade defines a higher-level interface that makes the subsystem easier to use."

Flyweight – (GOF 195): "Use sharing to support large numbers of fine-grained objects efficiently."

Proxy – (GOF 207): "Provide a surrogate or placeholder for another object to control access to it."

Behavioral Patterns

Chain of Responsibility – (GOF 223): "Avoid coupling the sender of a request to its receiver by giving more than one object a chance to handle the request. Chain the receiving objects and pass the request along the chain until an object handles it."

Command – (GOF 233): "Encapsulate a request as an object, thereby letting you parameterize clients with different requests, queue or log requests, and support undoable operations"

Interpreter – (GOF 243): "Given a language, define a representation for its grammar along with an interpreter that uses the representation to interpret sentences in the language."

Iterator – (GOF 257): "Provide a way to access the elements of an aggregate object sequentially without exposing its underlying representation."

Mediator – (GOF 273): "Define an object that encapsulates how a set of objects interact. Mediator promotes loose coupling by keeping objects from referring to each other explicitly, and it lets you vary their interaction independently."

Memento – (GOF 283): "Without violating encapsulation, capture and externalize an object's internal state so that the object can be restored to this state later."

Observer – (GOF 293): "Define a one-to-many dependency between objects so that when one object changes state, all its dependents are notifies and updated automatically."

State – (GOF 305): "Allow an object to alter its behavior when its internal state changes. The object will appear to change its class."

Strategy – (GOF 315): "Define a family of algorithms, encapsulate each one, and make them interchangeable. Strategy lets the algorithm vary independently from clients that use it."

Template Method – (GOF 325): "Define the skeleton of an algorithm in an operation, deferring some steps to subclasses. Template Method lets subclasses redefine certain steps of an algorithm without changing the algorithm's structure."

Visitor – (GOF 331): "Represent an operation to be performed on the elements of an object structure. Visitor lets you define a new operation without changing the classes of the elements on which it operates."

J2EE Design Patterns

The following descriptions are taken from:
http://java.sun.com/blueprints/patterns/catalog.html

Business Delegate - *Reduce coupling between Web and Enterprise JavaBeansTM tiers.*

The Business Delegate pattern manages the complexity of distributed component lookup and exception handling, and may adapt the business component interface to a simpler interface for use by views.
http://java.sun.com/blueprints/patterns/BusinessDelegate.html
Composite Entity - *Model a network of related business entities.*
The ***Composite Entity*** design pattern offers a solution to modeling networks of interrelated business entities. The composite entity's interface is coarse-grained, and it manages interactions between fine-grained objects internally. This design pattern is especially useful for efficiently managing relationships to dependent objects.
http://java.sun.com/blueprints/patterns/CompositeEntity.html
Composite View - *Separately manage layout and content of multiple composed views.*
A ***Composite View*** is a view built using other reusable sub-views. A single change to a sub-view is automatically reflected in every composite view that uses it. Furthermore, the composite view manages the layout of its sub-views and can provide a template, making consistent look and feel feel easier to achieve and modify across the entire application.
http://java.sun.com/blueprints/patterns/CompositeView.html
Data Access Object (DAO) - *Abstract and encapsulate data access mechanisms.*
The DAO pattern allows data access mechanisms to change independently of the code that uses the data
http://java.sun.com/blueprints/patterns/DAO.html
Fast Lane Reader - *Improve read performance of tabular data.*
The ***Fast Lane Reader*** design pattern provides a more efficient way to access tabular, read-only data. A fast lane reader component directly accesses persistent data using JDBC™ components, instead of using entity beans. The result is improved performance and less coding, because the component represents data in a form that is closer to how the data are used.
http://java.sun.com/blueprints/patterns/FastLaneReader.html
Front Controller - *Centralize application request processing.*
The ***Front Controller*** pattern defines a single component that is responsible for processing application requests. A front controller centralizes functions such as view selection, security, and templating, and applies them consistently across all pages or views. Consequently, when the behavior of these functions need to change, only a small part of the application needs to be changed: the controller and its helper classes.
http://java.sun.com/blueprints/patterns/FrontController.html
Intercepting Filter - *Pre- and post-process application requests.*
The ***Intercepting Filter*** pattern wraps existing application resources with a filter that intercepts the reception of a request and the transmission of a response. An intercepting filter can pre-process or redirect application requests, and can post-process or replace the content of application responses. Intercepting filters can also be stacked one on top of the other to add a chain of separate, declaratively-deployable services to existing Web resources with no changes to source code.
http://java.sun.com/blueprints/patterns/InterceptingFilter.html
Model-View-Controller - *Decouple data representation, application behavior, and presentation.*
The Model-View-Controller design pattern decouples data access, business logic, and data presentation and user interaction.
http://java.sun.com/blueprints/patterns/MVC.html
Service Locator - *Simplify client access to enterprise business services.*
The ***Service Locator*** pattern centralizes distributed service object lookups, provides a centralized point of control, and may act as a cache that eliminates redundant lookups. It also encapsulates any vendor-specific features of the lookup process.
http://java.sun.com/blueprints/patterns/ServiceLocator.html
Session Façade - *Coordinate operations between multiple business objects in a workflow.*

