1 XML : eXtensive Markup Language

1.1 hoofding – body – epiloog

1.2 basisconcepten

- elementen (met tekst of andere elementen)
- attributen
- namen
- entiteiten
  → ter vervanging van gereserveerde karakters, bv <
- cdata:
  → wordt beschouwd als ruwe tekst, niet geïnterpreteerd
  bv. <![CDATA[…]]>
- XML declaratie :
  <?xml version="1.0" encoding="…" standalone="…">
- Verwerkingsinstructies
  bv. <?target pseudoattributen ?>
- Commentaar
  bv. <!-- -->
- Goedgevormd : de regels van xml

1.3 Namespaces

URI: uniform resource identifier
<namespace:elementnaam>
prefix gebruiken : attribuut xmlns:prefix=…
standaard namespace : xmlns="…"
attributen horen niet automatisch ook bij de namespace

HFDST 2. DTD : document type definition

LEGT STRUCTUUR VAST:

- welke elementen en entiteiten
- waar de elementen voorkomen en in welke volgorde
- inhoud van de elementen
- welke attributen

HOE KOPPELEN?

- in xml-bestand zelf : <!DOCTYPE basiselement […]>
- op pc zelf : <!DOCTYPE basiselement SYSTEM “bestandsnaam”>
- op internet : <!DOCTYPE basiselement PUBLIC “naam” “url” >
- extern met uitbreidingen : <!DOCTYPE basiselement SYSTEM “bestandsnaam” […]>

ELEMENTEN VASTLEGGEN:

<!ELEMENT naam inhoud >

- <!ELEMENT naam (attr1, attr2*, attr3?, attr4+) → in volgorde
- <!ELEMENT naam (attr1 | attr2) → volgorde onbelangrijk
- <!ELEMENT naam EMPTY> → leeg
- <!ELEMENT (#PCDATA | attr | attr2)*> → afgewisseld met tekst

ATTRIBUTEN VASTLEGGEN:

<!ATTLIST naamelement
attr1 type1 standaard1
…
>
Mogelijke types : CDATA, NMTOKEN, opsomming, ID, IDREF
Mogelijke standaarden: #REQUIRED, #FIXED + waarde, #IMPLIED, standaardwaarde

ENTITEITEN VASTLEGGEN:

stukken xml vastleggen die herhaaldelijk voorkomen
<!ENTITY naamentiteit “waarde_entiteit”>
<!ENTITY naamentiteit SYSTEM “pad”> → in apart bestand

HFDST 3 : XML Schema

Is een xml-taal.

HOE KOPPELEN AAN XML-DOCUMENT?

- niet tot naamruimte : attribuut : xsi:noNameSpaceSchemaLocation=”schema.xsd”
- wel tot naamruimte : attribuut : xsi:schemaLocation=”schema.xsd”
- parser een schema meegeven
- parser zelf laten zoeken op basis van namespace

BASISELEMENT :

<xsd:schema xmlns:xsd=”URL” >

KINDELEMENT:

<xsd:element name=”…” type=”…”>

ATTRIBUTEN:

<xsd:attribute name=”…” type=”…”>

SIMPLE TYPES:

- bestaande types
- beperkingen van bestaande types:

```
<xs:simpleType name=”…”>
    <xs:restriction base=”basistype”>
        <xs:restrictietype value=”…”>
```

Mogelijke restricties:
enumeration, length, minLength, maxLength, pattern,
minInclusive, maxExclusive, totalDigits, fractionDigits

- unie van simple types
- lijst

COMPLEX TYPES:

Bestaan uit kindelementen en attributen.
anoniem : komt slechts 1 maal voor, geen naam : <xs:element … > <xs:complexType><…>

```
<xs:complexType name=”…”>
    <xs:sequence> → als er meerdere elementen zijn en volgorde belangrijk is
    <xsd:choice> → volgorde onbelangrijk
    <xs:element name=”…” type=”…” minOccurs=”…” maxOccurs=”…”/>
    <xs:attribute name=”…” type=”…”>
```

COMPLEX TYPE ALS UITBREIDING OF BEPERKING VAN ANDER TYPE:

Element met tekst en enkel attributen:

- xsd:simpleContent – tag

- xsd: extension met base=”simple_type”
- attributen

  Elementen met kindelementen:
  een complex type bestaande uit een ander complex type, uitgebreid met nog andere elementen
- xsd:complexContent
- xsd:extension base=”complex_type”
- xsd:sequence
- xsd:element …

QUALIFIED VS UNQUALIFIED:
Globaal: als element kind is van basiselement => behoort tot naamruimte => qualified
Lokaal: element kind van ander element => behoort niet tot naamruimte => unqualified

SCHEMA OF DTD?
DTD: basisvalidatie, voor beschrijvende xml-documenten, geen ondersteuning voor namespaces
Schema: complexe types, beperkingen, types afleiden, namespaces

HFDST 4 : XSLT : Extensible StyleSheet Language Transformations

Is een XML-taal om XML te transformeren!

1. CSS : Cascading Style Sheets

Bepaalt opmaak van elementen in XML, is geen XML-taal!
Bevat lijst van elementen met hun opmaak:

```
naamElem1, naamElem2,… {
        eig1: waarde1
        eig2: waarde2
}
```

Hoe koppelen?
processinginstructie: <?xml-stylesheet type="text/css" href="bestand.css"?>

2. XSLT : Extensible Stylesheet Language Transformation

Regels vastleggen om XML te transformeren naar ander document. Mogelijke opties: sorteren, filteren, info toevoegen enz. Is een XML-taal!

Werking:
sjabloon = patroon en beschrijving uitvoer
Overloop XML-doc, als patroon klopt => uitvoer genereren

Structuur:
basiselement: <xsl:stylesheet>
sjablonen: <xsl:template>
output: <xsl:output>

Elementen:
<xsl:template match="…">
<xsl:value-of select="…">
<xsl:apply-templates select="…">
<xsl:sort select="…" datatype="…" order="…">
<xsl:foreach select="…">
<xsl:if test="…">
<xsl:text>

Choose

```
<xsl:choose>
        <xsl:when test="…"> … </xsl:when>
        …
        <xsl:otherwise> … </xsl:otherwise>
</xsl:choose>
```

Variabelen:

<xsl:variable name="…" select="…"/> → moet leeg element zijn

<xsl:variable name="…"> … waarde … </xsl:variable> → beide variabelen gebruiken door: $naam

Parameters:

<xsl:param name="…" select="…"/> of <xsl:param name="…"> … waarde … </xsl:param>

Parameters gebruiken:

<xsl:apply-templates …>

<xsl:with-param name="…" select="…">

→ zal de gegeven template oproepen met de gespecificeerde parameter, deze kan in de template opgeroepen worden met $naam.

HFDST 5: XPATH: XML Path Language

Is geen XML-taal!
XPath beschouwd XML als boomstructuur en maakt het mogelijk knopen te identificeren.

1. De knopen en paden

- één basisknoop (is NIET het basiselement!) : "/"
- elementen : "naamelem" of "/naamelem"
- attribuutknopen : bv /kindelement/@naam
- commentaar- of textknopen : "text()", "comment()"
- verwerkingsinstr : "processing-instruction()"
- jokers:
    - o * = alle elementknopen
    - o @* = alle attr
    - o node() = alle knopen
    - o ook mogelijk met namespaec: prefix:*, prefix:@*
- AND-bewerking : elem1|elem2
- speciale tekens:
    - o . : de contextknoop
    - o // : alle nakomelingen en contextknoop zelf
    - o .. : ouderelement
- voorwaarden:
    - o /elem[@attr=…]
    - o /elem[kindelem="…"]
- Lange padnamen:
    - o parent()
    - o self()
    - o descendant()
    - o descendant-or-self()
    - o ancestor()
    - o ancestor-or-self()
    - o namespace()
    - o following-sibling()
    - o preceding-sibling()
    - o following()
    - o preceding()

2. Datatypes

- knopen
- getallen met bewerkingen zoals in java
- strings
- logische waarden

3. Functies

- position()
- last()
- count()
- id()
- concat()
- contains()
- starts-with()
- string()
- string-length()
- true()
- false()
- not()
- boolean()
- ceiling()
- floor()
- number()
- round()

HDST 6 : XML en programmeermodellen

1. 2 soorten parsers

Interpreteert het document en deelt het op in stukken (elementen, attr, …) Geeft de inhoud door aan de applicatie en geeft fouten als document niet welgevormd is.
Functies: oa.: documenten valideren, bommstructuur opbouwen, events genereren, welgevormdheid controleren,…

