PROGRAMMIEREN LERNEN MIT JAVA

BONUSKAPITEL - REFLECTION

MALTE LUTTERMANN

Reflection

Reflection erlaubt es unserem Programm, seinen eigenen Programmcode zu untersuchen und zu manipulieren. Dies kann sehr interessant sein, wenn man beispielsweise Informationen über Klassen während der Laufzeit des Programms herausfinden möchte.

Hinweis

Das in diesem Bonuskapitel vorgestellte Konzept ist für fortgeschrittene Entwickler, als Anfänger braucht man es in der Regel nicht.

Informationen über Klassen erhalten

Ein Beispiel: Wir möchten wissen, welche Methoden es alles in einer Klasse gibt. Dies wollen wir jedoch während der Laufzeit des Programms tun, weshalb wir dazu Reflection nutzen.

```
import java.lang.reflect.*;

public class Main {
    public static void main( String args[] ) {
        try {
            Class c = Class.forName( "Demo" );
            Method m[] = c.getDeclaredMethods();
            for( int i = 0; i < m.length; i++ ) {
                System.out.println( m[i].toString() );
            }
        }
        catch( ClassNotFoundException e ) {
            e.printStackTrace();
        }
        catch( ArrayIndexOutOfBoundsException e ) {
            e.printStackTrace();
        }
    }
}
```

Dazu brauchen wir zunächst ein Objekt vom Typ `Class`, über das wir die Methoden der angegebenen Klasse bekommen können. Dabei kann es passieren, dass die angegebene Klasse nicht gefunden wird. Um dies zu verhindern, sollte die angegebene Klasse zuerst kompiliert werden, bevor das Programm gestartet wird.

Sobald dies erfolgreich geschehen ist, können wir dieses Beispiel ausführen. Dabei sieht unsere Klasse `Demo` folgendermaßen aus:

```
public class Demo {
    public void methode1() {}
    public int methode2() {return 0;}
    public boolean methode3( int a, int b ) {return true;}
    public void methode4( int a ) {}
}
```

(Die Methoden sind hier nicht sinnvoll erstellt worden, sondern sollen nur zu Demonstrationszwecken dienen)

Dann erhalten wir die folgende Ausgabe:

```
public void Demo.methode1()
public int Demo.methode2()
public void Demo.methode4(int)
public boolean Demo.methode3(int,int)
```

(Dabei kann sich die Reihenfolge der Methoden jedes Mal, wenn wir das Programm ausführen, verändern)

Informationen über Methoden erhalten

Wir können nicht nur Informationen über Klassen, sondern auch Informationen über Methoden erhalten.

```
import java.lang.reflect.*;

public class Main {
    public static void main( String args[] ) {
        try {
            Class c = Class.forName( "Demo" );
            Method m[] = c.getDeclaredMethods();
            for( int i = 0; i < m.length; i++ ) {
                System.out.println( "Klasse: " + m[i].
                   getDeclaringClass() );
                System.out.println( "Methode: " + m[i].getName() );
                System.out.println( "Typen der Parameter: " );
                Class[] params = m[i].getParameterTypes();
                for ( int j = 0; j < params.length; j++ ) {
                    System.out.println( params[j] );
                }
                System.out.println( "Rueckgabetyp: " + m[i].
                   getReturnType() );
            }
        }
        catch( ClassNotFoundException e ) {
            e.printStackTrace();
        }
        catch( ArrayIndexOutOfBoundsException e ) {
            e.printStackTrace();
```

```
        }
    }
}
```

Ausgabe:

```
Klasse: class Demo
Methode: methode4
Typen der Parameter:
int
Rueckgabetyp: void
Klasse: class Demo
Methode: methode1
Typen der Parameter:
Rueckgabetyp: void
Klasse: class Demo
Methode: methode2
Typen der Parameter:
Rueckgabetyp: int
Klasse: class Demo
Methode: methode3
Typen der Parameter:
int
int
Rueckgabetyp: boolean
```

Informationen über Konstruktoren erhalten

Auf dieselbe Art und Weise kann man auch Informationen über Konstruktoren bekommen. Dafür nutzt man einfach die Methode `getDeclaredConstructors()` anstelle von `getDeclaredMethods()`. Die weiteren Informationen kann man dann mit denselben Methoden wie im obigen Beispiel bekommen.

