Java Beans sind kompliziert? Nicht die Bohne!

Knut Herrmann

NOTES & DOMINO ENTWICKLERCAMP

Knut Herrmann

- Diplom-Informatiker, Technische Universität Dresden
- seit 1992 Lotus Notes Anwendungsentwicklung
- in letzten Jahren mit Schwerpunkt XPages und Java
- Mitgründer und -Gesellschafter der Leonso GmbH

- Kontakt:
  - E-Mail: knut.herrmann@leonso.de
  - Twitter: @KnutHerrmann
  - Skype: knut.herrmann

Java Beans sind kompliziert?

Warum Java?

Komprimierte Einführung in Java

Agenda

Java im Domino Designer programmieren

Was macht eine Java Bean aus?

Notes-Daten lesen und schreiben

Java Beans in XPages einbinden

Fehler richtig behandeln

Java Beans einfach debuggen

Fremde Java APIs einbinden

Nächste Etappen

Nicht die Bohne!

Warum Java?

- In Java wurde schon (fast) alles programmiert
  - vieles frei verfügbar
  - warum das Rad neu erfinden?
- Java Klassen und APIs in XPages einfach verwendbar
- XPages basiert auf Java
  - aus XPages werden Java-Klassen generiert
  - lohnender Blick "unter die Haube"
- eigener Code in Java unabhängiger und zukunftssicherer
  - nicht an eine Plattform gebunden (z.B. XPages, Notes-Daten)
  - Trennung von Geschäftslogik, Bedienoberfläche und Datenhaltung

- Java ist eine der meistgenutzten Programmiersprachen

- Wie beginnen?
  - Java SE – Standard Edition
    - Java Grundfunktionalität
    - Basis-Klassen wie java.lang, java.util, java.io
  - (erst später Java EE mit JSF, JSP, …)
  - Buchempfehlung
    - "Java ist auch eine Insel" von Christian Ullenboom
    - frei zum Downloaden mit Übungsbeispielen
      openbook.galileocomputing.de/javainsel/

- unsere erste Java Klasse

```
package de.leonso.entwcamp;

public class Person {
    private String nachname = "";
    private String vorname = "";
    private boolean kunde = false;

    public void setNachname(String nachname) {
        this.nachname = nachname;
    }

    public String getNachname() {
        return nachname;
    }
    ...

    public String getName() {
        return vorname + " " + nachname;
    }
}
```

Komprimierte Einführung in Java

- Konventionen
  - Klassen sollen (weltweit) eindeutig sein → durch `package`
    "umgekehrte" eigene Domäne + eigene Gliederung
  - Klassennamen beginnen immer mit Großbuchstaben – und nur diese!
    - "CamelCase" für zusammengesetzte Wörter
    - Substantiv verwenden "Person", "Auftrag"
  - Variablen und Methoden
    - "camelCase" und erster Buchstabe immer klein
    - Methoden:
      mit Verb beginnen "getPerson", "berechneNeuenWert"
  - Konstanten – alles in Großbuchstaben
    - Worttrennung durch "_" MAXIMALE_ANZAHL_VERSUCHE

Komprimierte Einführung in Java

- Konventionen
  - geschweifte Klammern
    vom Ende der ersten Zeile `{`
    bis unter die letzte Zeile
    `}`

"ägyptische Klammerung"

  - maximal eine Anweisung pro Zeile `;`
  - Variablen unmittelbar vor erster Verwendung deklarieren
    - nicht wie in LotusScript alle Dim am Anfang
  - initiale Wertzuweisung gleich bei Variablendeklaration

  - siehe auch Code Conventions for the Java Programming Language

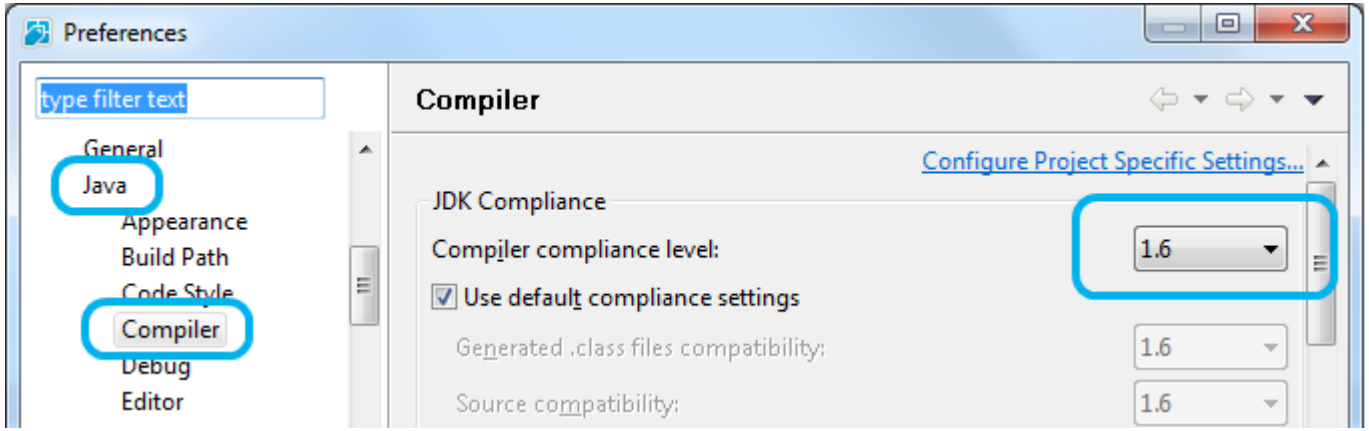
Komprimierte Einführung in Java

- Konzepte & Begriffe
  - Klasse
  - Objekt
  - Vererbung
  - Überschreiben
  - Überladen
  - Sichtbarkeit
  - Schnittstellen - Interface

