Java Message Service

Projekti 1C
Rinnakkaisohjelmointi
Syksy 2008, periodi 2
15.12.2008

Ismo Lehtonen
Ville Kesola
Ilkka Pullinen

1 Yleistä

1.1 Lupa käyttää materiaalia yliopistolla opetustarkoituksiin ja jatkokehittelyyn

Tätä dokumenttia saa käyttää yliopistolla opetustarkoituksiin ja jatkokehitykseen.

2 JMS Johdanto

2.1 Mikä on JMS?

JMS on Javan sovellusohjelmaliittymä (API, application program interface). JMS:n avulla voidaan luoda, lähettää, vastaanottaa sekä lukea viestijärjestelmästä järjestelmän viestejä. JMS:llä toteutettuja sovelluksia voidaan käyttää välikerrosohjelmistoissa, jotka sitovat yrityksissä useita komponentteja yhteen, esimerkiksi hajautetussa järjestelmässä viestien välitykseen palvelinten välillä. Viestijärjestelmät toimivat normaalisti lähteestä kohteeseen (point-to-point) tai julkaise-tilaa (publish-subscribe) mallin mukaan. Lähteestä kohteeseen mallissa jokainen viesti laitetaan jonoon, josta ne käsitellään järjestyksessä. Julkaise-tilaa mallissa vastaanottajia voi olla useampia toisin kuin PTP-yhteydessä.

JMS sovellukset tarvitsevat toimiakseen JMS tarjoajan, joka toteuttaa JMS API:n määrittämät palvelut sekä hoitaa viestien säilytyksen (tarvittaessa) ja välityksen JMS sovellusten välillä. JMS tarjoajia on useita, esimerkkeinä OpenJMS (http://openjms.sourceforge.net/) ja esimerkkien testauksessa ja toimintaan laitossa käytetty Sun:n Java EE sovelluspalvelin (http://java.sun.com/javaee/downloads/index.jspm, jossa JMS:n toteuttaa tarkalleen OpenMQ, https://mq.dev.java.net/)

2.2 JNDI - Java Naming and Directory Interface API

JNDI tarjoaa rajapinnan palveluun, jonne voidaan lisätä muita Java olioita tietyllä nimellä muiden olioiden saatavaksi. Olioiden JNDI:hin lisäyksen jälkeen muut sovellukset, esim JMS käyttävät eri osapuolet, voivat kysellä rajapinnan kautta halutun nimisen olion saatavuutta ja tämän yhteystietoja. JNDI toimii siis tavallaan kuten DNS, mutta kyselyn tuloksena saadaan haettava Java olio.

JMS sovellukset käyttävät JNDI:tä kyselemään tarvittavat JMS tarjoajan tarjoamat hallinnalliset oliot käytettyjen yhteyksien luontia varten JMS sovellusten/JMS tarjoajan välille ja etsimään käyttöönsä tarvitsevansa nimetyt jonot (Point-to-Point) ja/tai aihealueet (Publish-Subscribe).

Sovellukseen luotavalle JNDI oliolle voidaan antaa alustuksen yhteydessä tiedot juuri tai toisesta JNDI -instanssista, johon luotava JNDI olio liitetään. Näin ympäri verkkoa erillisillä palvelimilla olevat sovellusten tarjoamat JNDI:t liitetään yhdeksi JNDI avaruudeksi tietyn JNDI juuri olion avulla.

JNDI tarjoaa siis vain rajapinnan tietyille toiminnoille, jotka täytyy toteuttaa jonkin tarjoajan toimesta, joka liittää rajapinnan toiminnot esimerkiksi johonkin toiseen olemassa olevaan hakemisto-/nimipalveluun.

2.3 Arkkitehtuuri

JMS sovellukset koostuvat seuraavasta komponenteista: asiakkaat, ei-JMS asiakkaat, viestit, tuottaja sekä hallinnalliset oliot. Asiakkaat ovat Java-ohjelmia, jotka vastaanottajat ja lähettävät viestejä. Ei- JMS asiakkaat käyttävät järjestelmän omaa API:a JMS API:n sijaan, jolloin JMS tarjoaja muuntaa viestin muodon tarvittaessa JMS:n määrittämästä ei-JMS asiakkaan ymmärtämään muotoon ja toisinpäin. Viestit ovat joukko viestejä, joita käytetään asiakkaiden välillä. JMS-tarjoaja toteuttaa JMS-viestijärjestelmän sekä hallinnoi ja kontrolloi viestejä. Hallinnalliset oliot ovat valmiiksi konfiguroitu järjestelmään, jotka ylläpitäjä asettaa asiakkaiden käyttöön tarvittavien yhteyksien muodostamista varten JMS -asiakkaiden ja –tarjoajan välille sekä käytettyjen jonojen ja topikien kyselyyn käyttöä varten.

2.4 Yhteiset osat

JMS määrittää JMS sovelluksille yhteisen rajapinnan, joiden määrittämien metodien toteuttavia olioita käytetään hallinnallisten olioiden luomiseen JMS asiakkassovelluksen toimesta. JMS:n määrittämä rajapinta toteutetaan jonkin JMS tarjoajan toimesta.

