Aufgabe 1

a) Eine rekursive Methode ruft sich selbst auf.

b) Eine rekursive Methode hat mindestens einen Parameter (z.B. „länge“), dessen Wert sich beim rekursiven Aufruf ändert. Die Rekursion wird nicht mehr ausgeführt, wenn der Parameter einen bestimmten Wert erreicht (z.B. länge < 10).

c) Die Turtle befindet sich an der gleichen Position und schaut in die gleiche Richtung wie zu Beginn der Methode.

d) Die Turtle befindet sich an einer bestimmten Stelle, bevor die Rekursion aufgerufen wird. Wenn der Aufruf dann beendet ist, soll sie sich von dieser Position aus weiter bewegen (z.B. den zweiten Ast des Baums zeichnen). Wenn die Turtle nach dem Aufruf der Methode eine andere Position / Blickrichtung hätte, könnte man die Zeichnung kaum noch korrekt steuern.

Aufgabe 2

```
public void baumRekursiv(double länge)
{
   // Grundform
   t.forward(länge);

   // Rekursion
   if (länge > 3)
   {
      // Ast nach links
      t.left(45);
      baumRekursiv(länge * 0.67);
      // Ast nach rechts
      t.right(90);
      baumRekursiv(länge * 0.67);
      // Drehe in ursprüngliche Richtung
      t.left(45);
   }

   // zurück zur Ausgangsposition
   t.penUp();
   t.back(länge);
   t.penDown();
}
```

```
public void main()
{
   t.hideTurtle();
   t.setPos(0, -250);
   t.heading(0);
   baumRekursiv(200);
}
```

Aufgabe 3

```
public void teppichRekursiv(double länge)
{
   t.penDown();
   quadrat(länge);
   t.penUp();

   if (länge > 3.0)
   {
      // 1. links
      t.left(90.0);
      t.forward(länge * 2.0 / 3.0);
      t.right(90.0);
      t.forward(länge / 3.0);
      teppichRekursiv(länge / 3.0);
      // 2. links oben
      t.forward(länge);
      teppichRekursiv(länge / 3.0);
      // 3. oben
      t.right(90.0);
      t.forward(länge);
      t.left(90.0);
      teppichRekursiv(länge / 3.0);
      // 4. rechts oben
      t.right(90.0);
      t.forward(länge);
      t.left(90.0);
      teppichRekursiv(länge / 3.0);
      // 5. rechts
      t.left(180.0);
      t.forward(länge);
      t.right(180.0);
      teppichRekursiv(länge / 3.0);
      // 6. rechts unten
      t.left(180.0);
      t.forward(länge);
      t.right(180.0);
      teppichRekursiv(länge / 3.0);
      // 7. unten
      t.left(90.0);
      t.forward(länge);
      t.right(90.0);
      teppichRekursiv(länge / 3.0);
      // 8. unten links
      t.left(90.0);
      t.forward(länge);
      t.right(90.0);
      teppichRekursiv(länge / 3.0);
      // 9. zurück zum Ausgangspunkt
      t.forward(länge * 2.0 / 3.0);
      t.right(90.0);
      t.forward(länge * 2.0 / 3.0);
      t.left(90.0);
   }
}
```

```
public void quadrat(
   double länge)
{
   t.forward(länge);
   t.right(90);
   t.forward(länge);
   t.right(90);
   t.forward(länge);
   t.right(90);
   t.forward(länge);
   t.right(90);
}


public void main()
{
   t.hideTurtle();
   t.setPos(
      -122.0, -122.0);
   t.heading(0);

   // Länge, die sich
   // mehrmals durch 3
   // teilen lässt
   teppichRekursiv(243.0);
}
```

Aufgabe 4

```
public void dreieckRekursiv(double x1, double y1, double x2, double y2,
                            double x3, double y3, int tiefe)
{
   // Sonderfall: zeichne äußeres Dreieck
   if (tiefe == 0)
   {
      t.setPos(x1, y1);
      t.moveTo(x2, y2);
      t.moveTo(x3, y3);
      t.moveTo(x1, y1);
   }

   // Berechne Mittelpunkte der Seiten
   double mx1 = (x1 + x2) / 2.0;
   double my1 = (y1 + y2) / 2.0;

   double mx2 = (x2 + x3) / 2.0;
   double my2 = (y2 + y3) / 2.0;

   double mx3 = (x1 + x3) / 2.0;
   double my3 = (y1 + y3) / 2.0;

   // Zeichne inneres Dreieck
   t.setPos(mx1, my1);
   t.moveTo(mx2, my2);
   t.moveTo(mx3, my3);
   t.moveTo(mx1, my1);

   // Rekursion
   if (tiefe < maxtiefe - 1)    // maxtiefe als Attribut deklariert
   {
      dreieckRekursiv(x1, y1, mx1, my1, mx3, my3, tiefe+1);
      dreieckRekursiv(mx1, my1, x2, y2, mx2, my2, tiefe+1);
      dreieckRekursiv(mx3, my3, mx2, my2, x3, y3, tiefe+1);
   }
}


public void main()
{
   t.clear();
   t.hideTurtle();
   t.setPenColor("black");

   dreieckRekursiv(-350, -300, 0, 300, 350, -300, 0);
}
```

Aufgabe 5

```
/*
 * Zeichnet rekursiv eine Koch-Kurve.
 * Die Turtle muss an der richtigen Stelle
 * stehen und in die richtige Richtung
 * schauen.
 * Gezeichnet wird nur in der untersten
 * Rekursionstiefe.
 */
public void kochRekursiv(double länge)
{
   if (länge <= 3.0)
   {
      t.forward(länge);
   }
   else
   {
      kochRekursiv(länge / 3.0);
      t.left(60.0);
      kochRekursiv(länge / 3.0);
      t.right(120.0);
      kochRekursiv(länge / 3.0);
      t.left(60.0);
      kochRekursiv(länge / 3.0);
   }
}
```

```
/*
 * Zeichnet eine
 * Schneeflocke aus drei
 * Koch-Kurven.
 */
public void main()
{
   t.hideTurtle();

   t.setPos(-300.0, 180.0);
   t.heading(90);
   kochRekursiv(600.0);
   t.right(120);
   kochRekursiv(600.0);
   t.right(120);
   kochRekursiv(600.0);
}
```