

## Übungen zu Algorithmen

*Wintersemester 2003/2004*

### Blatt 10

#### Aufgabe 10.1 (25 Punkte)

Implementieren Sie eine Klasse `Figur2D`, die zweidimensionale geometrische Figuren repräsentieren soll. Die Klasse besitzt die folgenden Datenfelder:

```
public String farbe; // Enthält die Farbe der Figur
public double x, y; // Die x- und y-Koordinate der Figur in der Ebene
```

Die Klasse besitzt zwei Konstruktoren: Beim ersten werden alle Datenfelder in der Parameterliste übergeben. Der zweite ist der Defaultkonstruktor und weist dem Attribut `farbe` immer den Wert "schwarz" zu und platziert die Figur im Koordinatenursprung.

Implementieren Sie die folgenden Methoden:

- `public void verschiebeUm(double deltaX, double deltaY)`, verschiebt die Figur in der 2D-Ebene um `deltaX` in x- und um `deltaY` in y-Richtung.
- `public double flaeche()`, liefert den Flächeninhalt dieser `Figur2D`. Da bei einer allgemeinen Figur nicht klar ist, wie groß ihre Fläche ist, wird `0.0` geliefert.
- `public double umfang()`, liefert den Umfang dieser `Figur2D`. Da bei einer allgemeinen Figur nicht klar ist, wie groß ihr Umfang ist, wird `0.0` geliefert.
- `public double abstand(Figur2D other)`, liefert den Abstand zwischen den Positionen dieser `Figur2D` und der `Figur2D other` in euklidischer Metrik.
- `public String toString()`, liefert alle Datenfelder der `Figur2D` in einem `String` zurück. Ist diese Methode neu oder überschrieben? Wenn sie überschrieben ist, von welcher Klasse erbt `Figur2D` sie und welche Informationen liefert sie beim Aufruf in der Superklasse?

Testen Sie die Klasse `Figur2D`.

#### Musterlösung vom 07.01.2004:

```
/****** Aufg10_1.java *****/
```

```
import AlgoTools.IO;
```

```

/**
 *   Programmbeschreibung:
 *
 *   Testet die Klassen Figur2D, Kreis, Rechteck und Quadrat
 *
 *   @author Ralf Kunze    (rkunze@informatik.uni-osnabrueck.de)
 *   @author Olaf Mueller  (olaf@informatik.uni-osnabrueck.de)
 *
 *   @version 15.12.2003
 */

public class Aufg10_1 {

    public static void main(String[] argv) {
        // Test der Klasse Figur2D
        Figur2D f = new Figur2D();
        IO.println("Figur2D f:"+f);
        f.farbe = new String("gruen");
        f.verschiebeUm(1.0, 1.0);
        IO.println("Figur2D f:"+f+"; Flaeche="+f.flaeche()+"; Umfang="+f.umfang());
        IO.println("Abstand zum Ursprung="+f.abstand(new Figur2D()));
        f.x = 2;
        f.y = 0;
        IO.println("Abstand zum Ursprung="+f.abstand(new Figur2D()));

        // Test der Klasse Kreis
        Kreis k = new Kreis();                // Einheitskreis (nicht Teil
                                                // der Aufgabenstellung)

        IO.println("Kreis k:"+k);
        if(k.setRadius(1.5)) {
            IO.println("Neuer Radius="+k.getRadius());
        }
        else {
            IO.print("Setzen des Radius nicht erfolgreich. ");
            IO.println("Alter Radius wird beibehalten.");
        }

        if(k.setRadius(-3)) {
            IO.println("Neuer Radius="+k.getRadius());
        }
        else {
            IO.print("Setzen des Radius nicht erfolgreich. ");
            IO.println("Alter Radius wird beibehalten.");
        }
        IO.println("Kreis k:"+k);
        k = new Kreis(new String("blau"), 3, 0, 2);
        IO.println("Kreis k:"+k);
        IO.println("Abstand zum Ursprung="+k.abstand(new Figur2D()));
        IO.println("Abstand zu f="+k.abstand(f));
        IO.println("Abstand zum Einheitskreis im Ursprung="+k.abstand(new Kreis()));
        IO.println("Flaeche="+k.flaeche()+"; Umfang="+k.umfang());

        // Test der Klasse Rechteck
    }
}

