Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Language Technology

Prof. Dr. Chris Biemann
Seid M. Yimam

Language Technology Lab
Universität Hamburg

Overview

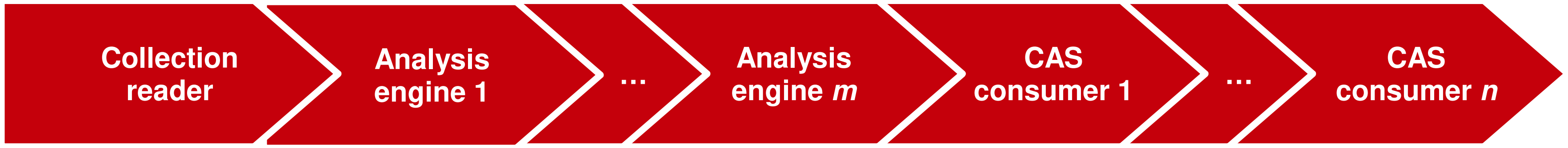
- In Tutorial 1, you set up your development environment with Eclipse, Maven, and UIMA
- In this tutorial, you will deepen your understanding of UIMA and associated libraries:
  - You will learn about the nature and purpose of the uimaFIT and DKPro libraries
  - You will learn how to create a DKPro-dependent application from scratch using Maven and Eclipse
  - You will be introduced to analysis engine configuration parameters

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Review: UIMA

- UIMA = Unstructured Information Management Architecture
- A component-based architecture for analysis of unstructured information
- "Analysis" means deriving a structure from the unstructured data
- Works like an assembly line:
  - take the raw material
  - refine it step by step
  - drive off with a nice car

