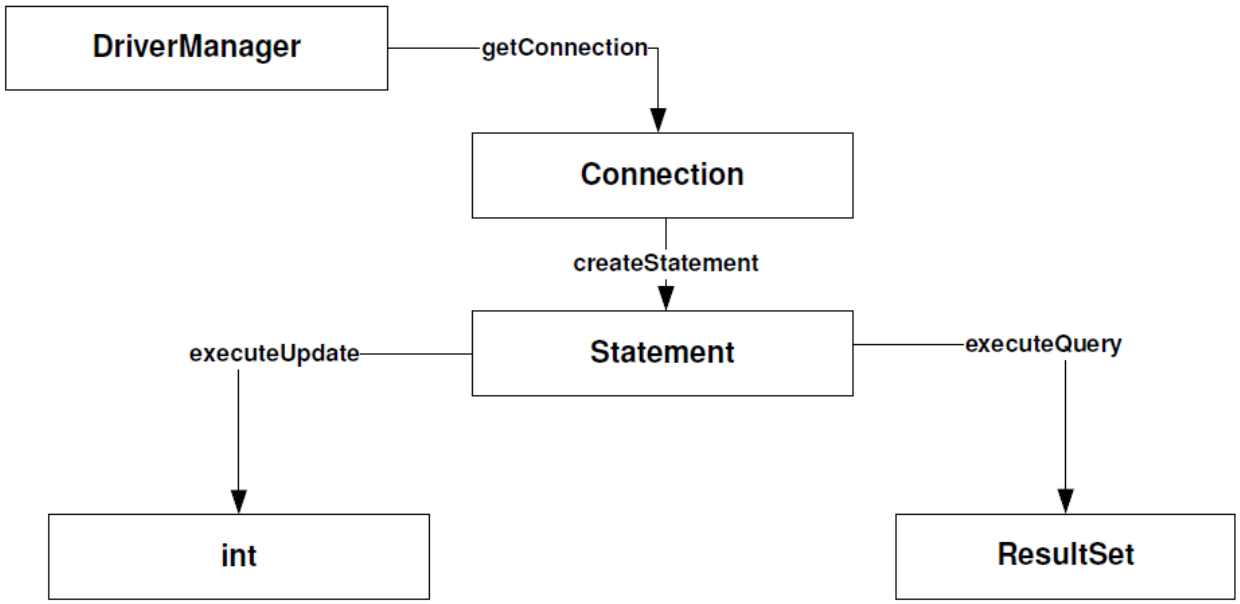
INM21 – Code Snippets

Datenbank

JDBC Treiber

```
// Der Treiber muss in die Laufzeitumgebung geladen werden (z.B. in der main()-Methode,
// bzw. bevor die DriverManager-Klasse aufgerufen wird).

Class.forName("org.postgresql.Driver");
//Beispiel für MySQL: Class.forName("com.mysql.jdbc.Driver");
```

Verbindung




```
String url = "jdbc:postgresql://147.88.100.100:5432/raum_db";
String user = "student";
String pwd = "geheim";

// Aufbau der Verbindung
Connection con = DriverManager.getConnection(url, user, pwd);

// Statement Objekt erstellen
Statement stm = con.createStatement();

// Delete-Query (gleiches Vorgehen für „insert“ „update“)
int anz = 0;
String delQuery =
"DELETE FROM tbl_raum WHERE id_raum=2";
anz = stm.executeUpdate(delQuery);
```

```
// Select-Query
String query = "SELECT * FROM tbl_raum";
ResultSet rs = stm.executeQuery(query);

String str = null;
while(rs.next()){
    str = "Raum: " + rs.getString("bezeichnung");
    str += ", Anz. Plaetze: " + rs.getInt("anz_plaetze");
    System.out.println(str);
}
```

