6 Abstrakte Klassen und Schnittstellen

6.1 Abstrakte Klassen und Methoden

6.1.1 Grundsätzliches

Abstrakte Klassen

- Eine Klasse kann mit dem Schlüsselwort `abstract` deklariert werden, um anzuzeigen, dass sie (technisch oder logisch) unvollständig ist.
- Von einer abstrakten Klasse können keine Objekte erzeugt werden.
- Eine abstrakte Klasse kann trotzdem Konstruktoren besitzen, die (nur) von Konstruktoren ihrer Unterklassen aufgerufen werden können (und daher sinnvollerweise `protected` sind).
- Aufgrund der üblichen Untertyp-Polymorphie, können Variablen einer abstrakten Klasse Objekte von (konkreten) Unterklassen referenzieren.

Abstrakte Methoden

- Eine Methode kann mit dem Schlüsselwort `abstract` deklariert werden, um lediglich ihre Signatur und ihren Resultattyp zu vereinbaren, aber noch keine Implementierung anzugeben. (Private, unveränderliche und statische Methoden können nicht abstrakt sein.)

- Abstrakte Methoden können in Unterklassen durch konkrete Methoden mit derselben Signatur überschrieben (d. h. implementiert) werden.

- Eine Klasse, die abstrakte Methoden deklariert oder erbt, aber nicht implementiert, muss abstrakt sein.

6.1.2 Beispiel

```
// Abstrakte Klasse: Allgemeines geometrisches Objekt.
abstract class Figure {
  // Unterklassenöffentliche Objektvariablen: Breite und Höhe.
  protected double width, height;

  // Unterklassenöffentlicher Konstruktor:
  // Breite und Höhe initialisieren.
  protected Figure (double w, double h) {
    width = w;
    height = h;
  }

  // Öffentliche Objektmethoden: Breite und Höhe abfragen.
  public double width () { return width; }
  public double height () { return height; }

  // Öffentliche abstrakte Methode: Fläche berechnen.
  public abstract double area ();
}
```

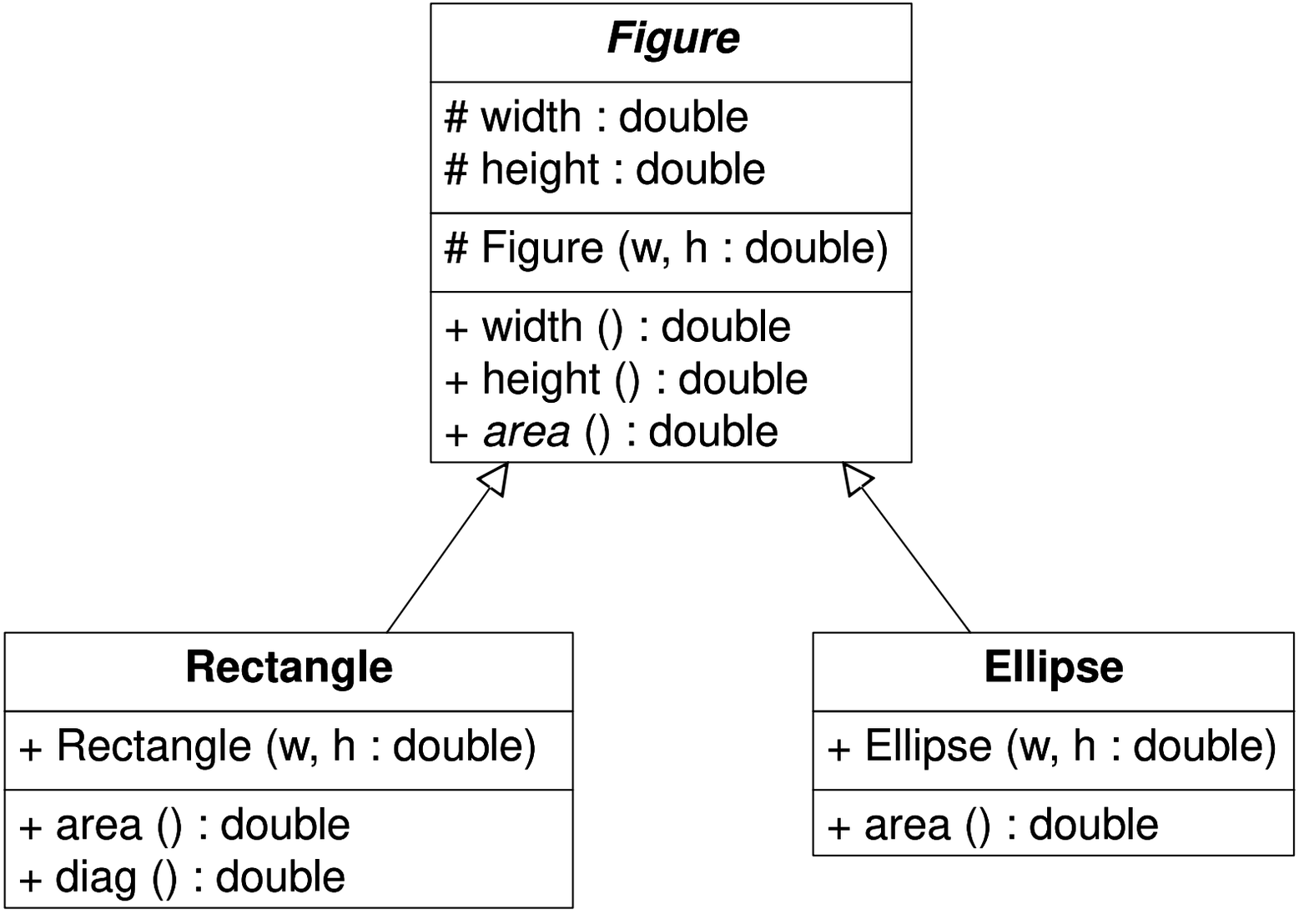
```
// Konkrete Unterklasse von Figure: Rechteck.
class Rectangle extends Figure {
  // Öffentlicher Konstruktor.
  public Rectangle (double w, double h) { super(w, h); }

  // Implementierung der geerbten abstrakten Methode area.
  public double area () { return width * height; }

  // Zusätzliche Objektmethode: Diagonale berechnen.
  public double diag () {
    return Math.sqrt(width*width + height*height);
  }
}

// Konkrete Unterklasse von Figure: Ellipse.
class Ellipse extends Figure {
  // Öffentlicher Konstruktor.
  public Ellipse (double w, double h) { super(w, h); }

  // Implementierung der geerbten abstrakten Methode area.
  public double area () { return Math.PI/4 * width * height; }
}
```



Die Namen von abstrakten Klassen und Methoden werden in UML kursiv geschrieben.

```
// Testklasse (kann prinzipiell abstrakt sein).
abstract class Test {
  // Aufruf zum Beispiel: java Test r 2.5 3.2 e 1.7 5 ...
  public static void main (String [] args) {
    // Unterschiedliche geometrische Objekte erzeugen.
    Figure [] figs = new Figure [args.length/3];
    for (int i = 0; i < args.length; i += 3) {
      char x = args[i].charAt(0);
      double w = Double.parseDouble(args[i+1]);
      double h = Double.parseDouble(args[i+2]);
      if (x == 'r') figs[i/3] = new Rectangle(w, h);
      else figs[i/3] = new Ellipse(w, h);
    }
    // Fläche und ggf. Diagonale aller Objekte ausgeben.
    for (Figure fig : figs) {
      System.out.print(fig.area());
      if (fig instanceof Rectangle r) {
        System.out.print(" " + r.diag());
      }
      System.out.println();
    }
  }
}
```