1.1 event based parsers

Leest het document 1 keer in van begin tot einde en pauzeert enkel om externe bronnen op te halen. Tijdens het lezen wordt een stuk context bijgehouden.
Tijdens het lezen worden events gegenereerd, bv start element, einde element, inhoud element, …

Voordeel:

- moeten maar een beperkte hoeveelheid informatie bijhouden:
  inhoud van dtd of schema, stack voor elementnamen en namespaces bijhouden,
  dus niet de volledige inhoud: overlopen = verwijderen
  => performant!

Nadeel:

- applicatie is complexer: moet context bijhouden en informatie naar juiste plaats leiden, rekening houden met eventuele fouten => eventueel rollback
- kan niet steunen op informatie die nog moet komen => ~~Xpath~~

1.2 boomstructuur

Maakt gebruik van een API en een onderliggende bibliotheek die een objectmodel gebruiken om de xml-boom voor te stellen, volledige structuur wordt doorgegeven als het parsen gelukt is.

Voordeel:

- applicaties hebben direct de volledige boom => applicatie loopt door de boom, verandert boom, breidt boom uit. Rollbacks en parsen gebeurt door API.

Nadeel:

- veel geheugen nodig
- doorzoeken vergt ook veel instructies

2. SAX : simple API for XML

Is event-based. Bestaat uit verzameling interfaces.

XMLREADER:
Stelt de parser voor.
SAXFactory.newInstance() -> factory.getSAXParser() -> parser.getXMLReader()
(->reader.setContentHandler(…))
-> reader.parse(uri of inputsource)

- getSAXParser kan exception geven als configuratie niet goed is voor parser of een general SAXException
- parse() is synchroon

CONTENTHANDLER:
Interface die geimplementeerd moet worden en die verschillende methodes bevat die aangeroepen worden door de parser. Een overzicht:

- startDocument
- endDocument
- startElement
- endElement
- characters
- ignorableWhiteSpace
- processingInstruction
- skippedEntity
- startPrefixMapping
- endPrefixMapping
- setDocumentLocator
  → Locator geeft plaats (lijn- en kolomnr)

Nadelen van SAX: zelf opbouwen van datastructuur en rollback-mogelijkheden is moeilijk!

DEFAULTHANDLER:
Er bestaan nog 3 andere interfaces waar de Reader gebeurtenissen bij genereert, namelijk: ErrorHandler, DTDHandler, EntityResolver.
Alles samen zit in een soort adapterklasse : DefaultHandler
Als je een DefaultHandler gebruikt heb je geen Reader nodig, je werkt rechtstreeks uit de SAXParser:
->parser.parse(bestand, handler)

VALIDEREN:
valideren tov DTD => factory juist instellen: factory.setValidating(true);
valideren tov Schema => factory.setNameSpaceAware(true) en parser.setProperty(…,…)
de properties schemalanguage en schemasource instellen

3. DOM : Document Object Model

Maakt een boomstructuur, beschrijft hoe hiërarchische documenten kunnen opgeslaan worden in het geheugen. Bestaat uit verzameling interfaces. (Interface Description Language IDL). Belangrijk verschil met XPath: ook DocumentTypes, CDATA en entiteiten zijn knopen.

DE STRUCTUUR:

NODE
Is het basiselement, de rest is hiervan afgeleid. De meeste eigenschappen zijn readonly, behalve nodeValue en prefix.
NodeList: geordende verzameling
NamedNodeMap: niet-geordende verzamelingen

DOCUMENT
1 Document per XML-document, de rootknoop in feite.
Document heeft methodes om Nodes aan te maken, en toe te voegen aan het document. Kopiëren van andere Documenten gaat enkel via importNode(). Ook methodes om elementknopen te selecteren (getElementsByName)

ELEMENT
Bevat een tagname die de naam van Element voorstelt. Methodes om kindNodes toe te voegen (NIET om ze te maken), te overlopen, …

