N Überblick über die 13. Übung

- Einführung
- Konfiguration eines JXTA-Peers
- Peer Group und Peer
- Discovery Service
- JXTA-IDs
- Messages
- Advertisements
- Pipes und Pipe Service
- Module

N.1 Dokumentation

- Project JXTA v2.3: Java Programmer' s Guide
  - http://www.jxta.org/docs/JxtaProgGuide_v2.3.pdf
- Spezifikation zu JXTA
- Information zu Klassen im JavaDoc-Format
- Quellen
- Komplettes Paket (Version 2.3.6) in **`/local/jxta`**

N.2 Konfiguration eines Peers

- JXTA-Referenzimplementierung setzt sich aus einer Menge von Bibliotheken zusammen:
  - **`jxta.jar -`** Eigentliche JXTA-Implementierung mit allen Protokollen und wichtigen Elementen zum Betrieb eines Peers
  - **`Bcprov-jdk14.jar -`** API zur Zertifikatverwaltung und Sicherheitsmechanismen
  - **`log4j.jar -`** Logging-Mechanismen

N.2 Konfiguration eines Peers

- Übersetzen von JXTA-Anwendungen

```
>javac -classpath /local/jxta/lib/jxta.jar \
SimpleJxtaApp.java
```

- Ausführung einer JXTA-Anwendung

```
>java -classpath /local/lib/jxta.jar:\
                 /local/lib/log4j.jar:\
                 /local/jxta/lib/Bcprov-jdk14.jar
            -DJXTA_HOME=${HOME}/.jxta SimpleJxtaApp
```

- **JXTA_HOME** legt fest wo die Konfiguration des Peers und alle Metadaten wie Advertisements abgelegt werden

N.2 Konfiguration eines Peers

- Mit dem ersten Start einer JXTA-Applikation wird automatisch ein Konfigurationsdialog angezeigt
- Folgende Angaben können gemacht werden
  - Basic - Name des Peers und Proxy-Server (bei einer Firewall)
  - Advanced - TCP- und HTTP-Verbindungen sowie Regelung ob eingehende Verbindungen möglich sind (NAT /Firewall)
  - Rendezvous/Relay - Angabe einer Liste von Rendezvous und Relay Peers
  - Security - Username und Passwort
- Alle Konfigurationsdaten werden unter PlatformConfig im JXTA_HOME abgelegt. Zur Neukonfiguration kann die Datei gelöscht oder eine Datei *reconf* im JXTA_HOME angelegt werden.

N.3 Peer Group

- Peer Groups bilden Gemeinschaften von Peers mit gleichen Intressen und Diensten
- Zur Teilnahme an einen JXTA-Netzwerk muss zuerst die *Plattform* initialisiert bzw. der *World Peer Group* beigetreten werden
- JXTA bietet hierzu die **`PeerGroupFactory`**

```
import net.jxta.peergroup.*;

    PeerGroup platform = PeerGroupFactory.newPlatform();
```

  - Plattform stellt minimal erforderlichen Dienste zur Verfügung um andere Peer Groups zu finden oder zu erzeugen
  - Falls das lokale Peer noch keine ID besitzt wird sie durch die Plattform zugewiesen

N.3 Peer Group

- Eine weiterer Möglichkeit zur Initialisierung eines JXTA-Peers bildet die *Net Peer Group*

```
try {
    netPeerGroup = PeerGroupFactory.newNetPeerGroup();
} catch (PeerGroupException e) {
    System.out.println("group creation failure");
    ..
}
```

  - Die Erzeugung der Net Peer Group bedingt eine Initialisierung der Plattform
  - Die Net Peer Group wird im lokalen Netzwerk gesucht und falls sie nicht gefunden werden kann wird eine Default-Konfiguration verwendet
  - Die Default-Konfiguration kann nach den Erfordernissen des Administrators überschrieben werden

N.3 Peer Group

- Schnittstelle zu Peer Groups **`net.jxta.peergroup.PeerGroup`**
- Status Operationen

```
public PeerID getPeerID();

public String getPeerName();

public PeerGroupID getPeerGroupID();

public String getPeerGroupName();
```

- Verwaltungs Operationen

```
public PeerGroup getParentGroup();

public PeerGroup newGroup(PeerGroupID gid,
                          Advertisement impl,
                          String name,
                          String description)
                   throws PeerGroupException;
```

