13 Import of external libraries

13.1 Introduction

In many applications, which are tackled by the use of a programming language, the standard libraries (API's) are not sufficient to work effectively. For example, if applications are to be implemented which have to deal with image data (e.g. modify, scale, transform images, etc.) the use of a library is recommended, which provides readily implemented classes and methods for those modifications of image data. Typically, such libraries contain classes too, which pattern the respective object (in this case an image) with its properties. Optimally after the import of such a library the data type "image" is provided.

13.2 Packages

Typically, in Java libraries are contained in so-called packages. Here, a Java packet is a collection of class files, which are organized according to specific criteria within directories[6]. In most cases these packages are packed in so-called JAR archives.

First a JAR archive is a special ZIP file. The most important usage of JAR archives is to collect all data, which belong to a Java program (.class-, image-, sound files, etc.) in a single file.

13.3 Load packages

As an example for an external library the project Apache-PDF-Box may serve (http://pdfbox.apache.org/), a Java library to work with PDFs (creation, display, extraction, join, etc.).

This is the link to the newes version of this JAR archive:

https://repo1.maven.org/maven2/org/apache/pdfbox/pdfbox/2.0.21/pdfbox-2.0.21.jar

Further we need a JAR archive for the API documentation:

https://repo1.maven.org/maven2/org/apache/pdfbox/pdfbox/2.0.21/pdfbox-2.0.21-javadoc.jar

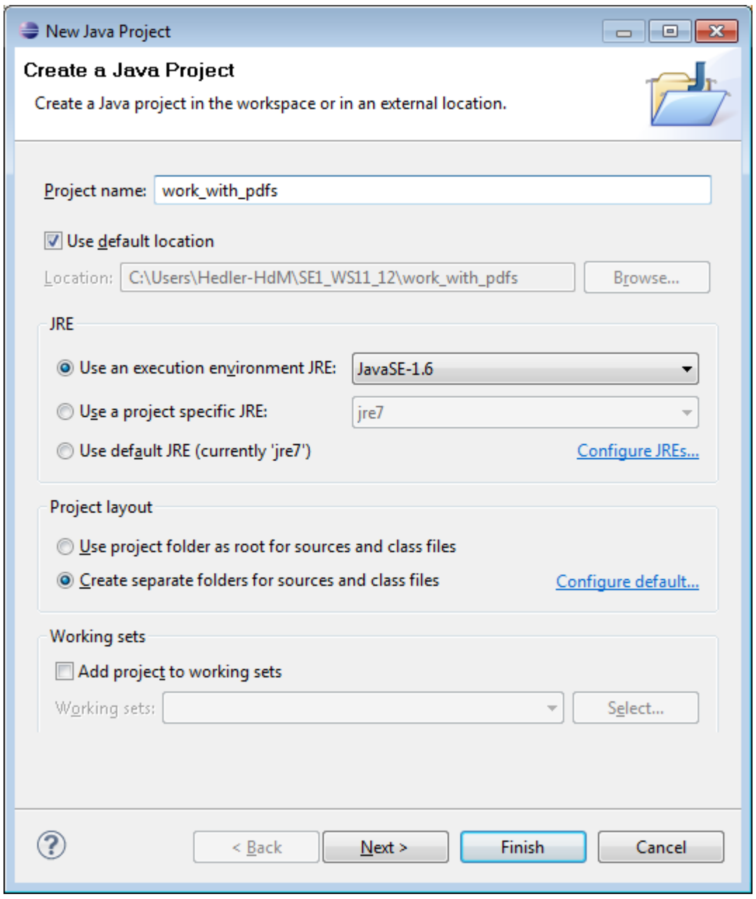
13.4 Add packages in Eclipse

Both JAR files have to be downloaded and optimally stored in an arbitrary directory, which belongs to the project. Alternatively you find them in Moodle in the exercises folder of this lecture.

To be able to import a specific packet with the import instruction, the JAR archive has to be made known to the IDE (in our case Eclipse)[7].

[6] The creation of such packages is not part of this lecture.
[7] Eclipse knows about the location of the standard API, but of course does not know about other, external libraries.

Generate a new Java project in Eclipse now (the version number of your JavaSE may differ):


New Java Project

Create a Java Project

Create a Java project in the workspace or in an external location.

