合肥學院
HEFEI UNIVERSITY

OOP with Java

Homework 05: Maven

Thomas Weise · 汤卫思

tweise@hfuu.edu.cn · http://iao.hfuu.edu.cn

Hefei University, South Campus 2
Faculty of Computer Science and Technology
Institute of Applied Optimization
230601 Shushan District, Hefei, Anhui, China
Econ. & Tech. Devel. Zone, Jinxiu Dadao 99

合肥学院 南艳湖校区/南2区
计算机科学与技术系
应用优化研究所
中国 安徽省 合肥市 蜀山区 230601
经济技术开发区 锦绣大道99号

Outline

website

Introduction

- We want to practice Maven, Libraries, and JUnit
- This homework is comprised of four tasks for one project
- Send me a zip archive named `hw05_[your_student_id].zip` (where `[your_student_id]` is replaced with your student id) containing the whole project folder (see later)

Maven, Libraries and JUnit

- Maven is a tool to build Java applications

Maven, Libraries and JUnit

- Maven is a tool to build Java applications
- Java applications are usually built and delivered as `jar` executable files

Maven, Libraries and JUnit

- Maven is a tool to build Java applications
- Java applications are usually built and delivered as `jar` executable files
- They often use `jar` libraries as well

Maven, Libraries and JUnit

- Maven is a tool to build Java applications
- Java applications are usually built and delivered as `jar` executable files
- They often use `jar` libraries as well
- Maven can manage the dependencies of a project on external libraries

Maven, Libraries and JUnit

- Maven is a tool to build Java applications
- Java applications are usually built and delivered as `jar` executable files
- They often use `jar` libraries as well
- Maven can manage the dependencies of a project on external libraries
- We want to test this.

General Concept: Maven for Libraries

- We want to make a software which can produce `pdf` documents

General Concept: Maven for Libraries

- We want to make a software which can produce `pdf` documents
- The software has a main routine which takes a file name as command line argument.

General Concept: Maven for Libraries

- We want to make a software which can produce `pdf` documents
- The software has a main routine which takes a file name as command line argument.
- It should produce a `pdf` file of that name with the text "Hello World!"

General Concept: Maven for Libraries

- We want to make a software which can produce `pdf` documents
- The software has a main routine which takes a file name as command line argument.
- It should produce a `pdf` file of that name with the text "Hello World!"
- In order to produce `pdf` files, we will use the *Apache PDFBox 2.0.5* library (`https://pdfbox.apache.org/`)

General Concept: Maven for Libraries

- We want to make a software which can produce `pdf` documents
- The software has a main routine which takes a file name as command line argument.
- It should produce a `pdf` file of that name with the text "Hello World!"
- In order to produce `pdf` files, we will use the *Apache PDFBox 2.0.5* library (`https://pdfbox.apache.org/`)
- PDFBox in turn depends on a set of other libraries (such as Apache FontBox) with specific versions

General Concept: Maven for Libraries

- We want to make a software which can produce `pdf` documents
- The software has a main routine which takes a file name as command line argument.
- It should produce a `pdf` file of that name with the text "Hello World!"
- In order to produce `pdf` files, we will use the *Apache PDFBox 2.0.5* library (`https://pdfbox.apache.org/`)
- PDFBox in turn depends on a set of other libraries (such as Apache FontBox) with specific versions
- We will therefore use Maven to manage these dependencies and to automatically link to the libraries for us.

General Concept: Maven for Libraries

- We want to make a software which can produce `pdf` documents
- The software has a main routine which takes a file name as command line argument.
- It should produce a `pdf` file of that name with the text "Hello World!"
- In order to produce `pdf` files, we will use the *Apache PDFBox 2.0.5* library (`https://pdfbox.apache.org/`)
- PDFBox in turn depends on a set of other libraries (such as Apache FontBox) with specific versions
- We will therefore use Maven to manage these dependencies and to automatically link to the libraries for us.
- The Maven assembly plugin can even help us to build a `jar` of our application which already and directly contains all these required libraries, so we do not need to distribute them with our software

General Concept: Maven for JUnit

- Since we are using a Maven-based build process, we can also directly use its JUnit integration

General Concept: Maven for JUnit

- Since we are using a Maven-based build process, we can also directly use its JUnit integration
- We want to make three simple test cases for our `Main` class

General Concept: Maven for JUnit

- Since we are using a Maven-based build process, we can also directly use its JUnit integration
- We want to make three simple test cases for our `Main` class:
  - Check that the `static void main(String[] args)` method actually produces an output file

General Concept: Maven for JUnit

- Since we are using a Maven-based build process, we can also directly use its JUnit integration
- We want to make three simple test cases for our `Main` class:
  - Check that the `static void main(String[] args)` method actually produces an output file
  - Check whether it throws an `NullPointerException` if `args==null`

General Concept: Maven for JUnit

- Since we are using a Maven-based build process, we can also directly use its JUnit integration
- We want to make three simple test cases for our `Main` class:
  - Check that the `static void main(String[] args)` method actually produces an output file
  - Check whether it throws an `NullPointerException` if `args==null`
  - Check whether it throws an `IllegalArgumentException` if `args.length<1`

What to do?