```

```

//Rechteck r = new Rechteck(); // Compilerfehler, da kein
// Defaultkonstruktor vorhanden
Rechteck r = new Rechteck("rot", 0, 1, 2, 3);
IO.println("Rechteck r:"+r);
IO.println("Flaeche="+r.flaeche()+"; Umfang="+r.umfang());
r.verschiebeUm(-1,-1);
// r.b = 4; // Compilerfehler, da private
if(r.setBreite(4)) {
    IO.println("Neue Breite="+r.getBreite());
}
else {
    IO.print("Setzen der Breite nicht erfolgreich. ");
    IO.println("Alte Breite wird beibehalten!");
}
IO.println("Rechteck r:"+r);
IO.println("Flaeche="+r.flaeche()+"; Umfang="+r.umfang());
IO.println("Abstand zum Ursprung="+r.abstand(new Figur2D()));
IO.println("Abstand von r zu k="+r.abstand(k));
IO.println("Abstand von k zu r="+k.abstand(r));
Rechteck r2 = new Rechteck(new String("gelb"), 1, 0, 2, 1.5);
IO.println("Rechteck r2:"+r2);
IO.println("Abstand von r zu r2="+r.abstand(r2));
IO.println("Abstand von r2 zu r="+r2.abstand(r));

// Test der Klasse Quadrat
Quadrat q = new Quadrat(new String("braun"), 0, 0, 4);
IO.println("Quadrat q:"+q);
if(q.setBreite(3)) {
    IO.println("Neue Breite="+q.getBreite());
}
else {
    IO.print("Setzen der Breite nicht erfolgreich. ");
    IO.println("Alte Breite wird beibehalten!");
}
IO.println("Hoehe von q:"+q.getHoehe());
if(q.setHoehe(2)) {
    IO.println("Neue Hoehe="+q.getHoehe());
}
else {
    IO.print("Setzen der Hoehe nicht erfolgreich. ");
    IO.println("Alte Hoehe wird beibehalten!");
}
IO.println("Quadrat q:"+q);
}
}

```

```

/***** Figur2D.java *****/

```

```

/**
 *   Programmbeschreibung:

```

```

*
*   Diese Klasse repraesentiert eine allgemeine Figur in der
*   zweidimensionalen Ebene. Jedes Objekt besitzt eine Farbe und eine
*   Position in der Ebene.
*
*   @author Ralf Kunze    (rkunze@informatik.uni-osnabrueck.de)
*   @author Olaf Mueller (olaf@informatik.uni-osnabrueck.de)
*
*   @version 15.12.2003
*/

public class Figur2D {

    public String farbe;           // Farbe dieser Figur
    public double x,y;            // x- und y-Koordinate

    /**
     * Instanziert eine neue Figur2D mit der Farbe 'schwarz' und dem
     * Koordinatenursprung als Position.
     */
    public Figur2D() {
        this(new String("schwarz"), 0.0, 0.0);
    }

    /**
     * Instanziert eine neue Figur2D mit der uebergebenen Farbe farbe und
     * der Position (x, y).
     *
     * @param farbe Die Farbe der neuen Figur2D
     * @param x     Die x-Koordinate der neuen Figur2D
     * @param y     Die y-Koordinate der neuen Figur2D
     */
    public Figur2D(String farbe, double x, double y) {
        this.farbe = farbe;
        this.x = x;
        this.y = y;
    }

    /**
     * Verschiebt diese Figur2D in der 2D-Ebene um deltaX in x-Richtung und
     * um deltaY in y-Richtung.
     *
     * @param deltaX Der Wert, um den diese Figur2D in x-Richtung verschoben wird.
     * @param deltaY Der Wert, um den diese Figur2D in y-Richtung verschoben wird.
     */
    public void verschiebeUm(double deltaX, double deltaY) {
        x += deltaX;
        y += deltaY;
    }

    /**
     * Liefert den Abstand von dieser Figur2D zu der uebergebenen Figur2D other.
     * In dieser Version der Methode wird der Abstand zwischen der durch (x,y)