Informationen über Attribute erhalten

Informationen über Attribute lassen sich auch herausfinden, dazu nutzen wir die Methode `getDeclaredFields()`. Für das nächste Beispiel passen wir zuerst einmal unsere Klasse `Demo` an:

```
public class Demo {
    private int z;
    protected String s;
    public boolean b;
    double d;
}
```

Da unsere Klasse `Demo` nun Attribute besitzt, können wir jetzt auch Informationen darüber bekommen (nicht vergessen: Bevor wir das eigentliche Programm ausführen, müssen wir die Klasse `Demo` erneut kompilieren!).

```
import java.lang.reflect.*;

public class Main {
    public static void main( String args[] ) {
        try {
            Class c = Class.forName( "Demo" );
            Field f[] = c.getDeclaredFields();
            for( int i = 0; i < f.length; i++ ) {
                System.out.println( "Klasse: " + f[i].
                   getDeclaringClass() );
                System.out.println( "Attribut: " + f[i].getName() );
                System.out.println( "Datentyp: " + f[i].getType() );
                System.out.println( "Zugriffsmodifikator: " +
                   Modifier.toString(f[i].getModifiers()) );
            }
        }
        catch( ClassNotFoundException e ) {
            e.printStackTrace();
        }
        catch( ArrayIndexOutOfBoundsException e ) {
            e.printStackTrace();
        }
    }
}
```

Ausgabe:

```
Klasse: class Demo
Attribut: z
Datentyp: int
Zugriffsmodifikator: private
Klasse: class Demo
Attribut: s
Datentyp: class java.lang.String
Zugriffsmodifikator: protected
Klasse: class Demo
Attribut: b
Datentyp: boolean
Zugriffsmodifikator: public
Klasse: class Demo
Attribut: d
Datentyp: double
Zugriffsmodifikator:
```

Nun haben wir gelernt, wie wir Informationen über Klassen, Konstruktoren, Metho-

den und Attribute bekommen können.

Methoden mittels Reflection aufrufen

Jetzt schauen wir uns folgendes Szenario an: Angenommen, wir möchten während der Laufzeit eine Methode ausführen, wir wissen aber erst während der Laufzeit des Programms den Namen dieser Methode. Wir verwenden Reflection, um dies zu tun.

Zunächst einmal erstellen wir uns eine Methode für das kommende Beispiel:

```
public class Demo {
    public int add( int a, int b ) {
        return a + b;
    }
}
```

Dann suchen wir eine Methode mit dem Namen `add`, die zwei Parameter vom Datentyp `Integer` erwartet. Daraufhin können wir diese Methode aufrufen:

```
import java.lang.reflect.*;

public class Main {
    public static void main( String args[] ) {
        try {
            Class c = Class.forName( "Demo" );
            Class types[] = { Integer.TYPE, Integer.TYPE };
            Method m = c.getMethod( "add", types );
            Demo d = new Demo();
            Object params[] = { new Integer(14), new Integer(15) };
            Object o = m.invoke( d, params );
            System.out.println( ((Integer) o).intValue() );
        }
        catch( NoSuchMethodException e ) {
            e.printStackTrace();
        }
        catch( IllegalAccessException e ) {
            e.printStackTrace();
        }
        catch( ClassNotFoundException e ) {
            e.printStackTrace();
        }
        catch( InvocationTargetException e ) {
            e.printStackTrace();
        }
    }
}
```

Ausgabe:

```
29
```

Mittels Reflection ist es ebenfalls möglich, private Methoden aufzurufen. Dabei muss sich die die gesuchte Methode jedoch direkt in der Klasse befinden, auf die wir zu-

greifen wollen, d.h. wir können nicht auf vererbte Methoden zugreifen. Darüber hinaus könnte es passieren, dass ein `SecurityManager` verhindert, dass wir die Methode `setAccessible( boolean )` aufrufen (dafür gibt es auch Lösungen, auf die wir hier jedoch nicht näher eingehen werden).