  - … wer schon mit LotusScript Klassen gearbeitet hat, hat's einfacher ☺

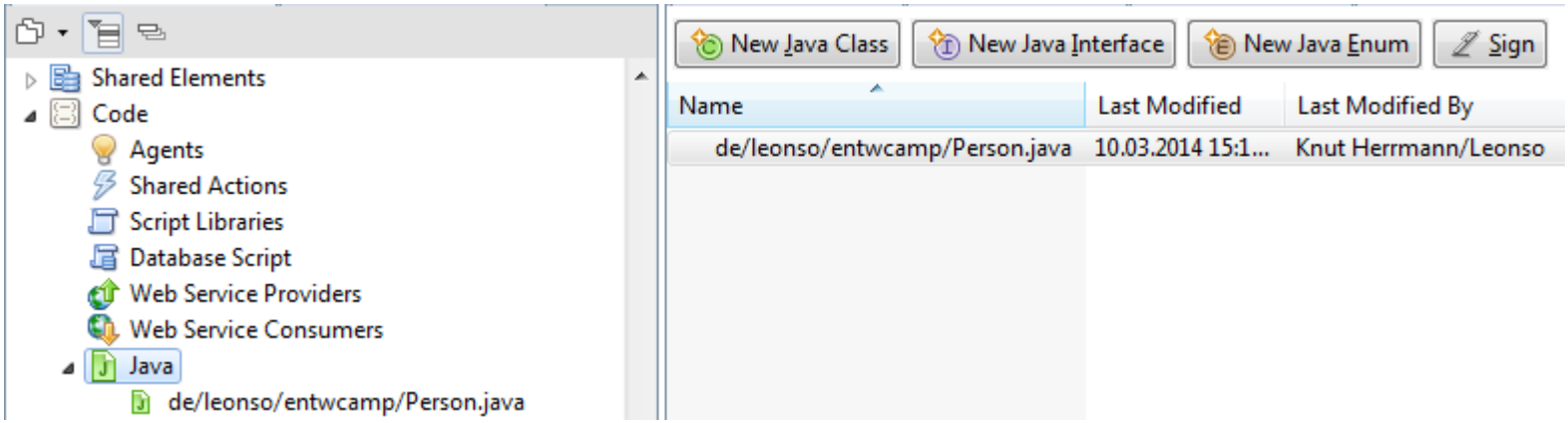
Java im Domino Designer programmieren

- Domino Designer ist angepasste Eclipse Version
- Java Versionen
  - J2SE 5.0 (1.5.0) Notes 8
  - JSE 6.0 Notes 8.5 und 9
  - wichtig: nicht alle Java APIs unterstützen diese älteren Versionen aktuelle Version ist JSE 7.0
  - in File > Preferences die Version 1.6 einstellen (Standard 1.5)



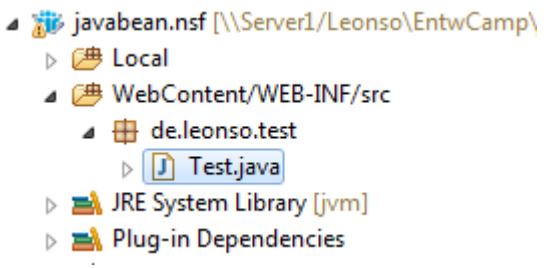
Java im Domino Designer programmieren

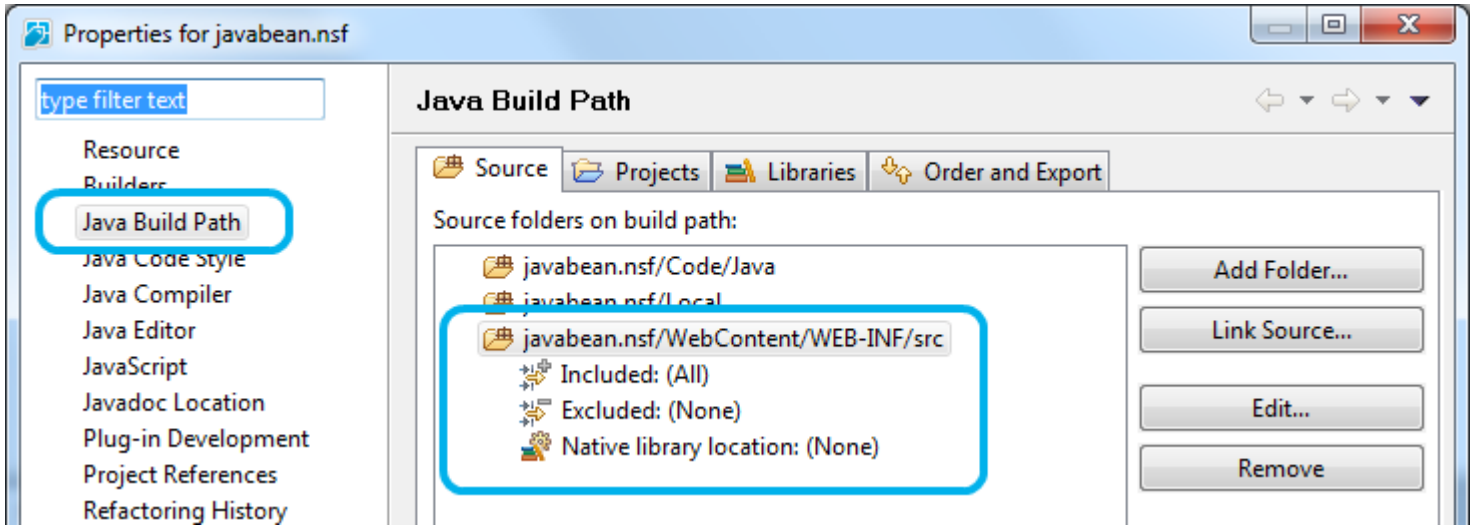
- Java-Klassen
  - in Code > Java erstellen



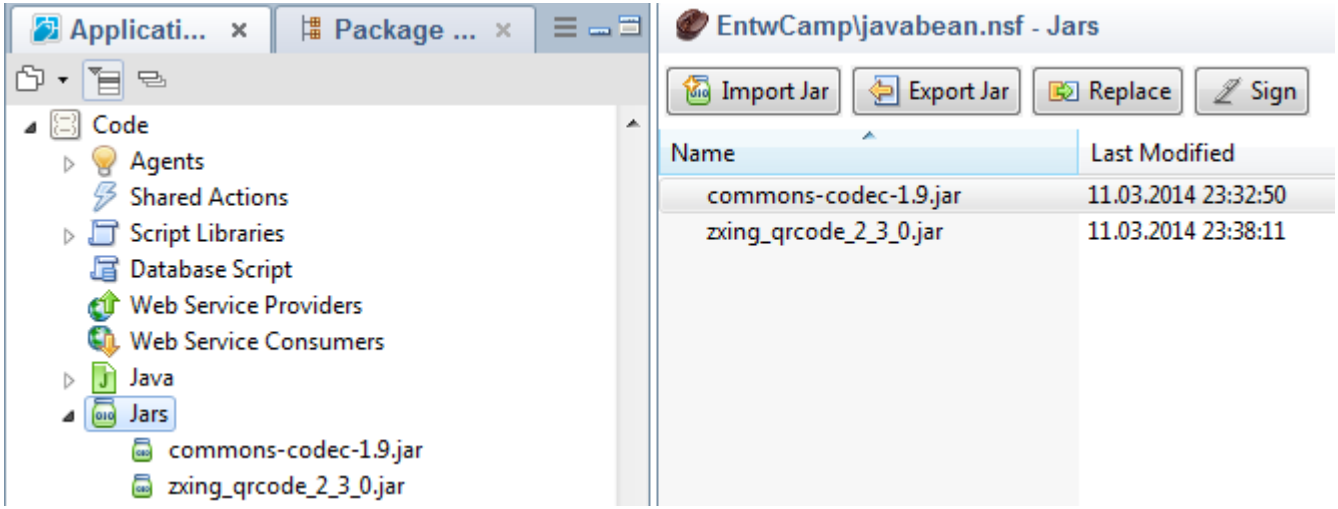
  - werden automatisch in XPages gefunden, da sie im Klassensuchpfad (classpath) liegen