Methoden für ResultSet

```
public int getInt (int columnIndex) throws SQLException
public int getInt (String columnName) throws SQLException
public long getLong (int columnIndex) throws SQLException
public long getLong (String columnName) throws SQLException
public String getString (int columnIndex) throws SQLException
public String getString (String columnName) throws SQLException
public Object getObject (int columnIndex) throws SQLException
public Object getObject (String columnName) throws SQLException

public void beforeFirst () throws SQLException
public void afterLast () throws SQLException
public boolean absolute (int m) throws SQLException
public boolean previous () throws SQLException
public void first () throws SQLException
public void last () throws SQLException
public boolean isFirst() throws SQLException
public boolean isLast() throws SQLException
public boolean isBeforeFirst() throws SQLException
public boolean isAfterLast() throws SQLException
public int getRow() throws SQLException
```

Transaktionen

```
try {
    connection.setAutoCommit(false);
    statement.executeUpdate("INSERT ... ");
    statement.executeUpdate("UPDATE ... ");
    statement.executeUpdate("DELETE ... ");
    connection.commit();
}
catch (SQLException e) {
    if (connection != null){
```

```
            connection.rollback();
        }
}
```

Property-Datei

```
#=== Datei wird als .properties abgelegt.

#=== DB-URL
jdbc.url=jdbc:mysql://147.88.111.111:3306/raum_db

#=== Treiberklasse
jdbc.drivers=com.mysql.jdbc.Driver

#=== Benutzername
jdbc.user=student

#=== Password
jdbc.password=geheim
```

Aufruf

```
// Properties-Objekt erzeugen
Properties dbProperties = new Properties();

// Klassenloader holen
ClassLoader cLoader = this.getClass().getClassLoader();

// Properties laden
dbProperties.load(cLoader.getResourceAsStream("db.properties");

// Treiber-Klasse auslesen
String driverClass = dbProperties.getProperty("jdbc.drivers");

// Treiber laden
Class.forName(driverClass);

// URL, Benutzername und das Kennwort auslesen
String dbUrl = dbProperties.getProperty("jdbc.url");
String user = dbProperties.getProperty("jdbc.user");
String pwd = dbProperties.getProperty("jdbc.password");

// Verbindung zur Datenbank herstellen
con = DriverManager.getConnection(dbUrl, user, pwd);
...
```

Metadaten

Datenbank

```
DatabaseMetaData dbMetaData = connection.getMetaData();

// Tabellennamen auslesen
ResultSet rSet = dbMetaData.getTables(null, null, null, new String[]{"TABLE"});

// Tabellennamen ausgeben
while(rSet.next()){
    System.out.println("TABLE: " + rSet.getString("TABLE_NAME"));
}
```

ResultSet

```
ResultSetMetaData rsMetaData = rSet.getMetaData();

// Anzahl Spalten auslesen
int anzahlSpalten = rsMetaData.getColumnCount();

// Spaltennamen und SQL-Typ ausgeben
if(rsMetaData.next()) {
    for (int i = 1; i <= rsMetaData.getColumnCount(); i++) {
        System.out.println("COLUMN-NAME: " + rsMetaData.getString(i) + "TYPE: " +
                    rsMetaData.getColumnTypeName());
    }
}
```

OR-Mapping

Java Persistence API (JPA)

EntityManagerFactory

```
public class JpaUtil {
    private static EntityManagerFactory entityManagerFactory = null;

    public static synchronized EntityManagerFactory getEntityManagerFactory() {
        if (entityManagerFactory == null) {
            entityManagerFactory = Persistence.createEntityManagerFactory("BooksPU");
        }
        return entityManagerFactory;
    }
}
```

EntityManager

```
EntityManagerFactor factory = JpaUtil.getEntityManagerFactory();
EntityManager em = factory.createEntityManager();

// Transaktion "starten":
em.getTransaction().begin();

// Objekte erstellen, Speichern …
Adresse adr = new Adresse ( … );
em.persist(adr);

// Änderungen in die Db speichern:
em.getTransaction().commit();

// EntityManager schliessen:
em.close();
```

Methoden für EntityManager

```
void persist (Object entity)
<T> T find(Class<T>, Object primaryKey)
<T> T merge (T entity)
void remove (Object entity)
...
```