DOM IN JAVA
Parser aanmaken:
Factory-instance aanmaken, een DocumentBuilder-object creëren. Dit DocumentBuilder-object bevat de methode parse(bestand). Er kan een ErrorHandler geregistreerd worden bij de builder (builder.setErrorHandler(handler)).
Valideren: idem als SAX: factory.setValidating(true), factory.setNameSpaceAware(true)
MAAR: **factory**.setAttribute(…,…) om schemalanguage en schemasource in te stellen
Zoeken:
NodeList knopen=document.getElementsByTagName(…)

Document aanmaken of veranderen:
doc=builder.newDocument() → Document aanmaken
doc.createElement(), doc.createTextNode(), doc.appenChild(),… → toevoegen

4. XML-transformaties in Java

Ene XML-formaat omzetten naar andere. Populair:xslt

Een Transformer-object zet een bronobject om tot een resultaatobject.
Bron: gegeneerd door XMLReader, een Node, een inputstream.
Resultaat: ContentHandler, Node, OutputStream
Transformer-object: aangemaakt via factory, opgebouwd met tranformer)instructies

4 pakketten: java.xml.transform(=*) , *.dom , *.sax en *.stre

5. StAX = Streaming API for XML

Nadelen van DOM en SAX worden weggewerkt in nieuw model: pull parsing.
Principe: iterator-model: met behulp van de iterator kan de applicatie telkens het volgende stukje xml opvragen. De applicatie stuurt de parser aan.

6. XML Databinding

XML-document gebruiken, maar niet met expliciete XML-structuur werken => een extra laag die het werken met boomstructuren en gebeurtenissen verbergt. XML-documenten worden omgezet naar objecten en omgekeerd aan de hand van XML-schema's.

6.1 JAXP = Java API for XML Processing

Gebruikt DOM, SAX en XSLT, je kunt kiezen hoe je de gegevens wil zien => flexibel

6.2 JAXB = Java Architecture for XML Binding

Er wordt een verbinding gelegd tussen XML schema's en Java voorstellingen van deze gegevens. De XML-documenten worden omgezet naar Java-objecten en omgekeerd (zie figuur p91)
Het bindingsproces bestaat uit 5 stappen:

- klassen genereren op basis van het xml schema
- klassen compileren
- unmarshal: xml-documenten die voldoen aan het schema worden omgezet naar objecten
- valideren: gebeurt voor de omzetting (in beide richtingen)
- marshal: java-objecten omzetten naar xml-documenten

DEEL 2 : Toegang tot gegevensbanken in applicaties

HFDST1 : JDBC : Java Database Connectivity

1. Verschillende versies

JDBC 1.0 : toegang tot databanken
JDBC 2.0 : extra: aanpasbare resultsets, batch, programmatorisch aanpassen van db
JDBC 3.0 : niet enkel gegevensbanken, ook andere tabelstructuren

2. JDBC Drivers

Implementatie voor de interfaces uit de JDBC API zodat de algemene JDBC-opdrachten vertaald worden naar de productonafhankelijke opdrachten (van de gegevensbanken).

SOORTEN DRIVERS:

JDBC/ODBC-brug: ODBC = Open DB Connectivity, Microsoft's algemene API voor gegevensbanken. De brug vertaalt opdrachten van JDBC naar ODBC, en die worden dan door ODBC naar gegevensbankopdrachten vertaald. (inefficiënt en beperkt tot ODBC-functionaliteit)

Deels Java, deels productonafhankelijk: deze drivers vertalen methodes uit JDBC-API naar methodes uit productonafh. API. Dit is efficiënter en API maakt gebruik van volledige API van product. Er moet een stuk binaire code specifiek voor het db-systeem geïnstalleerd worden op de client-pc.

JDBC-Net : gebruiken tussenliggende db-server, clients kunnen met verschillende db's connecteren via deze server. De tussenliggende server gebruikt de productafhankelijke protocollen om te communiceren met de db's. Opdracht gaat via JDBC-driver naar tussenliggende server, die het verder afhandelt. (geschikt voor gebruik met verschillende db's)

Pure Java : volledig in Java en maken direct verbing met db (socket), er worden productafhankelijke methodes gebruikt. (meest efficiënt que performantie, ontwikkelingstijd en installatie)

DRIVERS LADEN:

constante string met volledige klassenaam van driver gebruiken in Class.forName(driverstring)
Deze methode laadt de driver in de virtuele machine en maakt een instantie van zichzelf en registreert zich bij de DriverManager. Deze klasse vormt de gemeenschappelijke toegangslaag tot de verschillende drivers.