JMS tarjoajan määrittelemät ja tarjoamat hallinnalliset oliot JMS asiakkaiden käyttöön ovat *ConnectionFactory* ja *Destination*. *ConnectionFactory:*ä käytetään luomaan yhteys tuottajaan sisältäen tarjoajan määrittämät tarvittavat asetukset yhteyden luomista varten. *Destination* oliot pitävät kirjaa siitä, mihin viestit ovat tarkoitus lähettää (jono, aihealue) tarjoten mahdollisuuden käyttää haluttua nimeämiskäytäntöä käytetyissä osoitteissa. *ConnectionFactory* sekä *Destination* oliot ovat samassa nimiavaruudessa, josta JMS-asiakas voi kysellä JNDI:n avulla esimerkiksi olion saatavuutta. Nämä oliot tukevat samanaikaisuutta.

JMS asiakkaat luovat *Connection* olion saadun *Connectionfactory* luokan avulla. *Connetion* olio luodaan yhteyden autentikoinnin yhteydessä ja se on yhteysolio asiakkaan ja JMS tarjoajan välille luoden käytetyn TCP socket:n. *Connection* olion avulla luodaan yhteyttä käyttävät *Session* oliot (istunto) lopullista viestien lähetystä ja vastaanottamista varten. Yhteyden luonnin yhteydessä määritellään myös joko JMS tarjoajan toimesta tai asiakassovelluksen puolesta asiakastunniste (Client Identifier) yhteydelle. *Connection* olio sisältää myös mahdollisuuden määrittää poikkeuskäsittelijän (ExceptionListener), jonka kautta JMS tarjoaja ilmoittaa JMS asiakkaalle yhteysongelmasta (*JMSException* luokan olioiden avulla), jota JMS tarjoaja ei ole pystynyt korjaamaan. *Connection* olion määrittämä yhteys on luonnin jälkeen pysäytetyssä (stop) tilassa, jolloin viestejä asiakkaalle ei vielä tule. *Connection* olio asetetaan käynnissä (start) tilaan vasta sen jälkeen, kun tarvittava(t) viestienkäsittelijä olio(t) on luotu, jolloin tuleville viesteille on tarvittava käsittelijä sovelluksessa. JMS sovelluksen lopulliseen viestien lähettämiseen ja vastaanottoon tarvittavat oliot *MessageConsumer* ja *MessageProvider* luodaan *Session* olion avulla, ja vaikka niitä voidaan luoda monta yhdelle istunnolle, ne eivät tue rinnakkaisuutta vaan niitä tulee käyttää sarjallisesti. *Connection* olion luoma yhteys voidaan pysäyttää ja käynnistää käytön aikana ja asiakas voi lähettää *MessageProvider* olion avulla viestejä ulospäin yhteyden ollessa pysäytettynä. JMS asiakassovelluksen

sammutuksen yhteydessä luodut *Session* olio ja *Connection* olion yhteys tulee sulkea hallitusti.

Session olio siis tarjoaa muun muassa metodit yhteyden viestien käsittelyyn ja luomiseen, tarvittavien viestien lähetys ja vastaanotto olioiden luontia varten (*MessageProvider*, *MessageConsumer*), JMS-tarjoaja optimoidut metodit lähetettävien viestiolioiden luomista varten sekä metodit tarvittavien olioiden luomiseen väliaikaisten ja dynaamisten jonojen tai aihealueiden hallintaa ja käyttöä varten (*TemporaryQueues*, *TemporaryTopics*, *Queues*, *Topics*). *Session* olio mahdollistaa myös tarvittaessa transaktioihin perustuvat viestien lähetyksen ja käsittelyn ja pitää huolta vastaanotettujen viestien kuittaamisesta sessioon määritetyllä tavalla. Session olio pitää huolta viestien vastaanotosta oikeassa järjestyksessä.

Viestien vastaanottoon luotu *MessageConsumer* olio voi vastaanottaa viestejä joko synkronisesti *receive()* -metodin avulla tai asynkronisesti luomalla *MessageListener* rajapinnan toteuttavan *MessageConsumer* olion tätä varten.

Viestien lähetykseen luotu *MessageProvider* olio luodaan antaen käytetty *Queue* tai *Topic* –olio (Kohdealueen jonon tai aihealueen toimintojen rajapinta *Destination* oliolle) parametrina olion luovalle metodille määrittäen kohteen, johon lähetettävät viestit lähetetään. JMS määrittää kaksi eri viestien lähetys tapaa, joista toisessa viestit lähetetään korkeintaan kerran (NON_PERSISTENT, eli sallitaan viestien hukkuminen matkalla) ja toisessa viestit tulee lähettää kerran ja vain kerran (PERSISTENT, eli lähetettävää viestiä ei saa hukata ja lähetyksen tulee onnistua). Lähetettävälle viestille voidaan määrittää myös *time-to-live* aika, jonka jälkeen viesti poistetaan JMS tarjoajan toimesta jos sitä ei ole saatu lähetettyä vastaanottajalle.

