Applicability of J2EE notes.

General.
The most commonly used J2EE technologies (in order of use) are :

- Java Servlets / JSPs
- JDBC
- JNDI
- EJB
- JMS
- Java XML Pack
- Java Mail
- Java IDL / RMI-IIOP
- JTA / JTS

Identify application aspects that are suited to implementation using J2EE technology (i.e. why J2EE ?)
J2EE draws on the main strengths of Java but viewed in the larger context of the enterprise.

An enterprise environment / application with the following qualities would be suitable for J2EE technologies:

- Heterogeneous environment
  Any medium to large sized organisation will have a variety of servers/platforms in use in the enterprise (e.g. Windows, Linux, Solaris, AS400). As JVMs are available for most platforms, Java provides Write Once Run Anywhere. Consequently use of J2EE enables code reuse, skills consolidation, etc. across the enterprise.

- Requirement for vendor neutrality and/or enterprise integration.
  A primary function of Java and J2EE is to provide generic interfaces for both developers (using APIs) and service providers (using SPIs). This enables an organisation to pick and choose products as they see fit without the fear of vendor lock-in. In addition, integration between enterprise components is quicker and easier.

- Productivity
  J2EE provides a feature-rich set of high quality APIs (and some default/reference implementations). Developers can use the APIs to reduce the time spent coding. E.g. sending email from an application used to be time-consuming as a developer had to write a cut-down email client or integrate classes from a third party; email support can now be added using Java Mail in a few minutes.

- Future-proofing
  Whenever there's a core demand in the development / service provider community, Sun works in partnership with other suppliers of enterprise components to provide Java and J2EE APIs and implementations that support the demand. By using J2EE, an organisation can be reasonably certain that any developments within the industry will : *1)* be available in J2EE and *2)* be available in a timely-fashion. E.g. support for web services was added to J2EE as soon as it became apparent that there was a demand in the community.

- Requirement for Enterprise class components.
  J2EE is high quality and provides fully integrated support for enterprise concepts – e.g. security, transactions, etc.

Identify application aspects that are suited to implementation using EJB (i.e. why use EJB ?)
EJB draws on the use of Java to provide a generic interface for both developers and service providers. EJB provides a standard execution and packaging environment which enables the development of components that are distributed, container managed and fault tolerant. In addition, EJB provides support for declarative programming – for transactions, security, etc.

An enterprise environment / application with the following qualities would be suitable for EJB technologies:

- Large scale deployment.
  The application has to service a large number of users whilst maintaining high levels of performance and availability. EJB facilitates such requirements through support for clustering (multiple instances; location transparency via JNDI and smart stubs), demand-based instance pooling, caching, etc.

- Transactional in nature.
  EJB provides ACID support for distributed transactions either automatically using CMT (Container Managed Transactions) or programmatically using BMT (Bean Managed Transactions with JTA/JTS). If the application is transactional in nature (e.g. transferring amount between bank accounts) it is suitable for EJB

- Requirement for fine-grained security.
  EJB has integrated support for declarative, fine-grained security

- Requirement for supporting a variety of clients.
  EJB splits the architecture in to n-tiers – typically client tier, web tier, EJB tier and EIS tier.
  The business logic is centralized in the EJB tier. Consequently, there is a greater level of reuse and application clients are thinner.

- Requirement for enhanced productivity / reliability.
  EJB follows the "iceberg" model – the majority of the low-level functionality (e.g. threading, pooling, CMT, CMP/R, etc.) has been developed by experts and is automatically bestowed upon the application developers. Consequently the developers can concentrate on writing the business logic. However, the productivity benefits may be mixed – EJBs take longer to write; an application may have to integrate with an existing database or a non-DB source so has to use BMP (Bean Managed Persistence), etc.

- Re-use.
  Components can be written that can be re-used across several projects.

- Component provider.
  If the business of the organisation is providing components for others (e.g. a SAP library), it makes sense to harness the strengths of EJB as a component model and packaging mechanism.

An enterprise environment / application with the following qualities wouldn't necessarily benefit from EJB:

- simple, small scale website – might be better to use Servlets and JSP with local Java classes
- no transactional requirements
- only basic security requirements
- an application that needs to use facilities "forbidden" by EJB (e.g. socket servers, JNI wrappers, etc.)
- an application that may needs to be scaled-down (e.g. a self-contained client version that won't have access to an EJB container)
- a development team who aren't comfortable with EJB development

Identify suitable J2EE technologies for the implementation of specified application aspects.

J2EE Technology	*Usage Pattern*
Servlets / JSPs	• need to process dynamic requests from HTTP clients A dynamic request is one where the response can be tailored on a per request basis – e.g. filter by IP, get customer details from DB and render as HTML • use Servlets for lower level functionality such as processing requests, routing, etc. • use JSPs for presentation aspects such as combining data and template for HTML output
JDBC	• use to access a vendor specific database in a generic way
JNDI	• use to access vendor specific directory or naming products in a generic way – e.g. used by JDBC for connections, access LDAP, etc.
EJB	• large scale deployment • need transaction / security (with the added bonus of declarative programming) • need to support a variety of clients • no existing DB (use CMP)
JMS	• use for systems that require asynchronous features • use to integrate with legacy systems • use to simulate threads in EJB
Java XML Pack	• use for B2B message exchange and/or HTTP application clients • use to produce web services
Java Mail	• use to send/receive email with plain or MIME content
Java IDL/RMI-IIOP	• use Java IDL to write Java-CORBA services (i.e. Java CORBA clients and/or Java CORBA object servers) • use RMI-IIOP to allow CORBA clients to interoperate with EJB objects or Java RMI object servers
JTA/JTS	• use when you need coarse grained transactions • use when EJB-CMT isn't an option