3. Verbindingen met een DB

Connection is afhankelijk van driver => DriverManager.getConnection(JDBC-URL, user, passw)
De JDBC-URL bestaat uit:

- protocol (JDBC)
- subprotocol (gegevensbank-driver)

- subname (gebruikte gegevensbank: naam, server, poort)

Altijd try-finally om bij fouten de connectie te sluiten (close())!!!

4. SQL-Opdrachten

4.1 Statement-object:

aanmaken: conn.createStatement();
sluiten: stmt.close();
→ in finally, geeft gereserveerde geheugen in db en in vm (virtuele machine) vrij
DDL-opdrachten : stmt.executeUpdate(sql_string)
Zoek-opdrachten: stmt.executeQuery(sql_string)
→ geeft ResultSet terug
ResultSet: rs.next() , rs.getXxx(kolomindex) , rs.getXxx(kolomnaam) , close() of automatisch

4.2 Prepared Statements:

aanmaken: conn.prepareStatement(sql_string) → doorgestuurd naar db en gecompileerd
sluiten: prepstat.close() → in finally!
parameters: in sql_string ? zetten
prepstat.setXxx(paramindex, waarde)
uitvoeren: executeUpdate() of executeQuery()

Waarom prepared statements?

- Gebruik van strings met ‘
- Veiligheid: voorkomt dat ook andere queries meegegeven worden (=SQL-injectie)

4.3 Callable Statements:

Worden gebruikt om stored procedures op te roepen. De naam en nodige parameters moeten gekend zijn!

aanmaken: conn.prepareCall(call_opdracht)
→ call-opdracht tussen accolades: escape syntax, opdrachten die niet in de standaard SQL-syntax kunnen geformuleerd worden, de driver vertaald dit dan naar de correcte syntax van het db-systeem
parameters: definiëren in call-opdracht met ‘?’ als resultaat of argument van functie!
invoer: callstat.setXxx(paramindex, waarde)
uitvoer: instellen: callstat.registerOutParameter(paramindex, sqlType)
uitvoer: ophalen: callstat.getXxx(paramIndex)
invoer-uitvoer: beiden instellen!!!

5. Transacties

Om alles in 1 stuk uit te voeren of niet (commit vs rollback). Bij twee verschillende, parallelle transacties moeten er 2 verschillende connections zijn.

instellen: conn.setAutoCommit(false)
uitvoeren: commit() → meestal na alle opdrachten
ongedaan maken: rollback() → meestal in exceptions

6. Programmatorisch aanpassen van DB met Scrollable ResultSet

= Gegevens in de ResultSet aanpassen met blijvend effect in de DB, sinds JDBC 2.0!
Er wordt een scrollable resultset-object gebruikt die de hele tijd moet openblijven, dus ook de connection en statement moeten openblijven, wat redelijk wat geheugenruimte in DB vergt.

AANMAKEN DOOR OPTIES IN STATEMENT:

conn.createStatement(int resultSetType, int resultSetConcurrency)
met resultSetType als ResultSet.*:

- TYPE_FORWARD_ONLY : enkel vooruit met 1 stap
- TYPE_SCROLL_INSENSITIVE : in alle richtingen, aanpassingen in DB niet zichtbaar
- TYPE_SCROLL_SENSITIVE : in alle richtingen, aanpassingen in DB zichtbaar

met resultSetConcurrency als ResultSet.*:

- CONCUR_READ_ONLY
- CONCUR_UPDATABLE

METHODES VAN DE RESULTSET:

- Doorlopen van RS:
  - afterlast
  - previous
  - absolute(rijnr)
- Aanpassen van rijen in RS (en DB):
  - updateXxx(kolomnaam/-nr, waarde) → verandert in ResultSet maar niet in DB
  - updateRow() → alle waarden van de rij worden geupdate, ook in DB (cursor niet verplaatsen tussen updateXxx en UpdateRow)
  - cancelRowUpdate()
- Toevoegen van rijen in RS (en DB):
  - moveToInsertRow() → verplaatst cursor naar invoegrij
  - updateXxx(kolomnaam/-nr,waarde)
  - insertRow()
  - moveToCurrentRow() → terug naar rij voor de insert
- Rij verwijderen:
  - deleteRow()