2.5 JMS Viestit

JMS määrittelee myös siinä käytettyjen viestien muodon. Viesti sisältää seuraavat pääosat: Otsikko kentät, joilla määritellään viestin vakiotiedot ja joilla annetaan ohjeita viestin käsittelyyn JMS tarjoajalle ja JMS asiakkaille, lisäasetukset (ominaisuudet), joilla sovellukset voivat lisätä tarvitsemiaan otsikkokenttiä sekä viimeisenä itse viestitieto (runko, body).

Jotkin otsikkokenttien tiedoista voidaan sopia käyttämättömiksi viestin koon minimoimiseksi (JMS tarjoajan salliessa tämän ominaisuuden), jolloin sovittujen kenttien arvot sisältävät vain nolla tai tyhjämerkin.

Lisäasetuksiin voidaan määrittää vain tietyn muotoisia arvoja ja jotkin ominaisuuksien nimet ovat JMS:n varaamia tulevaa käyttöä varten (kuten JMSX –alkuiset ominaisuuksien nimet).

3 Point-to-point viestintä (PTP)

Point-to-point viestintä on lähettäjien ja vastaanottajien välistä suoraa viestintää, jossa yhdellä viestillä on vain yksi vastaanottaja. Jokaisella vastaanottajalla on oma jononsa jonne lähettäjät välittävät sille tarkoitetut viestit. Jonot toimivat kuten geneeriset postilaatikot, toisin sanoen kunkin jonon luonti ja ylläpito vaatii resursseja. Tavallisesti vastaanottajan kaikki viestit toimitetaankin yhteen jonoon. Vastaanottaja poimii sitten jonostaan viestin ja vahvistaa viestin purettuaan sen onnistuneen purun. Viestit säilytetään jonossa, kunnes vastaanottaja poimii ne tai ne vanhentuvat. Viestin lähettäjän ja vastaanottajan välillä ei näin ollen ole aikariippuvuutta, vastaanottaja voi poimia viestin kun se pääsee suoritukseen.

JMS PTP malli määrittelee kuinka käyttäjät toimivat jonojen kanssa, kuinka ne löydetään, ja kuinka niihin välitetään ja niistä haetaan viestejä.

Jono oliossa (*Queue*) on määritelty tarjoajakohtainen jonon nimi, jonka perusteella käyttäjät voivat määrittää jonon identiteetin JMS:n metodeille. Tarvittaessa ajon aikana voidaan luoda myös *TemporaryQueue* (väliaikainen jono) tyyppisiä jonoja ja ne ovat olemassa vain yhteyden ajan (*QueueConnection* olion olemassa olon ajan). Jonot siis luodaan yhteyden muodostuksessa. JMS ei tarjoa palveluita pitempikestoisten jonojen käyttöön. Tämä ei ole ongelma, koska useimmat palvelunkäyttäjät käyttävät staattisesti määriteltyjä jonoja. Jos jonossa on vastaanottajan käsittelemiä, mutta vahvistamattomia viestejä yhteyden sulkeutuessa pitää nämä viestit säilyttää ja lähettää uudelleen kunnes vastaanottaja seuraavankerran ottaa yhteyden jonoon.

Palvelunkäyttäjät käyttävät *QueueConnectionFactory* oliota luodessaan yhteyden *QueueConnection* olion avulla JMS PTP palveluntarjoajaan. Edelleen *QueueConnection* oliota käytetään luomaan *QueueSession* olioita, joilla puolestaan tarvitaan viestien luontiin ja vastaanottoon. *QueueSession* luokkaa tarjoaa metodit *QueueReceiver*, *QueueSender*, *QueueBrowsers* ja *TemporaryQueues* olioiden luontiin. Vastaanottaja käyttää *QueueReceiver* olion metodeja hakiessaan viestejä jonosta. Olioon voidaan määritellä, että jonosta haetaan vain tiettyjä viestejä toisin sanoen viestejä ei ole välttämätöntä käsitellä saapumisjärjestyksessä. Vastaanottaja käyttää *QueueBrowser* oliota jonossa olevien viestien tutkimiseen, ilman, että niitä täytyy käsitellä/poistaa jonosta. Selausmetodi palauttaa *java.util.Enumeration*, jota voidaan käyttää joko koko jonon sisällön tai vain jonon sisältämien tiettyjen viestien tutkimiseen.

PTP rajapinta	Yleinen JMS rajapinta
QueueConnectionFactory	ConnectionFactory
QueueConnection	Connection
Queue	Destination
QueueSession	Session
QueueSender	MessageProducer
QueueReceiver	MessageConsumer

4 Publish / Subscibe viestintä (PUB/SUB)

Pub/Sub malli määrittelee kuinka JMS-käyttäjät julkaisevat ja hakevat viestejä tunnetuista, sisällön perusteella järjestetyistä "solmuista". JMS:ssä näitä solmuja kutsutaan aihekokonaisuuksiksi (topic). Aihekokonaisuus voidaan käsittää viestien välittäjäksi, joka kokoaa ja jakaa sille toimitettuja viestejä. Tässä mallissa aihekokonaisuuksien välittäjä toimii siis välikätenä viestien julkaisijoiden ja vastaanottajien välillä (vrt. PTP malli).