```

```

    * gegebenen Position dieser Figur2D in der 2D-Ebene zu der durch
    * (other.x, other.y) gegebenen Position der Figur2D other in euklidischer
    * Metrik berechnet.
    *
    * @param other Die Figur2D zu der der Abstand von dieser Figur2D berechnet
    * werden soll.
    */
public double abstand(Figur2D other) {
    return Math.sqrt((x-other.x)*(x-other.x) + (y-other.y)*(y-other.y));
}

/**
 * Liefert den Flaecheninhalt dieser Figur2D.
 */
public double flaeche() {
    return 0.0;
}

/**
 * Liefert den Umfang dieser Figur2D.
 */
public double umfang() {
    return 0.0;
}

/**
 * Liefert eine Repraesentation dieser Figur2D in Form eines Strings. Alle
 * nach aussen sichtbaren Attribute werden zusammen mit ihren Werten
 * aufgelistet.
 */
public String toString() {
    return new String("Farbe="+farbe+"; Position=["+x+"; "+y+"]");
}
}

```

### Aufgabe 10.2 (20 Punkte)

Leiten Sie von der Klasse `Figur2D` die Klasse `Kreis` ab. Bei der Klasse `Kreis` soll die durch  $(x, y)$  definierte Position den Mittelpunkt des Kreises darstellen. Die Klasse `Kreis` wird erweitert um das neue Attribut `private double radius`, das den Radius des Kreises speichert. Warum ist das Attribut `private` und welche Konsequenzen hat diese Wahl? Implementieren Sie zwei Methoden

```

public double getRadius() // liefert den Radius dieses Kreises
public boolean setRadius(double radius) // setzt den Radius dieses Krei-
// ses auf den Wert radius. Liefert true, falls erfolgreich; false sonst.

```

Versehen Sie die Klasse mit einem passenden Konstruktor, der jedes Datenfeld in der Parameterliste erhält.

Überschreiben Sie die Methoden `flaeche` und `umfang` in der für Kreise spezifischen Weise. Fügen Sie dazu der Klasse eine Konstante `PI` hinzu, die den Wert `3.141592653589793238` hat.



```

    if(!setRadius(radius)) {
        setRadius(1.0);
    }
}

/**
 * Setzt den Radius dieses Kreises auf den uebergebenen Wert.
 * Falls der Parameter radius < 0 sein sollte, behaelt der Kreis
 * seinen alten Radius.
 * Liefert true, wenn der neue Radius erfolgreich gesetzt wurde; false sonst.
 *
 * @param radius Der neue Radius dieses Kreises.
 */
public boolean setRadius(double radius) {
    if(radius < 0) {
        return false;
    }

    this.radius = radius;
    return true;
}

/**
 * Liefert den Radius dieses Kreises.
 */
public double getRadius() {
    return radius;
}

/**
 * Liefert den Abstand von dieses Kreises zu der uebergebenen Figur2D other.
 * Falls other auch ein Kreis ist, wird der korrekte Abstand der beiden
 * Kreise geliefert; sonst das, was figur2D.abstand() liefert.
 *
 * @param other Die Figur2D zu der der Abstand von dieser Figur2D berechnet
 * werden soll.
 */
public double abstand(Figur2D other) {
    double d = super.abstand(other); // Falls Kreis: Abstand der
                                     // Mittelpunkte bestimmen
                                     // sonst: fertiges Ergebnis

    if(other instanceof Kreis) { // Falls Kreis:
        Kreis k = (Kreis)other; // Cast, um an radius zu kommen
        d = d - k.radius - radius; // Radien abziehen
        d = d<0? 0 : d; // liegt Schnitt vor?
    }

    return d;
}

/**
 * Liefert den Flaecheninhalt dieses Kreises.
 */

```

```

public double flaeche() {
    return PI*radius*radius;
}

/**
 * Liefert den Umfang dieses Kreises.
 */
public double umfang() {
    return 2*PI*radius;
}

/**
 * Liefert eine Repraesentation dieses Kreises in Form eines Strings. Alle
 * nach aussen sichtbaren Attribute werden zusammen mit ihren Werten
 * aufgelistet.
 */
public String toString() {
    return new String(super.toString()+"; Radius="+radius);
}
}

```

### Aufgabe 10.3 (25 Punkte)

Leiten Sie von der Klasse `Figur2D` die Klasse `Rechteck` ab. Bei der Klasse `Rechteck` soll die durch  $(x, y)$  definierte Position die linke untere Ecke des Rechtecks darstellen. Die Klasse erhält neue Attribute: `Rechteck` wird erweitert um `private double breite` und `private double hoehe`, die die Breite und die Höhe des Rechtecks speichern. Implementieren Sie `get`- und `set`-Methoden für beide Attribute.