Project name: work_with_pdfs

[✓] Use default location

Location: C:\Users\Hedler-HdM\SE1_WS11_12\work_with_pdfs

JRE

(●) Use an execution environment JRE: JavaSE-1.6

( ) Use a project specific JRE: jre7

( ) Use default JRE (currently 'jre7')

Project layout

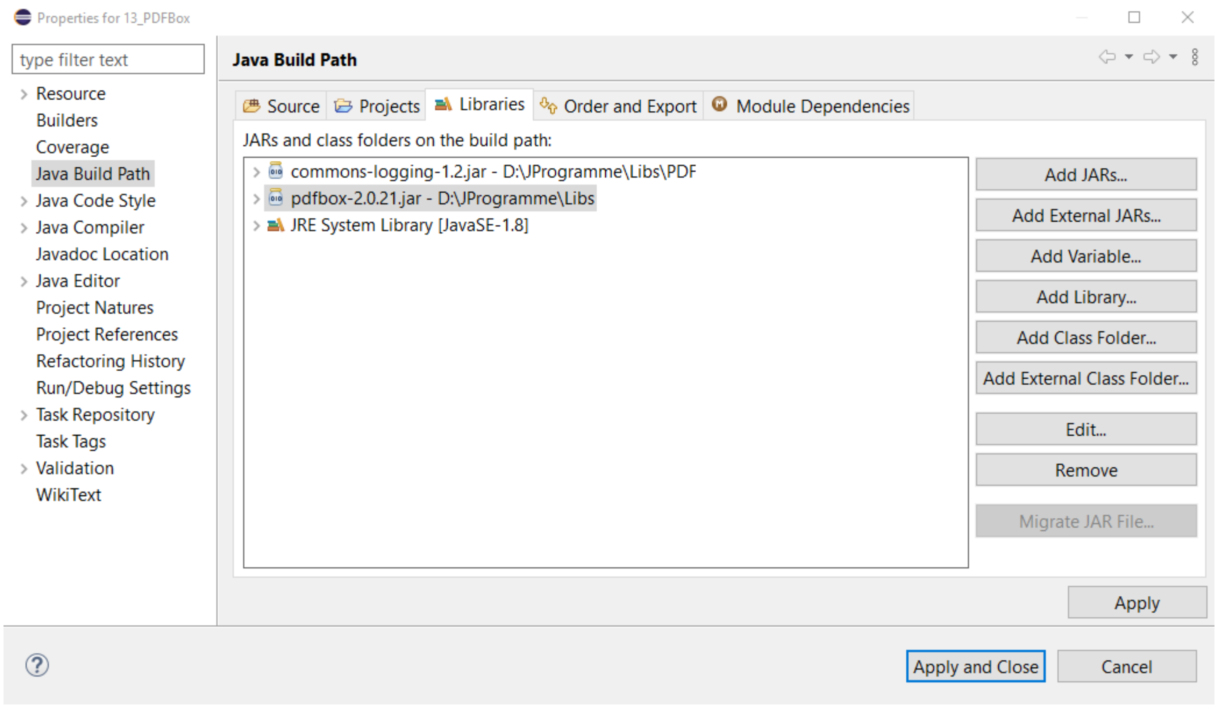
( ) Use project folder as root for sources and class files

(●) Create separate folders for sources and class files

Working sets

[ ] Add project to working sets

Klick “Next“.


