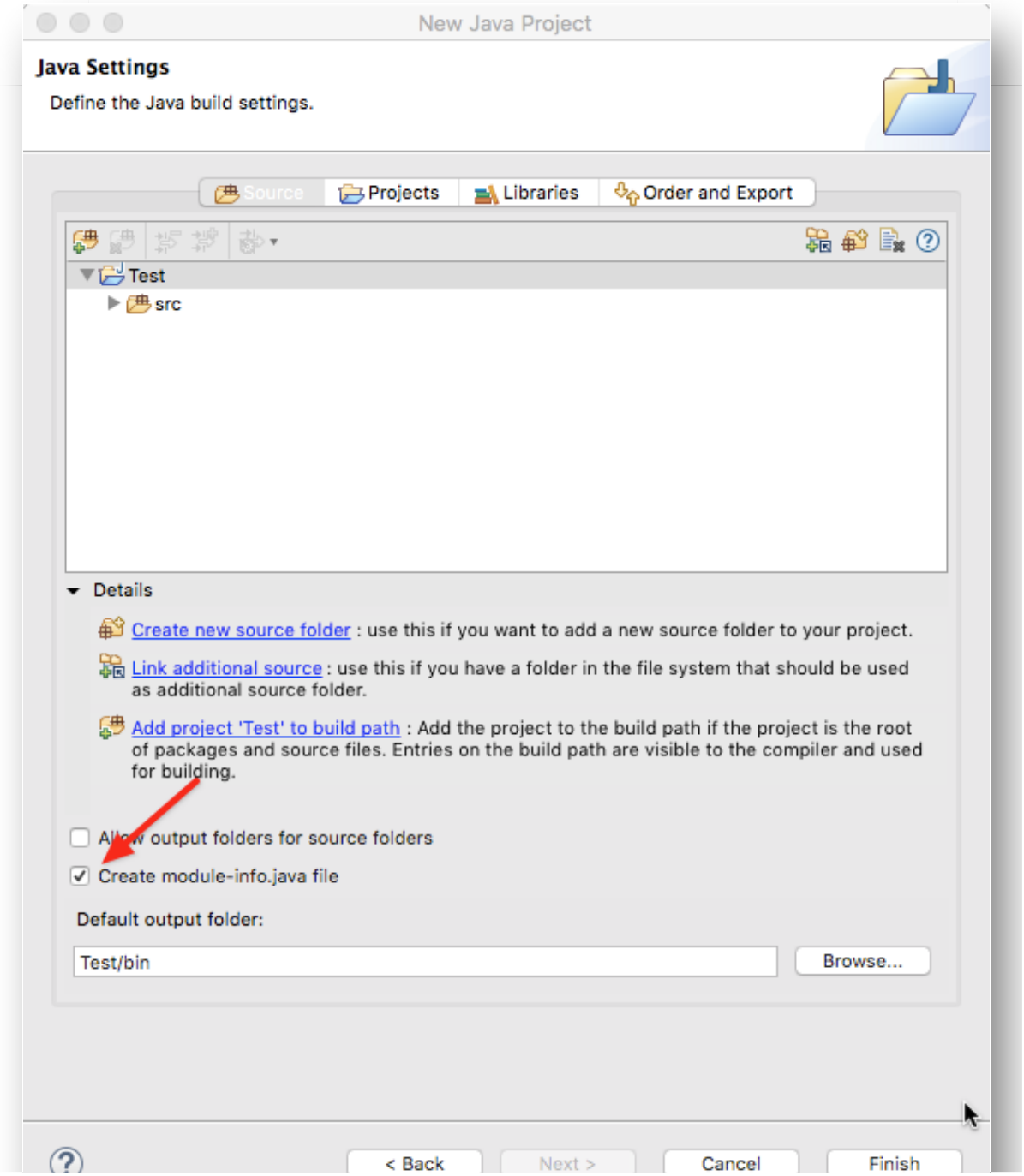
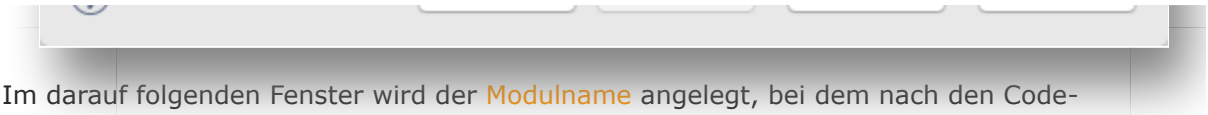
Javabeginners - Module verwenden

Anlegen der Projekte

Bei der Anlage eines neuen Projektes wird im zweiten Fenster abgefragt, ob die Anlage einer Datei `module-info.java` gewünscht wird. Sie dient als Moduldeklaration, in der durch spezielle Direktiven angegeben wird, welche Bestandteile das Modul enthält und auf welche davon und auf welche Weise von außen darauf zugegriffen werden kann.



