Sorgt dafür, dass der Compiler alle Klassen der Simple Game Engine kennt.

```
import sge.*;

public class Haus
{

    private Rechteck mauer;
    private Dreieck dach;
    private Rechteck tuer;

    public Haus(){

        mauer = new Rechteck(100,100,210,200);
        mauer.setFuellfarbe("weiß");

        dach = new Dreieck(205,0,210,100);
        dach.setFuellfarbe("rot");

        tuer = new Rechteck(170,200,70,100);
        tuer.setFuellfarbe("braun");

    }


    public void tuerAuf(){
        tuer.setFuellfarbe("schwarz");
    }

    public void verschieben(double dx, double dy){
        mauer.verschieben(dx, dy);
        dach.verschieben(dx, dy);
        tuer.verschieben(dx, dy);
    }
}
```

public (="öffentlich") bedeutet, dass die Klasse überall im Code verwendet werden darf. Die Klassendefintion reicht von der { bis zur korrespondierenden } .

Deklaration (="Bekanntmachung") der **Attribute**.
Bem.: Hier wird noch **kein** Objekt instanziert (erzeugt)!
private bewirkt, dass man nur im Code innerhalb der Klasse darauf zugreifen kann.

Datentyp (="Art") des Attributs

Bezeichner (="Name") des Attributs

Erzeugt ein neues `Rechteck` –Objekt im Arbeitsspeicher des Rechners und speichert die Referenz darauf im Attribut **mauer.**

Konstruktor-Methode

Sie wird beim Erzeugen eines neuen Haus-Objekts mittels

```
new Haus();
```

automatisch aufgerufen.

public (="öffentlich") bedeutet, dass die Methode `tuerAuf` überall im Code verwendet werden darf.

Danach kommt der **Datentyp des Rückgabewertes** der Methode. **void** (="Leere") bedeutet, dass die Methode keinen Wert zurückgibt. Die Methodendefintion reicht von der { bis zur korrespondierenden } .

Ende der Methode `verschieben`

Ende der Klasse `Haus`

Die Methode `verschieben` besitzt die beiden **Parameter** `dx` und `dy` mit dem **Datentyp** **double** (="Doppelt genaue Fließkommazahl").

Wird bspw.

```
Haus.verschieben(10,20)
```

aufgerufen, so erzeugt der Rechner zwei neue Variablen (=benannte Speicherbereiche) `dx` und `dy` , schreibt den Wert 10 in die Variable `dx` und den Wert 20 in die Variable `dy` und arbeitet dann die Anweisungen im Methodenrumpf beginnend mit

```
mauer.verschieben(dx, dy);
```

ab. Beim Aufruf von `mauer.verschieben(dx, dy)` kopiert der Computer die aktuellen Werte aus den Variablen `dx` und `dy` (also 10 und 20) und ruft damit die Methode verschieben des **mauer** -Objekts auf.

`dx` und `dy` nennt man **formale Parameter** der Methode, die tatsächlich übergebenen **Werte** (z.B. 10 und 20) nennt man **aktuelle Parameter**. Oft sagt man aber einfach nur „Parameter" und „Wert des Parameters" dazu.