- Java-Klassen
  - alternativ im Package Explorer Verzeichnis anlegen und im Java Build Path als Source definieren
    - Beispielpfad: WebContent/WEB-INF/src



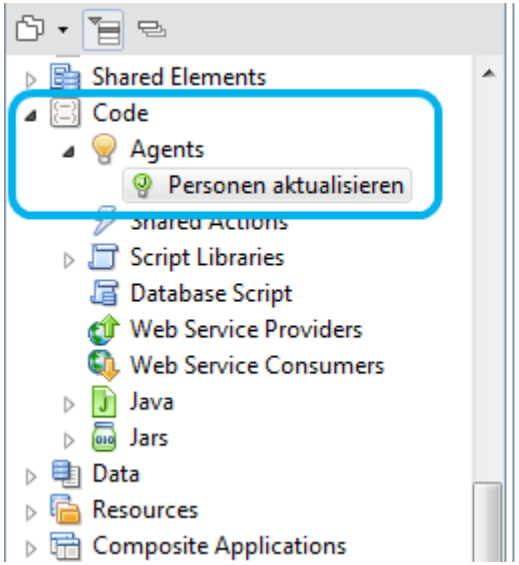


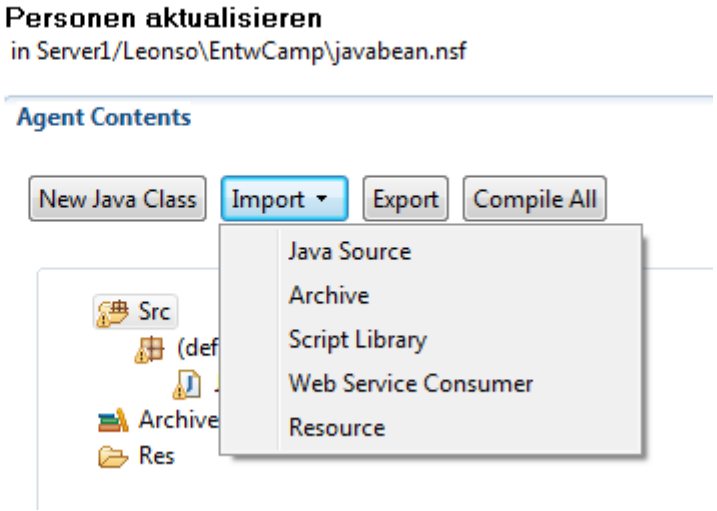
- Java-Archive .jar
  - in Code > Jars (ab Notes 9)



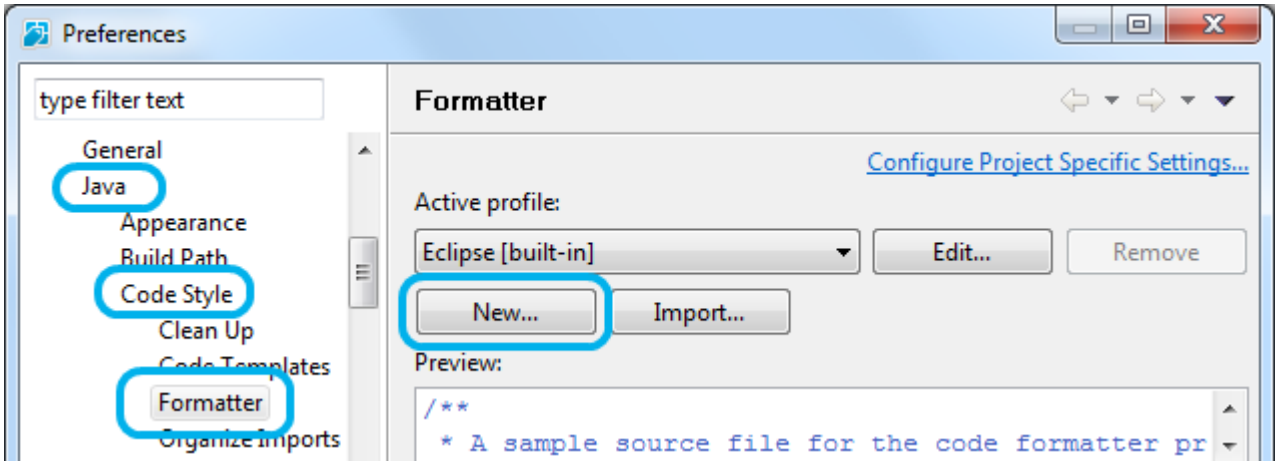
  - alternativ im Package Explorer im Verzeichnis WebContent/WEB-INF/lib
  - sind damit automatisch im Klassensuchpfad

- Java-Agenten
  - leben in ihrer eigenen "Java-Welt"
    - enthalten alle verwendeten Klassen, Java Archive und Ressourcen
    - können Java Bibliotheken (Script Libraries) einbinden
    - haben kein Zugriff auf Code > Java, Code > Jars und WebContent

