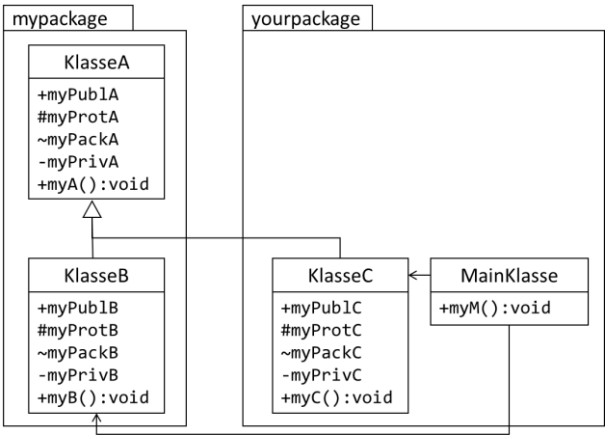
7 Package

Bevor wir in Java mit der Kapselung und somit den Zugriffsmodifikatoren beginnen, müssen wir das Konzept des Packages kurz klären. Ein Package ist eine Sammlung von zusammengehörigen Klassen. Packages werden in Java mit dem Schlüsselwort `package` markiert und der zugehörige Code muss in einem Unterordner liegen. Packagenamen sind üblicherweise aus Kleinbuchstaben aufgebaut und sollten idealerweise global eindeutig sein, weshalb man oftmals die Firmen-URL einbaut (bspw. `com.oracle.…`). Die Nutzung der Packages durch „fremde" Klassen erfolgt durch den import Befehl. Im Zusammenhang mit den Zugriffsmodifikatoren ist es nun wichtig, wer von welcher Klasse auf welche Eigenschaften zugreifen darf:



8 Zugriffsmodifikatoren

In Java existieren 4 Zugriffsmodifikatoren:

Name	Symbol	Bedeutung
`public`	+	Jeder darf von außen auf die Eigenschaften/Methoden zugreifen.
`protected`	#	Nur Nachfahren dürfen zugreifen
`package`	~	Alle innerhalb des Packages dürfen zugreifen
`private`	-	Nur die Klasse selbst darf zugreifen

Gehen wir vom oben gezeigten Klassendiagramm aus. Markieren Sie, welche Variablennutzungen einen Compilerfehler hervorrufen würden:

```
public void myA() {
  int val1 =  myPublA + myPrivA + myProtA + myPackA;

  KlasseA valA = new KlasseA();
  int val2 = valA.myPublA + valA.myPrivA + valA.myProtA + valA.myPackA;

  KlasseB valB = new KlasseB();
  int val3 = valB.myPublA + valB.myPrivA + valB.myProtA + valB.myPackA;
  int val4 = valB.myPublB + valB.myPrivB + valB.myProtB + valB.myPackB;

  KlasseC valC = new KlasseC();
  int val5 = valC.myPublA + valC.myPrivA + valC.myProtA + valC.myPackA;
  int val6 = valC.myPublC + valC.myPrivC + valC.myProtC + valC.myPackC;
}
```

```
public void myB() {
  int val1 =  myPublA + myPrivA + myProtA + myPackA;
  int val2 =  myPublB + myPrivB + myProtB + myPackB;
  KlasseA valA = new KlasseA();
  int val3 = valA.myPublA + valA.myPrivA + valA.myProtA + valA.myPackA;
}
```

```
public void myC() {
  int val1 =  myPublA + myPrivA + myProtA + myPackA;
  int val2 =  myPublC + myPrivC + myProtC + myPackC;
  KlasseA valA = new KlasseA();
  int val3 = valA.myPublA + valA.myPrivA + valA.myProtA + valA.myPackA;
}
```

```
public static void myM() {
  KlasseB valB = new KlasseB();
  int val1 = valB.myPublA + valB.myPrivA + valB.myProtA + valB.myPackA;
  int val2 = valB.myPublB + valB.myPrivB + valB.myProtB + valB.myPackB;
  KlasseC valC = new KlasseC();
  int val3 = valC.myPublA + valC.myPrivA + valC.myProtA + valC.myPackA;
  int val4 = valC.myPublC + valC.myPrivC + valC.myProtC + valC.myPackC;
}
```

9 Getter und Setter

Die wichtigste Frage bei Zugriffsmodifikatoren ist, warum wir die Eigenschaften einer Klasse überhaupt hinter einer privaten Variablen "verstecken" sollten. Man möchte ja, dass von außen der Nutzer des Objektes gewisse Werte eintragen oder verändern kann. Sehen wir hierzu folgenden Code der Klasse `Rechteck` an, bei der die beiden Eigenschaften `laenge` und breite jeweils `public` sind und wir den Umfang nun als Variable halten:

```
public class Rechteck extends Geo2DObjekt {
  public double laenge;
  public double breite;
  public Rechteck(int farbeInit, double laenge, double breite) {
    super(farbeInit);
    this.laenge = laenge;
    this.breite = breite;
    berechneUmfang();
  }
  public void berechneUmfang() {
    umfang = 2 * (laenge + breite);
  }
}
```

Nun nutzen wir sie wie folgt im Hauptprogramm:

```
Rechteck myR = new Rechteck(0xff00aa, 10, 20);
System.out.println("Umfang: " + myR.umfang);
myR.breite = 10;
System.out.println("Umfang: " + myR.umfang);
```

Welche Ausgabe erhalten wir:

Was hätte der Nutzer der Klasse tun müssen, um das zweite, falsche Ergebnis zu vermeiden:

Das Problem ist also die fehlende Kontrolle über die Eigenschaften. Nur wenn wir den Zugriff von außen verhindern, laufen wir nicht in die Gefahr von fehlender Synchronisation der internen Strukturen. Dieses Vorgehen nennt man „**Kapselung**“ oder „information Hiding“. Ändern wir also unseren Code:

```
public class Rechteck extends Geo2DObjekt {
  private double laenge;
  private double breite;
  public Rechteck(int farbeInit, double laenge, double breite) {
…
  public void setLaenge(double laenge) {
    this.laenge = laenge;
    berechneUmfang();
  }
  public double getLaenge() {
    return laenge;
  }
}
```

Ergänzen Sie nun den Setter und Getter für die Breite:

Welche Berechnung müsste denn theoretisch noch in den beiden Settermethoden hinterlegt werden:

Getter- und Settermethoden erlauben uns also, die Kontrolle über unsere Eigenschaften zu behalten. Folgende Eigenschaften sind für Getter- und Setter festzuhalten:

- Getter und Setter sind `public`
- Die Namen sind in der Regel „`get`“ plus den Namen der Variable
- Setter haben einen Parameter gleichen Datentyps wie die Variable
- Getter geben den gleichen Datentyp wie die Variable zurück
- Getter haben im Regelfall keinen Parameter. Ausnahmen sind mitunter Getter von einzelnen Positionen eines Arrays.