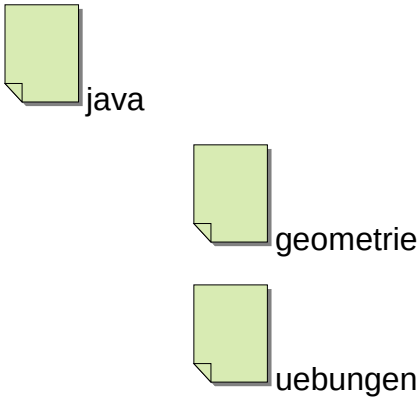
Packages

Packages dienen in größeren Softwareprojekten der Organisation von Klassen. Packages korrespondieren dabei mit der Ordnerstruktur des jeweiligen Betriebssystems.



geometrie und *uebungen* sind Unterordner von *java*. Im Ordner *geometrie* könnten sich die Klassen Rechteck, Quader und Kreis befinden. Im Ordner *uebungen* befänden sich z.B. die Klassen TestRechteck, TestQuader und TestKreis.

Der Startpunkt der Ordnerstruktur lässt sich über die Umgebungsvariable *classpath* festlegen. Unter Windows 10 funktioniert es folgendermaßen:

1. Gleichzeitiges Drücken der Windows- und Pause-Taste
2. Erweiterte Systemeinstellungen
3. Umgebungsvariablen
4. Benutzervariablen: Neu
5. Name der Variablen: CLASSPATH
6. Wert der Variablen: z.B. u:\semester2\

In diesem Beispiel wäre das Verzeichnis *semester2* direkt oberhalb des Verzeichnisses *java*.

Die Klassen heißen jetzt nicht mehr Rechteck oder Quader oder Kreis, sondern java.geometrie.Rechteck, java.geometrie.Quader und java.geometrie.Kreis. Entsprechend verhält es sich mit den Klassen im Package *uebungen*.

Beispiel:

```
package java.geometrie;
class Rechteck {
. . .
}
```

Mittels der Package-Anweisung wird bestimmt, in welchem Package sich die Klasse Rechteck befindet.

```
package java.uebungen;
class TestRechteck {                                    // Alternative 1
    public static void main(String[] args) {
        java.geometrie.Rechteck R1;
```

```
        R1 = new java.geometrie.Rechteck();
        . . .
        }
    }
```

Jeder Package-Name (und damit auch der Verzeichnis-Name) besteht nur aus Kleinbuchstaben. Keine Klasse darf so heißen wie eines der Packages. Packages und ihre Unterpackages sind durch einen Punkt getrennt.

In der Testklasse muss in unserem Beispiel für die Klasse Rechteck der sogenannte full-qualified-name java.geometrie.Rechteck benutzt warden. Über eine import-Anweisung lässt sich wieder der normale Klassenname nutzen:

```
package java.uebungen;
import java.geometrie.Rechteck;
class TestRechteck {                                    // Alternative 2
    public static void main(String[] args) {
        Rechteck R1;
        R1 = new Rechteck();
        . . .
        }
    }
```

Zur Erinnerung: Ohne Packages würde die Laufzeitumgebung von Java eine Klasse im aktuellen Verzeichnis suchen.

Bemerkungen zum Import:

import completePackageName.ClassName;

→ importiert nur die eine Klasse

import completePackageName.*;

→ importiert alle Klassen aus dem Package

→ importiert keine Klassen aus Unter-Packages

Klassen aus Unter-Packages müssen mit eigener import-Anweisung importiert werden

Besitzen Klassen aus unterschiedlichen Packages denselben Namen ⇒ verwenden Sie bitte den full-qualified-name, um Mehrdeutigkeiten zu vermeiden

Start eines Programms, das sich in einem Package befindet: nur über den full-qualified-name

→ java java.uebungen.TestRechteck

Zugriffsrechte in Packages

```
class Rechteck {                        // Zugriff nur aus Package
    private int Breite;                 // Zugriff nur aus Klasse
    private int Laenge;                 // Zugriff nur aus Klasse

    Rechteck() {                        // Zugriff nur aus Package
        Breite = 1;
        Laenge = 1;
        }

    int berechneFlaeche() {             // Zugriff nur aus Package
        return Breite * Laenge;
        }
    . . .
    }
```

Wie schon bekannt, erlaubt der Zugriffsmodifizierer *private* nur einen Zugriff aus der eigenen Klassen. Ist kein Zugriffsmodifizierer angegeben, ist der Zugriff aus allen Klassen desselben Packages erlaubt.

Falls aus Klassen anderer Packages zugegriffen werden soll, muss der Zugriffsmodifizierer *public* verwendet werden.

```
public class Rechteck {
    . . .
    public Rechteck() {
        . . .
        }

    public int berechneFlaeche() {
        . . .
        }
    . . .
}
```

Bemerkungen zu Zugriffsrechten:

- beim Vererben bleiben die Zugriffsrechte erhalten
- Zugriffsrechte sollten sorgsam ausgewählt werden.

