Transfer der resultierenden `de.firma.moda-1.0.jar` zum Computer von Team 2 in das modules-Verzeichnis und dann auf dem zweiten Computer:

```
1. javac -p modules -d classes\de.firma.modmain
     ↪ src\de.firma.modmain\*.java
     ↪ src\de.firma.modmain\de\firma\modmain\*.java
2. jar --create
     ↪ --file modules/de.firma.modmain-1.0.jar
     ↪ --module-version 1.0
     ↪ --main-class de.firma.modmain.modmain
     ↪ -C classes/de.firma.modmain .
```

Ausführung der Anwendung:

```
java -p modules -m de.firma.modmain
```

Was in der praktischen Arbeit mit Modulen immer wieder vorkommt, ist der Umstand, dass verschiedene Module, die gleichen Abhängigkeiten benötigen. Bei der Modellierung solcher Beziehungen, hilft die Unterstützung von transitiven Abhängigkeiten durch das Modulsystem, wie sie im folgenden Kapitel vorgestellt wird.

3.2.2 Transitive Abhängigkeiten

Die im vorangegangenen Beispiel verwendeten Module `de.firma.modmain` und `de.firma.moda` werden um die beiden weiteren Module `de.firma.modb` und `de.firma.modc` ergänzt, wobei Modul `de.firma.moda` von diesen beiden abhängt. Modul `de.firma.modmain` soll in diesem Beispiel nun ebenfalls Klassen aus `de.firma.modc` verwenden können. Dies lässt sich durch die Deklaration einer transitiven Abhängigkeit modellieren. Abbildung 3–11 zeigt den Modulgraphen.

Zunächst wird eine passende Verzeichnisstrukur erzeugt:

```
1. cd \MeinWorkspace
2. mkdir FourModulesTransDep
3. cd FourModulesTransDep
4. mkdir src\de.firma.modmain\de\firma\modmain
5. mkdir src\de.firma.moda\de\firma\moda
6. mkdir src\de.firma.modb\de\firma\modb
7. mkdir src\de.firma.modc\de\firma\modc
```

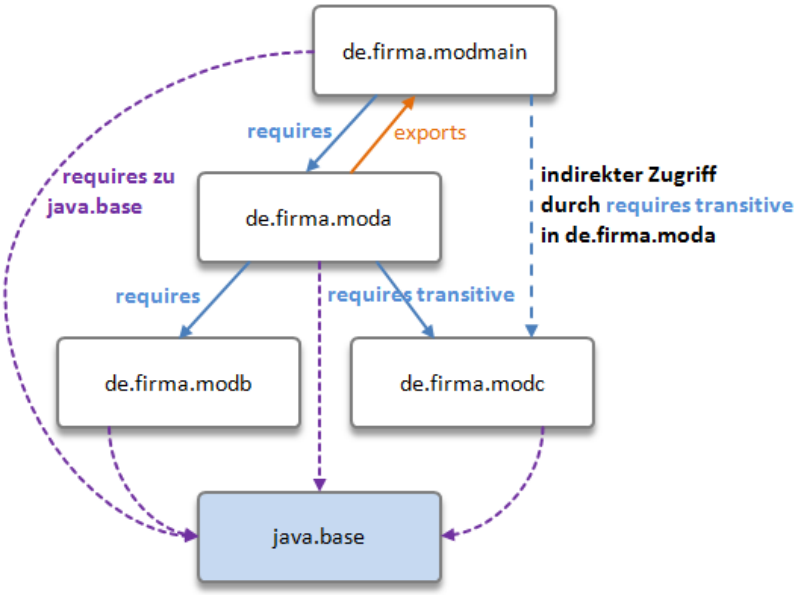

Abb. 3–11
Deklaration und Implementierung eines Java Moduls

Im Moduldeskriptor von Modul `de.firma.modmain` hat sich nichts geändert, wie folgendes Listing zeigt:

Listing 3–6
module-info.java

```
module de.firma.modmain {
   // benötigtes Modul
   requires de.firma.moda;
}
```

Der Moduldeskriptor von Modul `de.firma.moda` wurde um die beiden benötigten Abhängigkeiten erweitert:

Listing 3–7
module-info.java

```
module de.firma.moda {
   // benötigtes Module
   requires de.firma.modb;
   requires transitive de.firma.modc;

   // exportiertes Paket
   exports de.firma.moda;
}
```

Zudem zeigt das Listing eine mit **requires transitive** deklarierte transitive Abhängigkeit. Dadurch wird das Modul `de.firma.modc` für jedes andere Modul zugreifbar, welches Modul `de.firma.moda` importiert.

