Listing 1: Unique songs nested in unique singers with no attributes

```
<Songs>
 <Sgr><name>Eagles</name>
  <Song><genre>rock</genre><lg>en</lg>Hotel California</Song>
 </Sgr>
 <Sgr><name>H Belafonte</name>
  <Song><genre>folk</genre><lg>cpe</lg>Day O</Song>
  <Song><genre>calypso</genre><lg>en</lg>Jamaica Farewell</Song>
 </Sgr>
 <Sgr><name>J Prasad</name>
  <Song><lg>pa</lg><genre>folk</genre>Mera Dil Darda</Song>
 </Sgr>
</Songs>
```

Listing 2: Singers nested within songs grouped by genre

```
<Songs>
 <rock>
  <Song lg="en">Hotel California<Sgr name="Eagles"/>
  </Song>
 </rock>
 <folk>
  <Song lg="cpe">Day O<Sgr name="H␣Belafonte"/>
  </Song>
  <Song lg="pa">Mera Dil Darda<Sgr name="J␣Prasad"/>
  </Song>
 </folk>
 <calypso>
  <Song lg="en">Jamaica Farewell<Sgr name="H␣Belafonte"/>
  </Song>
 </calypso>
</Songs>
```

1. (a) (8 points) Of the following statements, identify all that hold about heterogeneity:
   - A. Heterogeneity often arises due to historical reasons
   - B. Once the components of an information system are correctly integrated, we don't have to worry about heterogeneity any more
   - C. We try to avoid heterogeneity because heterogeneous solutions are fragile
   - D. Heterogeneity arises because different components embody incompatible conceptual modeling assumptions

   Solution: A and D $\quad -2$ for each choice that is wrongly checked or unchecked

   (b) (8 points) Of the following statements, identify all that hold about servlet containers and EJB containers:
   - A. A servlet container shields servlets from one another
   - B. A servlet container guarantees transactional consistency, which is essential for electronic business applications
   - C. A servlet container helps programmers and system administrators do each other's jobs
   - D. Unlike an EJB container, a servlet container provides a single thread for each servlet class, thus preventing race conditions among different users

Solution: A −2 for each choice that is wrongly checked or unchecked

(c) (6 points) List three uses or benefits of enterprise models (in about 25 words total).

Solution:

- Document requirements and designs of information resources
- Capture organizational structure
- Capture design rationales thus facilitating change management
- Enable reusability
- Enable verification and validation of system implementations with respect to requirements

(d) (6 points) List three of the things that the XML Infoset specifies (in about 25 words total).

Solution:

- That there is a unique document root element
- That the ordering of attributes is irrelevant
- That element nodes may contain attributes and other elements
- What the main node types are (elements, attributes, comments, namespace declarations, processing instructions)

2. Mark the appropriate choices to complete the following XML Schema snippets for Listing 1. Ignore the missing components and ignore namespaces.

```
<xsd:element name="Songs" type="SongsT"/>

<xsd:complexType name="SongsT">
 <xsd:sequence>
    <xsd:element name="Sgr" type="SgrT" maxOccurs="unbounded"/>
 </xsd:sequence>
</xsd:complexType>

<!-- PART(a) deals with SgrT -->

<!-- PART(b) deals with SongT -->
```

(a) (12 points) Identify all snippets that correctly capture SgrT

```
A.
<xsd:complexType name="SgrT">
 <xsd:sequence>
    <xsd:element name="name" type="xsd:string"/>
    <xsd:element name="Song" type="SongT" maxOccurs="unbounded"/>
 </xsd:sequence>
</xsd:complexType>
```

```
B.
<xsd:complexType name="SgrT" mixed="true">
 <xsd:sequence>
    <xsd:element name="name" type="xsd:string"/>
    <xsd:element name="Song" type="SongT" maxOccurs="unbounded"/>
 </xsd:sequence>
</xsd:complexType>

C.
<xsd:complexType name="SgrT">
 <xsd:all>
    <xsd:element name="name" type="xsd:string"/>
    <xsd:element name="Song" type="SongT" maxOccurs="unbounded"/>
 </xsd:all>
</xsd:complexType>

D.
<xsd:complexType name="SgrT">
 <xsd:all>
    <xsd:element name="name" type="xsd:string"/>
    <xsd:element ref="Song" maxOccurs="unbounded"/>
 </xsd:all>
</xsd:complexType>

<xsd:element name="Song" type="SongT"/>
```

Solution: A and B.