JDK Packages

Java selbst bringt eine große Anzahl von Klassen mit sich, die in unterschiedlichen Packages organisiert sind. Diese Klassen sind in Java-Archivdateien gepackt und werden bei der Installation von Java automatisch im System gespeichert. Für die Nutzung dieser Klassen ist kein CLASSPATH notwendig.

Java Packages (Auszug):

java.lang	→ Standard-Funktionalitäten, z.B. Object → wird automatisch importiert
java.io	→ Datenein- und -ausgabe
java.math	→ Mathematische Funktionen
java.net	→ Netzwerkzugriffe und -kommunikation
java.util	→ Datenstrukturen, Hilfsklassen
javax.awt	→ grafische Benutzeroberfläche
javax.swing	→ grafische Benutzeroberfläche

Dokumentation der JDK-Packages

- siehe z.B. http://docs.oracle.com/javase/8/docs/api/

Interfaces

- Ein Interface enthält nur Methodenköpfe und Konstanten
- Methodenköpfe sind automatisch public

[public] interface identifier [extends interface1 [, interface2 …]] {

. . .

}

Beispiel: java\geometrie\Forms2D.java

```
package java.geometrie;

public interface Forms2D {

        double berechneFlaeche();

        double berechneUmfang();

        }
```

Ein Interface ist eine Vorschrift. In unserem Beispiel schreibt das Interface vor, welche Methoden für ein 2D-Objekt mindestens vorhanden sein müssen.

Verwendung:
class Rechteck implements Forms2D
→ alle Interface-Methoden müssen implementiert werden

Eine Klasse kann mehrere Interfaces implementieren
class BlaBla implements Forms2D, GrafikObjekt

Verwendung: java\geometrie\Rechteck.java

```
package java.geometrie;

public class Rechteck implements java.geometrie.Forms2D {

        . . .

        public double berechneFlaeche() {

                return getBreite() * getLaenge();

                }

        public double berechneUmfang() {

                return 2.0 * (getBreite() + getLaenge());

                }

        }
```

Verwendung: java\geometrie\Kreis.java

```
package java.geometrie;
public class Kreis implements java.geometrie.Forms2D {
    . . .
    public double berechneFlaeche() {
        return Math.PI * getRadius() * getRadius();
        }
    public double berechneUmfang() {
        return 2.0 * Math.PI + getRadius();
        }
    }
```

Interface Polymorphismus

Wie schon im Fall der Vererbung von Klassen gibt es Polymorphismus auch im Zusammenhang mit Interfaces.

```
class GeometrischeObjekte {
    public static void main(String[] args) {
        Forms2D R1;
        R1 = new Rechteck();
        Forms2D K1;
        K1 = new Kreis();
        . . .
        }
    }
```

Default Methoden:

Bis zur Version Java 7 standen in einem Interface nur Methodenköpfe. Seit Java 8 kann ein Interface unter Anderem auch Default-Methoden enthalten. Eine Default-Methode ist eine Methode mit einer Default-Implementierung.

```
package java.geometrie;
public interface Forms2D {
    default double berechneFlaeche() {return 0;}
    default double berechneUmfang() {return 0;}
    }
```

Hintergrund dieser neuen Möglichkeit ist folgender:

Ohne Default-Methoden ist es mitunter sehr aufwändig bis unmöglich, ein bestehendes Interface um eine Methode zu erweitern, denn das würde bedeuten, dass weltweit alle Klassen, die dieses Interface implementiert haben, angepasst werden müssen (die zusätzliche Methode muss in allen Klassen implementiert werden), auch wenn diese Klassen diese neue Methode gar nicht benötigen.

Das Problem mit dieser Neuerung ist, dass es dem Compiler nicht mehr auffält, wenn Sie z.B. in der Klasse Rechteck vergessen, die Methode berechneUmfang() zu implementieren (der Compiler sieht die Default-Implementierung und daher ist es für den Compiler ok). Damit wird der Grundgedanke von Interfaces, eine Vorschrift darzustellen, welche Methoden in einer Klasse unbedingt implementiert werden müssen, verwässert.

Meine Empfehlung lautet daher: Verzichten Sie in der Regel auf Default-Methoden.

Unterschiede Interface – abstrakte Klasse

- Eine abstrakte Klasse dient in der Regel als Mutterklasse für abgeleitete Klassen und wird mit *extends* geerbt.
- Ein Interface ist eine Vorschrift, welche Methoden unbedingt in der Klasse, die dieses Interface mittels *implements* implementiert, programmiert werden müssen.
- Alle Methoden eines Interfaces sind quasi "abstract", da in einem Interface nur Methodenköpfe stehen.
- Ein Interface hat keine Variablen bzw. Attribute.
- Eine Klasse kann mehrere Interfaces implementieren.
- Eine Klasse kann von einer anderen Klasse erben und zusätzliche Interfaces implementieren.