Properties for 13_PDFBox

Java Build Path

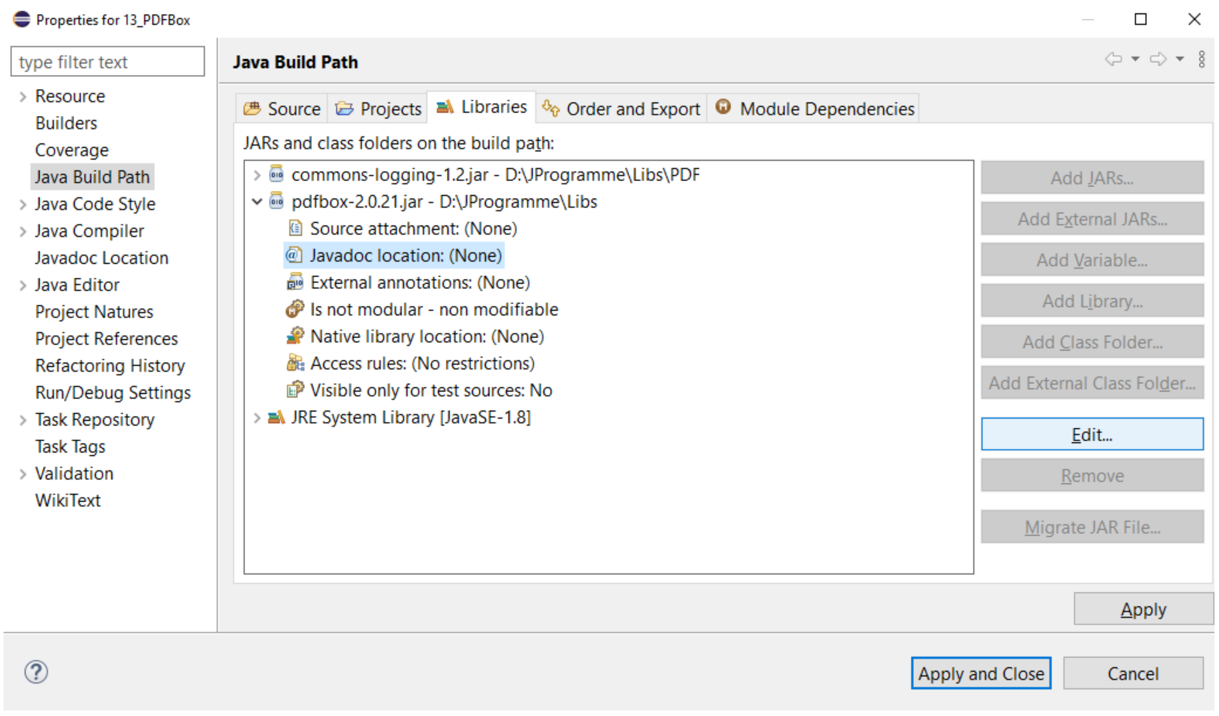
Source | Projects | Libraries | Order and Export | Module Dependencies

JARs and class folders on the build path:

- commons-logging-1.2.jar - D:\JProgramme\Libs\PDF
- pdfbox-2.0.21.jar - D:\JProgramme\Libs
- JRE System Library [JavaSE-1.8]

In the tab “Libraries“ external JARs can be added now. Add the library `pdfbox-app-2.0.21.jar`.

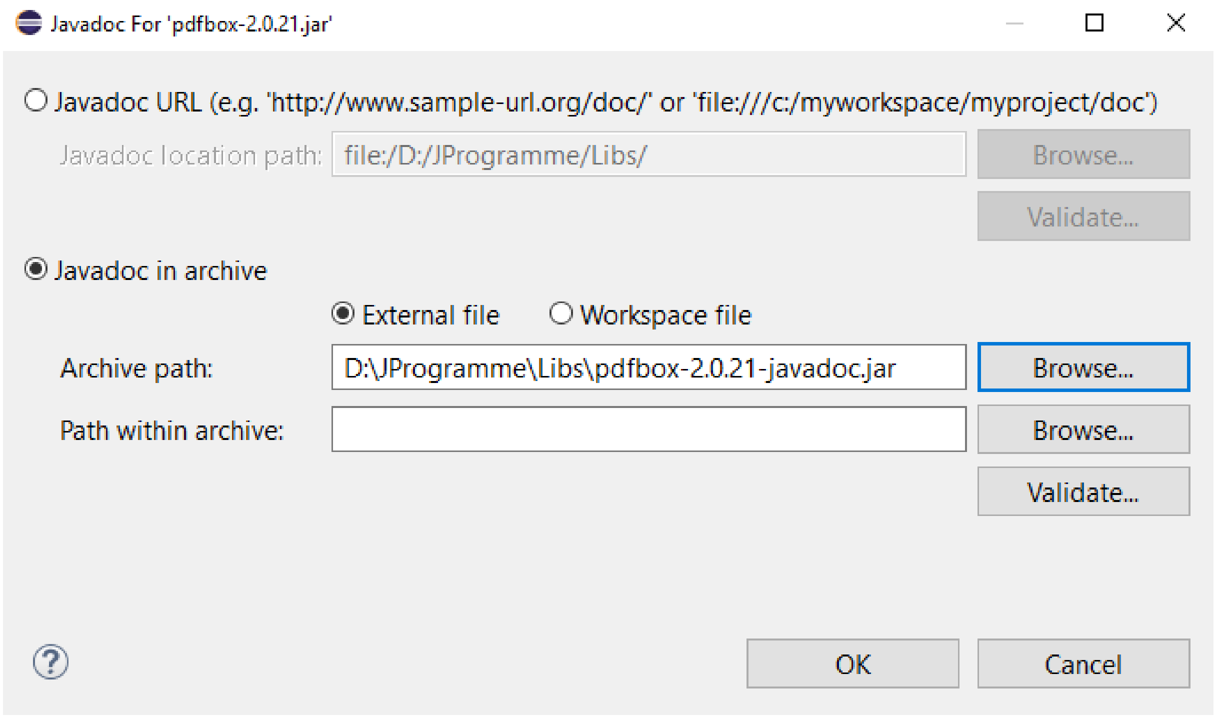
Now open the directory of the archive, mark “Javadoc location“ and press the “Edit...“ button.