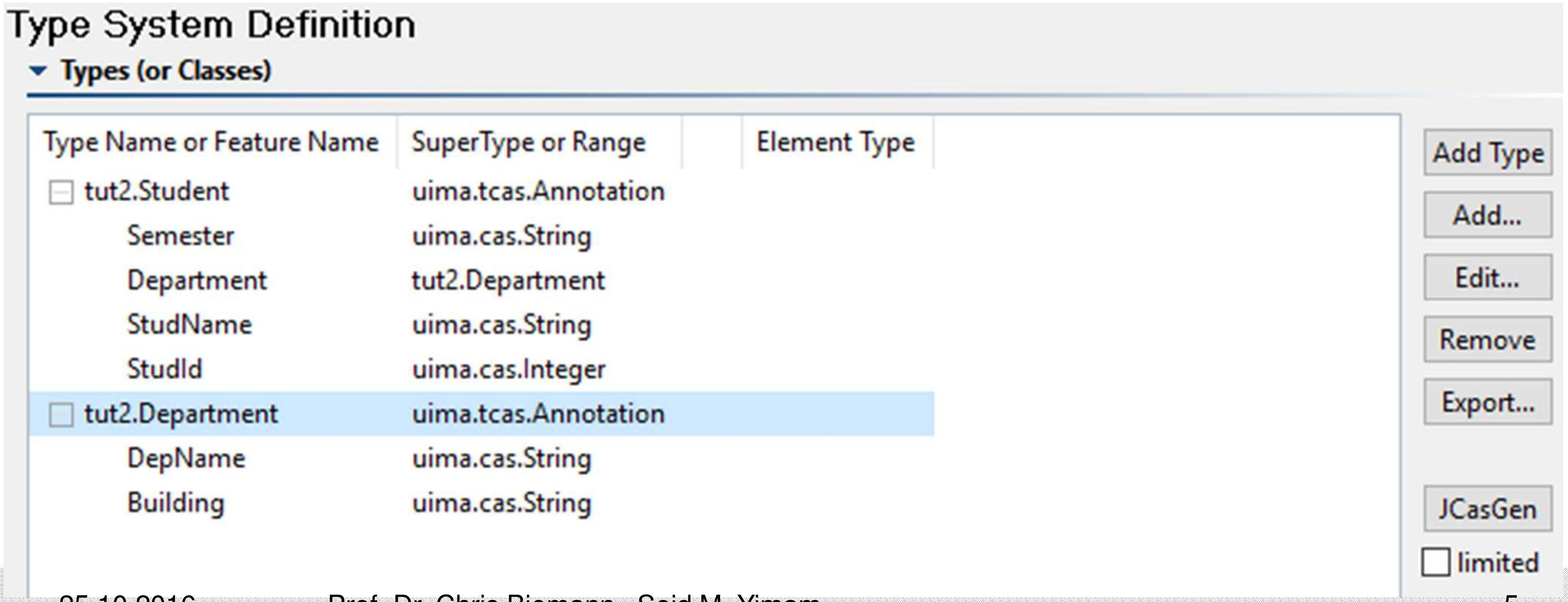
Review: UIMA aggregate analysis engine



- **Aggregate analysis engines (AAEs)** specify a "source to sink" flow of data
- **Collection readers** acquire source data
- **Analysis engines** label regions of documents with user-defined data types called **annotations**
- **CAS consumers** then do something useful with the annotated documents

Review: UIMA annotations

- Annotation types are defined in **XML descriptor files**
- The UIMA Eclipse plugin provides the **component descriptor editor**, a GUI for creating and editing these XML descriptor files
- The editor automatically generates Java classes from the XML



Type System Definition

Types (or Classes)

Type Name or Feature Name	SuperType or Range	Element Type
tut2.Student	uima.tcas.Annotation	
Semester	uima.cas.String	
Department	tut2.Department	
StudName	uima.cas.String	
StudId	uima.cas.Integer	
tut2.Department	uima.tcas.Annotation	
DepName	uima.cas.String	
Building	uima.cas.String	

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

XML descriptor files

- Collection readers, analysis engines, and CAS consumers must also be defined in XML descriptor files
- In Eclipse, these descriptors are normally created and edited with the component descriptor editor, just as with the annotation type descriptors
- Having to first create the XML descriptors for each AAE component is time-consuming
- It's also difficult to keep the XML descriptors in sync with their associated Java code
- Data flows are defined using a special GUI, the **collection processing engine configurator**

uimaFIT

- uimaFIT is a library which automatically generates the XML descriptor files from Java code
- With uimaFIT
  - no XML descriptor needs to be manually edited to create an Annotator
  - a pipeline can be easily sticked together without editing any XML (which is all done behind the scene)
- The only time you need to use the component descriptor editor is when defining annotation types
- Further information and documentation is available from the uimaFIT website : https://uima.apache.org/uimafit.html