N.3 Peer Group

- Operationen zum Zugriff auf Dienste einer Peer Group

```
public MembershipService getMembershipService();

public DiscoveryService getDiscoveryService();

public PipeService getPipeService();

public PeerInfoService getPeerInfoService();

public Service lookupService(ID name)
                    throws ServiceNotFoundException
```


N.3 Peer Group

- HelloWorld-Beispiel

```
import net.jxta.peergroup.*;

public class SimpleJxtaApp {

    static PeerGroup netPeerGroup = null;

   private void startJxta() {
       try {

          // create and start the default JXTA NetPeerGroup
          netPeerGroup=PeerGroupFactory.newNetPeerGroup();

       } catch (PeerGroupException e) {
          System.out.println("group creation failure");
          e.printStackTrace();
          System.exit(1);
       }
   }
   ...
```


N.3 Peer Group

- HelloWorld-Beispiel (Fortsetzung)

```
public static void main(String args[]) {

      System.out.println("Starting JXTA ....");

      SimpleJxtaApp myapp = new SimpleJxtaApp();

      myapp.startJxta();

      System.out.println("Hello from JXTA group " +
                        netPeerGroup.getPeerGroupName() );

      System.out.println("  Group ID = " +
         netPeerGroup.getPeerGroupID().toString());

      System.out.println("  Peer name = " +
         netPeerGroup.getPeerName());

      System.out.println("  Peer ID = " +
         netPeerGroup.getPeerID().toString());

      myapp.netPeerGroup.stopApp();
      System.exit(0);
   }
```


N.3 Discovery Service

- Discovery Service ermöglicht die Veröffentlichung, sowie die Suche von *Advertisements*
- Advertisements beschreiben alle wichtigen Ressourcen wie Peers, Peer Groups, Pipes, ... oder Endpoints
- Veröffentlichen von Advertisements

```
public void publish(Advertisement advertisement)
            throws IOException;

public void publish(Advertisement adv,
                    int type,
                    long lifetime,
                    long lifetimeForOthers)
            throws IOException;

public void remotePublish(Advertisement adv);

public void remotePublish(Advertisement adv,
                          long lifetime);

public void remotePublish(String peerid,
                          Advertisement adv);
```

N.3 Discovery Service

- Operationen des **`DiscoveryService`** zur Suche nach *lokalen* Advertisements

```
public Enumeration getLocalAdvertisements(int type,
                                          String attribute,
                                          String value)
                                          throws IOException;
```

- Parameter
  - **`type`** - Art der Ressource **`PEER, GROUP`** oder **`ADV`**
  - **`attribute`** - Name des gesuchten Attributes innerhalb eines Advertisements
  - **`value -`** Wert des Attributes z.B. *test* aber auch Wildcards möglich **test**
- Rückgabewert
  - Gefundende Advertisements

N.3 Discovery Service

- Operationen des **`DiscoveryService`** zur Suche nach *entfernten* Advertisements

```
public int getRemoteAdvertisements(String peerid,
                                   int type,
                                   String attribute,
                                   String value,
                                   int threshold,
                                   DiscoveryListener listener);
```

- Parameter
  - **`type`** - Art der Ressource **`PEER, GROUP`** oder **`ADV`**
  - **`attribute`** - Name des gesuchten Attributes
  - **`value -`** Wert des Attributes z.B. *test* aber auch Wildcards möglich **test**
  - **`threshold -`** Maximale Anzahl der zurückgelieferten Advertisements
  - **`listener -`** Objekt welches bei Antworten benachrichtigt werden soll
- Rückgabewert
  - ID zur Identifikation der Suchanfrage

N.3 Discovery Service

- Zur Verarbeitung von Suchanfragenergebnissen wir das **`DiscoveryListener`** Interface verwendet

```
public void discoveryEvent(DiscoveryEvent e);
```

- Jede Anwort eines Peers wird als **`DiscoveryEvent`** an die Applikation zugestellt. Ein **`DiscoveryEvent`** bietet folgende Methoden an:

```
public DiscoveryResponseMsg getResponse();

public int getQueryID();

public Enumeration getSearchResults()
```