Models

Verleger Model

```
@Entity
//@Inheritance(strategy= InheritanceType.JOINED) bei Vererbung
public class Verleger implements Serializable{
    @Id
    @GeneratedValue
    private Integer id;
    private String name;
    @OneToMany (mappedBy="verleger")
    private List<Buch> buchListe;

// Konstruktoren und Methoden …
}
```

Buch Model

```
@Entity
public class Buch implements Serializable {
    @Id
    @GeneratedValue
    private Integer id;
    private String titel;
    private String isbn;
    @ManyToOne
    private Verleger verleger;
// Weitere Attribute, Konstruktoren und Methoden …
}
```

Ausführung

```
…
em.getTransaction().beginn();
Verleger verleger = em.find(Verleger.class, 3);
Buch buch = em.find(Buch.class, 33);
buch.setVerleger(verleger); // 1. Verleger in Buch setzen
verleger.getBuchListe().add(buch); // 2. Buch zum Verleger hinzufügen
em.getTransaction().commit();
…
```

Weitere Annotationen

```
@OneToOne
```

```
@OneToMany
@ManyToOne
@ManyToMany
// (mappedBy="projects") für bidirektionale Verknüpfungen
// (cascade=CascadeType.ALL) für kaskierende Aktionen

@Column(name="Nachname")
@JoinTable(name="student_lerngruppe")
@JoinColumn(name="lerngruppe_id")
…
```

Abfragen

```
EntityManager em = JpaUtil.getEntityManagerFactory().createEnitityManager();
```

Query

```
Query q = em.createQuery("SELECT p FROM Person p");
Person p = (Person) q.getSingleResult(); // muss gecastet warden
List<Person> pList = (List<Person>) q.getResultList(); muss gecastet warden
```

TypedQuery

```
TypedQuery<Person> tQuery = em.createQuery("SELECT p FROM Person p", Person.class);
Person p = tQuery.getSingleResult();
List<Person> pList = tQuery.getResultList();
for (Person p: pList){
    System.out.println(p.getName() + " " + p.getVorname());
}
```

NativeQuery

```
String sql = "SELECT * FROM adresse WHERE plz > 6000";
Query q = em.createNativeQuery(sql, Adresse.class);
List<Adresse> adrList = q.getResultList();
…
```

Parameter binding

```
TypedQuery<Person> q = em.createTypedQuery("SELECT p FROM Person p where p.name=?1 AND
p.vorname=?2");
q.setParameter(1, "Pechvogel");
q.setParameter(2, "Hansli");
```

Java Persistence Query Language (JPQL)

SELECT

```
SELECT p FROM Person p
SELECT p, a FROM Person p, Adresse a
```

WHERE

```
SELECT p FROM Person p WHERE p.vorname= 'Hansli'
SELECT p FROM Person p WHERE p.name='Meier' AND p.vorname LIKE '%and'
SELECT a FROM Adresse a WHERE a.plz > 10000 OR ort LIKE '%ns'
SELECT a FROM Adresse a WHERE a.plz between '6000' AND '7000'
SELECT a FROM Adresse a WHERE a.plz IN ('6000', '6010', '6030', '6048')
```

ORDER BY

```
SELECT p FROM Person p WHERE p.name='Meier' ORDER BY p.vorname DESC
SELECT a FROM Adresse a ORDER BY a.plz ASC
```

JOIN

```
// INNER JOIN
SELECT p, a FROM Person p INNER JOIN p.adresse a

// LEFT OUTER JOIN
SELECT p, a FROM Person p LEFT JOIN p.adresse a
```

Sockets

TCP-Sockets

Server Code

```
// ServerSocket erzeugen
ServerSocket server = new ServerSocket(port);

// ClientSocket holen, wenn eine Verbindung gewuenscht wird
Socket client = server.accept();

// Informationen ueber den Client ausgeben
String hostName = client.getInetAddress().getHostName();
int p = client.getPort();
System.out.println("Verbindung mit: " + hostName + ", Port: " + p + "\n");

// InputStream vom Client holen
InputStream is = client.getInputStream();

// vom Client zugestellten Daten ausgeben
int c = 0;
while ((c = is.read()) != -1){
    System.out.print((char) c);
}
```