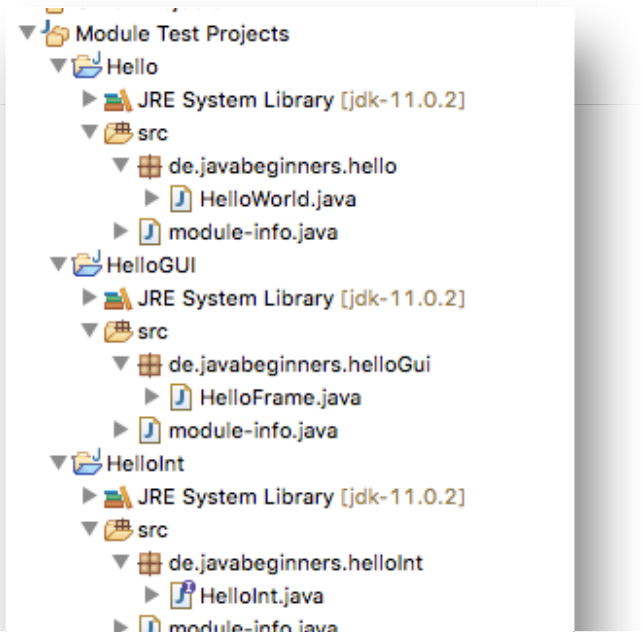


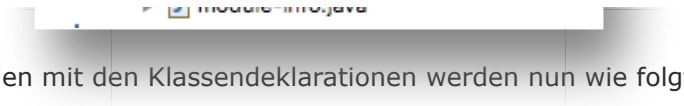
Im darauf folgenden Fenster wird der Modulname angelegt, bei dem nach den Code-Konventionen darauf geachtet werden sollte, dass er mit einem Kleinbuchstaben beginnt und keine Leerzeichen enthält. Wir geben hier 'hello' ein.
Nach Abschluss der Projekterzeugung findet sich `module-info.java` auf oberster Ebene, also direkt im Verzeichnis `src`. Sie kann auch auf einer höheren package-Ebene angelegt werden, muss jedoch zwingend im Wurzelverzeichnis des Moduls liegen. Die Datei enthält einen durch das Schlüsselwort `module` und den Modulnamen eingeleiteten, zunächst noch leeren Block in geschweiften Klammern:

```
module hello {
}
```

Ein Modul kann eine beliebige package-Struktur enthalten, deren Zugriff detailliert gesteuert werden kann.
Im Beispielprojekt wird zunächst das package `de.javabeginners.hello` und darin eine Klasse `HelloWorld` in einer Datei `HelloWorld.java` erzeugt.

Auf die gleiche Weise werden nun zwei weitere Projekte angelegt: `HelloGUI` mit einem Modul `helloGUI`, einem package `de.javabeginners.helloGUI` und der darin deklarierten GUI-Klasse `HelloFrame`, sowie ein Projekt `HelloInt` mit dem Modul `helloInt`, einem package `de.javabeginners.helloInt` und einem Interface `HelloInt`. Zur besseren Übersicht lassen sich die drei Projekte in Eclipse bei Bedarf zu einem Working Set zusammenfassen:
Die Struktur im package manager sieht nun folgendermaßen aus:





Die drei Quelltextdateien mit den Klassendeklarationen werden nun wie folgt ergänzt. Man beachte, dass `HelloFrame` von `JFrame` abgeleitet wird und `HelloInt` implementiert, somit zwei verschiedene Formen der Abhängigkeit vorliegen:

```
package de.javabeginners.hello;

import de.javabeginners.helloGui.HelloFrame;

public class HelloWorld {

    public static void main(String[] args) {
        HelloFrame frame = new HelloFrame();
        frame.setSize(300, 300);
        frame.setDefaultCloseOperation(HelloFrame.EXIT_ON_CLOSE);
        var txt = "Text";
        frame.setText(txt);
        frame.sprich();
        frame.setVisible(true);
    }
}
```

HelloWorld.java

```
package de.javabeginners.helloGui;

import java.awt.GridBagLayout;

import javax.swing.JFrame;
import javax.swing.JLabel;

import de.javabeginners.helloInt.HelloInt;

public class HelloFrame extends JFrame implements HelloInt {
    private JLabel label;

    public HelloFrame() {
        label = new JLabel();
        this.add(label);
        this.setLayout(new GridBagLayout());
    }
```

```
    public void setText(String txt) {
        label.setText(txt);
    }

    public void sprich() {
        System.out.println("Hallo Welt!");
    }
}
```

HelloFrame.java

```
package de.javabeginners.helloInt;

public interface HelloInt {
public void sprich();
}
```