N.3 Discovery Service

- Discovery-Beispiel:

```
private DiscoveryService discovery;

private void startJxta() {
    //Siehe HelloWorld-Beispiel
        ...
    discovery = netPeerGroup.getDiscoveryService();
}

public void run() {
    try {

        discovery.addDiscoveryListener(this);

        while (true) {
            discovery.getRemoteAdvertisements(null,
                                  DiscoveryService.PEER,
                                  null, null, 5);
            try {
              Thread.sleep(60 * 1000);
            } catch(Exception e) {}
        }
    } catch(Exception e) { e.printStackTrace();}
}
```

N.3 Discovery Service

- Discovery-Beispiel (Fortsetzung):

```
public void discoveryEvent(DiscoveryEvent ev) {

    DiscoveryResponseMsg res = ev.getResponse();


    System.out.println("Got a Discovery Response [" +
                        res.getResponseCount() +
                       " elements] ");

    PeerAdvertisement adv = null;

    Enumeration enum = res.getAdvertisements();
    if (enum != null ) {
        while (enum.hasMoreElements()) {
            adv = (PeerAdvertisement) enum.nextElement();
            System.out.println ("Peer name="+ adv.getName());
        }
    }
}
```


N.4 JXTA-ID

- Eine ID kann mit Hilfe der **`net.jxta.id.IDFactory`** erzeugt werden

```
public static PeerGroupID newPeerGroupID();

public static PeerGroupID newPeerGroupID(String idformat);

public static PeerGroupID newPeerGroupID(byte[] seed);

public static PipeID newPipeID(PeerGroupID groupID);
...
```

- Einlesen einer JXTA ID

```
public static ID fromURI(URI source)
                  throws URISyntaxException
```

- Ermittlung des Standard ID-Formats

```
public static String getDefaultIDFormat();
```

- Beispiel: Erzeugung einer Pipe-ID

```
PipeID id= IDFactory.newPipeID(netPGroup.getPeerGroupID());
```


N.4 JXTA-ID

- Alle Ressourcen in JXTA werden durch eindeutige IDs identifiziert
- Es gibt verschiedene Typen von IDs
  - Peer Group IDs, Peer IDs, Codat IDs, Pipe IDs ,Module Class IDs, Module Spec IDs
- Darstellung
  - Nach der Spezifikation *urn:jxta:<Format>-<Daten>*
  - Im Rahmen der Referenzimplementierung *urn:jxta:uuid-<Daten>*
- Basisklasse aller IDs bildet **`net.jxta.id.ID`**
- Für alle Typen von IDs gibt es spezifische Unterklassen
  - z.B. **`net.jxta.peer.PeerID`**

N.5 Advertisement

- Metadaten-Beschreibungen angebotener Netzwerk-Ressourcen
- Zu Verarbeitung werden sie als strukturierte Dokumente repräsentiert
  - Zugriff erfolgt auf Tupel aus Name und Blatt
- Beispiel: Peer Group Advertisment

```
<?xml version="1.0" ?>
<!DOCTYPE jxta:PGA>
<jxta:PGA xmlns:jxta="http://jxta.org">

    <GID> urn:jxta:jxta-NetGroup</GID>

    <MSID>urn:jxta:uuid-DEADBEEF2332342342...</MSID>

    <Name>NetPeerGroup</Name>

    <Desc>NetPeerGroup by default</Desc>

</jxta:PGA>
```

N.5 Advertisement

- Erzeugung eines Advertisments erfolgt mit Hilfe der **`net.jxta.document.AdvertisementFactory`**

```
public static Advertisement newAdvertisement(String
                                    advertisementType);

public static Advertisement newAdvertisement(
                                    MimeMediaType mimetype,
                                    InputStream stream
                                  )throws IOException;

public static Advertisement newAdvertisement(
                                    MimeMediaType mimetype,
                                    Reader source
                                  )throws IOException;
```

- Beispiel: Erzeugung eines Pipe-Advertisements

```
PipeAdvertisement pa = (PipeAdvertisment)
       AdvertisementFactory.newAdvertisement(
           PipeAdvertisement.getAdvertisementType()
       );
```


N.5 Advertisement

- Advertisement-Basisklasse

```
public static String getAdvertisementType();

public abstract ID getID();

public abstract Document getDocument(MimeMediaType
                                        asMimeType);
```

- Die Advertisement-Fabrik kennt schon zahlreiche Advertisement-Typen weiter können jedoch registriert werden

```
public static boolean registerAdvertisementInstance(
          String rootType,
          AdvertisementFactory.Instantiator instantiator);
```