Ändern wir also zunächst die Klasse `Demo` ab.

```
public class Demo {
    private void methode() {
        System.out.println( "methode() wurde ausgefuehrt." );
    }
}
```

Nun können wir die private Methode folgendermaßen aufrufen:

```
import java.lang.reflect.*;

public class Main {
    public static void main( String args[] ) {
        try {
            Demo d = new Demo();
            Method m = d.getClass().getDeclaredMethod( "methode" );
            m.setAccessible( true );
            Object r = m.invoke( d );
        }
        catch( NoSuchMethodException e ) {
            e.printStackTrace();
        }
        catch( InvocationTargetException e ) {
            e.printStackTrace();
        }
        catch( IllegalAccessException e ) {
            e.printStackTrace();
        }
    }
}
```

Ausgabe:

```
methode() wurde ausgefuehrt.
```

Konstruktoren mittels Reflection aufrufen

Auf ähnliche Art und Weise ist es auch möglich, neue Objekte über Konstruktoren zu erzeugen. Wie immer, ändern wir zunächst die Klasse `Demo` ein wenig ab.

```
public class Demo {
    public Demo() {
        System.out.println( "Parameterloser Konstruktor" );
    }
    public Demo( int a ) {
        System.out.println( "Angegebene Zahl: " + a );
    }
}
```

Daraufhin können wir folgenden Code nutzen, um neue Objekte über Konstruktoren zu erzeugen.

```
import java.lang.reflect.*;

public class Main {
    public static void main( String args[] ) {
        try {
            Class c = Class.forName( "Demo" );
            Class types = Integer.TYPE;
            Constructor ct = c.getConstructor( types );
            Object params = new Integer(12);
            Object obj = ct.newInstance( params );
        }
        catch( InstantiationException e ) {
            e.printStackTrace();
        }
        catch( ClassNotFoundException e ) {
            e.printStackTrace();
        }
        catch( NoSuchMethodException e ) {
            e.printStackTrace();
        }
        catch( IllegalAccessException e ) {
            e.printStackTrace();
        }
        catch( InvocationTargetException e ) {
            e.printStackTrace();
        }
    }
}
```

```
Angegebene Zahl: 12
```

Mittels Reflection auf Attribute zugreifen

Ebenfalls können wir Reflection verwenden, um auf Attribute zuzugreifen.

```
public class Demo {
    public int y = 100;
    private int z = 20;
}
```

```java
import java.lang.reflect.*;

public class Main {
    public static void main( String args[] ) {
        try {
            Class c = Class.forName( "Demo" );
            Field f = c.getField( "y" );
            Demo d = new Demo();
            f.setInt( d, 42 );
            System.out.println( "y=" + d.y );
            // privates Attribut
            f = d.getClass().getDeclaredField( "z" );
            f.setAccessible( true );
            f.setInt( d, 42 );
            System.out.println( "z=" + f.get( d ) );
        }
        catch( NoSuchFieldException e ) {
            e.printStackTrace();
        }
        catch( ClassNotFoundException e ) {
            e.printStackTrace();
        }
        catch( IllegalAccessException e ) {
            e.printStackTrace();
        }
    }
}
```

```
y=42
z=42
```

Anmerkungen

Referenzen

Dieses Bonuskapitel wurde basierend auf folgendem Tutorial erstellt:

- »Using Java Reflection«, `https://www.oracle.com/technetwork/articles/java/javareflection-1536171.html`, zuletzt aufgerufen am 17.09.2018

Weitere Referenzen:

- »java - What is reflection and why is it useful? - Stack Overflow«, `https://stackoverflow.com/questions/37628/what-is-reflection-and-why-is-it-useful`, zuletzt aufgerufen am 15.09.2018
- »java - Any way to Invoke a private method? - Stack Overflow«, `https://stackoverflow.com/questions/880365/any-way-to-invoke-a-private-method`, zuletzt aufgerufen am 17.09.2018