Versehen Sie die Klasse mit einem passenden Konstruktor, der jedes Datenfeld in der Parameterliste erhält.

Überschreiben Sie die Methoden `flaeche` und `umfang` in der für Rechtecke spezifischen Weise. Verändern Sie die Methode `toString`, so daß auch die neuen Attribute ausgegeben werden. Falls die Methode `abstand` ein `Rechteck` übergeben bekommt, soll sie den kleinsten Abstand zwischen einem Eckpunkt dieses Rechtecks und einem Eckpunkt des anderen Rechtecks liefern. Sonst liefert sie den Abstand zwischen der durch  $(x, y)$  gegebenen Position dieses Rechtecks und der Position der übergebenen `Figur2D`.

Testen Sie Ihre Klasse in einem geeigneten Testprogramm.

### Musterlösung vom 07.01.2004:

```

/***** Rechteck.java *****/

/**
 * Programmbeschreibung:
 *
 * Diese Klasse repraesentiert ein Rechteck in der
 * zweidimensionalen Ebene. Jedes Rechteck wird durch seine linke untere
 * Ecke und seine Breite und Hoehe repraesentiert.
 */

```

```

*   @author Ralf Kunze    (rkunze@informatik.uni-osnabrueck.de)
*   @author Olaf Mueller (olaf@informatik.uni-osnabrueck.de)
*
*   @version 15.12.2003
*/

public class Rechteck extends Figur2D {

    private double breite, hoehe;                // Breite und Hoehe des
                                                // Rechtecks

    /**
     * Instanziert ein neues Rechteck mit der uebergebenen Breite und Hoehe
     * der uebergebenen Farbe farbe und der linken unteren Ecke
     * an der Position (x, y).
     *
     * Falls einer der Parameter breite oder hoehe < 0 sein sollte,
     * wird der entsprechende Parameter auf 1.0 gesetzt.
     *
     * @param farbe Die Farbe des neuen Rechtecks
     * @param x     Die x-Koordinate der linken unteren Ecke des neuen Rechtecks
     * @param y     Die y-Koordinate der linken unteren Ecke des neuen Rechtecks
     * @param breite Die Breite dieses Neuen Rechtecks
     * @param hoehe Die Hoehe dieses Neuen Rechtecks
     */
    public Rechteck(String farbe, double x, double y, double breite, double hoehe) {
        super(farbe, x, y);
        if(!setBreite(breite)) {
            setBreite(1.0);
        }
        if(!setHoehe(hoehe)) {
            setHoehe(1.0);
        }
    }

    /**
     * Setzt die Breite dieses Rechtecks auf den uebergebenen Wert.
     * Falls der Parameter breite < 0 sein sollte, behaelt das Rechteck
     * seine alte Breite.
     * Liefert true, wenn die neue Breite erfolgreich gesetzt wurde; false sonst.
     *
     * @param breite Die neue Breite dieses Rechtecks.
     */
    public boolean setBreite(double breite) {
        if(breite < 0) {
            return false;
        }

        this.breite = breite;
        return true;
    }

    /**

```

```

    * Liefert die Breite dieses Rechtecks.
    */
public double getBreite() {
    return breite;
}

/**
 * Setzt die Hoehe dieses Rechtecks auf den uebergebenen Wert.
 * Falls der Parameter hoehe < 0 sein sollte, behaelt das Rechteck
 * seine alte Hoehe.
 * Liefert true, wenn die neue Hoehe erfolgreich gesetzt wurde; false sonst.
 *
 * @param hoehe Die neue Hoehe dieses Rechtecks.
 */
public boolean setHoehe(double hoehe) {
    if(hoehe < 0) {
        return false;
    }

    this.hoehe = hoehe;
    return true;
}

/**
 * Liefert die Hoehe dieses Rechtecks.
 */
public double getHoehe() {
    return hoehe;
}

/**
 * Liefert den Flaecheninhalt dieses Rechtecks.
 */
public double flaeche() {
    return breite*hoehe;
}

/**
 * Liefert den Umfang dieses Rechtecks.
 */
public double umfang() {
    return 2*breite+2*hoehe;
}

/**
 * Liefert den Abstand von diesem Rechteck zu der uebergebenen Figur2D other.
 * Falls other ebenfalls ein Rechteck ist, wird der korrekte Abstand zwischen
 * diesen beiden Rechtecken geliefert; sonst der Abstand, den
 * Figur2D.abstand() liefert.
 *
 * @param other Die Figur2D zu der der Abstand von dieser Figur2D berechnet
 * werden soll.