Client Code

```
// Socket erzeugen und Verbindung zum Server aufbauen
Socket socket = new Socket("localhost", 10001);

// Benachrichtigung
System.out.println("Verbindung mit Server hergestellt!");

// OutputStream vom Socket holen
OutputStream os = socket.getOutputStream();

// Meldung dem Server senden
String msg = "Das ist eine Test-Meldung!";
os.write(msg.getBytes());

// Verbindung schliessen
socket.close();
```

UDP-Sockets

Server Code

```
DatagramSocket socket = null;
DatagramPacket req = null, res = null;
byte[] buf = new byte[508];

// UDP-Socket erzeugen
socket = new DatagramSocket(9001);

// Request-DatagramPacket erzeugen
req = new DatagramPacket(buf, buf.length);
while(true){
    // Request empfangen (entgegennehmen)
    socket.receive(req);

    // Response erzeugen
    res = new DatagramPacket(req.getData(), req.getLength(),
    req.getAddress(), req.getPort());

    // Response senden
    socket.send(res);
}
```

Client Code

```
String msg = "Das ist eine Test-Meldung!";
byte [] resBuf = new byte[508];
int port = 9001;
InetAddress server = InetAddress.getByName("localhost");

// Socket erzeugen
DatagramSocket socket = new DatagramSocket();

// Request erzeugen
DatagramPacket req = new DatagramPacket(msg.getBytes(), msg.length(),
server, port);

// Request senden
socket.send(req);
DatagramPacket res = new DatagramPacket(resBuf, resBuf.length);

// Response empfangen
socket.receive(res);
System.out.println("Antwort: " + new String(res.getData()));
```

Multicast-Sockets

Sender Code

```
String msg = "Diese Meldung geht an alle Gruppenmitglieder!";
MulticastSocket s = null;
DatagramPacket mOut = null;
String mcIp= "230.2.2.2";
InetAddress group = InetAddress.getByName(mcIp);

// Socker erzeugen
s = new MulticastSocket(4004);

// Der Gruppe beitretten
s.joinGroup(group);

// Nachricht erzeugen
mOut = new DatagramPacket(msg.getBytes(), msg.length(), group, 4004);

// Nachricht senden
s.send(mOut);

// Gruppe verlassen
s.leaveGroup(group);
if (s != null) s.close();
```