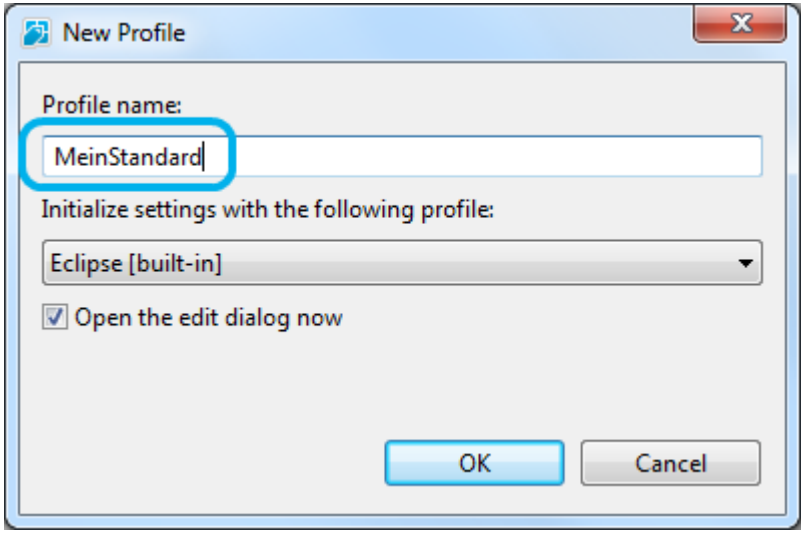




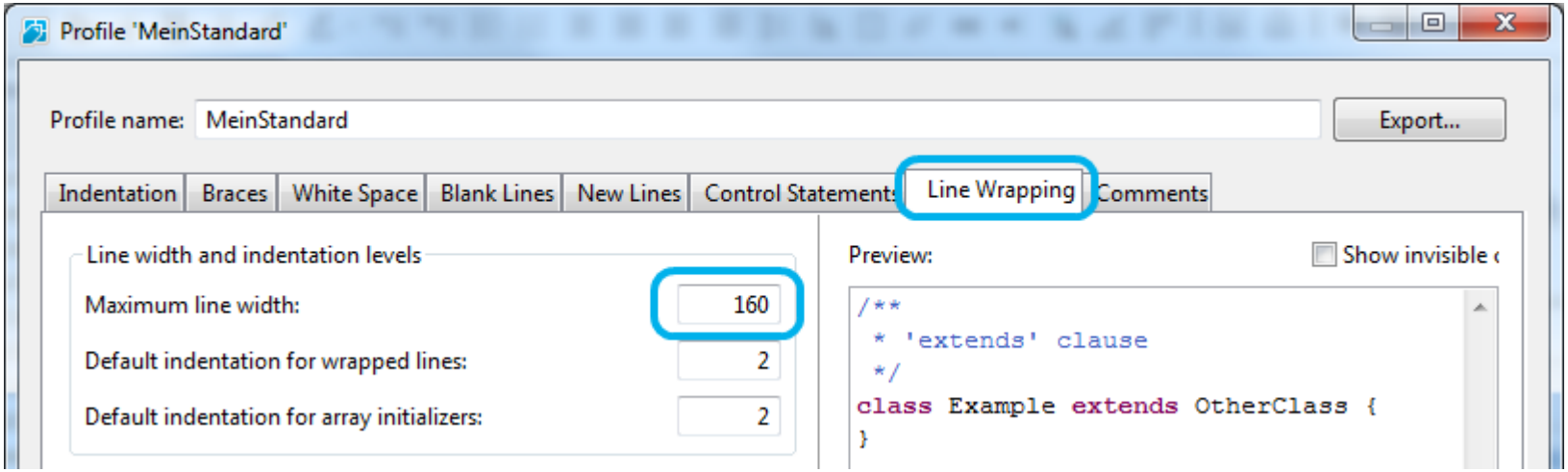
- Domino Designer – Java Code Formatierung anpassen
  - Code kann automatisch formatiert werden
  - alle Projektmitarbeiter sollten gleiche Formatierungs-Einstellung benutzen(!)
    - wichtig bei Nutzung einer Versionsverwaltung (Version Control System)
  - Empfehlung: Anzahl Zeichen pro Zeilen vergrößern (Standard ist 80)
    - Designer > File > Preferences > Java > Code Style > Formatter
      Standardprofil kann nicht verändert werden, deshalb neues Profil erstellen



Java im Domino Designer programmieren



- Profilnamen vergeben
- Tabulator "Line Wrapping"
- Anzahl Zeichen pro Zeilen auf zum Beispiel 160 setzen



Java im Domino Designer programmieren

- Der Java Editor "denkt" oft mit und macht einem das Programmierer-Leben leicht
  - Tastenkombinationen
    - Strg + Leertaste — Autovervollständigung
    - Strg + 1 — Vorschläge zur Vervollständigung oder Korrektur
    - Strg + Shift + C — Zeilen kommentieren / dekommentieren
    - Strg + Shift + F — Code neu formatieren
    - Strg + Shift + O — Importzeilen neu organisieren (entfernen/hinzufügen)
    - Alt+Shift+R — Klasse, Methode oder Variable umbenennen
  - Kontextmenü
    - Source/…
    - Refactor/…
    - Surround With/…

Demo

Was macht eine Java Bean aus?

- Java Bean ist "normale" Java Klasse ohne große Abhängigkeiten
  = Plain Old Java Object (POJO)

- sie genügt bestimmten Konventionen:
  - **1**. öffentlicher Standard-Konstruktor ohne Parameter
  - **2**. öffentliche Getter und Setter für das Lesen und Schreiben von Werten
  - **3**. implementiert "Serializable"

- durch diese Standardisierung können andere Programme
  die Java Beans einheitlich instanziieren und nutzen,
  z.B. XPages ☺

Was macht eine Java Bean aus?

- **1**. öffentlicher Standard-Konstruktor ohne Parameter

```
public class Person {
    …
    public Person() {
        // Werte initialisieren
    }
    …
}
```

  - kann weggelassen werden, wenn es nichts zu initialisieren und keine anderen Konstruktoren gibt

Was macht eine Java Bean aus?

- **2**. öffentliche Getter und Setter für das Lesen und Schreiben von Werten
  - Variablen/Felder **`private`**
  - Zugriff von außen über **`public`**:
    - getFeld()
    - setFeld(typ feld)
    - isFeld() für boolean

```
public class Person {
    private String nachname = "";
    private boolean kunde = false;

    public void setNachname(String nachname) {
        this.nachname = nachname;
    }

    public String getNachname() {
        return nachname;
    }

    public void setKunde(boolean kunde) {
        this.kunde = kunde;
    }

    public boolean isKunde() {
        return kunde;
    }
```

Was macht eine Java Bean aus?

- **3**. implementiert "Serializable"

```
import java.io.Serializable;

public class Person implements Serializable {
    private static final long serialVersionUID = 1L;
```

  - sorgt dafür, dass ein Objekt dieser Klasse (auf Platte) gespeichert und später wieder original hergestellt werden kann

  - übrigens, Notes Objekte sind nicht serializable ☹

Was macht eine Java Bean aus?

- Ist eine Java Bean eine Managed Bean?
  - nein, sie kann aber eine werden – dazu später

Notes-Daten lesen und schreiben

- Um Domino Klassen nutzen zu können, brauchen wir eine gültige **lotus.domino.Session**
  - Java Beans im XPages Umfeld können sich Session aus FacesContext ermitteln

```
public Session getSession() {
        FacesContext context = FacesContext.getCurrentInstance();
        return (Session) context.getApplication().getVariableResolver()
                .resolveVariable(context, "session");
}
```

  - alternativ kann SSJS die globale Variable "session" der Java Bean Methode als Parameter mitgeben
  - (Java Agenten liefern Session "frei Haus")

Notes-Daten lesen und schreiben

- Ab **Session** können wir (fast) so programmieren wie in LotusScript
  - Domino Klassen heißen genauso, nur ohne "Notes" am Anfang

```
import lotus.domino.Database;
import lotus.domino.Document;
import …

public class PersonNotes extends Person {
    private static final long serialVersionUID = 1L;
    private static final String DATENBANK_PFAD = "EntwCamp/Testdaten.nsf";
    private static final String ANSICHT_KUNDENLISTE = "Kundenliste";

    public void readPerson(String nachname) throws NotesException {
        Session session = getSession();
        Database db = session.getDatabase(session.getServerName(),
                                          DATENBANK_PFAD);
        View view = db.getView(ANSICHT_KUNDENLISTE);
        Document doc = view.getDocumentByKey(nachname);
        setNachname(doc.getItemValueString("Nachname"));
        doc.recycle();
        view.recycle();
        db.recycle();
    }
```