```

```

*/
public double abstand(Figur2D other) {
    if(other instanceof Rechteck) {
        Rechteck r = (Rechteck)other;
        // Falls Rechteck:
        // Cast, um an breite und hoehe
        // dranzukommen
        double minx = Double.POSITIVE_INFINITY; // Minimum in x-Richtung
        double miny = minx; // Minimum in y-Richtung

        double dx; // akt. Abstand in x-Richtung
        double dy; // akt. Abstand in y-Richtung

        // vorlaeufiger kleinster Abstand in x-Richtung
        minx = Math.abs(r.x - x);

        // aktueller Abstand in x-Richtung
        dx = Math.abs(r.x+r.breite - x);
        // neuer kleinster Abstand?
        if(dx < minx) {
            minx = dx;
        }

        // aktueller Abstand in x-Richtung
        dx = Math.abs(r.x - x - breite);
        // neuer kleinster Abstand?
        if(dx < minx) {
            minx = dx;
        }

        // aktueller Abstand in x-Richtung
        dx = Math.abs(r.x + r.breite - x - breite);
        // neuer kleinster Abstand?
        if(dx < minx) {
            minx = dx;
        }

        // bisheriger kleinster Abstand in y-Richtung
        miny = Math.abs(r.y - y);

        // aktueller Abstand in y-Richtung
        dy = Math.abs(r.y + r.hoehe - y);
        // neuer kleinster Abstand?
        if(dy < miny) {
            miny = dy;
        }

        // aktueller Abstand in y-Richtung
        dy = Math.abs(r.y - y - hoehe);
        // neuer kleinster Abstand?
        if(dy < miny) {
            miny = dy;
        }

        // aktueller Abstand in y-Richtung

```

```

    dy = Math.abs(r.y + r.hoehe - y - hoehe);
    // neuer kleinster Abstand?
    if(dy < miny) {
        miny = dy;
    }

    return Math.sqrt(minx*minx + miny*miny);
}

return super.abstand(other);           // Falls other kein Rechteck
}

/**
 * Liefert eine Repraesentation dieses Rechtecks in Form eines Strings. Alle
 * Attribute werden zusammen mit ihren Werten aufgelistet.
 */
public String toString() {
    return new String(super.toString()+"; Breite="+breite+"; Hoehe="+hoehe);
}
}

```

#### Aufgabe 10.4 (15 Punkte)

Fügen Sie der in den Aufgaben 10.1 bis 10.3 entstandenen Klassenhierarchie eine Klasse Quadrat hinzu. Es gibt zwei Klassen, von denen Quadrat sinnvoll abgeleitet werden könnte. Diskutieren Sie die Vor- und Nachteile beider Möglichkeiten schriftlich, entscheiden Sie sich für die Ihrer Meinung nach bessere und implementieren Sie die Klasse. Testen Sie Quadrat.

#### Musterlösung vom 07.01.2004:

```

/***** Quadrat.java *****/

/**
 * Programmbeschreibung:
 *
 * Diese Klasse repraesentiert ein Quadrat in der
 * zweidimensionalen Ebene. Jedes Quadrat wird durch seine linke untere
 * Ecke und seine identische Breite und Hoehe repraesentiert.
 *
 * Man koennte Quadrat entweder von Figur2D oder von Rechteck ableiten.
 * Das Ableiten von Figur2D haette den Vorteil, dass man nur ein neues
 * Attribut 'seitenlaenge' braeuchte; beim Ableiten von Rechteck wird
 * der Speicherplatz fuer ein double (entweder b oder h) verschwendet.
 *
 * Andererseits spart das Ableiten von Rechteck sehr viel
 * Implementationsaufwand. Die Methoden umfang, flaeche und abstand
 * koennen unveraendert von Rechteck geerbt werden (und liessen sich (bis
 * auf umfang() im Falle von nur einer Seitenlaenge (4*a statt 2*b+2*h))
 * auch nichteffizienter implementieren). Die Methoden
 * getBreite und getHoehe bleiben auch unveraendert, wobei die Tatsache,
 * dass man nur eine braeuchte, um an die gewuenschte Information zu
 * kommen, keinen Nachteil darstellt.
 */

