J2EE Basics

- What is J2EE?
  - Stands for Java 2 Enterprise Edition
  - An umbrella for several things, including web apps
  - Provides a standardized way of developing webapps in Java
  - Defines a structure for code deployed to a server
  - Glassfish is the reference implementation

Request/Response cycle

- The client sends a request (described by javax.servlet.http.HttpServletRequest) containing a method (GET, POST, etc), headers, parameters (?firstName=Bob), a document (XML, JSON, etc)
- The server sends back a response (described by javax.servlet.http.HttpServletResponse), containing a code (EG, 200 OK), headers and a document.

Containers and Servers

- A Container provides only basic services, just barely runs a web app.
  - Provides JNDI
  - Sometimes provides database connection pooling
- A Server also provides extra goodies
  - Support for EJBs, a fancy web UI, reporting, monitoring, authentication via LDAP, etc

Basic Structure

- A webapp is a zip file using the suffix .war
- WEB-INF is the only special directory
  - WEB-INF/classes contains your compiled class files
  - WEB-INF/lib contains libraries and frameworks
  - WEB-INF/web.xml is the Deployment Descriptor
  - The client is not allowed to directly access anything in WEB-INF via a URL

Servlets

- A Servlet responds to a URL
  - Uses provided parameters to locate data
  - Typically doesn't render the response itself
  - Stores data in request attributes and forwards to a view
  - The view renders the data in the response
  - Request attributes are just a Map<String, Object> used to pass data from the servlet to the view
  - Generally used in MVC pattern – model is data in the database, view is JSP or JSF, controller is servlet

Filters

- A Filter intercepts handling of a URL before a Servlet gets a chance to respond to it
  - Typically doesn't render any response
  - Redirect to another URL
  - Respond with an HTTP error code
  - May modify request details
  - Useful to perform a pre action that has to occur on every access to one or more URLs
  - Examples:
    - Redirect old URLs to new ones
    - Validate and/or sanitize parameters
    - Detect security issues (SQL injection, XSS)

Listeners

- A listener receives events related to servlets startting/stopping, session created/modified/destroyed, request begin/end, etc.
- Unable to make modifications
- Example uses:
  - Log every servlet starting, every request
  - When a session is created, cache user details

Deployment Descriptor

- Must be stored in WEB-INF/web.xml
- Don't have to have one, but most projects do
- Schemas for various versions available from Oracle at: http://www.oracle.com/webfolder/technetwork/jsc/xml/ns/javaee/index.html
- Describes the following (not a complete list):
  - Configuration parameters
  - Servlets, and their mapping to URLs
  - Filters, and their mapping to URLS/servlets
  - Listeners
  - JNDI resources
  - Security
  - Welcome files

Configuration parameters

- Key/value pairs
- Can be configured in multiple places
  - Globally
  - Servlets
  - Filters
- Inherited
  - A key can be defined both globally and on servlets/filters. The servlets/filters override global values with the same key.

Servlet/Filter mappings

- Servlets are defined by a unique name and a fully qualified Java class that extends javax.servlet.http.HttpServlet
  - Multiple mappings can be defined for different url patterns
- Filters are defined by a unique name and a fully qualified Java class that implements javax.servlet.filter
  - Multiple mappings can be defined for different url patterns and/or servlet names
- Can be configured via Annotations instead

Listeners

- Listeners are just defined by a fully qualified Java class that implements one of the various listener interfaces such as javax.servlet.ServletRequestListener
- Can be configured via javax.servlet.annotation.WebListener annotation instead

Common Considerations

- Convert parameters/document received in a request into Java object(s)
- Validate Java object(s) to ensure client sent valid data (cannot rely on client validation)
- Read/write Java object(s) to a database
- Use a view to render web page responses
- Logging
- Authentication
- Authorization
- Localisation
- Multi-threading
- Connection pooling
- Transaction Management
- Multiple methods of access
- Reporting

Multi-Threading

- Most webapps don't need to do anything
- Thread pool created at start, grows and shrinks
- When a request is made, the container selects or waits for an unused thread in the pool
- When the request is completed, the thread is returned to the pool
- Handled entirely by the server
- Classes and methods serving the response use global fields and local variables
- Largely "just works"
- One common exception is background tasks

Connection Pooling

- Near 100% of webapps using a database use connection pooling
- Pool created on startup, grows and shrinks
- Webapp always connects as the same database user, a user unrelated to any real person
- When connections are closed by Java code, they aren't really closed – just returned to pool for use by another thread later

Multiple methods of access

- Not all accesses are an HTTP Request
  - Might use email, FTP, JMS queues, database rows
- Some accesses might come from other systems rather than people
- Not all HTTP requests are for web pages
  - XML, JSON, CSV sent and/or returned
  - APIs like REST/SOAP