- Luckily, most of this has already been prepared

The answer to this homework is a zip archive of the complete project folder after completing task hw05-4

What to do?

- Luckily, most of this has already been prepared:
  - an Eclipse Maven project has already been created

The answer to this homework is a zip archive of the complete project folder after completing task hw05-4

What to do?

- Luckily, most of this has already been prepared:
  - an Eclipse Maven project has already been created
  - a Maven `pom.xml` file exists with some basic settings

The answer to this homework is a zip archive of the complete project folder after completing task hw05-4

What to do?

- Luckily, most of this has already been prepared:
  - an Eclipse Maven project has already been created
  - a Maven `pom.xml` file exists with some basic settings
  - the `Main` class exists in folder `src/main/java` and package `cn.edu.hfuu.iao`

The answer to this homework is a zip archive of the complete project folder after completing task hw05-4

What to do?

- Luckily, most of this has already been prepared:
  - an Eclipse Maven project has already been created
  - a Maven `pom.xml` file exists with some basic settings
  - the `Main` class exists in folder `src/main/java` and package `cn.edu.hfuu.iao`
  - the JUnit test cases are already there, in folder `src/test/java` and package `cn.edu.hfuu.iao` as class `MainTest`

The answer to this homework is a zip archive of the complete project folder after completing task hw05-4

What to do?

- Luckily, most of this has already been prepared:
  - an Eclipse Maven project has already been created
  - a Maven `pom.xml` file exists with some basic settings
  - the `Main` class exists in folder `src/main/java` and package `cn.edu.hfuu.iao`
  - the JUnit test cases are already there, in folder `src/test/java` and package `cn.edu.hfuu.iao` as class `MainTest`
- But several things are missing

The answer to this homework is a zip archive of the complete project folder after completing task hw05-4

What to do?

- Luckily, most of this has already been prepared:
  - an Eclipse Maven project has already been created
  - a Maven `pom.xml` file exists with some basic settings
  - the `Main` class exists in folder `src/main/java` and package `cn.edu.hfuu.iao`
  - the JUnit test cases are already there, in folder `src/test/java` and package `cn.edu.hfuu.iao` as class `MainTest`
- But several things are missing:
  - The Maven `pom.xml` file lacks the dependency on the PDFBox library, so it cannot compile

The answer to this homework is a zip archive of the complete project folder after completing task hw05-4

What to do?

- Luckily, most of this has already been prepared:
  - an Eclipse Maven project has already been created
  - a Maven `pom.xml` file exists with some basic settings
  - the `Main` class exists in folder `src/main/java` and package `cn.edu.hfuu.iao`
  - the JUnit test cases are already there, in folder `src/test/java` and package `cn.edu.hfuu.iao` as class `MainTest`
- But several things are missing:
  - The Maven `pom.xml` file lacks the dependency on the PDFBox library, so it cannot compile
  - Some of the JUnit test cases may fail and the `cn.edu.hfuu.iao.Main` class may need to be adapted so that all tests pass

The answer to this homework is a zip archive of the complete project folder after completing task hw05-4

What to do?

- Luckily, most of this has already been prepared:
  - an Eclipse Maven project has already been created
  - a Maven `pom.xml` file exists with some basic settings
  - the `Main` class exists in folder `src/main/java` and package `cn.edu.hfuu.iao`
  - the JUnit test cases are already there, in folder `src/test/java` and package `cn.edu.hfuu.iao` as class `MainTest`
- But several things are missing:
  - The Maven `pom.xml` file lacks the dependency on the PDFBox library, so it cannot compile
  - Some of the JUnit test cases may fail and the `cn.edu.hfuu.iao.Main` class may need to be adapted so that all tests pass
  - After this is done, you can build the executable containing all the dependencies...