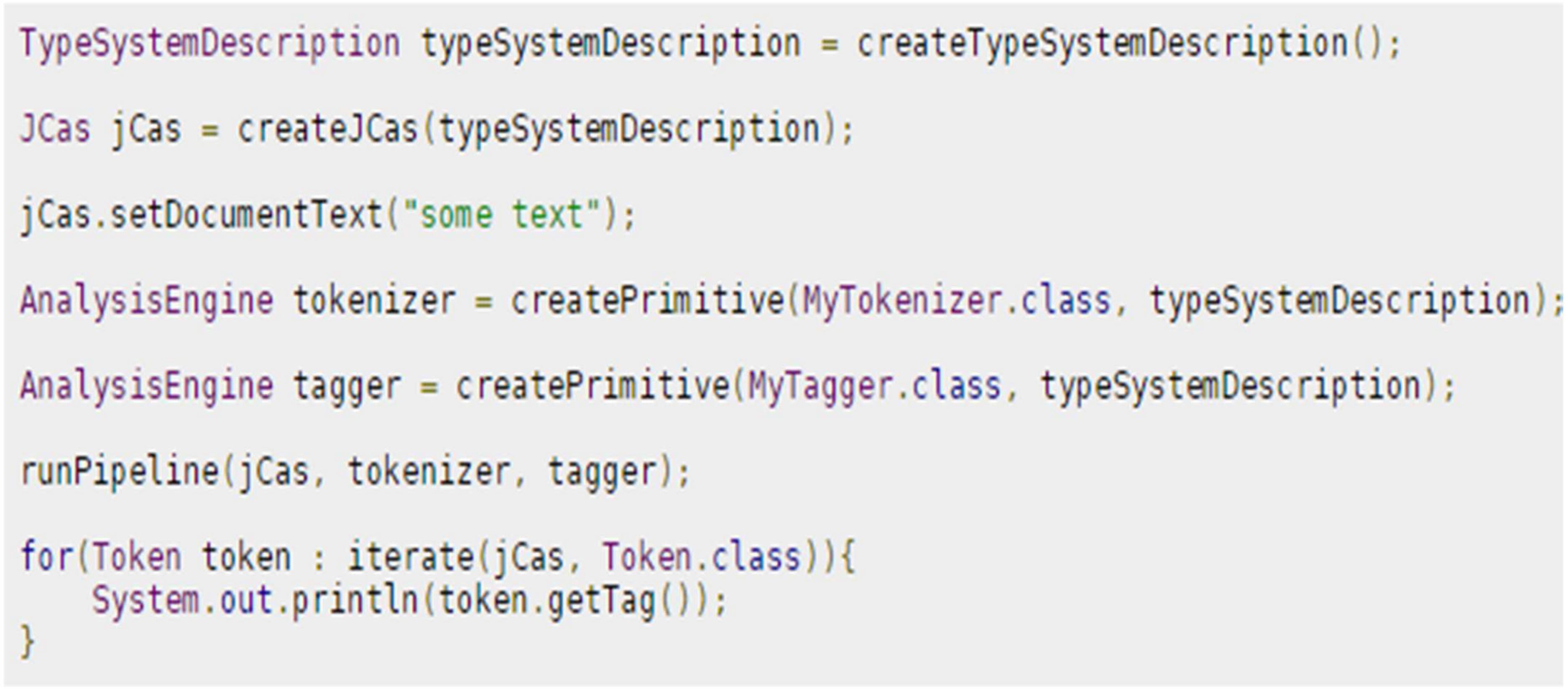
uimaFIT – Its power

- What does it take to "just tag some text?" Here's a list of things you must do with the traditional approach:
  1. wrap your tagger as a UIMA analysis engine
  2. write a descriptor file for your analysis engine
  3. write a CAS consumer that produces the desired output
  4. write another descriptor file for the CAS consumer
  5. write a descriptor file for a collection reader
  6. write a descriptor file that describes a pipeline
  7. invoke the Collection Processing Manager with your pipeline descriptor file

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

uimaFIT – Its power

- Here's a simple snippet of Java code that demonstrates "tagging some text" with uimaFIT



```java
TypeSystemDescription typeSystemDescription = createTypeSystemDescription();

JCas jCas = createJCas(typeSystemDescription);

jCas.setDocumentText("some text");

AnalysisEngine tokenizer = createPrimitive(MyTokenizer.class, typeSystemDescription);

AnalysisEngine tagger = createPrimitive(MyTagger.class, typeSystemDescription);

runPipeline(jCas, tokenizer, tagger);

for(Token token : iterate(jCas, Token.class)){
    System.out.println(token.getTag());
}
```