Käyttäjä käyttää *TopicConnectionFactory* luokkan oliota luodakseen yhteyden *TopicConnection* olion avulla JMS PUB/SUB palveluntarjoajaan. *TopicConnection* on aktiivinen yhteys palveluntarjoajaan. Käyttäjä käyttää *TopicConnection* oliota luodakseen yhden tai useampia *TopicSession* olioita viestien luontiin ja vastaanottoon. *TopicSession* luokka tarjoaa metodit *TopicPublisher*, *TopicSubsciber* ja *TemporatyTopics* luokkien olioiden luontiin. Se tarjoaa myös *unsubscibe* metodin asiakkaan kestoiällä varustetun kirjautumisten (durable subscription) poistamiseen. Julkaisijat ja vastaanottajat ovat aktiivisia, kun niiden edustamat java oliot ovat olemassa. JMS tukee myös vaihtoehtoista durability (kestoikä) ominaisuutta vastaanottajille, jonka avulla JMS muistaa viestin vastaanottajan, vaikka tämä ei olisikaan aktiivinen. Jos järjestelmässä on kuittaamattomia viestejä *TopicSession* olion määrittämän istunnon sulkemisen aikaan, kestoiällä varustetun *TopicSubscriber* olion täyttyy säilyttää ja uudelleen välittää ne. Jos kirjautuminen on toteutettu normaalilla *TopicSubscriber* luokalla näin ei tarvitse menetellä. Kestoiällä varustetut *TopicSubscribe* oliot vievät JMS palveluntarjoajalta enemmän resursseja. *TopicPublisher* oliota käytetään aiheeseen liittyvien viestien lähettämiseen aihekokonaisuuteen. *TopicSubsciber* oliota käytetään vastaavasti aiheeseen liittyvien viestien vastaanottoon. Viestin valitsimen (message selector) filtteroi viestejä ja määrittelee mitkä toimitetaan vastaanottajalle.

Käyttäjän näkökulmasta palveluntarjoaja rekisteröi julkaisijat ja vastaanottajat dynaamisesti kirjautumisen yhteydessä. Palveluntarjoajan kannalta näiden luontiin voidaan tarvita muutakin. JMS ei määrittele aiheiden/viestien varastointiin allokoitua tilaa tai resurssien ylivuodon seurauksia JMS tarjoajien toteuttaessa nämä kukin omalla tavallaan.

JMS PUB/SUB palveluntarjoajan täytyy taata yhtäläinen toimivuus JMS käyttäjille huolimatta siitä käyttääkö JMS käyttäjäohjelma PUB/SUB rajapintaa vai yleistä JMS rajapintaa.

PUB/SUB rajapinta	**Yleinen JMS rajapinta**
Topic connection factory	Connection Factory
Topic connection	Connection
Topic	Destination
TopicSession	Session
TopicPublisher	MessageProducer
TopicSubscriber	MessageConsumer

4.1 PUB/SUB vasteaika

Kaikissa PUB/SUB järjestelmissä on tyypillisesti jonkin verran viivettä. Vastaanottajan näkemät viestit voivat vaihdella riippuen siitä kuinka nopeasti JMS palveluntarjoaja aktivoi vastaanottajan olemassaolon ja kuinka kauan itse viestien välitys kestää. Esim. vastaanottaja voi olla saamatta joltain kaukaiselta julkaisijalta lähetettyä viestiä, koska palveluntarjoaja ei ole ehtinyt julkaista uuden vastaanottajan olemassaoloa koko järjestelmään. Toisaalta aktivoitu vastaanottaja voi saada myös vanhoja viestejä jotka ovat palvelutarjoajalla vielä tallella.

JMS ei määrittele tarkkaa välitys- tai toimintalogiikkaa sille ajalle kun palveluntarjoaja aktivoi uuden käyttäjän. Toiminta taataan vasta kun uusi käyttäjä on määritelty koko järjestelmään.

4.2 Kestoikä

Vastaanottaja näkee julkaistut viestit ainoastaan ollessaan aktiivinen. Vastaanottaja voi määritellä itselleen erillisen, kestoiällä varustetun, vastaanottajan (durable subscriber) statuksen kirjautuessaan järjestelmään. Tällöin JMS säilyttää välitettyjä viestejä, vaikka vastaanottaja ei olisikaan aktiivinen, kunnes vastaanottaja aktivoituu ja saa viestit tai kunnes viestit vanhenevat.

Kaikkien JMS palveluntarjoajien pitää pystyä ajamaan JMS sovelluksia, jotka dynaamisesti luovat ja tuhoavat kestoiällä varustettuja kirjautumisia.

4.3 Aihekokonaisuuksien hallinta (Topic management)

JMS ei määrittele toimintoja aihekokonaisuuksien hallintaan, luontiin ja poistamiseen. Jotkin tuotteen vaativat, että aiheet on staattisesti määritelty, mutta joillain ei ole minkäänlaista aiheiden hallintaa.