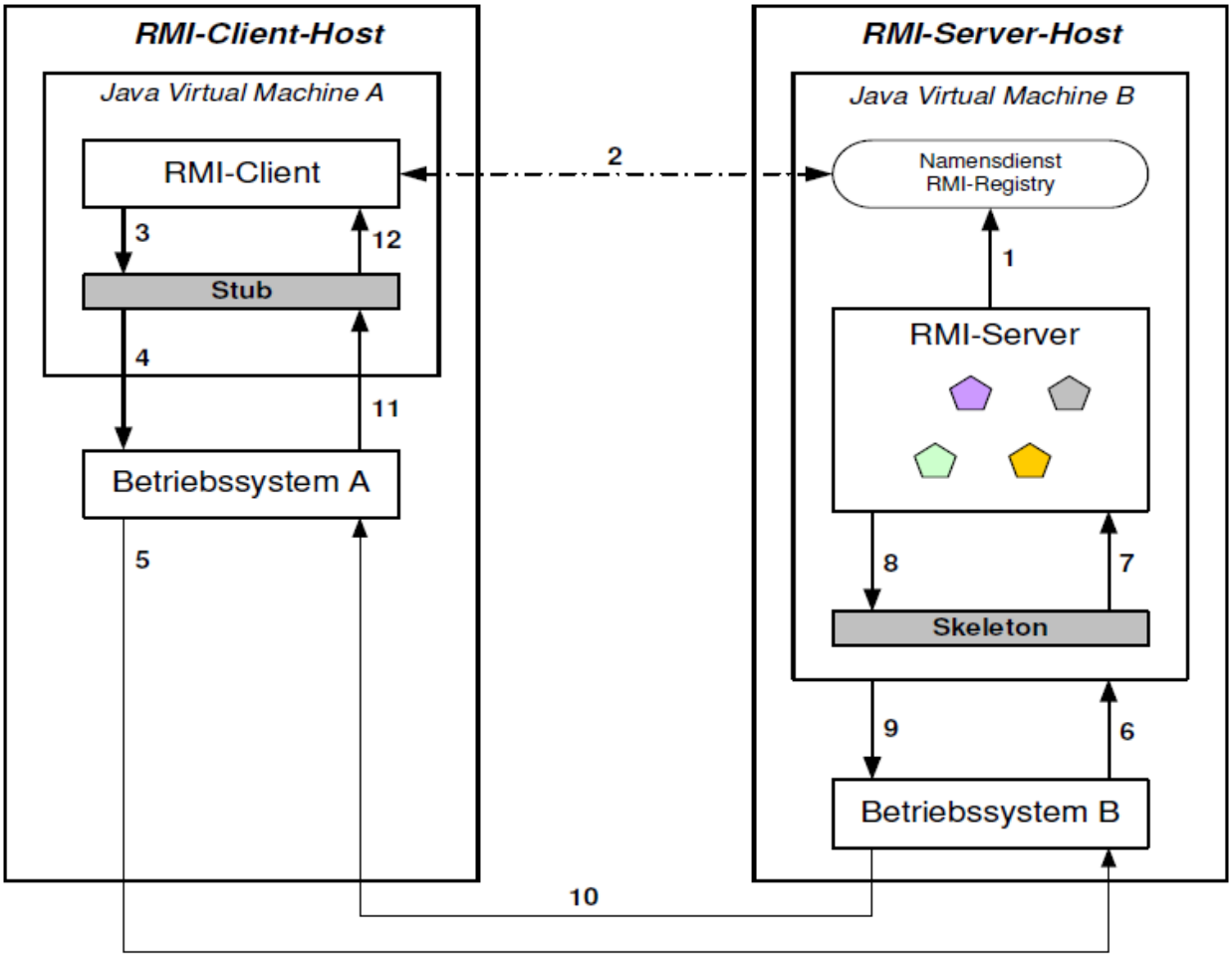
Receiver Code

```
MulticastSocket s = null;
DatagramPacket mIn = null;
String mcIp= "230.2.2.2";
InetAddress group = InetAddress.getByName(mcIp);

// Socker erzeugen
s = new MulticastSocket(4004);

// Der Gruppe beitretten
s.joinGroup(group);
mIn = new DatagramPacket(buf, buf.length, group, 4004);

// Nachricht empfangen
s.receive(mIn);

// Nachricht ausgeben
System.out.println("Empfangen: " + new String(mIn.getData()));

// Gruppe verlassen
```

```
s.leaveGroup(group);
if (s != null) s.close();
```

RPC

RMI



Interface

```
import java.rmi.*;

public interface Adder extends Remote {
    int add(int x, int y) throws RemoteException;
}
```

Implementierung

```
import java.rmi.*;
import java.rmi.server.*;

public class AdderImpl extends UnicastRemoteObject implements Adder {
```

```
    public AdderImpl() throws RemoteException

    public int add(int x, int y) throws RemoteException {
        return x + y;
    }
}
```

Security Policy Datei

```
// Datei im Root-Verzeichnis ablegen als adder.policy
grant {
    permission java.security.AllPermission;
};
```

Server Code

```
public static void main(String[] args){
    try {
        // Entferntes Objekt erzeugen
        Adder adder = new AdderImpl();

        // Registry-Instanz erzeugen bzw. starten
        Registry reg = LocateRegistry.createRegistry(port);

        // Entferntes Objekt beim Namensdienst registrieren
        if (reg != null) {
            reg.rebind("AdderObjekt", adder);
            System.out.print("Adder bound!");
        }

        // Ausgabe - Server bereit
        System.out.println("Adder bound");
    } catch (RemoteException re){
        re.printStackTrace();
    } catch (MalformedURLException me){
        me.printStackTrace();
    }
}
```