HelloInt.java

Nach Ergänzung der Dateien werden in den Quelltexten eine Reihe an Fehlern durch rote Schlangenlinien angezeigt. Es fällt auf, dass selbst die korrekt angegebenen `import`-Anweisungen derartig gekennzeichnet sind.
Die Ursache liegt darin, dass in modularisierten Projekten die benötigten Zugriffe in der zugehörigen Datei `module-info.java` geregelt werden müssen. Dies gilt auch für die Bausteine der JDK-Bibliothek.
Ergänzt man die Datei `module-info.java` des Projektes `HelloGUI` folgendermaßen

```
module helloGUI {
    requires java.desktop;
}
```

so verschwindet diese Fehleranzeige in der Klasse `HelloFrame`. Der Zugriff auf die Elemente des Moduls `java.desktop`, hier die benötigten AWT- und Swing-Bibliotheken, wird durch die Angabe des Moduls nach dem Schlüsselwort `requires` in `module-info.java` auf diese Weise erlaubt. Das Schlüsselwort `requires` kennzeichnet die Abhängigkeit eines Moduls von einem anderen.
Ein Überblick über die hier möglichen Schlüsselworte und deren Funktionen findet sich im Abschnitt Schlüsselworte der Moduldirektiven.

In der Klasse `HelloWorld` kennzeichnet der Compiler alle Einträge als fehlerhaft, die die

In der Klasse `HelloWorld` kennzeichnet der Compiler alle Einträge als fehlerhaft, die die Klasse `HelloFrame` referenzieren. Man vermutet zurecht, dass ein entsprechender Eintrag in `module-info.java` des Moduls `hello` benötigt wird, um Zugriff auf die Klasse zu erhalten:

```
module hello {
    requires helloGUI;
}
```

Erstaunlicherweise behebt dieser Eintrag jedoch nicht alle Fehleranzeigen. Das ist erst dann der Fall, wenn das package `de.javabeginners.helloGui` explizit in der zugehörigen Moduldeklaration zum Export vorgesehen wird:

```
module helloGUI {
    requires java.desktop;
    exports de.javabeginners.helloGui;
}
```

Nun fehlt noch die Behebung eines Fehlers in der Klasse `HelloFrame`. Es verwundert nun nicht mehr, dass der Import des Interfaces noch markiert ist: Das package muss noch exportiert werden:

```
module helloInt {
    exports de.javabeginners.helloInt;
}
```

Der Fehler wird hierdurch jedoch immer noch nicht vollständig behoben, da die Nutzung des Interfaces durch das Modul `helloGUI` noch angezeigt werden muss:

```
module helloGUI {
    exports de.javabeginners.helloGui;
    requires java.desktop;
    requires helloInt;
    uses de.javabeginners.helloInt.HelloInt;
}
```

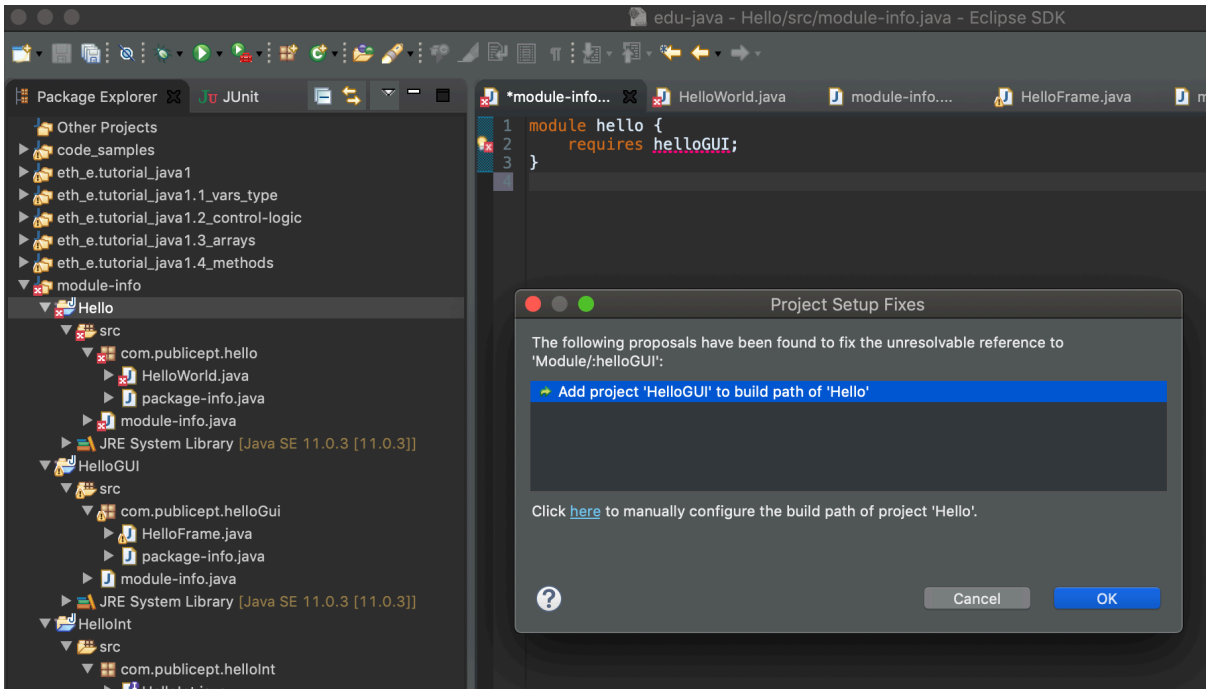
Bei der Nutzung des Interface muss sowohl eine `requires`-, als auch eine `uses`-Direktive