N.5 Advertisement

- Ein/Ausgabe eines Pipe-Advertisements

```
..
try {
    FileInputStream is = new FileInputStream(NAME);
    pipeadv = (PipeAdvertisement)
        AdvertisementFactory.newAdvertisement(
                        new MimeMediaType("text/xml"), is);
    is.close();
} catch (java.io.IOException e) {
    System.out.println("failed to read/parse pipe adv.");
    e.printStackTrace();
    System.exit(-1);
}

StructuredTextDocument doc = (StructuredTextDocument)
    pipeadv.getDocument(new MimeMediaType("text/plain"));

StringWriter out = new StringWriter();

doc.sendToWriter(out);

System.out.println(out.toString());
...
```


N.6 Message

- Grundlegende Einheit zum Transfer von Daten
- Gliedert sich in (Name, Type, Wert)-Tupel deren Reihenfolge beachtet wird
- Datenformat ist binär oder XML-basiert je nach Protokoll
- Transport erfolgt über den End Point-Service oder via Pipes über den Pipe-Service
- **`net.jxta.endpoint.Message`** bildet einen Container für Instanzen der abstrakten Klasse **`net.jxta.endpoint.MessageElement`** bzw. den konkreten Unterklassen:
  - **`TextMessageElement`**
    - ➤ **`StringMessageElement`**, **`TextDocumentMessageElement`**
  - **`ByteArrayMessageElement`**
  - **`InputStreamMessageElement`**

N.6 Message

- Methoden zur Manipulation einer Message

```
public void addMessageElement(MessageElement add);

public void addMessageElement(String namespace,
                              MessageElement add);

public MessageElement replaceMessageElement(
                             MessageElement replacement);

public MessageElement replaceMessageElement(
                             String namespace,
                             MessageElement replacement);

public MessageElement getMessageElement(String name);

public MessageElement getMessageElement(String namespace,
                                        String name);

public Message.ElementIterator getMessageElements();

public Message.ElementIterator getMessageElements(
                                                String name);
...
```


N.6 Message

- Erzeugung einer Message

```
Message msg = new Message();

Date date = new Date(System.currentTimeMillis());

StringMessageElement sme = new StringMessageElement(
                  "DATE", date.toString() , null);

msg.addMessageElement(null, sme);
```


N.7 Pipe

- Virtuelle Verbindungen die zu verschiedenen Zeitpunkten mit unterschiedlichen Peer-Endpoints verbunden sein können
- Jede Pipe kann durch eine JXTA ID eindeutig indentifiziert werden
- Pipes sind unidirektional und besitzen mit *Output Pipe* eine Datenquelle und mit *Input Pipe* eine Datensenke
- Es gibt zwei Arten von Pipes:
  - *Point-to-Point* Pipes für unidirektionale, asynchrone Verbindungen
    - ➤ Es gibt kein Acknowledgment oder Reply
    - ➤ Antworten werden über eine umgekehrt gerichtete Pipe versendet
  - *Propagate Pipes* für Multicast-Verbindungen

N.7 Pipe

- Sowohl Input als auch Output-Pipes werden durch den Pipe-Service erzeugt
- Erzeugen einer Input-Pipe

```
public InputPipe createInputPipe(PipeAdvertisement adv)
                                 throws IOException;

public InputPipe createInputPipe(PipeAdvertisement adv,
                                 PipeMsgListener listener)
                                 throws IOException;
```

- Parameter
  - **`adv`** - Zu bindendes Advertisement
  - **`listener`** - Listener für alle eintreffenden Nachrichten

N.7 Pipe

- Erzeugen einer Ouput-Pipe

```
public OutputPipe createOutputPipe(PipeAdvertisement adv,
                                   long timeout)
                            throws IOException;

public OutputPipe createOutputPipe(PipeAdvertisement adv,
                                Enumeration resolvablePeers,
                                   long timeout)
                            throws IOException;

public void createOutputPipe(PipeAdvertisement adv,
                             PeerID peerid,
                             OutputPipeListener listener)
                      throws IOException;
```

- Parameter
  - **adv** - Zu bindendes Advertisement
  - **timeout** - Zeitspanne bis zum Abbruch des Verbindungsversuches
  - **resolvablePeers** - Menge von anzufragenden Peers mit PeerID
  - **listener** - Listener der angesprochen wird sobald ein Bindung erfolgt