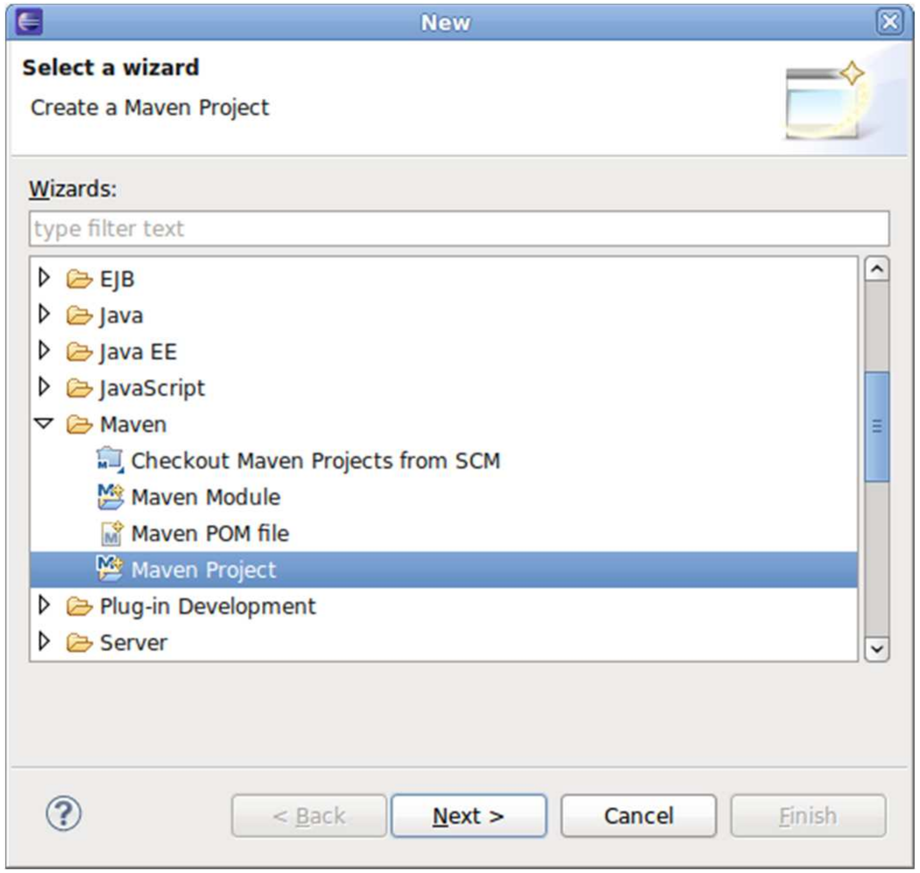
Darmstadt Knowledge Processing Repository (DKPro)

- A collection of software components for natural language processing based on the Apache UIMA framework (with uimaFIT)
- Components cover the whole range of NLP-related processing tasks
- Provides wrappers for third-party NLP tools as well as original NLP components:
  - tokenization, segmentation (Stanford segmenter, OpenNLP Segm.)
  - compound splitting (Banana Split, JWordSplitter)
  - stemming (Snowball)
  - part-of-speech tagging (TreeTagger)
  - spelling correction (Jazzy)
  - I/O support for various formats (text, XML, PDF, wikitext, etc.)

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Exercise - Create a new Maven project

- File → New → Other… → Maven → Maven Project


New

Select a wizard

Create a Maven Project

Wizards:

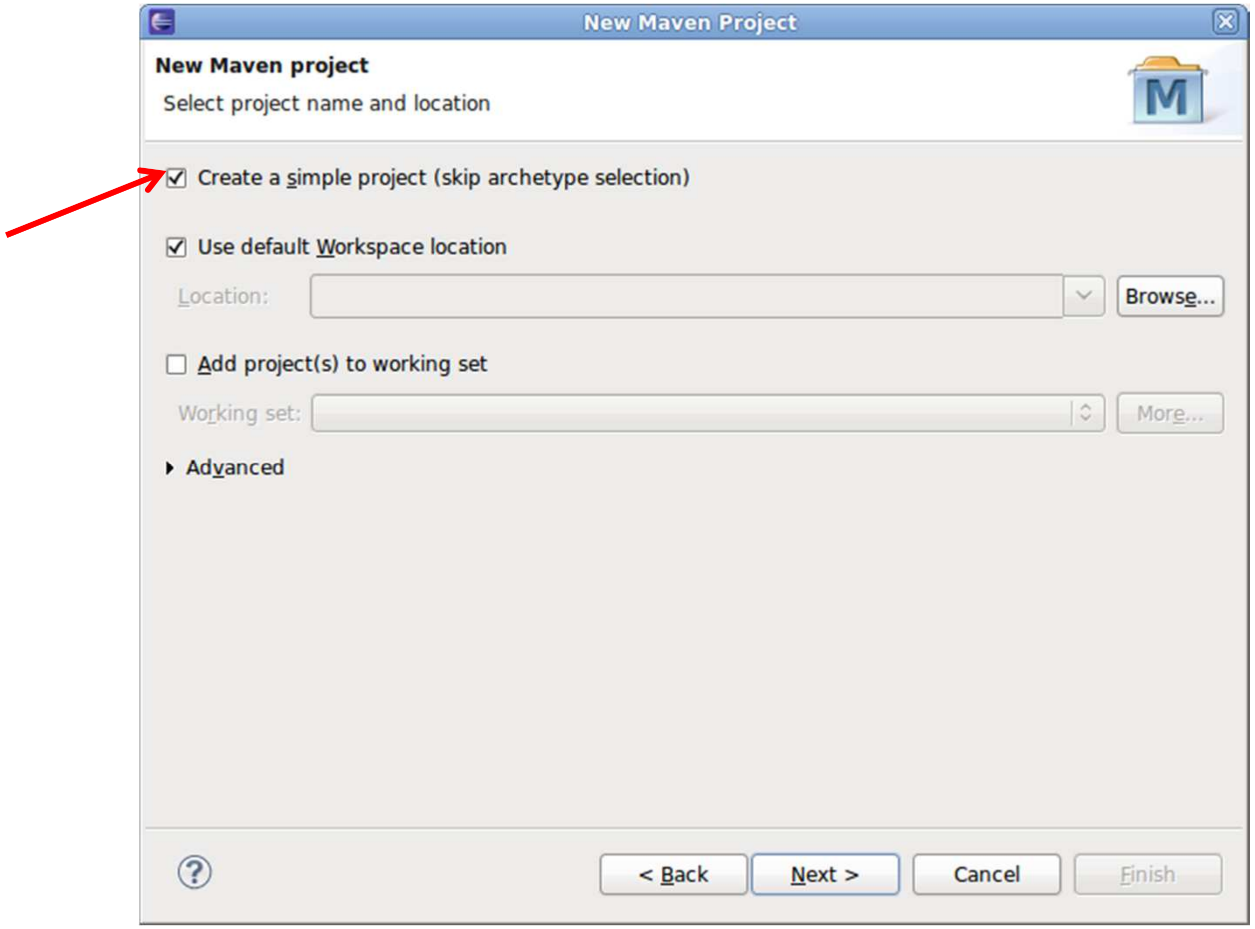
type filter text

- EJB
- Java
- Java EE
- JavaScript
- Maven
  - Checkout Maven Projects from SCM
  - Maven Module
  - Maven POM file
  - Maven Project
- Plug-in Development
- Server

< Back | Next > | Cancel | Finish

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Exercise - Create a new Maven project


New Maven Project

New Maven project
Select project name and location

- [✓] Create a simple project (skip archetype selection)
- [✓] Use default Workspace location