Client Code

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

        // policy-Datei angeben
        System.setProperty("java.security.policy", "adder.policy");
```

```
        // SecurityManager installieren
        System.setSecurityManager(new RMISecurityManager());

        // URL definieren
        String url = "rmi://196.168.1.25:1099/AdderObjekt";

        // Referenz auf das entfernte Objekt holen
        Adder adderObj = (Adder) Naming.lookup(url);

        // Methode 'add' des entfernten Objekts aufrufen
        int sum = adderObj .add(13, 37);

        // Ergebnis ausgeben
        System.out.println("13 + 37 = " + sum);
    }
}
```

WebService

JAX-WS

Interface

```
package time.model;
import javax.jws.*;
import java.util.List;

@WebService
public interface Time {
    @WebMethod
    long getCurrentTime(@WebParam(name = "cityName") String cityName);
    @WebMethod
    List<String> getAvailableCityNames() throws Exception;
}
```

Implementierung

```
package time.business;
import javax.jws.WebService;
import time.model.Time;

@WebService(endpointInterface = "time.model.Time")
public class TimeImpl implements Time {
    public long getCurrentTime(String cityName) {
    // Implementierung ...
    }

    public List<String> getAvailableCityNames() throws Exception {
    // Implementierung ...
    }
}
```

Publisher

```
package time;

public class Publisher {
    public static void main(String[] args) {

        // Service-Objekt erstellen
        Time service = new TimeImpl();
```

```
        // URI definieren
        String uri= "http://localhost:9090/time";

        // WebService publizieren
        Endpoint ePoint = Endpoint.publish(uri,service);

        // Dialog zum Beenden des WebServices anzeigen
        JOptionPane.showMessageDialog(null, "Server beenden");

        // WebService-Ausführung beenden
        ePoint.stop();
    }
}
```

Client-Artefakte generieren

```
…> wsimport -keep http://localhost:9090/time?wsdl
```

Client

```
package time.client;

import java.util.*;
import time.business.Time;
import time.business.TimeImpl;

public class Client {
    public static void main(String[] args) {

        // Service kreieren
        TimeImpl service = new TimeImpl ();

        // Proxy kreieren (Client-Stub)
        Time proxy = service.getTimeImplPort();

        // Aktuelle Zeit in Wolgograd abfragen
        long timeInMillis = proxy.getCurrentTime("Wolgograd");
        Date d = new Date(timeInMillis);

        // Ausgabe
        System.out.println("Zeit hier: " + new Date());
        System.out.println("Zeit in Wolgograd: " + d);
    }
}
```

MultiThreading

Thread Klasse

Vererbung

```
import java.text.SimpleDateFormat;
import java.util.Date;

public class Uhr extends Thread {
    public void run() {
        SimpleDateFormat sdf = new SimpleDateFormat("hh:mm:ss:SSS");
        while (true) {
            try {
                Date d = new Date();
                System.out.println("Zeit: " + sdf.format(d));
                Thread.sleep(1000);
            } catch (InterruptedException e) {
                // Ausnahmebehandlung ...
            }
        }
    }
}
```

Ausführen

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

        // Klasse Uhr instanzieren
        Uhr timeThread = new Uhr();

        // Thread-Ausfuehrung starten
        timeThread.start();
    }
}
```

Runnable Interface

Superklasse ZeitHandler

```
import java.text.SimpleDateFormat;
```

```
public class ZeitHandler {
    protected SimpleDateFormat sdf = null;

    public ZeitHandler() {

        // SimpleDateFormat erstellen
        sdf = new SimpleDateFormat("dd.MM.yyyy 'at' hh:mm:ss");
    }
}
```

Unterklasse Uhr

```
public class Uhr extends ZeitHandler implements Runnable {
    public void run() {
        while (true) {
            try {
                String dateAsString = sdf.format(new java.util.Date());
                System.out.println("Date & Time: " + dateAsString);
                Thread.sleep(1000);
            } catch (InterruptedException e) {
                // Ausnahmebehandlung …
            }
        }
    }
}
```