The answer to this homework is a zip archive of the complete project folder after completing task hw05-4

Setup

1. Extract the archive `hw05_sources.zip` from the website

Setup

1. Extract the archive `hw05_sources.zip` from the website
2. Import the extracted project into Eclipse

Setup

1. Extract the archive `hw05_sources.zip` from the website
2. Import the extracted project into Eclipse
3. You will get lots of compiler errors: Ignore them until Task hw05-2.

Task hw05-1: Adding Developer Information

1. Add your `developer` information to the Maven `pom.xml` file, similar to what we discussed in the lectures

Task hw05-1: Adding Developer Information

1. Add your `developer` information to the Maven `pom.xml` file, similar to what we discussed in the lectures
2. The answer to this question is the updated `pom.xml` file.

Task hw05-2: Adding Dependency

1. In order to use *Apache PDFBox 2.0.5* library, include its dependency information at the right position into the Maven `pom.xml` file

Task hw05-2: Adding Dependency

1. In order to use *Apache PDFBox 2.0.5* library, include its dependency information at the right position into the Maven `pom.xml` file
2. The newest library version can be found at `https://mvnrepository.com/artifact/org.apache.pdfbox/pdfbox`, where we select `2.0.5`

Task hw05-2: Adding Dependency

1. In order to use *Apache PDFBox 2.0.5* library, include its dependency information at the right position into the Maven `pom.xml` file
2. The newest library version can be found at `https://mvnrepository.com/artifact/org.apache.pdfbox/pdfbox`, where we select `2.0.5`
3. The following page provides the dependency information as given below

Listing: `pom.xml` The PDFBox Dependency Information

```
<groupId>org.apache.pdfbox</groupId>
<artifactId>pdfbox</artifactId>
<version>2.0.5</version>
```

Task hw05-2: Adding Dependency

1. In order to use *Apache PDFBox 2.0.5* library, include its dependency information at the right position into the Maven `pom.xml` file
2. The newest library version can be found at `https://mvnrepository.com/artifact/org.apache.pdfbox/pdfbox`, where we select `2.0.5`
3. The following page provides the dependency information as given below
4. Insert this information as dependency into the Maven `pom.xml` file

Listing: `pom.xml` The PDFBox Dependency Information

```
<groupId>org.apache.pdfbox</groupId>
<artifactId>pdfbox</artifactId>
<version>2.0.5</version>
```

Task hw05-2: Adding Dependency

1. In order to use *Apache PDFBox 2.0.5* library, include its dependency information at the right position into the Maven `pom.xml` file
2. The newest library version can be found at `https://mvnrepository.com/artifact/org.apache.pdfbox/pdfbox`, where we select `2.0.5`
3. The following page provides the dependency information as given below
4. Insert this information as dependency into the Maven `pom.xml` file
5. Then click on the project, select "Maven" and then "Update Project..."

Listing: `pom.xml` The PDFBox Dependency Information

```
<groupId>org.apache.pdfbox</groupId>
<artifactId>pdfbox</artifactId>
<version>2.0.5</version>
```

Task hw05-2: Adding Dependency

1. In order to use *Apache PDFBox 2.0.5* library, include its dependency information at the right position into the Maven `pom.xml` file
2. The newest library version can be found at `https://mvnrepository.com/artifact/org.apache.pdfbox/pdfbox`, where we select `2.0.5`
3. The following page provides the dependency information as given below
4. Insert this information as dependency into the Maven `pom.xml` file
5. Then click on the project, select "Maven" and then "Update Project..."
6. The compiler errors should now disappear

Listing: `pom.xml` The PDFBox Dependency Information

```
<groupId>org.apache.pdfbox</groupId>
<artifactId>pdfbox</artifactId>
<version>2.0.5</version>
```

Task hw05-2: Adding Dependency

1. In order to use *Apache PDFBox 2.0.5* library, include its dependency information at the right position into the Maven `pom.xml` file
2. The newest library version can be found at `https://mvnrepository.com/artifact/org.apache.pdfbox/pdfbox`, where we select `2.0.5`
3. The following page provides the dependency information as given below
4. Insert this information as dependency into the Maven `pom.xml` file
5. Then click on the project, select "Maven" and then "Update Project..."
6. The compiler errors should now disappear
7. The answer to this question is the updated `pom.xml` file (which includes also the answer to hw05-1).

Listing: `pom.xml` The PDFBox Dependency Information

```
<groupId>org.apache.pdfbox</groupId>
<artifactId>pdfbox</artifactId>
<version>2.0.5</version>
```

Task hw05-3: Running JUnit Tests

1. Then right click the class `cn.edu.hfuu.iao.MainTest` in folder `src/test/java` and select "Run as..." and then "JUnit test".