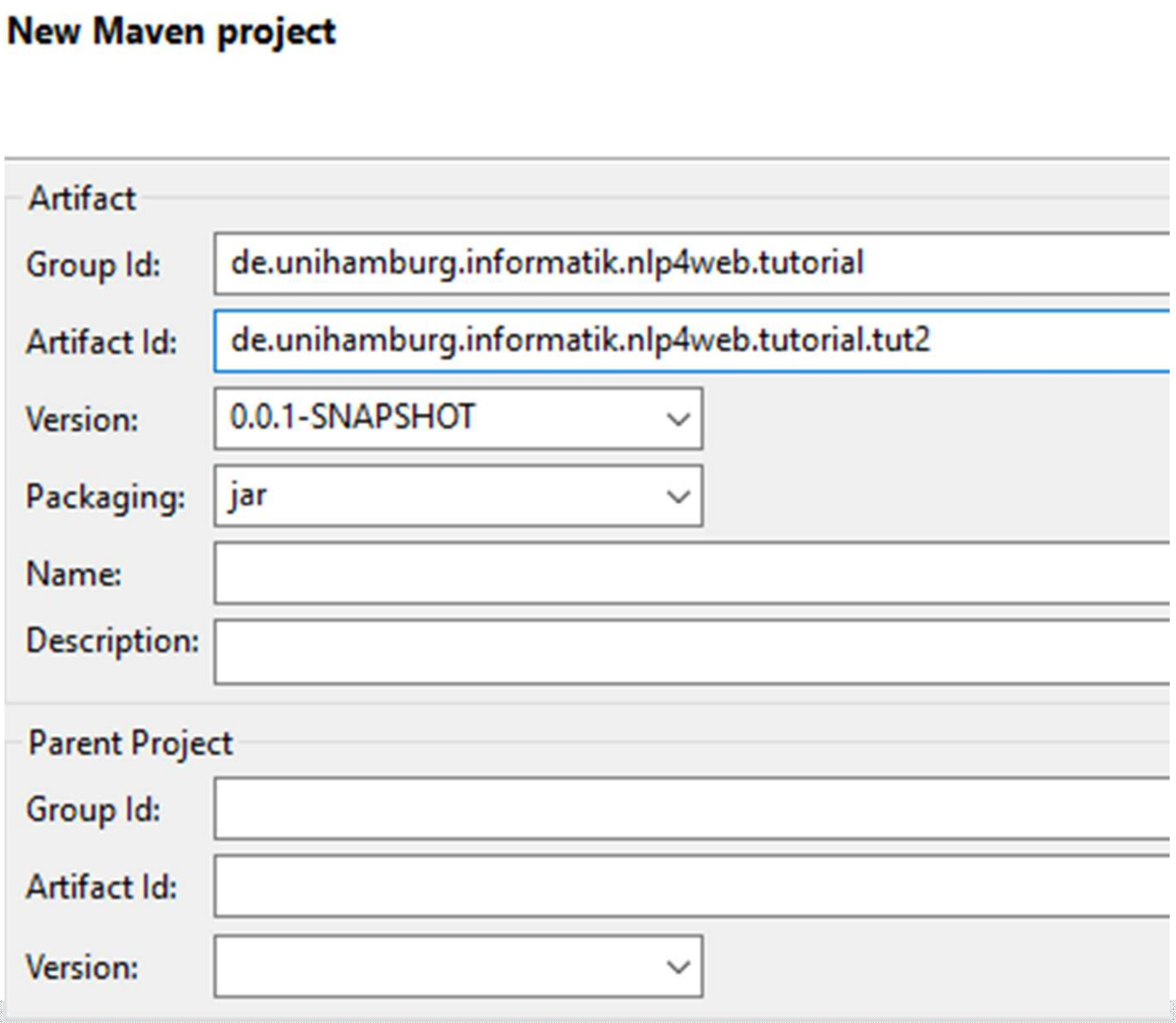
Location:

- [ ] Add project(s) to working set

Working set:

▸ Advanced

Exercise - Create a new Maven project



New Maven project

Artifact

Group Id:	de.unihamburg.informatik.nlp4web.tutorial
Artifact Id:	de.unihamburg.informatik.nlp4web.tutorial.tut2
Version:	0.0.1-SNAPSHOT
Packaging:	jar
Name:	
Description:	

Parent Project

Group Id:	
Artifact Id:	
Version:	