4.4 Aihe (Topic)

Topic olioiden avulla viestien välittäjä määrittelee aiheen identiteetin JMS metodeille. Monet PUB/SUB palveluntarjoajat määrittelevät aiheet hierarkioiksi ja tarjoavat monia tapoja kirjautua hierarkian eri osiin. JMS ei määrittele rajoituksia koskien mitä Topic olio edustaa. Se voi olla oksa aihehierarkiassa tai se voi olla suurempi osa hierarkiasta. Aiheiden ja niihin liittyvien kirjautumisien organisointi on tärkeä osa SUB/PUB sovelluksen arkkitehtuuria. JMS ei määrittele tapaa kuinka tämä tulisi toteuttaa. Väliaikainen aihe (*TemporaryTopic*) on erityisoolio, joka luodaan yhteyden (*Connection*) tai aiheyhteyden (*TopicConnection*) ajaksi. Se on aihe, jota voi käyttää ainoastaan sen luonut yhteys.

4.5 TopicRequestor (aihe pyytäjä)

JMS tarjoaa *TopicRequestor* apuluokan palvelupyyntöjen yksinkertaistamiseen. *TopicRequestor* luokan konstruktorille annetaan parametreina *TopicSession* olio ja haluttu aihe. Se luo *TemporaryTopic* olion vastauksille ja tarjoaa *request()* metodin joka välittää halutun viestin ja odottaa sen vahvistusta.

5 Ohjelma esimerkit

Alla olevat esimerkit saa käännettyä ja ajettua seuraavan ohjeen mukaan (Esimerkit toimivat myös muiden JMS tarjoajien avulla jolloin JMS hallintaresurssien luonti saattaa tapahtua erilailla). Ohje esimerkkisovellusten ajoon pätee vain ympäristössä, jossa sekä JMS tarjoaja, että esimerkkisovellukset ajetaan samalla palvelimella. Jos sovellukset ajetaan eri palvelimilla (kuten oikeassa elämässä tarkoitus on), tulee JNDI alustaa toisella tavalla (Ottamaan yhteys juuri JNDI olioon).

1. Lataa Java 2 EE "GlassFish" (http://java.sun.com/javaee/downloads/index.jsp, "GlassFish Java EE", "Download Java EE 5 SDK Update 6 for Windows, English")
2. Asenna sovellus. Windows puolella normaali windows asennus. Internet sivustojen mukaan Vistassa kannattaa kohdepolkuna käyttää jotain muuta, kuin oletuksena annettavaa "Program Files" hakemistoa.
3. Lisää CLASSPATH ymäristömuuttujaan seuraavat jar -paketit:
    1. <javaEESDK asennushakemisto>\lib\javaee.jar
    2. <javaEESDK asennushakemisto>\lib\appserv-rt.jar
    3. <javaEESDK asennushakemisto>\lib\appserv-admin.jar
    4. <javaEESDK asennushakemisto>\lib\install\applications\jmsra\imqjmsra.jar
4. Käynnistä Java EE SDK (GlassFish) oletus palvelin
    - Windows: Start -> Programs -> Sun Microsystems -> Java EE 5 SDK -> Start default server, HUOM! Vistassa (Home 64bit) tuota polkua ei löydy ollenkaan yhden testatun asennuksen tapauksessa)
    - Komentokehoitteesta: asadmin start-domain domain1
5. Käynnistys pitäisi onnistua ilman ongelmia, olettaen että tarvittavat portit eivät ole varattuna.
6. Mene selaimella "Java System Application Server Consoleen" joka löytyy oletus asennuksella osoitteesta http://localhost:4848
7. Kirjaudu asennuksen yhteydessä määrittämälläsi administrator tunnuksella sisään
8. Mene vasemman menun polkuun: *Configuration -> Java Message Service*
9. Klikkaa oikealle ilmestynyttä ping ikonia testataksesi että JMS provider on toiminnassa, ilmoitus on "Ping succeeded: JMS service is running" kun kaikki ok JMS tarjoajan kohdalla.
10. Mene vasemman menun polkuun: *Resources -> JMS Resources -> Destination Resources*
11. Klikkaa "new" ikonia oikealla luodaksesi esimerkeissä käytetyt jonot (ja topic:t)
12. Poin-To-Point Jonon (Queue) luonti (aseta seuraavat tiedot näkyviin kenttiin)
    1. **JNDI Name**: jmsTestiJono
       (Ulospäin näkyvä jonon nimi)
    2. **Physical Destination Name**: jono1
       (Sisäinen jono "fyysinen" jono jonka JMS provider luo automaattisesti ja jonka kautta ne viestit liikkuu)
    3. **Resource Type**: javax.jms.Queue
       (Jonon tyyppi)
13. Publish-subscribe aiheen (Topic) luonti (aseta seuraavat tiedot näkyviin kenttiin)

1. **JNDI Name**: jmsTestiTopic
   (Ulospäin näkyvä aihealueen nimi)
2. **Physical Destination Name**: aihe1
   (Sisäinen jono "fyysinen" jono jonka JMS provider luo automaattisesti ja jonka kautta ne viestit liikkuu)
3. **Resource Type**: javax.jms.Topic
   (Jonon tyyppi)

14. Yhteyden toteuttajan luominen (Connection factory)
    1. **JNDI Name**: connectionFactory
       (Ulospäin näkyvä nimi)
    2. **Resource Type**: javax.jms.ConnectionFactory
       (Resurssin tyyppi)

    Käännä esimerkit normaalisti javac <tiedostonnimi.java>
15. Käännä esimerkit normaalisti **javac <tiedostonmini.java>**
16. Aja esimerkit normaalisti **java <tiedostonimi>** ottaen huomioon mahdolliset kommentit esimerkin alussa.