N.7 Pipe

- Methoden einer Input-Pipe **net.jxta.pipe.InputPipe**

```
public Message waitForMessage()
                       throws InterruptedException;

public Message poll(int timeout)
            throws InterruptedException;

public void close();
```

  - Rückgabewert **null** indiziert das die Verbindung geschlossen wurde oder aber die Wartezeit abgelaufen ist
  - Auslesen von Informationen über die Pipe

```
public PipeAdvertisement getAdvertisement();
public ID getPipeID();
```


N.7 Pipe

- Methoden einer Output-Pipe **net.jxta.pipe.OutputPipe**

```
public boolean send(Message msg) throws IOException;
```

  - Rückgabewert zeigt an ob die Nachricht erfolgreich versendet werden konnte. Ist dies nicht der Fall kann es erneut versucht werden da kein schwerwiegender Fehler vorliegt.

```
public void close();
public boolean isClosed();
```


N.7 Pipe

- Bidirektionaler Datenaustausch durch automatischen Aufbau einer zweiten umgekehrten Pipe mittels **net.jxta.util.JxtaBiDiPipe**

```
public void connect(PeerGroup group,
                    PipeAdvertisement pipeAd)
             throws IOException;

public void connect(PeerGroup group,
                    PeerID peerid,
                    PipeAdvertisement pipeAd,
                    int timeout,
                    PipeMsgListener listener)
             throws IOException;

public Message getMessage(int timeout)
                   throws InterruptedException;

public boolean sendMessage(Message msg)
                    throws IOException;

public void setListener(PipeMsgListener listener);
// Für JxtaServerPipe
public void setListener(PipeEventListener listener);

...
```

N.7 Pipe

- Annahme von Verbindungswünschen ähnlich des ServerSockets durch **`net.jxta.util.JxtaServerPipe`**

```
public void bind(PeerGroup group,
                 PipeAdvertisement pipeadv)
          throws IOException;

public JxtaBiDiPipe accept() throws IOException;

public void setPipeTimeout(int timeout)
                   throws SocketException;

public void close() throws IOException;
```


N.8 Module

- Module in JXTA sind einfache funktionale Einheiten die durch ein definiertes Protokoll über das Netzwerk angesprochen werden können
- Es gibt zwei verschiedene Ausprägungen von Modulen
  - Service
  - Applikation
- Um ein Modul zu veröffentlichen sind folgende Advertisements nötig
  - **`Module Class Advertisement`** - Allgemeine Klasse einer Funktionalität
  - **`Module Specifiaction Advertisement`** - Spezifikationen
  - **`Module Implementation Advertisement`** - Implementierungen

N.8 Module

- Parameter eines Module Class Advertisements
  - MCID - ID der Klasse
  - Name - Name der Modul Klasse
  - Desc - Beschreibung
- Parameter eines Module Specification Advertisements
  - MSID - ID des Moduls
  - Vers - Version der Spezifikation
  - Name - Name der Modulspezifikation
  - Desc - Beschreibung des Moduls
  - Crtr - Erzeuger der Spezifikation
  - SURI - Link zur Spezifikation
  - Parm - zusätzliche Parameter
  - PipeAdvertisement

N.8 Module

- Parameter eines Module Implementation Advertisement
  - MSID - ID des Moduls
  - Comp - Kompatibilitätsinformationen
  - Code - Codereferenz, in der aktuellen Implementierung ein Klassename
  - PURI - URL zum Code
  - Prov - Entwickler des Moduls
  - Desc - Beschreibung
  - Parm - Weitere Parameter
- Module Implementation Advertisements besitzen keine eigene ID

N.8 Module

- Jedes Modul implementiert das Interface **`net.jxta.platform.Module`** mit folgenden Methoden

```
public void init(PeerGroup group,
                 ID assignedID,
                 Advertisement implAdv)
          throws PeerGroupException;

public int startApp(String[] args);

public void stopApp();
```

- Ein Service erbt von der Schnittstelle und bietet folgende zusätzliche Funktionen an

```
public Service getInterface();

public Advertisement getImplAdvertisement();
```


N.8 Module

- Laden und starten von Modulen erfolgt über die Mechanismen der Klasse **`PeerGroup`**

```
public Module loadModule(ID assignedID,
                         Advertisement impl)
                  throws ProtocolNotSupportedException,
                         PeerGroupException;

public Module loadModule(ID assignedID,
                         ModuleSpecID specID,
                         int where);
```