The ***Session Facade*** pattern defines a higher-level business component that contains and centralizes complex interactions between lower-level business components. A Session Facade is implemented as a session enterprise bean. It provides clients with a single interface for the functionality of an application or application subset. It also decouples lower-level business components from one another, making designs more flexible and comprehensible.
http://java.sun.com/blueprints/patterns/SessionFacade.html

Transfer Object - *Transfer business data between tiers.*

A ***transfer object*** is a serializable class that groups related attributes, forming a composite value. This class is used as the return type of a remote business method. Clients receive instances of this class by calling coarse-grained business methods, and then locally access the fine-grained values within the transfer object. Fetching multiple values in one server roundtrip decreases network traffic and minimizes latency and server resource usage.
http://java.sun.com/blueprints/patterns/TransferObject.html

Value List Handler - *Efficiently iterate a virtual list.*

The Value List Handler design pattern provides a more efficient way to iterate a large, read-only list across tiers. A value list handler provides a client with an iterator for a virtual list that resides in another application tier. The iterator typically accesses a local ordered collection of Transfer Objects, representing a sub range of the large list. A Data Access Object usually manages access to the list data, and may provide caching.
http://java.sun.com/blueprints/patterns/ValueListHandler.html

View Helper - *Simplify access to model state and data access logic.*

A View Helper is a class that does data retrieval for the view. It adapts a data resource to a simple API usable by application views. The View Helper pattern decouples business and application classes from one another and allows them to vary at their own rates. Decoupling also promotes reuse, because each business or presentation component has fewer dependencies. So the view can focus on formatting and presentation logic, and let the View Helper handle the processing and retrieval of data.
http://java.sun.com/blueprints/patterns/ViewHelper.html

VIII. Messaging

Messaging: Enterprise Messaging allows two or more applications to exchange information in the form of messages.

Middleware Architectures:

a. Synchronous, tightly coupled communication between distributed components: This is the model of CORBA, RMI, EJB and so on. The programming model is called ***Remote Procedure Call (RPC).***
Note: EJB 2.0 has a new kind of bean called a Message Driven Bean, which acts as a message listener for processing asynchronous requests.

b.

Asynchronous, loosely coupled communication between components: This is the Message Oriented Middleware or *MOM* model. The programming model is called ***Messaging.***

1.

Publish Subscribe Messaging: Generally Pub/Sub is used when a one to many broadcast of messages is required. 'Producers' sends messages to many clients via virtual channels called 'Topics.' 'Consumers' receive messages by subscribing to topics. Consumers receive a copy of all messages in the topic they have subscribed to. The Publish Subscribe Architecture is generally a push-based model. Consumers may optionally establish 'durable' subscriptions that allow them to collect messages after periods of inactivity.

Point-to-Point Messaging: ***Point-to-point:*** The point to point messaging model allows both 'send and receive' and 'send and forget' messages, via virtual channels called 'queues.' The p2p model typically uses a 'pull' or 'polling' model. In this model, clients generally request messages from queues.

IX. Internationalization

1. **Internationalization:** Adapting a program for use in any country is called Internationalization.
2. **Localization:** The process of adapting a program for use in a particular country is referred to as Localization. During Localization the language of the text, message icons, colors used, dialogs, number formats, time representation and even sorting algorithms are subject to change.

List of items that may be subject to Internationalization:

- €Language for Messages
- €Formats – Numeric, Date and so on
- €Dictionary sort order
- €Currency symbol and position
- €Tax and other legal rules
- €Cultural preferences

Java support for Internationalization:

- Properties
- Locale
- Resource Bundle
- Unicode
- Java.text Package
- InputStreamReader
- OutputStreamWriter

Locale:

The following is taken from:
http://java.sun.com/j2se/1.3/docs/api/java/util/Locale.html

A `Locale` object represents a specific geographical, political, or cultural region. An operation that requires a `Locale` to perform its task is called*locale-sensitive* and uses the `Locale` to tailor information for the user. For example, displaying a number is a locale-sensitive operation--the number should be formatted according to the customs/conventions of the user's native country, region, or culture.

The `Properties` class represents a persistent set of properties. The `Properties` can be saved to a stream or loaded from a stream. Each key and its corresponding value in the property list is a string.

Every Java application has a single instance of class `Runtime` that allows the application to interface with the environment in which the application is running. The current runtime can be obtained from the `getRuntime()` method.

Properties:

The following is taken from:
http://java.sun.com/j2se/1.3/docs/api/java/util/Properties.html

The `Properties` class represents a persistent set of properties. The `Properties` can be saved to a stream or loaded from a stream. Each key and its corresponding value in the property list is a string.

A property list can contain another property list as its "defaults"; this second property list is searched if the property key is not found in the original property list.