Notes-Daten lesen und schreiben

- Recyceln
  - ist eine Notes-Altlast und für Java-Programmierer eine ~~Last~~ Pflicht
  - jedes Domino Objekt bekommt Handle – die sind begrenzt
  - Objekte sofort recyceln, wenn sie nicht mehr gebraucht werden
  - absolute Pflicht in Schleifen

```
View view = db.getView(ANSICHT_KUNDENLISTE);
Document doc = view.getFirstDocument();
while (doc != null) {
    // mit doc arbeiten
    Document tmpdoc = view.getNextDocument(doc);
    doc.recycle();
    doc = tmpdoc;
}
```

  - … wer keinen Bock auf Recyceln hat: OpenNTF Domino API

Java Beans in XPages einbinden

- in SSJS Java Bean instanziieren und verwenden

```
var person = new de.leonso.entwcamp.PersonNotes();
person.readPerson("Meier");
person.getVorname() + " " + person.getNachname() +
" ist " + (person.isKunde() ? "ein" : "kein") + " Kunde"
```

  - wenn mehrere Klassen eines Package verwendet werden, ist es sinnvoll, das Package zu importieren

```
importPackage(de.leonso.entwcamp);
var person = new PersonNotes();
person.readPerson("Meier");
var abteilung = new Abteilung();
abteilung.setzeZustaendigeAbteilung(person);
person.getVorname() + " " + person.getNachname()
+ " ist " + (person.isKunde() ? "ein" : "kein")
+ " Kunde und wird von der Abteilung "
+ abteilung.getName() + " betreut."
```

Java Beans in XPages einbinden

- Java Bean für eine XPage instanziieren, in viewScope speichern und in XPage-Elementen nutzen

```
<xp:this.beforePageLoad><![CDATA[#{javascript:
      viewScope.person = new de.leonso.entwcamp.PersonNotes();
      viewScope.person.readPerson("Meier")}]]>
</xp:this.beforePageLoad>
<xp:inputText
   id="inputTextNachname"
   value="#{viewScope.person.nachname}" />
<xp:button
   value="Speichern"
   id="button1">
   <xp:eventHandler
      event="onclick"
      submit="true"
      refreshMode="complete"
      immediate="false"
      save="true"
      action="#{javascript:viewScope.person.save()}" />
</xp:button>
```

Java Beans in XPages einbinden

- Java Beans entfalten ihre Eigenschaften in der Expression Language (EL)

```
<xp:inputText
    id="inputTextNachname"
    value="#{viewScope.person.nachname}" />
```

  - mit `person.nachname` erwartet die EL in der Java Bean die Methoden getNachname() und setNachname()
    - sind beide Methoden in der Klasse enthalten, ist das Textfeld bearbeitbar
      - beim Rendern der Seite wird automatisch mit getNachname() der Feldinhalt gesetzt
      - beim Commit wird der eventuell geänderte Wert mit setNachname() in der Java Bean gesetzt
    - gibt es kein setNachname(), wird das Textfeld automatisch im Lesemodus angezeigt
    - die Schreibweise muss genau diesem Muster folgen, insbesondere auch die Groß-/Kleinschreibung

Java Beans in XPages einbinden

- Java Beans und die Expression Language (EL)
  - die EL-Schreibweise kann auch innerhalb eines JavaScript-Blocks verwendet werden
    - das SSJS Bespiel kann damit verkürzt werden

```
var person = new de.leonso.entwcamp.Person();
person.readPerson("Meier");
person.vorname + " " + person.nachname +
" ist " + (person.kunde ? "ein" : "kein") + " Kunde"
```

  - die EL-Version in XPages (< 2.0) erlaubt leider keine runden Klammern
    - damit können Setter nicht als EL innerhalb von JavaScript verwendet werden
      `person.nachname("Müller");` funktioniert NICHT!

Java Beans in XPages einbinden

- Java Bean wird zur **Managed Bean**, wenn wir sie managen lassen
  - Wer ist der Manager? - XPages/JSF
  - eine Managed Bean wird beim ersten Aufruf automatisch instanziiert und bleibt eine spezifizierte Lebensdauer am Leben
    - application-Scope — lebt über gesamte Lebensdauer der Anwendung (nsf) und ist für alle Benutzer gleich
    - session-Scope — lebt für die Dauer einer Sitzung, in der der Benutzer mit der Anwendung verbunden ist
    - view-Scope — lebt in der aktuellen Seite / XPage
    - request-Scope — lebt für die Zeitdauer einer HTTP-Anfrage

Java Beans in XPages einbinden

- eine Managed Bean wird in faces-config.xml definiert

```
<faces-config>
    <managed-bean>
        <managed-bean-name>person</managed-bean-name>
        <managed-bean-class>de.leonso.entwcamp.PersonNotes</managed-bean-class>
        <managed-bean-scope>view</managed-bean-scope>
    </managed-bean>
    …
</faces-config>
```

- und kann im SSJS ohne zu instanziieren einfach verwendet werden

```
person.readPerson("Meier");
person.vorname + " " + person.nachname +
" ist " + (person.kunde ? "ein" : "kein") + " Kunde"
```

Java Beans in XPages einbinden

- unser Beispiel verkürzt sich mit einer Managed Bean auf

```
<xp:this.beforePageLoad><![CDATA[#{javascript:
     person.readPerson(param.name)}]]>
</xp:this.beforePageLoad>
<xp:inputText
   id="inputTextNachname"
   value="#{person.nachname}" />
<xp:button
   value="Speichern"
   id="button1">
   <xp:eventHandler
      event="onclick"
      submit="true"
      refreshMode="complete"
      immediate="false"
      save="true"
      action="#{javascript:person.save()}" />
</xp:button>
```

- …trotzdem sollte nicht jede Java Bean eine Managed Bean werden

Demo

Fehler richtig behandeln

- Fehler (Exceptions) können durch fremde und eigene Klassen "geworfen" werden
  - in der Regel wird damit auf falsche Parameter, unerwartete (System-) Zustände oder Programmierfehler reagiert
  - an der Fehlerstelle wird die Bearbeitung der Methode abgebrochen und der Fehler an die aufrufende Methode weitergereicht
- Fehler können in einem try-catch Block abgefangen werden
  - der Fehler sollte behandelt und beseitigt werden, wenn eine sinnvolle Weiterarbeit möglich ist

```
try {
    // Code der gesichert läuft
} catch (Exception exception) {
    // hier können Fehler behandelt oder beseitigt werden
    // für kurzfristige Fehlersuche kann mit e.printStackTrace();
    // der Fehler auf der Konsole ausgegeben werden
}
```

Fehler richtig behandeln

- der Fehler sollte mit eigenem Fehlerinformationen angereichert und hochgereicht werden, wenn eine Weiterarbeit sinnlos wäre

```
try {
    // Code der gesichert läuft
} catch (YourException exception) {
    throw new MyException("genaue Fehlerbeschreibung", exception);
}
```