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

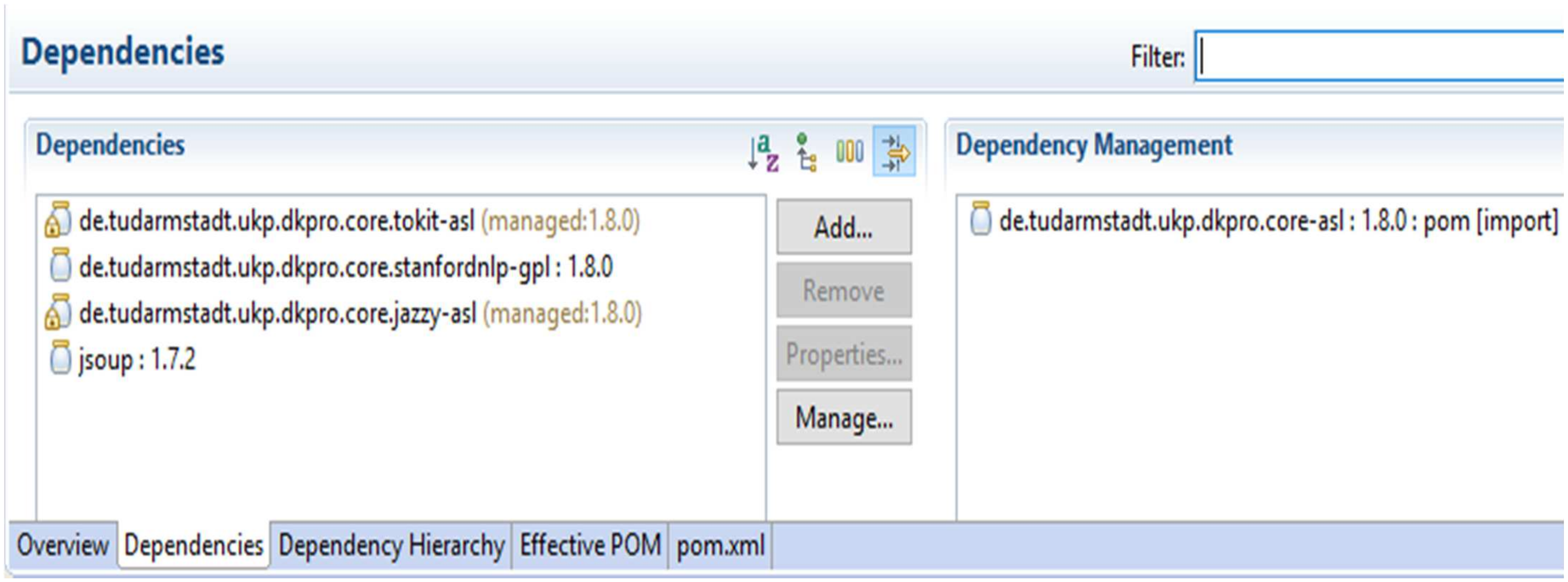
Exercise - Add DKPro dependencies to the POM

- Pom.xml → Dependencies → Dependency Management→Add
  - de.tudarmstadt.ukp.dkpro.core-asl (Version 1.8.0)
    (you can search for the package, by just typing **core-asl,**
    use pom as type and import as scope)

- Pom.xml → Dependencies → Dependencies →Add
  - de.tudarmstadt.ukp.dkpro.core.tokit-asl

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Exercise - Add DKPro dependencies to the POM



de.tudarmstadt.ukp.dkpro.core.tokit-asl (managed:1.8.0)
de.tudarmstadt.ukp.dkpro.core.stanfordnlp-gpl : 1.8.0
de.tudarmstadt.ukp.dkpro.core.jazzy-asl (managed:1.8.0)
jsoup : 1.7.2

de.tudarmstadt.ukp.dkpro.core-asl : 1.8.0 : pom [import]

Exercise - Create a webpage reader

Task: Create a collection reader which reads the text of a webpage

- Create a class that extends **JCasCollectionReader_ImplBase**
  Use the class that is located at org.apache.uima.fit.component.JCasCollectionReader_ImplBase
- Use jsoup (http://jsoup.org) for retrieving the page and extracting the text
  - First add jsoup to the dependencies in your pom.xml

```
<dependency>
        <groupId>org.jsoup</groupId>
        <artifactId>jsoup</artifactId>
        <version>1.7.2</version>
 </dependency>
```

```
Document doc = Jsoup.connect("http://example.com/").get();
String title = doc.title();
```

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Exercise - Create a collection reader