Because `Properties` inherits from `Hashtable`, the `put` and `putAll` methods can be applied to a `Properties` object. Their use is strongly discouraged as they allow the caller to insert entries whose keys or values are not `Strings`. The `setProperty` method should be used instead. If the `store` or `save` method is called on a "compromised" `Properties` object that contains a non-`String` key or value, the call will fail.

java.text package:

The following is taken from:
http://java.sun.com/j2se/1.3/docs/api/java/text/package-summary.html

Provides classes and interfaces for handling text, dates, numbers, and messages in a manner independent of natural languages. This means your main application or applet can be written to be language-independent, and it can rely upon separate, dynamically linked localized resources. This allows the flexibility of adding localizations for new locations at any time.

ResourceBundle:

The following is taken from:
http://developer.java.sun.com/developer/technicalArticles/Intl/IntlIntro/

Resource Bundles - This internationalization feature of the JDK provides a mechanism for separating user interface (UI) elements and other locale-sensitive data from the application logic in a program. Separating locale-sensitive elements from other code allows easy translation. It allows you to create a single code base for an application even though you may provide 30 different language versions. Although you might be predisposed to think of text only, remember that any localizable element is a resource, including buttons, icons, and menus.

The JDK uses resource bundles to isolate localizable elements from the rest of the application. The resource bundle contains either the resource itself or a reference to it. With all resources separated into a bundle, the Java application simply loads the appropriate bundle for the active locale. If the user switches locales, the application just loads a different bundle.

Resource bundle names have two parts: a base name and a locale suffix. For example, suppose you create a resource bundle named `MyBundle`. Imagine that you have translated `MyBundle` for two different locales, ja_JP and fr_FR. The original `MyBundle` will be your default bundle; the one used when others cannot be found, or when no other locale-specific bundles exist. However, in addition to the default bundle, you'll create two more bundles. In the example these bundles would be named `MyBundle_ja_JP` and `MyBundle_fr_FR`. The `ResourceBundle.getBundle` method relies on this naming convention to search for the bundle used for the active locale.

The `java.util.ResourceBundle` class is abstract, which means you must use a subclass of `ResourceBundle`. The JDK provides two subclasses:`PropertyResourceBundle` and `ListResourceBundle`. If these don't meet your needs, you can create your own subclass of `ResourceBundle`."

X. Security

1. **Security Basics:** Identification refers to identifying who someone is. Authentication refers to verifying that a person / application is indeed who he / she / it is claiming to be. Authorization is verifying if that someone has access to a particular resource. Confidentiality means that the data has not being read by anyone other than the intended recipient. (i.e. the data is encrypted). Data Integrity means that the data hasn't been altered in transit (i.e. the data is sealed).
2. **Virtual Private Networks (VPN):** A network created between two other networks (these may not be located in the same place, geographically). VPN authenticates the user and uses Encryption for communication. Normally the VPN is a trusted network on top of an untrusted network (such as the Internet).
3. **Tunneling:** Tunneling is used to pass one protocol through a port that it does not, by default run on. For example if the only free port on the firewall was port 80 and you needed to pass JRMP through the firewall you would "tunnel" JRMP through the firewall. (JRMP by standard runs on port 1099). Basically JRMP would run on top of HTTP. However Tunneling should generally be avoided and should only be used as a last resort.
4. **Sandbox Model:** In JDK 1.0, all remote code was untrusted and all local code was trusted. This was the original sandbox model. The JDK 1.1 sandbox allowed applets using signed jar files to be trusted, if the necessary public key entries were available in the keystore. In JDK 1.2 and above, all code, local and remote, can be subjected to a policy file. This makes the Java 2 Platform security model totally flexible.

    The following is taken from:
    http://java.sun.com/docs/books/tutorial/security1.2/overview/index.html

"JDK 1.1 introduced the concept of a "signed applet," as illustrated in the next figure. A digitally signed applet is treated like local code, with full access to resources, if the public key used to verify the signature is trusted. Unsigned applets are still run in the sandbox. Signed applets are delivered, with their respective signatures, in signed JAR (Java Archive) files.

JDK 1.2 introduces a number of improvements over JDK 1.1. First, all code, regardless of whether it is local or remote, can now be subject to a security policy. The security policy defines the set of permissions available for code from various signers or locations and can be configured by a user or a system administrator. Each permission specifies a permitted access to a particular resource, such as read and write access to a specified file or directory or connect access to a given host and port.

The runtime system organizes code into individual domains, each of which encloses a set of classes whose instances are granted the same set of permissions. A domain can be configured to be equivalent to the sandbox, so applets can still be run in a restricted environment if the user or the administrator so chooses. Applications run unrestricted, as before, by default but can optionally be subject to a security policy."

5. **DMZ:** DMZ stands for Demilitarized zone. To set up a DMZ you need two firewalls and you create 3 separate regions. The different regions are the Internet, DMZ (in the middle) and the third is your network. You would have a server in the DMZ that is accessible to both the public and your network. The Internet may not access your network but they can access the server in the DMZ. Your network must access the Internet through the server in the DMZ.