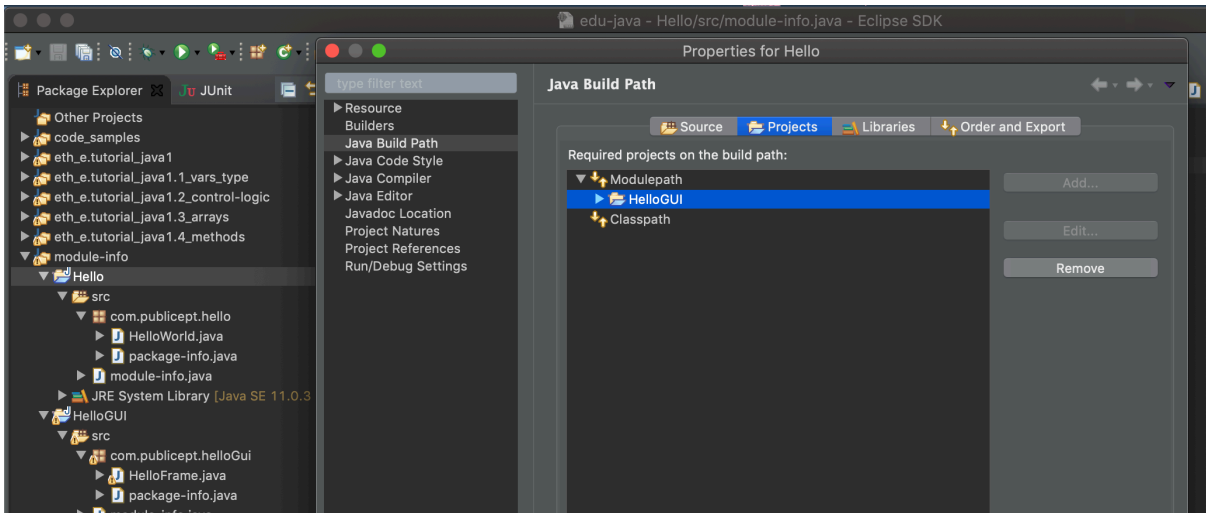
angegeben werden. `requires` kennzeichnet hier wiederum die Abhängigkeit des einen Moduls von dem anderen, `uses` bezeichnet den 'Service', hier also das Implementieren des Interface `HelloInt`. Statt der `uses`-Direktive könnte hier übrigens auch

```
provides de.javabeginners.helloInt.HelloInt with
de.javabeginners.helloGui.HelloFrame;
```

angegeben werde.

Project build path hinzufügen:





Die JDK-Module

Auch wenn Eclipse mit dem Content-Assistenten beim Fehlen einer Modul-Direktive die notwendigen Vorschläge macht, kann es nicht schaden, gelegentlich einen Überblick über die zur Verfügung stehenden JDK-Module zu erhalten.

Fanden sich in den Java-Versionen vor der Version 9 die Bibliotheken innerhalb des Verzeichnisses $JAVA_HOME/jre/lib, so wurde, nach Einführung der Modularisierung mit Java 9, das Verzeichnis jre durch jmods ersetzt. In diesem befinden sich die JDK-Module. Deren Auflistung kann über die Kommandozeile durch Aufruf der ausführbaren Datei java im bin-Verzeichnis der Installation mit der Option --list-modules erfolgen:

```
# java --list-modules
```

Eine ausführliche Erläuterung der einzelnen Module findet sich dann in der Java-Dokumentation der verwendeten JDK-Version.

Schlüsselworte der Moduldirektiven

Die im Folgenden gelisteten Schlüsselworte sind im Kontext einer Moduldeklaration für die jeweilige Funktion reserviert und dürfen nicht in einem anderen Kontext verwendet werden.

requires <modulname>

requires transitive <modulname

exports <modulname>

uses <modulname>

provides <type> with <class>

opens <package>

opens <package> to <modullist

open module <modulname>

Quellen

http://www.javamagazine.mozaicreader.com/SeptOct2017#&pageSet=18&page=0