GEBRUIK DE JUISTE ZOEKOPDRACHT!

criteria om aanpasbare RS te krijgen:

- zoekopdracht verwijst naar 1 tabel
- bevat geen joins of group by
- selecteert primaire sleutel
- gebruiker heeft lees- en schrijfpermissies

bij toevoegen ook:

- zoekopdracht selecteert alle rijen die niet leeg mogen zijn
- selecteert ook alle rijen zonder defaultwaarde

7. Batch

Een aantal opdrachten (geen zoek-opdrachten) als eenheid uitvoeren, dit maakt de uitvoering soms efficiënter!
In JDBC 1.0 zijn alle opdrachten een apart proces, zelf de opdrachten in een transactie.
In JDBC 2.0 kunnen de statement-objecten een lijst van opdrachten bijhouden. De lijst is in eerste instantie leeg en kan als één batch doorgestuurd worden.

METHODES VAN STATEMENT:

- addBatch(sql_string)
- clearBatch()
- executeBatch() → geeft rij integers terug met aantallen aangepaste rijen uit DB

BATCH UITVOEREN VAN GEWONE STATEMENTS:

stmt = conn.createStatement()
conn.setAutoCommit(False) → batch moet in transcactie doorgestuurd worden!!!!
stmt.addBatch(sql_string) → paar keer uitvoeren
stmt.executeBatch()
conn.commit() of conn.rollback()

BATCH UITVOEREN VAN PREPARED AND CALLABLE STATEMENTS:

eerst parameters zetten, nadat dit gebeurt is: prepstat.addBatch()
opnieuw parameters zetten, prepstat.addBatch()
enz.
Procedures met uitvoerparameters zijn niet toegelaten!

8. Metadata

Metadata-interfaces kunnen informatie over een bepaalde gegevensbank, driver, … opvragen.

DATABASEMETADATA:

informatie over DB: naam, maxconn, sql-syntax, stored procedures?, tabelinfo, batch ondersteund?, resultsets ondersteund?, transacties ondersteunt?
metadata = conn.getMetaData()

RESULTSETMETADATA:

informatie over RS-object: aantal kolommen, namen van kolommen, type van kolommen, …
rs.getMetaData()

ParameterMetaData:
informatie over een bepaalde parameter van een prepared statement: naam, type, …
enkel in JDBC 3.0

9. SQL3-gegevenstypes

BLOB : Binary Large Object
CLOB : Character Large Object
ARRAY : tabel van waarden
gestructureerd SQL-type: struct in Java (zelfgemaakt)
REF : verwijzing naar zelgemaakt gestruct SQL type
DISTINCT: nieuw type afgeleid van basistype

getXxx() en setXxx() gebruiken, maar deze geven verwijzingen…

BIJ CLOB, BLOB EN ARRAY
=> deze zijn verwijzingen naar het object in de databank en bevatten dus niet de eigenlijke waarde, om de waarde binnen te halen in de java-appl moet de data geconverteerd worden via methodes uit de respectievelijke interfaces.

GESTRUCTUREERDE SQL-TYPES:
create type naam (…) moet geconverteerd worden naar een struct in Java, en dan met getAttributes de attributen als Objects uit de struct halen! Ook mogelijk om gestruct sql-type om te zetten naar zelfgemaakte java-klassen, als er relaties gelegd werden.

DISTINCT TYPE:
create naam as type => aanspreken in java via basistype met getXxx en SetXxx

REFERENTIES:
corresponderen met java-type Ref: getRef(), setRef() updateRef()

10. Het gebruik van DataSource

In plaats van een connectie maken met een DriverManager, een DataSource-object gebruiken. Het voordeel is dat de naam van de driverklasse niet in de code moet opgenomen worden. Tweede voordeel is dat connection pooling en gedistribueerde transacties kunnen gebruikt worden. Derde voordeel is dat makkelijk van datasource, JDBC-Driver, gegevensbank, … kan gewisseld worden.