- Methoden müssen Fehler, die sie oder untergeordnete Methoden werfen, anzeigen, damit aufrufende Programme darauf reagieren können

```
public void readPerson(String nachname) throws MyException {
```

- optional kann ein finally-Block hinzugefügt werden, der immer, auch nach einem Fehler, ausgeführt wird

```
try {
    // Code der gesichert läuft
} catch (ExceptionKlasse exception) {
    // hier können Fehler behandelt oder beseitigt werden
} finally {
    // wird in jedem Fall ausgeführt
    // hier können Objekte geordnet zurückgesetzt werden
}
```

Fehler richtig behandeln

- Beispiel: try-catch-finally-Block für readPerson()

```
public void readPerson(String nachname) throws Exception {
    Session session = getSession();
    Database db = null;
    View view = null;
    Document doc = null;
    try {
        db = session.getDatabase(session.getServerName(), DATENBANK_PFAD);
        view = db.getView(ANSICHT_KUNDENLISTE);
        doc = view.getDocumentByKey(nachname);
        setNachname(doc.getItemValueString("Nachname"));
    } catch (Exception e) {
        throw new Exception("Dokument konnte nicht gelesen werden!", e);
    } finally {
        doc.recycle();
        view.recycle();
        db.recycle();
    }
}
```

- schlecht: im finally-Block können Fehler auftreten und geworfen werden, die den eigentlichen Fehler überdecken

Fehler richtig behandeln

- der finally-Block müsste korrekt so aussehen,

```
} finally {
    if (doc != null) {
        doc.recycle();
    }
    if (view != null) {
        view.recycle();
    }
    if (db != null) {
        db.recycle();
    }
}
```

  da nicht gesichert, dass alle Objekte zum Zeitpunkt des Fehlers bereits erstellt wurden

- mit Hilfsfunktionen wie dieser
  github.com/OpenNTF/XSnippets/.../xsnippets/Utils.java
  lässt sich das aber elegant verkürzen

```
} finally {
    Utils.recycleObjects(doc, view, db);
}
```

Java Beans einfach debuggen

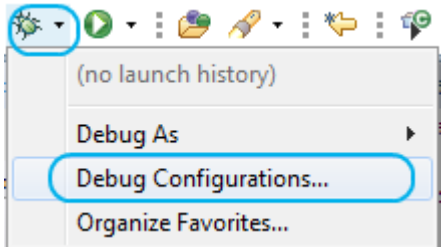
- die schnelle Lösung - System.out.println()

```
System.out.println("getNachname(): " + getNachname());
```

  - entspricht dem SSJS print() – Ausschrift ist auf der Serverkonsole sichtbar
- Ziel: Java Beans zur Laufzeit auf Server richtig debuggen
  - notes.ini vom Server erweitern um

    JavaEnableDebug=1
    JavaDebugOptions=transport=dt_socket,server=y,suspend=n,address=8000
  - nach Server-Neustart sollte auf der Konsole erscheinen:

    NOTES.INI contains the following *DEBUG* parameters:
    JAVADEBUGOPTIONS=transport=dt_socket,server=y,suspend=n,address=8000
    JAVAENABLEDEBUG=1
    Warning: Debug parameters could impact operation or performance.
  - Java Klasse öffnen (!) und Debugger starten

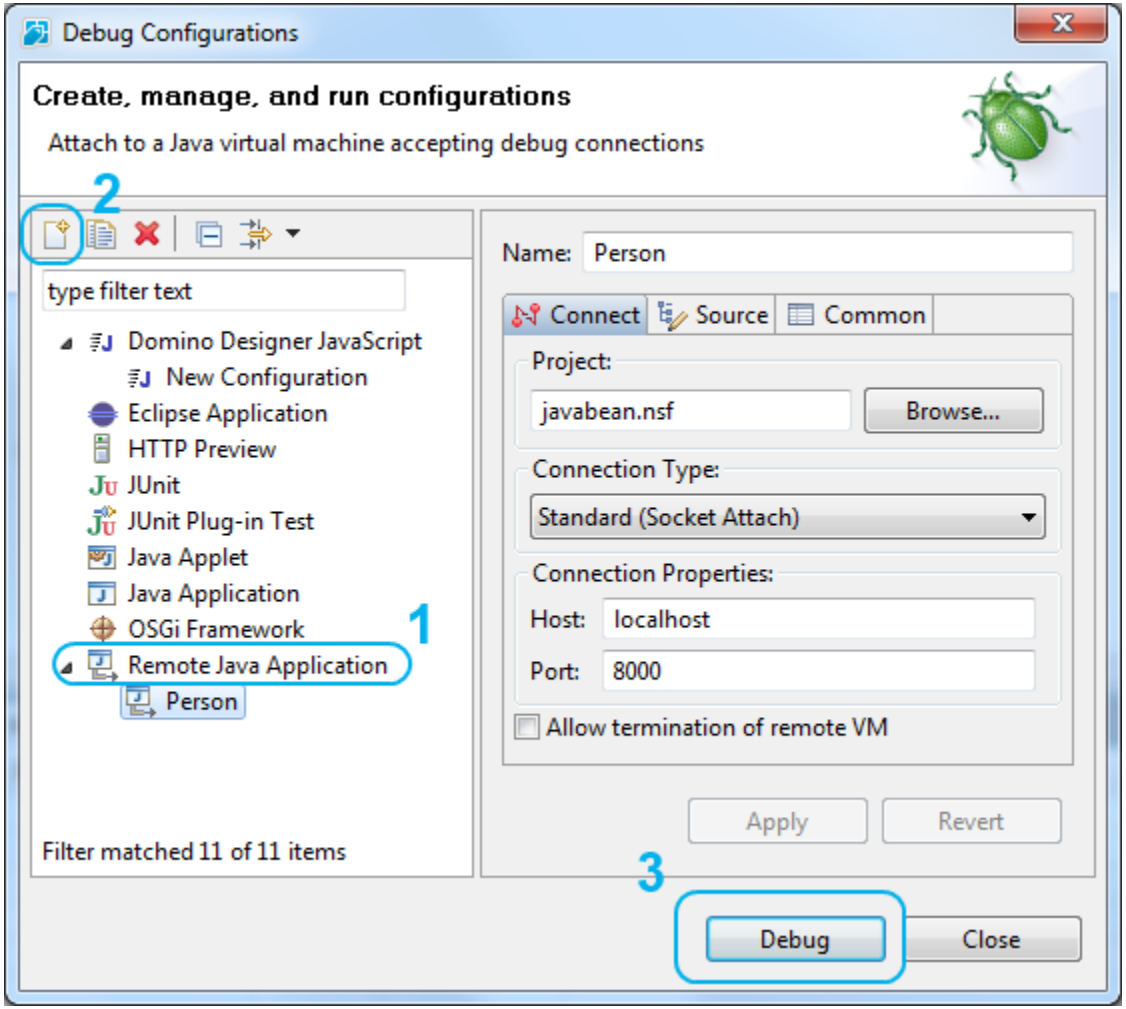


Java Beans einfach debuggen

- 1 - 2 - 3 nacheinander klicken
- im Java Code auf gewünschte Zeilennummer doppelklicken

```
19⊖     public void
20          this.nac
21      }
```

  bis Breakpoint angezeigt wird
- XPages Anwendung mit Java Kode starten
- Frage, ob debugget werden soll, bejahen