5.1 Point-to-Point esimerkki: Producer-Consumer kahdella prosessilla

Point-to-Point esimerkissä on toteutettu Producer-Consumer ongelman ratkaisu kahdelle prosessille käyttäen apuna JMS:ä, jolloin bufferina toimii JMS:n jono (queue). Ohjelmat voi kääntää 4. kohdan ohjeen mukaan. Ohjelmat voi ajaa joko samaan aikaan tai eri aikaan tai peräjälkeen. Consumer prosessi jää odottamaan ikuisesti kulutettavaa, jolloin producer on hyvä käynnistää jossain vaiheessa.

5.1.1 Producer.java

```
import javax.naming.*;
import javax.jms.*;

public class Producer {
  public static void main(String[] args) {
    int maxMessages, messagesCreated;
    ConnectionFactory connectionFactory=null;       /* JMS tarjoajan antama olio
                                                     * yhteyksien muodostamista varten
                                                     */
    Connection myConnection=null;                   /* Yhteys luokka joka muodostetaan
                                                     * ConnectionFactoryn avulla
                                                     */
    Session mySession=null;                         /* Istunto luokka joka muodostetaan
                                                     * yhteys olion avulla
                                                     */
    Context JNDIContex=null;                        /* Paikallinen olio JNDI:lle */
    Queue myQueue=null;                             /* JNDI:n kautta kyseltävä käytettävä
                                                     * jono JMS tarjoajalta
                                                     */
    MessageProducer myProducer;                     /* Istunto olion avulla luotava
                                                     * viestien lähettäjä olio
                                                     */
    TextMessage myMessage;                          /* Istunto olion avulla luotava
                                                     * lähetettävä viesti olio
                                                     */
```

```
maxMessages=messagesCreated=0;

/* Parametrinä annetaan monta viestiä tuotetaan ja lähetetään maksimissaan */
if (args.length<1) {
  System.out.println("Anna parametrina monta viestiä tuotetaan maksimissaan.");
  System.exit(1);
}

try {
  maxMessages=new Integer(args[0]);
} catch (Exception e) {
  System.out.println(e);
  System.exit(1);
}

if (maxMessages<1) {
  System.out.println("Parametrinä annettava viestien määrä tulee "
                     +"olla positiivinen kokonaisluku.");
  System.exit(1);
}

/* Instanssin käyttämä JNDI olio, jonka olettaa käytettävien
 * JNDI oletus asetusten olevan jndi.properties tiedostossa
 * joka löytyy CLASSPATH:n määritellystä appserv-rt.jar paketista.
 */
try {
  JNDIContex=new InitialContext();
} catch (NamingException e) {
  System.out.println(e);
  System.exit(1);
}

/* JMS tarjoaman ConnectionFactoryn kysely ja haku JNDI:stä */
try {
  connectionFactory=(ConnectionFactory)JNDIContex.lookup("connectionFactory");
} catch (NamingException e) {
  System.out.println(e);
  System.exit(1);
}

/* Käytetyn jono -olion hakeminen JNDI:stä */
try {
  myQueue=(Queue)JNDIContex.lookup("jmsTestiJono");
} catch (NamingException e) {
  System.out.println(e);
  System.exit(1);
}

try {
  /* Käytetyn yhteyden ja yhteys olion luominen */
  myConnection = connectionFactory.createConnection();

  /* Yhteydessä käytetyn istunto olion luominen
   * false = Ei transaktioiden käyttöä
   * Session.AUTO_ACKNOWLEDGE = Automaattinen viestien kuittaus tapa
   */
  mySession=myConnection.createSession(false, Session.AUTO_ACKNOWLEDGE);

  /* Viestien tuottaja olion luominen */
  myProducer=mySession.createProducer(myQueue);

  //myConnection.start();
  while (messagesCreated<maxMessages) {
    messagesCreated++;
    /* Odotetaan maks 2 sekuntia viestien välillä */
    int i=(int)(Math.random()*2000);
    System.out.println("\nTuotetaan tuotettavaa (odotetaan "+i+" millisekuntia)...");
    try {
      /* Odotetaan maks 2 sekuntia viestien välillä */
      Thread.sleep( i);
    } catch (InterruptedException e) {
```

```
          System.out.println(e);
        }
        /* Luodaan lähetettävä TextMessage -tyyppinen viesti olio */
        myMessage=mySession.createTextMessage();
        myMessage.setText("Tuotettava tieto nro "+messagesCreated);

        System.out.println("Lähetetään tuotettu tieto viestinä eteenpäin...");
        myProducer.send(myMessage);

        System.out.println("Viesti lähetetty eteenpäin.");
      } /* while (...) */

      System.out.println("\nLähetetään viesti sisältäen tiedon lähetyksen lopusta.");
      myMessage=mySession.createTextMessage();
      myMessage.setText("end of production");
      myProducer.send(myMessage);

      System.out.println("\nSuljetaan yhteys");
      mySession.close();
      myConnection.close();

    } catch (JMSException e) {
      System.out.println(e);
      System.exit(1);
    } /* try */

    /* Sammutetaan ohjelma tylysti. Testatessa esimerkit jäivät hengaamaan loppuun
     * odottaen, jonkin taustalle luodun threadin sammumista.
     * Seuraavissa linkeissä on asiasta keskustelua:
     *    - http://forums.java.net/jive/message.jspa?messageID=268625
     *    - https://glassfish.dev.java.net/issues/show_bug.cgi?id=1429
     */
    System.exit(0);

  } /* main() */

} /* class Producer */
```