DE JNDI API
Interfaces en klassen om gebruik te maken van naming en directory services, onafhankelijk van het soort services.
Naming service:
Verbindt verstaanbare namen met computerobjecten (bv DNS, bestandssysteem). De naamservice bewaart de verbindingen tussen naam en object en biedt bovendien de mogelijkheid om een object op te zoeken en terug te vinden op basis van zijn naam! De naming service bepaalt de syntax waaraan de gebruikte namen moeten voldoen (naamconventie). Naming service bestaat uit een

aantal contexten die de zelfde naamconventie gebruiken. Deze context bevat een opzoekactie. Een naam kan verbonden zijn met een andere context met de zelfde naamconventie => subcontext.
Directory service: associeert attributen met de objecten in de naming service. Attributen bestaan uit een naam en mogelijke waarden. Dit maakt het mogelijk te zoeken op basis van attributen ipv op naam.

EEN DATASOURCE OBJECT AANMAKEN:
DataSource-object = factory voor connections (niet alleen met DB, ook bestanden bv.)
DataSource-object moet geregistreerd zijn bij een naming service zodat hij kan opgevraagd worden via de JNDP API.

⇨ de verschillende stappen:

- DataSource-klasse instellen: JDBC-driver nodig die data sources ondersteunt, klassenaam van data source van die driver opgeven zodat object kan worden aangemaakt door tool van application server
- Eigenschappen DataSource instellen: oa: de naam, server, poort van DB (via get/set in datasource)
- DataSource registreren: bij de naming service die gebruikt wordt door application server

⇨ DataSource-object is beschikbaar, opvragen:

- Context ctx = new InitialContext()
- DataSource ds = (DataSource) ctx.lookup("jdbc/naamDB")
- Connection conn = ds.getConnection("username,password")

3 IMPLEMENTATIES VAN DE INTERFACE DATASOURCE:

- basis
- met connection pooling
- met gedistribueerde transacties

CONNECTION POOLING:
Er zijn constant een aantal verbindingen met de DB beschikbaar, wanneer nodig wordt 1 gebruikt en nadien terug losgelaten. Dit gebeurt automatisch: bij GetConnection() wordt geen nieuwe verbinding gemaakt maar een referentie teruggegeven naar 1 uit de pool. Close() sluit de verbinding niet, maar stelt ze weer ter beschikking.
Om te werken moet ook een ConnectionPoolDataSource-object beschreven en geregistreerd worden.

GEDISTRIBUEERDE TRANSACTIES:
= transacties die data manipuleren in verschillende gegevensbronnen. De acties op die verschillende gegevensbronnen vormen een eenheid die ofwel volledig, ofwel niet wordt uitgevoerd.
De code van de acties op de DB's en de code die transacties maakt, beheert en uitvoert worden gescheiden. In de code zal dus geen spoor van transacties zijn.

2 HFDST 2 : C# in een noteNdop

Al genoeg bladverspilling in de cursus, dus lees het daar maar eens, misschien de belangrijkste verschillen met Java:

- string-type
- object-type
- boxing, unboxing
- enum
- readonly velden
- struct
- referentieparameters vs uitvoerparameters
- tabel van parameters in methode
- afleiden en interfaces : naamKlasse : basisklasse Iface1 Iface2 { … }
- Eigenschappen
- Indexers (gedragen als virtuele tabellen of hashtabellen)
- namespaces = packages
- virtual (=overridable in basisklasse)
- override (in afgeleide klasse)
- constructor van afgeleide klasse :

  mijnKlasse() : base() { … }
- compileren:
  - csc bestand.cs
  - cscs /out:Naam.exe bestand1.cs bestand2.cs bestand3.cs
  - cscs /target:library /out:bib.dll bestand1.cs bestand2
  - csc /reference:bib.dll program.cs

HFDST 3 : ADO.NET = ActiveX Data Objects

Data van verschillende bronnen (niet enkel DB's) op te halen en te manipuleren.
Losse koppeling => geen constante verbinding met gegevensbron, deel van de data lokaal opgeslagen

3.1 DataProvider

biedt de mogelijkheid om verbinding te maken met gegevensbron, opdrachten uit te voeren in de bron.