Demo

Fremde Java APIs einbinden

- Java APIs werden als .jar-Dateien geliefert
- können in der nsf oder auf dem Server gespeichert werden
  - in nsf
    - >= Notes 9 in Code > Jars
    - < Notes 9 in WebConent > WEB-INF > lib
  - auf Server
    - im Verzeichnis /jvm/lib/ext
- populäre Java APIs
  - Apache Commons – bewährte Erweiterungen zum Standard-Java
  - Apache POI – Excel und Word Dateien erstellen (OpenNTF Projekt)
  - iText – pdf-Dokumente erstellen und bearbeiten - leider nicht mehr frei

Fremde Java APIs einbinden

- unser Fokus:

  QRCode-Generierung auf Basis der Java API ZXing ("Zebra Crossing")

  - github.com/zxing/zxing

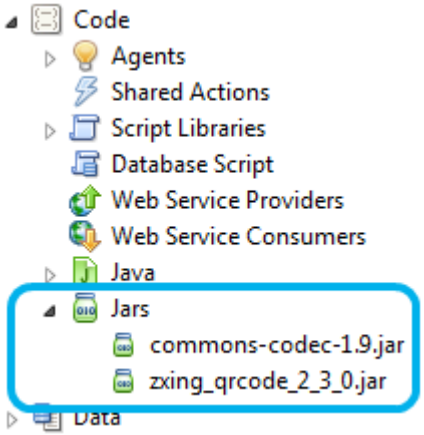
- unser Gesellenstück:
  - eine XPage soll unter Nutzung von Java Beans Personendaten aus einer Notes-Datenbank lesen und diese zusammen mit einem QRCode anzeigen
  - der QRCode wird auf Basis der Java API ZXing erstellt
  - der generierte QRCode soll durch Smartphones eingelesen und direkt als Kontakt gespeichert werden können

Fremde Java APIs einbinden

- 1. Schritt - Java API ZXing einrichten
  - ZXing downloaden – z.B. Version 2.3.0 github.com/zxing/zxing/releases/tag/zxing-2.3.0
  - leider gibt es keine fertige .jar Datei
  - ZXing hat einen riesigen Funktionsumfang den wir größtenteils nicht brauchen wie Barcode lesen oder Smartphoneunterstützung
  - durch ausprobieren habe ich die notwendigen Java-Klassen zusammengesammelt und in einer zxing_qrcode_2_3_0.jar gespeichert
  - Vorteil: es genügen ganze 53K für die QRCode-Generierung
  - zxing_qrcode_2_3_0.jar wird unter Code > Jars gespeichert

Fremde Java APIs einbinden

- 2. Schritt – Base64-Konvertierung bereitstellen
  - der generiert QRCode soll nicht erst irgendwo auf dem Server oder in einem Notes-Dokument gespeichert werden, sondern direkt als Base64-Stream an den Browser gesendet werden
  - für die Konvertierung nutzen wir aus der Apache Commons Codec API commons.apache.org/proper/commons-codec/ die Klasse Base64OutputStream
  - commons-codec-1.9-bin.zip downloaden und unter Code > Jars speichern
  - damit haben wir zwei .jar Dateien eingebunden



Fremde Java APIs einbinden

- 3. Schritt – Zeichenfolge für den QRCode erstellen
  - QRCode verschlüsselt nur Zeichenfolgen, d.h ganz normalen Text
  - Smartphones lesen QRCode, stellen die Zeichenfolge wieder her und zeigen sie an
  - gute Apps für QRCode-Lesen sind "i-nigma" und "Barcode Scanner"
  - bei bestimmten Inhalten "reagieren" die Apps, wenn sie z.B. mit "BEGIN:VCARD", "http://" oder tel: beginnen
  - da wir einen Kontakt erstellen möchten, programmieren wir eine zusätzliche Methode in der Personenklasse, die vCards erzeugt wie

```
BEGIN:VCARD VERSION:4.0
N:Herrmann;Gottfriede
FN:Gottfriede Herrmann
ADR;TYPE=home:Kitzbühler Straße 65;Essen;;45359
TEL;TYPE=voice,home:05472-83774855018
EMAIL;TYPE=home:gottfriede.herrmann@essesno.de
END:VCARD
```

Fremde Java APIs einbinden

- 3. Schritt – Zeichenfolge für den QRCode erstellen
  - zusätzliche Methode getVcard() in der Klasse Person

```
public String getVcard() {
    String vCard = "BEGIN:VCARD VERSION:4.0\n";
    vCard += "N:" + getNachname() + ";" + getVorname() + "\n";
    vCard += "FN:" + getVorname() + " " + getNachname() + "\n";
    vCard += "ADR;TYPE=home:" + getStrasse() + ";" + getOrt() + ";;"
                  + getPlz() + "\n";
    vCard += "TEL;TYPE=voice,home:" + getTelefon() + "\n";
    vCard += "EMAIL;TYPE=home:" + getMail() + "\n";
    vCard += "END:VCARD";
    return vCard;
}
```

- 4. Schritt – QRCode für Zeichenfolge als Base64 erzeugen
  - Klasse Qrcode mit Methode getImgBase64() erstellen

```
01  private int laenge = 110;
02  private String format = "png";
03
04  public String getImgBase64(String text) throws Exception {
05      Hashtable<EncodeHintType, Object> hints =
06                                  new Hashtable<EncodeHintType, Object>();
07      hints.put(EncodeHintType.CHARACTER_SET, "UTF-8");
08      hints.put(EncodeHintType.MARGIN, 0);
09      QRCodeWriter writer = new QRCodeWriter();
10      BitMatrix bitmatrix = writer.encode(text, BarcodeFormat.QR_CODE,
11                                              laenge, laenge, hints);
12      ByteArrayOutputStream out = new ByteArrayOutputStream();
13      Base64OutputStream out64 = new Base64OutputStream(out);
14      MatrixToImageWriter.writeToStream(bitmatrix, format, out64);
15      out64.close();
16      out.close();
17      return "" + out.toString();
18  }
```

Fremde Java APIs einbinden

- 5. Schritt – alles in einer XPages zusammenführen
  - Objekte für Person und QRCode in beforePageLoad anlegen

```
<xp:this.beforePageLoad><![CDATA[#{javascript:
   viewScope.person = new de.leonso.entwcamp.PersonNotes();
   viewScope.qrcode = new de.leonso.entwcamp.Qrcode()}]]>
</xp:this.beforePageLoad>
```