Transitive Abhängigkeiten helfen bei einer sauberen Strukturierung, da alle neben `moda` für `modmain` noch benötigten Abhängigkeiten vollständig mit der Deklaration von `moda` gekapselt werden können. Des

Weiteren hilft dieser Mechanismus bei der Erzeugung von Aggregator-Modulen; also Module, die einzig und alleine dafür da sind, eine Menge bestimmter Module zusammenzuführen und gebündelt nach außen anzubieten. Dadurch müssen anderen Module nur dieses eine Aggregatormodul lesen, ohne eine ganze Phalanx an Abhängigkeiten deklarieren zu müssen. Als Beispiele wären hier bei einer Schichtenarchitektur die mögliche Zusammenfassung einer Schicht als Modul zu nennen oder die mit Java 8 eingeführten Compact-Profiles, die im Kapitel über das modularisierte JDK nochmals genauer betrachtet werden.

Die entsprechenden Moduldeskriptoren der Module `de.firma.modb` und `de.firma.modc` sehen dabei wie folgt aus:

Listing 3–8 *module-info.java Dateien*

```
module de.firma.modb {
   // exportiertes Paket
   exports de.firma.modb;
}

module de.firma.modc {
   // exportiertes Paket
   exports de.firma.modc;
}
```

Im vorliegenden Beispiel bekommen die Module `de.firma.modb` und `de.firma.modc` noch jeweils eine Klasse. Modul `de.firma.modb` enthält die Klasse `TestB`:

Listing 3–9 *TestB.java*

```
package de.firma.modb;

public class TestB {
         public static String getName() {
      return TestB.class.getName();
   }
}
```

Und Modul `de.firma.modc` enthält die Klasse `TestC`:

Listing 3–10 *TestC.java*

```
package de.firma.modc;

public class TestC {
   public static String getName() {
      return TestC.class.getName();
   }
}
```

Die Main-Methode der Startklasse `App` wird nun um einen Zugriff auf das Modul `de.firma.modc` erweitert, obwohl in dem Moduldeskriptor von `de.firma.modmain` keine entsprechende Abhängigkeit explizit deklariert wurde:

Listing 3–11
App.java

```
package de.firma.modmain;

import de.firma.moda.TestA;
import de.firma.modc.TestC;

public class App {
  public static void main(String[] args) {
    System.out.println("Name des importierten
                Moduls A: " + TestA.getName());
                System.out.println("Name des transitiv
                erreichbaren Moduls C: " + TestC.getName());
  }
}
```

Transitive Abhängigkeiten

Transitive Abhängigkeiten

Transitive Abhängigkeiten werden im Moduldeskriptor `module-info.java` mit **requires transitive** `de.firma.meinModul` deklariert. Wenn die Abhängigkeit nur für bestimmte Module weitergereicht werden soll, kann auch **requires transitive to** geschrieben werden, mit dem entsprechenden Modul hinter **to**.

Beispiel:

```
module de.firma.moda {
  requires transitive de.firma.modb;
  requires transitive de.firma.modc to de.firma.modmain;
}
```

Neben den fest deklarierten Abhängigkeiten zwischen Modulen, die bereits statisch während der Kompilierung vorliegen, gibt es auch einen Mechanismus, wo die direkte Abhängigkeit erst zur Laufzeit aufgelöst wird und was Thema des nächsten Kapitels ist.

3.3 Services

Seit Java SE 6 existiert in der Spezifikation ein Service-Provider-Mechanismus, der eine bessere Entkopplung zwischen Klassen unterstützt, indem Klassenabhängigkeiten erst zur Laufzeit aufgelöst werden. Dadurch wird eine erheblich stärkere Entkopplung zwischen den Provider-Klassen und den später auf diese zugreifenden Klassen erreicht. Letzte-