Task hw05-3: Running JUnit Tests

1. Then right click the class `cn.edu.hfuu.iao.MainTest` in folder `src/test/java` and select "Run as..." and then "JUnit test".
2. Notice whether any of the tests fail: write down all failing test cases

Task hw05-3: Running JUnit Tests

1. Then right click the class `cn.edu.hfuu.iao.MainTest` in folder `src/test/java` and select "Run as..." and then "JUnit test".
2. Notice whether any of the tests fail: write down all failing test cases
3. If a test fails, investigate why it fails (read JavaDoc of test methods, compare to code in `Main` class)

Task hw05-3: Running JUnit Tests

1. Then right click the class `cn.edu.hfuu.iao.MainTest` in folder `src/test/java` and select "Run as..." and then "JUnit test".
2. Notice whether any of the tests fail: write down all failing test cases
3. If a test fails, investigate why it fails (read JavaDoc of test methods, compare to code in `Main` class)
4. Fix the `main` method in class `Main`

Task hw05-3: Running JUnit Tests

1. Then right click the class `cn.edu.hfuu.iao.MainTest` in folder `src/test/java` and select "Run as..." and then "JUnit test".
2. Notice whether any of the tests fail: write down all failing test cases
3. If a test fails, investigate why it fails (read JavaDoc of test methods, compare to code in `Main` class)
4. Fix the `main` method in class `Main`
5. Run the JUnit tests again. If everything works: OK, if not: start again at step 1.

Task hw05-3: Running JUnit Tests

1. Then right click the class `cn.edu.hfuu.iao.MainTest` in folder `src/test/java` and select "Run as..." and then "JUnit test".
2. Notice whether any of the tests fail: write down all failing test cases
3. If a test fails, investigate why it fails (read JavaDoc of test methods, compare to code in `Main` class)
4. Fix the `main` method in class `Main`
5. Run the JUnit tests again. If everything works: OK, if not: start again at step 1.
6. The answer to this question is the updated `Main.java` file and a text file with the list of tailing test cases

Task hw05-3: Running JUnit Tests

1. Then right click the class `cn.edu.hfuu.iao.MainTest` in folder `src/test/java` and select "Run as..." and then "JUnit test".
2. Notice whether any of the tests fail: write down all failing test cases
3. If a test fails, investigate why it fails (read JavaDoc of test methods, compare to code in `Main` class)
4. Fix the `main` method in class `Main`
5. Run the JUnit tests again. If everything works: OK, if not: start again at step 1.
6. The answer to this question is the updated `Main.java` file and a text file with the list of tailing test cases
7. Do not worry much about lots of console output, PDFBox is very verbose and might do things such as initializing font caches and so on. You can ignore all of this, even Exceptions printed to the console. The only thing that matters is JUnit output.

Task hw05-4: Performing Maven Build

1. Now everything is done and we can finally perform a Maven build.

Task hw05-4: Performing Maven Build

1. Now everything is done and we can finally perform a Maven build.
2. Inside Eclipse, run the project as Maven build with the goals

```
clean test install package
```

Task hw05-4: Performing Maven Build

1. Now everything is done and we can finally perform a Maven build.
2. Inside Eclipse, run the project as Maven build with the goals
   `clean test install package`
3. You will find a new folder called `target`, containing a file called
   `hw05-task-0.0.1-jar-with-dependencies.jar`

Task hw05-4: Performing Maven Build

1. Now everything is done and we can finally perform a Maven build.
2. Inside Eclipse, run the project as Maven build with the goals
   `clean test install package`
3. You will find a new folder called `target`, containing a file called
   `hw05-task-0.0.1-jar-with-dependencies.jar`
4. This is the stand-alone executable version of our PDF generator. It includes all dependencies.

Task hw05-4: Performing Maven Build

1. Now everything is done and we can finally perform a Maven build.
2. Inside Eclipse, run the project as Maven build with the goals `clean test install package`
3. You will find a new folder called `target`, containing a file called `hw05-task-0.0.1-jar-with-dependencies.jar`
4. This is the stand-alone executable version of our PDF generator. It includes all dependencies.
5. The file `hw05-task-0.0.1-jar-with-dependencies.jar` is the answer to this task.

谢谢

Thank you

Thomas Weise [汤卫思]
tweise@hfuu.edu.cn
http://iao.hfuu.edu.cn

Hefei University, South Campus 2
Institute of Applied Optimization
Shushan District, Hefei, Anhui,
China

Caspar David Friedrich, “Der Wanderer über dem Nebelmeer”, 1818
http://en.wikipedia.org/wiki/Wanderer_above_the_Sea_of_Fog