XML data structure design and XML processing
- Use validation
- Use schema for validation when possible
- Exploit reuse of code within schema
- Avoid redundancy in XML instance documents
- Apply guidelines about time/date representations

XML programming in Java (with JAXB)
- Abstract factory pattern – understand it and use it correctly
- Understand JAXB annotations and binding principles and use them correctly
- Understand importance of developing robust Java code (i.e. avoid runtime exceptions and manage all exceptions that may arise)
- Management of generated code: don't modify generated code but use it
- Use XML processors (e.g. JAXB marshaller/unmarshaller) for parsing/serialization of XML documents rather than direct reading/writing of XML
- When reading an XML document, validate it against a schema so that the code can be made robust with a limited effort
- Achieve portability of code (absence of dependencies on locale, location)
- Understand the importance of build automation and use it correctly

Consuming web services in Java (with JAX-RS)
- Robustness: catch (and manage) possible runtime exceptions in your client
- Know the standard meaning of HTTP methods and status codes and use them accordingly
- Follow links instead of building them when made possible by the service

REST API design
- Understand/apply the principles, best practices, guidelines about interface design:
  - do not expose more than strictly necessary (information hiding)
  - choose adequate granularity level for resources (performance)
  - use resource concept correctly (no resource that represents action, no multiple URIs for same resource, etc)
  - minimize the number of interactions necessary to perform operations (performance)
  - limit the size of messages (let clients choose what to get, etc)
  - use HTTP methods consistently with their standard meaning and features
  - foresee all particular cases and decide how errors will be communicated to clients
  - use HTTP status codes consistently with their standard meaning
  - support HTTP headers as much as possible/convenient
  - use HTTP media-type headers to control types
  - use idempotent methods when possible and convenient
  - provide hyperlinks so as to enable HATEOAS
  - exploit HTTP features for efficiency (e.g. use GET whenever possible because it is more efficient (caching), etc)
  - use stateless interactions only (no server status about sessions apart from resources: use only resources to store state in service)
  - use HTTP self-documentation features
  - complement self-documentation features with user documentation (make sure a user has all the necessary information about the service)