Task: Create a collection reader which reads the text of a webpage

- Override:
  - `initialize(UimaContext context)`
    Call super.initialize(context) and fetch the webpage.
  - `getProgress()`
    Return `new Progress[] {new ProgressImpl(`*`current`*`, `*`max`*`, Progress.ENTITIES)}`, where *`max`* is the total number of documents in the collection, and *`current`* is the index of the current document.
  - `hasNext()` ,
    Return true if there are more documents to fetch with getNext(). Otherwise return false.
  - `getNext(Jcas j)`
    Add the webpage's to the CAS: `j.setDocumentText(s);`
    - Where `s` is the text of the webpage

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Exercise - Create a CAS consumer

Task: Implement a CAS consumer which outputs the contents of each CAS

- Superclass **`JCasConsumer_ImplBase`**
- Override `process(JCas jcas)` to perform the output, using the iterators (see next slide) . The following `Annotation` methods can be used to retrieve interesting annotation data:
  - `getType()`
  - `getShortName()`
  - `getBegin()`
  - `getEnd()`
  - `getCoveredText()`
- For accessing the logger, use **`getContext().getLogger()`**

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Exercise - How to iterate annotations?

```
// Efficiently select specific annotation types.
for (Sentence s : JCasUtil.select(jcas, Sentence.class)) {
        for (Token t : JCasUtil.selectCovered(jcas, Token.class, s)) {
                String text = t.getCoveredText();
                // ...
        }
}

// Iterate over all annotations. Slower, but fetches all annotations.
for (Annotation a : JCasUtil.select(jcas, Annotation.class)) {
        String shortName = a.getType().getShortName();
        String text = a.getCoveredText();
        // ...
}
```

Exercise - Create an Aggregate Analysis Engine

Task: Create a UIMA pipeline which runs your collection reader, analysis engines, and your CAS consumer.

- Follow the example of `Pipeline.java` from the previous tutorial (tutorial 1)
- For the analysis engine, use `de.tudarmstadt.ukp.dkpro.core.tokit.BreakIteratorSegmenter`, a DKPro module for **performing tokenization** using Java's `BreakIterator`.
- Instead of the `BreakIteratorSegmenter`, try using **`StanfordSegmenter`** `(Artefact: de.tudarmstadt.ukp.dkpro.core.stanfordnlp-gpl).` (You will first have to add a dependency to your POM.) How does it compare?

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Configuration parameters

- Analysis engines can be written to take **configuration parameters**—name–value pairs which affect the behaviour of the engine
- In uimaFIT, configuration parameter names and values are passed to the engine as arguments to ***createEngine***( )
- An example is the **Jazzy spellchecker**, which requires you to supply a dictionary file. Try adding Jazzy (search for the name jazzy in the dependency and use the dkpro version) to your dependencies and then to your AAE with something like the following:

```
AnalysisEngine jazzy = createEngine(JazzyChecker.class,
JazzyChecker.PARAM_MODEL_LOCATION, "/usr/share/dict/words");
```

Jazzy performs **spellchecking** on `Token` annotations, so make sure it is invoked after a tokenizer.

Defining configuration parameters

- You can specify configuration parameters for your own UIMA components by using the `@ConfigurationParameter` Java annotation when declaring member variables:

```
import org.apache.uima.fit.descriptor.ConfigurationParameter;
public static final string PARAM_TEXT_ENCODING =
   "TextEncoding";
@ConfigurationParameter(
   name = PARAM_TEXT_ENCODING,
   description = "Sets the file's text encoding for I/O",
   mandatory = true, defaultValue = "utf-8")
private String textEncoding;
```

- The four Java annotation elements are all optional; you can see their default values in the org.apache.uima.fit.descriptor Javadocs on the uimaFIT web page: `https://uima.apache.org/uimafit.html`

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Exercise - Defining configuration parameters

- Think back to the webpage reader you just implemented. How did you specify the URL to read the webpages from?
- You probably hard-coded the URL of the webpage. Add a mandatory configuration parameter to your collection reader which takes the URL.