Properties for 13_PDFBox

Java Build Path

JARs and class folders on the build path:
- commons-logging-1.2.jar - D:\JProgramme\Libs\PDF
- pdfbox-2.0.21.jar - D:\JProgramme\Libs
  - Source attachment: (None)
  - Javadoc location: (None)
  - External annotations: (None)
  - Is not modular - non modifiable
  - Native library location: (None)
  - Access rules: (No restrictions)
  - Visible only for test sources: No
- JRE System Library [JavaSE-1.8]

Now declare the JAR „pdfbox-2.0.21-javadoc.jar“ as documentation to the jar above:


Javadoc For 'pdfbox-2.0.21.jar'

Javadoc URL (e.g. 'http://www.sample-url.org/doc/' or 'file:///c:/myworkspace/myproject/doc')

Javadoc location path: file:/D:/JProgramme/Libs/

[✓ Javadoc in archive]

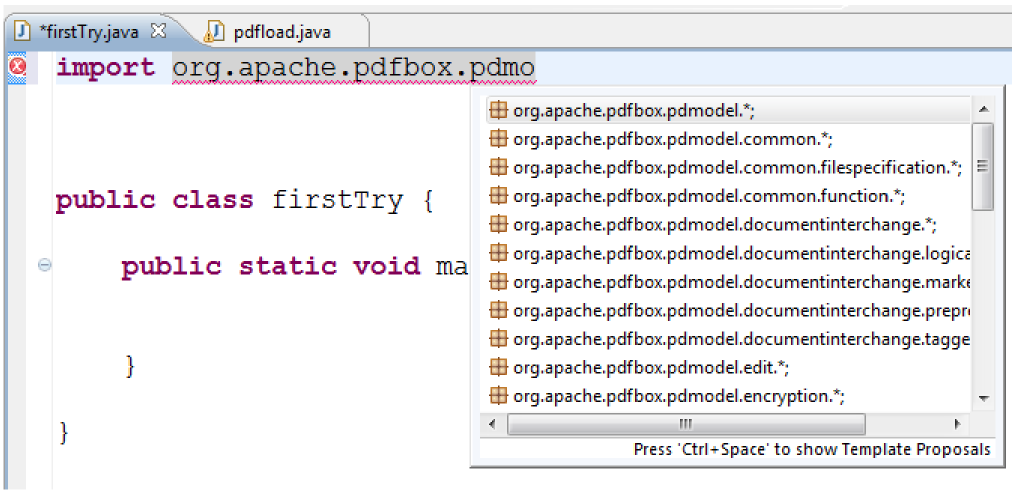
[✓ External file] Workspace file

Archive path: D:\JProgramme\Libs\pdfbox-2.0.21-javadoc.jar

Path within archive:

Both JARs are now in the “build path“ of Eclipse and are accessible now. Klick „OK“ and then „Finish“.

