**Application Design Concepts and Principles**

* Identify the effects of an object-oriented approach to system design including the effect of encapsulation, inheritance, and use of interfaces.
* Identify how the Separation of Concerns principle applies to the component model of a Java EE application; including client, the web and business component containers,  and the integration and resource layers.
* Identify the correct interpretation of Separation of Concerns as it applies to the Java EE service layers, including component APIs, run-time containers, the operating system, and hardware resources.
* Identify non-functional and quality-of-service requirements that influence application design, including trade-offs in performance, availability, and serviceability.

**Integration and Messaging**

* Identify the APIs available for a Java EE technology-based system to communicating with external resources, including JPA, JDBC, RMI, Web Services, JMS, and JCA. Outline the benefits and drawbacks of each approach.
* Describe the technologies used to integrate business components with Web Services, including XML over HTTP, JSON, SOAP and REST.
* Identify and detail the technologies used to integrate business components with external resources, including JMS and JCA.
* Identify how a Service Oriented Architecture (SOA) facilitates system integration and best practices.

**Web Tier Technologies**

* Identify the benefits and drawbacks of using URL rewriting and cookies to manage HTTP session state.
* Identify appropriate uses for JSP and Servlet technology, and JavaServer Faces in a given Java EE application.
* Identify the benefits of using an EJB container with a web container instead of a web container alone.
* Identify the differences between client pull and server push architectures.
* Identify the benefits and drawbacks of using a browser to access asynchronous, lightweight processes on the server.

**Security**

* Identify elements of the security model in the Java SE environment for remote clients, including Web Start, applets and the role of the SecurityManager class.
* Select appropriate locations to implement Java EE security technologies or features in a UML component and deployment diagram.
* Classify security threats to an enterprise application select measures an architect can propose to mitigate them.
* Identify techniques associated with declarative and programmatic security, including the use of annotations, deployment descriptors, and JAAS technology.
* Identify the security technologies  that apply to an application’s code, messaging and transport layers

**Common Architectures**

* Identify the appropriate strategy for deploying client applications to desktop and mobile platforms, the principles for designing a user interface and the benefits of applying client-tier patterns.
* Identify best practices for exception handling, logging, and business tier patterns.
* Identify design patterns that address specific challenges in the web tier, including authentication, authorization, and scaling and clustering to meet demand.
* Identify Java EE technologies, including JMS, JCA and Web Services, and design patterns that address specific challenges in enterprise integration.
* Identify the challenges in integrating enterprise resources, the Java EE technologies that address them (including JPA and JDBC), and the communication protocols that support tier-to-tier communication (including RMI, IIOP, and CORBA).

**Business Tier Technologies**

* Identify the correct EJB technology to apply for a given scenario, including entity classes, session beans, message-driven beans, timers, interceptors, and POJOs.
* Identify benefits and drawbacks of different persistence technologies such as BMP, CMP, and JPA, including ease of development, performance, scalability, extensibility, and security.
* Identify the benefits and drawbacks of implementing Web Services in the EJB component container.
* Select the appropriate use of JPA and JPQL in a given scenario.

**Design Patterns**

* Demonstrate knowledge of Java EE design patterns including: Service Starter, Singleton, Bean Locator, Resource Binder, Dependency Injection, Payload Extractor, Context Holder, and Thread Tracker.
* Select an appropriate pattern for a given application challenge from the following: Facade, Strategy, Observer, Composite, and Abstract Factory.
* Identify a design pattern, using a description of its features, from the following:  Facade, Strategy, Observer, Composite, and Abstract Factory.
* Identify the use of the law of leaky abstractions or a specific anti-pattern in a given scenario.

**Assume the following:**

* **Missing package and import statements**: If sample code do not include package or import statements, and the question does not explicitly refer to these missing statements, then assume that all sample code is in the same package, or import statements exist to support them.
* **No file or directory path names for classes**: If a question does not state the file names or directory locations of classes, then assume one of the following, whichever will enable the code to compile and run:
  + All classes are in one file
  + Each class is contained in a separate file, and all files are in one directory
* **Unintended line breaks**: Sample code might have unintended line breaks. If you see a line of code that looks like it has wrapped, and this creates a situation where the wrapping is significant (for example, a quoted String literal has wrapped), assume that the wrapping is an extension of the same line, and the line does not contain a hard carriage return that would cause a compilation failure.
* **Code fragments**: A code fragment is a small section of source code that is presented without its context. Assume that all necessary supporting code exists and that the supporting environment fully supports the correct compilation and execution of the code shown and its omitted environment.
* **Descriptive comments**: Take descriptive comments, such as "setter and getters go here," at face value. Assume that correct code exists, compiles, and runs successfully to create the described effect.