5.1.2 Consumer.java

```
import javax.naming.*;
import javax.jms.*;


public class Consumer {
  public static void main(String[] args) {
    ConnectionFactory connectionFactory=null;       /* JMS tarjoajan antama olio
                                                     * yhteyksien muodostamista varten
                                                     */
    Connection myConnection=null;                   /* Yhteys luokka joka muodostetaan
                                                     * ConnectionFactoryn avulla
                                                     */
    Session mySession=null;                         /* Istunto luokka joka muodostetaan
                                                     * yhteys olion avulla
                                                     */
    Context JNDIContex=null;                        /* Paikallinen olio JNDI:lle */
    Queue myQueue=null;                             /* JNDI:n kautta kyseltävä käytettävä
                                                     * jono JMS tarjoajalta
                                                     */
    MessageConsumer myConsumer;                     /* Istunto olion avulla luotava
                                                     * viestien vastaanottaja olio
                                                     */
    TextMessage myMessage;                          /* Vastaanottaja oliolta saatava
                                                     * vastaanotettu viesti olio
                                                     */
    String strMessage;

    /* Instanssin käyttämä JNDI olio, jonka olettaa käytettävien
```

```
 * JNDI oletus asetusten olevan jndi.properties tiedostossa
 * joka löytyy CLASSPATH:n määritellystä appserv-rt.jar paketista.
 */
try {
  JNDIContex=new InitialContext();
} catch (NamingException e) {
  System.out.println(e);
  System.exit(1);
}

/* JMS tarjoaman ConnectionFactoryn kysely ja haku JNDI:stä */
try {
  connectionFactory=(ConnectionFactory)JNDIContex.lookup("connectionFactory");
} catch (NamingException e) {
  System.out.println(e);
  System.exit(1);
}

/* Käytetyn jono -olion hakeminen JNDI:stä */
try {
  myQueue=(Queue)JNDIContex.lookup("jmsTestiJono");
} catch (NamingException e) {
  System.out.println(e);
  System.exit(1);
}

try {
  /* Käytetyn yhteyden ja yhteys olion luominen */
  myConnection=connectionFactory.createConnection();

  /* Yhteydessä käytetyn istunto olion luominen
   * false = Ei transaktioiden käyttöä
   * Session.AUTO_ACKNOWLEDGE = Automaattinen viestien kuittaus tapa
   */
  mySession=myConnection.createSession(false, Session.AUTO_ACKNOWLEDGE);

  /* Viestien tuottaja olion luominen, parametrinä käytettävän jonon olio */
  myConsumer=mySession.createConsumer(myQueue);

  myConnection.start();
  do {
    System.out.println("\nOdotetaan kulutettavaa...");
    /* Parametrinä arvo kuinka kauan odotetaan, jolloin 0 tarkoittaa, että odotetaan
     * kunnes seuraava viesti saadaan eli blokataan kuluttaja prosessi.
     */
    myMessage=(TextMessage)myConsumer.receive(0);

    System.out.println("Käsitellään vastaanotettua viestiä...");
    System.out.println("Saatu viesti: "+myMessage);
    strMessage=myMessage.getText();

  } while (strMessage!=null && !strMessage.equals("end of production"));
  System.out.println("\nLähetyspää lähetti viestin ilmoittaen lähetyksen lopusta.");

  System.out.println("\nSuljetaan yhteys");
  mySession.close();
  myConnection.close();

} catch (JMSException e) {
  System.out.println(e);
  System.exit(1);
} /* try */

/* Sammutetaan ohjelma tylysti. Testatessa esimerkit jäivät hengaamaan loppuun
 * odottaen, jonkin taustalle luodun threadin sammumista.
 * Seuraavissa linkeissä on asiasta keskustelua:
 *    - http://forums.java.net/jive/message.jspa?messageID=268625
 *    - https://glassfish.dev.java.net/issues/show_bug.cgi?id=1429
 */
System.exit(0);
```

```
  } /* main() */
} /* class Consumer */
```

5.2 Publisher-subscriber esimerkki

Publisher-subscriber esimerkkinä on toteutettu yksittäinen node luokka, joka lähettää tarvittaessa viestejä muille nodeille ja muuten käy tarkistamassa, onko viestejä muilta nodeilta tarjolla. Esimerkkiä voi käyttää jonkin hajautetun järjestelmän noden pohjana. Kääntö 4. kohdan ohjeen mukaan ja ajo käynnistämällä käännetty esimerkki ohjeen mukaan rinnakkain omiin komentokehoitteesiinsa pyörimään. Sovellus sammuvat vain painamalla CTRL-C.