Ausführung

```
public class Main {
    public static void main(String[] argv) {

        // Ein Runnable-Objekt erzeugen
        Uhr runnableObj = new Uhr();

        // Einen Thread erzeugen, wobei dem Konstruktor als Parameter ein
        // Runnable-Objekt uebergeben wird
        Thread timeObj = new Thread(runnableObj);

        // Ausfuehrung starten
        timeObj.start();
    }
}
```

Interrupt Methode

Aufruf

```
public class Main {
    public static void main(String[] args) {
        // Klasse Uhr instanzieren
        Uhr timeThread = new Uhr();

        // Thread-Ausfuehrung starten
        timeThread.start();
        try {
            // Haupt-Thread 30 Sekunden schlaffen lassen
            Thread.sleep(30000);

            // Den timeThread stoppen
            timeThread.interrupt();
        } catch (InterruptedException e) {
            // Ausnahmebehandlung …
        }
    }
}
```

Verwertung

```
import java.text.SimpleDateFormat;

public class Uhr extends Thread {
    public void run() {
        SimpleDateFormat sdf = new SimpleDateFormat("hh:mm:ss:SSS");
        try {
            while (!isInterrupted()) {
                String dateAsString = sdf.format(new java.util.Date());
                System.out.println("Zeit: " + dateAsString);
                Thread.sleep(1000);
            }
        } catch (InterruptedException e) {
            // Ausnahme behandeln falls nötig bzw. sinnvoll ...
        }
    }
}
```

Executor Framework

Implementierung

```
public class PrimeFactorsPrinter implements Runnable {
    private long number;

    public PrimeFactorsPrinter(long number) {
        this.number = number;
    }

    @Override
    public void run() {
        long currentValue = number;
        List<Long> prims = new ArrayList<>();

        for (long i = 2; i <= currentValue; i++) {
            if (currentValue % i == 0) {
                prims.add(i);
                currentValue /= i;
                i = 1;
            }
        }
        show(prims); // Resultat anzeigen (Code hier nicht angegeben)
    }
}
```

Ausführung

```
import java.util.concurrent.*;

public class Main {
    public static void main(String[] args) throws InterruptedException {

         // Anzahl Threads
         final int NUMBER_THREADS = 10;

         // ExecutorService erstellen
         ExecutorService executor = Executors.newFixedThreadPool(NUMBER_THREADS);

         for (long i = 1000000001; i < 1000000050; i++) {
             executor.execute(new PrimeFactorsPrinter(i));
         }

         executor.shutdown();
    }
}
```

Callable und Future

Implementierung

```
public class PrimeFactorsFinder<V> implements Callable<List<Long>> {
    private Long n;

    public PrimeFactorsFinder(Long n) { this.n = n; }

    @Override
    public List<Long> call() throws Exception {
        List<Long> primeFactorsList = findPrimeFactors(n);
        return primeFactorsList;
    }

    private List<Long> findPrimeFactors(long n) {
        List<Long> prims = new ArrayList<>();
        // Implementierung ...
        return prims;
    }
}
```

Ausführung

```
public class Main {
    public static void main(String[] args) {
        long startValue = 100000001;
        long endValue = 100000050;
        ExecutorService executor = Executors.newFixedThreadPool(5);

        for (Long i = startValue; i < endValue; i++) {
            Callable<List<Long>> c = new PrimeFactorsFinder<Long>(i);
            Future<List<Long>> prims = executor.submit(c);
            show(i, prims);
        }
        executor.shutdown();
    }

    private static void show(long number, Future<List<Long>> future) {
        try {
            List<Long> prims = future.get();
            if (prims.size() == 1) {
                 System.out.println(number + ": PRIMZAHL");
            } else {
                 String str = number + " = " + prims.get(0).longValue();
```

```
                for (int i = 1; i < prims.size(); i++) {
                    str += " * " + prims.get(i).longValue();
                }
                System.out.println(str);
            }
        } catch (InterruptedException | ExecutionException e) {
        // Ausnahme behandeln …
        }
    }
}
```