STANDAARD 2 DATA PROVIDERS:

- SQL Server.Net: via data-transfer-protocol van SQL Server, performant en efficiënt
- OLE DB Provider (Object Linking and Embedding for DataBases)
  iets minder efficient, maar uniforme toegang tot verschillende databronnen

BEVAT 4 SLEUTELOBJECTEN:

- Connection
- Command
- DataReader
- DataAdapter

3.1.1 Connection

conn = new SQLConnection(connstring)
of
conn = new SQLConnection(); conn.ConnectionString = connstring;
conn.Open();
conn.Close();

connectiestring bestaat uit naam/waarde-paren:
Provider (enkel OLE DB), Connect Timeout, Data Source/Server, Initial Catalog / Database, Integrated Security, Password, User ID

3.1.2 Command

Implementeert IDbCommand => SqlCommand of OleDbCommand – object

AANMAKEN:

SqlCommand comm = new SqlCommand (query, conn);
of
comm = conn.CreateCommand() en comm.CommandText = query;

METHODES:

comm.ExecuteReader() → heeft DataReader als resultaat
comm.ExecuteNonQuery() → om DDL-opdrachten of stored procedures uit te voeren
comm.CreateParameter) → maakt een Parameter aan (zie Parameters)
comm.Parameters.Add(parameter) → voegt parameter toe

EIGENSCHAPPEN:
comm.CommandText
comm.Parameters → van type IDataParameterCollection: heeft ook indexer met string

3.1.3 DataReader
Implementatie van IDataReader => SqlDataReader of OleDbDataReader
Met DataReader kun je de gegevensstroom van de databank éénmaal inlezen van begin tot eind. Het wordt verkregen als resultaat van de methode ExecuteReader.

AANMAKEN:
DataReader reader = Command.ExecuteReader();

METHODES:
reader.Read(); → resultaat is bool: nog rijen?
reader.GetXxx(int kolomnr); →alternatief: Indexer met int of string.
reader.Close();

3.1.4 Parameters

CONSTRUCTOR
new SqlParameter();
new SqlParameter("@naam", waarde);
new SqlParameter("@naam", type);

EIGENSCHAPPEN:
param.ParameterNamen
param.Value
param.SqlDbType
param.SourceColumn
param.SourceVersion

3.2 DataSet
= verzameling DataTable-objecten en hun onderliggende relaties, beperkingen (constraints)
Een DataSet wordt opgevuld met gegevens uit een gegevensbron maar heeft geen informatie over de onderliggende gegevensbron => alleenstaande component
Een DataSet kan gegevens inlezen uit databank, XML-bestand of –stream en de data die het bevat ook opnieuw uitschrijven in XML-formaat.
De communicatie gebeurt via de DataAdapter.

CONSTRUCTOR:
new DataSet();
new DataSet("naam");

EIGENSCHAPPEN:

- Tables
    - Columns (DataColumnCollection)
    - Rows (DataRowCollection)
    - PrimaryKey

METHODES:

datatabel = ds.Tables.Add("tabelnaam") → resultaat is DataTable
datakolom = datatabel.Columns.Add("kolomnaam",type) → resultaat is DataColumn
datatabel.PrimaryKey = new DataColumn [] {datakolom} → maakt constraint
datarij = datatabel.NewRow(); → maakt nieuwe rij (DataRow)
datarij[0]=waarde → stelt 0de kolom in
datatabel.Rows.Add(datarij); → voegt rij toe
datatabel.Rows.Find(primaire_sleutel) → zoekt op primaire sleutel
datarij.Delete() → markeert als te verwijderen
datarij.RejectChanges()

3.3 DataAdapter

Wordt gebruikt om DataSet op te vullen met gegevens uit databank en later ook om gegevens terug weg te schrijven.

CONSTRUCTOR:

new SqlDataAdapter()
new SqlDataAdapter(query, conn)

EIGENSCHAPPEN:

SelectCommand
InsertCommand
UpdateCommand
DeleteCommand
MissingSchemaAction

METHODES:

adapter.Fill(DataSet, "naamTabel"); → impliceert open verbinding en sluit nadien
adapter.MissingSchemaAction = MissingSchemaAction.AddWithKey
adapter.Update(DataSet, "naamTabel");

DataReader: performanter
DataSet: data manipuleren los van DB en pas later wijzigingen toebrengen

3.4 Transacties

CONSTRUCTIE EN METHODES:

SqlTransaction trans= conn.BeginTransaction()
command.TransAction = trans
opdrachten uitvoeren
trans.Commit() en trans.Rollback()