5.2.1 PSNode.java

```
import javax.naming.*;
import javax.jms.*;

public class PSNode {

  public static void main(String[] args) {
    int maxMessages, messagesCreated;
    TopicConnectionFactory connectionFactory=null;   /* JMS tarjoajan antama olio
                                                      * yhteyksien muodostamista varten
                                                      */
    TopicConnection myConnection=null;               /* Yhteys luokka joka muodostetaan
                                                      * ConnectionFactoryn avulla
                                                      */
    TopicSession mySession=null;                     /* Istunto luokka joka muodostetaan
                                                      * yhteys olion avulla
                                                      */
    Context JNDIContex=null;                         /* Paikallinen olio JNDI:lle */
    Topic myTopic=null;                              /* JNDI:n kautta kyseltävä käytettävä
                                                      * "topic" JMS tarjoajalta
                                                      */

    TopicPublisher mySender;                         /* Istunto olion avulla luotava
                                                      * viestien lähettäjä olio
                                                      */
    TopicSubscriber myReceiver;                      /* Istunto olion avulla luotava
                                                      * viestien vastaanottaja olio
                                                      */
    TextMessage myMessage;                           /* Istunto olion avulla luotava
                                                      * lähetettävä viesti olio
                                                      */


    /* Instanssin käyttämä JNDI olio, jonka olettaa käytettävien
     * JNDI oletus asetusten olevan jndi.properties tiedostossa
     * joka löytyy CLASSPATH:n määritellystä appserv-rt.jar paketista.
     */
    try {
      JNDIContex=new InitialContext();
    } catch (NamingException e) {
      System.out.println(e);
      System.exit(1);
    }

    /* JMS tarjoaman TopicConnectionFactoryn kysely ja haku JNDI:stä */
    try {
      connectionFactory=(TopicConnectionFactory)JNDIContex.lookup("connectionFactory");
    } catch (NamingException e) {
      System.out.println(e);
      System.exit(1);
```

```
}

/* Käytetyn topic -olion hakeminen JNDI:stä */
try {
  myTopic=(Topic)JNDIContex.lookup("jmsTestiTopic");
} catch (NamingException e) {
  System.out.println(e);
  System.exit(1);
}

try {
  /* Käytetyn yhteyden ja yhteys olion luominen */
  myConnection=connectionFactory.createTopicConnection();

  /* Yhteydessä käytetyn istunto olion luominen
   * false = Ei transaktioiden käyttöä
   * Session.AUTO_ACKNOWLEDGE = Automaattinen viestien kuittaus tapa
   */
  mySession=myConnection.createTopicSession(false, Session.AUTO_ACKNOWLEDGE);

  /* Viestien lähettäjä olion luominen */
  mySender=mySession.createPublisher(myTopic);

  /* Viestien vastaanottaja olion luominen */
  myReceiver=mySession.createSubscriber(myTopic);

  myConnection.start();
  while (true) {

    /* Odotetaan maks 2 sekuntia viestien välillä */
    int i=(int)(Math.random()*2000);
    System.out.println("\nTehdään noden omia juttuja (odotetaan "+i
                       +" millisekuntia)...");
    try {
      /* Odotetaan maks 2 sekuntia viestien välillä */
      Thread.sleep( i);
    } catch (InterruptedException e) {
      System.out.println(e);
    }

    /* Arvotaan, lähetetäänkö viestiä muille */
    i=((int)(Math.random()*100));
    if ( i<20) {
      System.out.println("Muodostetaan viestiä...");
      myMessage=mySession.createTextMessage();
      myMessage.setText("Viesti nodesta "+mySender);
      /* Asetetaan oma lisäasetus lähetettävään viestiin, josta
       * selviää viestin lähettäjän ID, jolloin saadaan selville, tuliko
       * viesti itseltä vai muualta.
       */
      myMessage.setStringProperty("SenderNodeID", mySender.toString());
      mySender.publish(myMessage);
    } else {
      System.out.println("Haetaan viestejä...");
      myMessage=(TextMessage)myReceiver.receive(1);
      if (myMessage!=null) {
        String SenderNodeID=myMessage.getStringProperty("SenderNodeID");
        /* Onko saatu viesti oma lähettämä? */
        if (SenderNodeID==null || !SenderNodeID.equals(mySender.toString())) {
          System.out.println("Saatiin viesti "+myMessage);
        } else {
          System.out.println("Saatu viesti oli itse lähetetty.");
        } /* if (...) */
      } /* if (...) */
    } /* if (...) */

  } /* while (...) */

} catch (JMSException e) {
  System.out.println(e);
  System.exit(1);
```

```
    } finally {
      try {
        mySession.close();
        myConnection.close();
      } catch (JMSException e) {
      } /* try */
    } /* try */

  } /* main() */

} /* class PSNode */
```

6 LÄHTEET

Wikipedia: Java Message Service
http://en.wikipedia.org/wiki/Java_Message_Service

Java Message Service Specification - version 1.1
http://java.sun.com/products/jms/docs.html