(b) (12 points) Identify all snippets that correctly capture SongT

```
E.
<xsd:complexType name="SongT">
 <xsd:all>
  <xsd:element name="lg" type="lgT"/>
  <xsd:element name="genre" type="genreT"/>
 </xsd:all>
</xsd:complexType>

F.
<xsd:complexType name="SongT" mixed="true">
 <xsd:sequence>
  <xsd:element name="lg" type="lgT"/>
  <xsd:element name="genre" type="genreT"/>
 </xsd:sequence>
</xsd:complexType>

G.
<xsd:complexType name="SongT">
 <xsd:sequence>
  <xsd:element name="lg" type="lgT"/>
  <xsd:element name="genre" type="genreT"/>
 </xsd:sequence>
</xsd:complexType>
```

```
H.
<xsd:complexType name="SongT" mixed="true">
 <xsd:all>
  <xsd:element name="lg" type="lgT"/>
  <xsd:element name="genre" type="genreT"/>
 </xsd:all>
</xsd:complexType>
```

Solution: H.

3. Mark the appropriate choices to complete the following XQuery listing to transform Listing 2 into a document like Listing 1 but placing genre before lg throughout.

```
xquery version "1.0";
declare boundary-space strip;

declare function local:createGenreElement($root)
{
 element { name($root) } {
  let $genre := $root/*
  let $songs := $genre/Song
  let $singers := $songs/Sgr
  for $singername in distinct-values($singers/@name)
  return
   element { "Sgr" } {
---------------(: PART (a) :)---------------

     {
     $singername
     },
      for $song in $songs
---------------(: PART (b) :)---------------

      return
       element { "Song" } {

---------------(: PART (c) :)---------------

---------------(: PART (d) :)---------------

         $song/text()
    }
   }
 }
};

local:createGenreElement(doc("genre-song-singer.xml")/Songs)
```

(a) (12 points) Identify all correct snippets for the element constructor of PART (a)

A. `element { name($singers[1]/@name) }`

```
B.  element { name($singers/@name) }

C.  element { distinct-values(name($singers[1]/@name)) }

D.  element { distinct-values(name($singers/@name)) }
```

Solution:

```
element { name($singers[1]/@name) }
AND
element { distinct-values(name($singers[1]/@name)) }
```

(b) (12 points) Identify all correct snippets for the where clause of PART (b)

```
E.  where every $x in $song/Sgr/@name satisfies $x = $singername

F.  where $song/child::node()[last()]/@name = $singername

G.  where $song/descendant::node()[last()]/@name = $singername

H.  where $song/descendant::node()/@name = $singername
```

Solution:

```
where every $x in $song/Sgr/@name satisfies $x = $singername
AND
where $song/descendant::node()/@name = $singername
```

(c) (12 points) Identify all correct snippets for the element constructor of PART (c)

```
I.  element { name($song/..) } { $genre  },

J.  element { name($song/..) } {   },

K.  element { "genre" } { name($song/..) },

L.  element { "genre" } { $song/.. },
```

Solution:

```
element { "genre" } { name($song/..) },
```

(d) (12 points) Identify all correct snippets for the element constructor of PART (d)

```
M.  element { "lg" } { $song/@lg },

N.  element { "lg" } { "{$song/@lg}" },

O.  element { "lg" } { "$song/@lg" },

P.  element { "lg" } { text {$song/@lg }},
```

Solution:

```
element { "lg" } { text {$song/@lg }},
```