With the help of the import statement various packages can be imported now, as the following screenshot shows:


*firstTry.java
pdfload.java

```
import org.apache.pdfbox.pdmo


public class firstTry {

    public static void ma

    }

}
```

```
org.apache.pdfbox.pdmodel.*;
org.apache.pdfbox.pdmodel.common.*;
org.apache.pdfbox.pdmodel.common.filespecification.*;
org.apache.pdfbox.pdmodel.common.function.*;
org.apache.pdfbox.pdmodel.documentinterchange.*;
org.apache.pdfbox.pdmodel.documentinterchange.logica
org.apache.pdfbox.pdmodel.documentinterchange.marke
org.apache.pdfbox.pdmodel.documentinterchange.prepr
org.apache.pdfbox.pdmodel.documentinterchange.tagge
org.apache.pdfbox.pdmodel.edit.*;
org.apache.pdfbox.pdmodel.encryption.*;
```

Press 'Ctrl+Space' to show Template Proposals

13.5 Working with the library PDFBox from Apache

For editing PDF documents, e.g. from a XML data source, libraries like Apache FOP are most appropriate, which have a grasp of the functionalities of a publishing program. PDFBox especially is suited to create simple PDF's and foremost process existing PDF's.

13.5.1 Create a new PDF document

This example shows how a new one-page PDF document can be created.

First an empty PDF document is generated (empty means here: without pages)

```
PDDocument document = new PDDocument();
```

As every document a PDF document needs at least one page. Therefore, we create the object of a page and add it to the object "document".

```
PDPage blankPage = new PDPage();
document.addPage( blankPage );
```

Now the document (which at the moment is in memory only) must be stored to disk.

```
document.save("BlankPage.pdf");
```

Finally the document must be closed.

```
document.close();
```

13.5.2 Load and modify PDF documents

An existing PDF document can be loaded and modified afterwards with PDF box.

Load a PDF document

```
PDDocument doc = PDDocument.load("test.pdf");
```

Now specific modifications can be performed, e.g. removing 2 pages.

```
doc.removePage(2);
```

Now the document (which at the moment is in memory only) must be stored into a different file to disk.

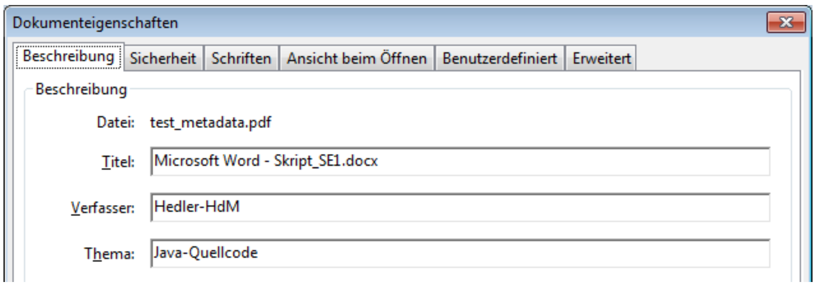
```
doc.save("test_mod.pdf");
```

At last the document must be closed.

```
doc.close();
```

13.6 Read PDF meta data

PDF documents contain divers meta data like originator, author., topic, etc. Normally you can display these meta data with Acrobat, as the following screenshot shows:



Beschreibung

Datei: test_metadata.pdf

Titel: Microsoft Word - Skript_SE1.docx

Verfasser: Hedler-HdM

Thema: Java-Quellcode

With the help of PDF-Box meta data can be extracted from a PDF document and then e.g. displayed.

First the PDF document must be loaded:

```
PDDocument document = PDDocument.load("test.pdf");
```

Then a meta data object (PDDocumentInformation) is queried from the document and is stored into the variable `info`.

```
PDDocumentInformation info = document.getDocumentInformation();
```

From this object „info“ various meta data of the PDF document can be queried and displayed:

```
System.out.println("Title=" + info.getTitle());
System.out.println("Author=" + info.getAuthor());
System.out.println("Subject=" + info.getSubject());
System.out.println("Keywords=" + info.getKeywords());
System.out.println("Creator=" + info.getCreator());
System.out.println("Producer=" + info.getProducer());
System.out.println("Creation Date=" + info.getCreationDate());
System.out.println("Modification Date=" + info.getModificationDate());
System.out.println("Trapped=" + info.getTrapped());
```

13.7 Extract text

A special application is the extraction of pure (unformatted) text data from PDF documents. Therefore, PDFBox provides the so-called stripper class.

First the PDF document has to be loaded again:

```
PDDocument document = PDDocument.load("test.pdf");
```

Then an object of class PDFTextStripper has to be generated.

```
PDFTextStripper stripper = new PDFTextStripper();
```

With this stripper object and its method `getText()` the PDF document's text can be parsed into a String:

```
String text= stripper.getText(doc);
```