  - gewünschte Person vom Nutzer abfragen und Personendaten lesen

```
<xp:inputText
      id="inputPerson"
      value="#{viewScope.inputPerson}" />
  <xp:button
      id="buttonRead"
      value="Einlesen">
      <xp:eventHandler
         event="onclick"
         submit="true"
         refreshMode="complete">
         <xp:this.action><![CDATA[#{javascript:
            viewScope.person.readPerson(viewScope.inputPerson)
         }]]></xp:this.action>
      </xp:eventHandler>
  </xp:button>
```

Fremde Java APIs einbinden

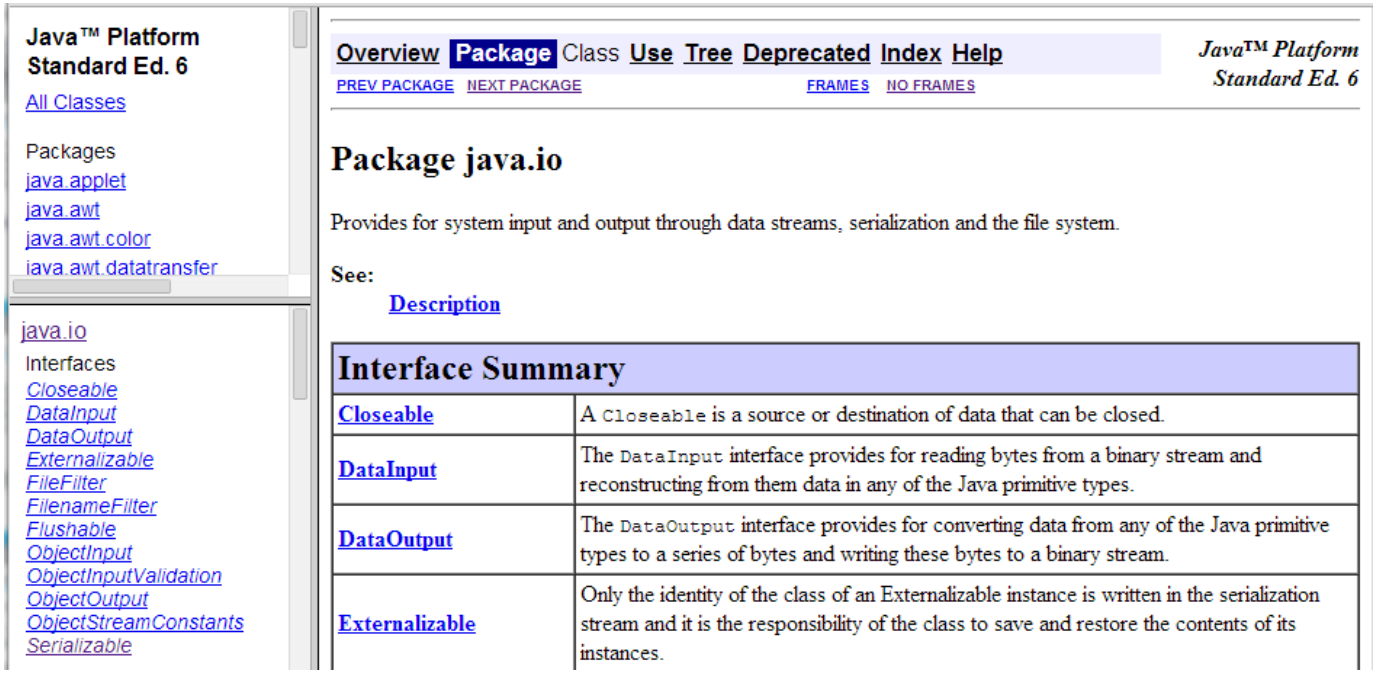
- 5. Schritt – alles in einer XPages zusammenführen
  - Personendaten und QRCode anzeigen

```
<xp:inputTextarea
        id="inputTextareaAnschrift"
        value="#{viewScope.person.anschrift}"
        cols="30"
        rows="3" />
    <br />
    <xp:image
        id="imageQrcode"
        url="#{javascript:viewScope.qrcode.getImgBase64(
                                    viewScope.person.vcard)}">
```

Demo

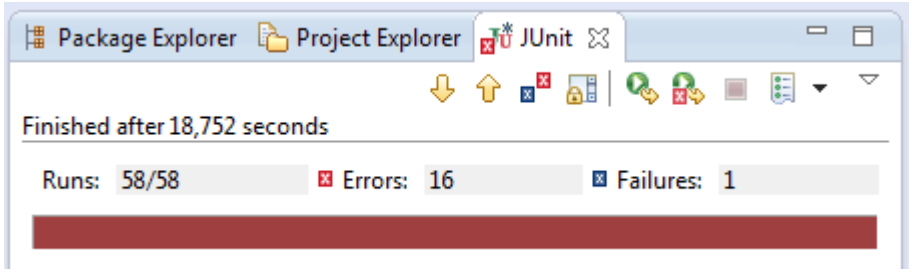
Nächste Etappen

- Javadoc
  - Java-Klassen so strukturiert dokumentieren, dass daraus eine HTML-Dokumentation generiert werden kann
  - Beschreibung: www.oracle.com/.../javadoc
  - Beispiel

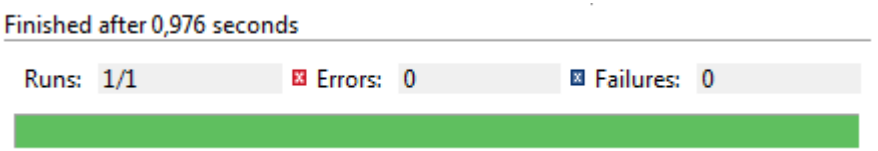


Nächste Etappen

- JUnit – automatisiertes Testen
  - parallel zum Code schreibt man Tests
  - Test liefert nur zwei Zustände: erfolgreich / fehlerhaft
  - nach Änderungen immer wieder alle Tests laufen lassen
  - = kontinuierliche Qualitätssicherung
  - junit.org



  - Ziel: grüner Balken



Nächste Etappen

- Einsatz von Version Control Software
  - integriert in Domino Designer im Kontextmenü "Team"
  - Nutzung von Git / Mercurial
- Installation und Nutzung des originalen Eclipse
  - insbesondere für größere Java-Projekte sinnvoll
- Tieferes Eindringen in die Java-Welt
  - Verwendung von Design Pattern
  - Java EE mit JSF, JSP, …

Fragen & Antworten

- Jetzt bitte Fragen stellen…

- … oder später
  - E-Mail: knut.herrmann@leonso.de
  - Twitter: @KnutHerrmann
  - Skype: knut.herrmann

- weitere Downloads zum Vortrag stehen unter www.leonso.de/ec14 zur Verfügung