```

```

* Die Methoden setBreite und setHoehe muessen ueberschrieben werden, was
* aber auch nicht mehr Aufwand verursacht als die komplette
* Reimplementation beim Ableiten von Figur2D.

```

```

*
* Abgesehen davon ergibt sich noch ein logischer Modellierungsvorteil
* wenn man von Rechteck ableitet. Der Benutzer der Klassenhierarchie
* versteht ein Quadrat als ein besonderes Rechteck, bei dem alle vier
* Seiten gleichlang sind. Durch das Ableiten der Klasse Quadrat von
* Rechteck, kann man jeder Referenz vom Typ Rechteck eine Instanz der
* Klasse Quadrat zuweisen (s. Aufgabe 10.5), was z.B. folgenden Code
* erlaubt:

```

```

*
* Rechteck[] rechtecke = new Rechteck[3];
* rechtecke[0] = new Rechteck(new String("schwarz"), 0, 0, 1, 2);
* rechtecke[1] = new Quadrat(new String("blau"), 1, 1, 2);
* rechtecke[2] = new Rechteck(new String("gelb"), 3, 4, 4, 3);
* for(int i=0; i<3; i++) rechtecke[i].setBreite(10);

```

```

* Auch korrekte Abstandsberechnungen waeren problemlos moeglich.
* double gesamtabstand = 0;
* for(int i=0; i<3; i++) {
*     gesamtabstand += rechtecke[i].abstand(rechtecke[(i+1)%3]);
* }

```

```

* Insgesamt ueberwiegen die Vorteile, die sich durch das Ableiten von
* Rechteck ergeben und deshalb ist diese Klasse hier auch
* dementsprechend implementiert.

```

```

* @author Ralf Kunze (rkunze@informatik.uni-osnabrueck.de)
* @author Olaf Mueller (olaf@informatik.uni-osnabrueck.de)

```

```

* @version 15.12.2003

```

```

*/

```

```

public class Quadrat extends Rechteck {

    /**
     * Instanziiert ein neues Quadrat mit der uebergebenen Seitenlaenge laenge,
     * der uebergebenen Farbe farbe und der Position (x, y).
     *
     * Falls der Parameter laenge < 0 sein sollte,
     * wird laenge auf 1.0 gesetzt.
     *
     * @param farbe Die Farbe des neuen Quadrats
     * @param x Die x-Koordinate des neuen Quadrats
     * @param y Die y-Koordinate des neuen Quadrats
     * @param laenge Die Laenge dieses Neuen Quadrats
     */
    public Quadrat(String farbe, double x, double y, double laenge) {
        super(farbe, x, y, laenge, laenge);
    }

    /**

```

```

    * Setzt die Breite und die Hoehe dieses Quadrats auf den uebergebenen Wert.
    * Falls der Parameter breite < 0 sein sollte, behaelt das Quadrat
    * seine alte Breite.
    * Liefert true, wenn die neue Breite erfolgreich gesetzt wurde; false sonst.
    *
    * @param breite Die neue Breite dieses Quadrats.
    */
public boolean setBreite(double breite) {
    super.setBreite(breite);
    return super.setHoehe(breite);
}

/**
 * Setzt die Hoehe und die Breite dieses Quadrats auf den uebergebenen Wert.
 * Falls der Parameter hoehe < 0 sein sollte, behaelt das Quadrat
 * seine alte Hoehe.
 * Liefert true, wenn die neue Hoehe erfolgreich gesetzt wurde; false sonst.
 *
 * @param hoehe Die neue Hoehe dieses Quadrats.
 */
public boolean setHoehe(double hoehe) {
    return setBreite(hoehe);
}
}

```

### Aufgabe 10.5 (15 Punkte)

Beantworten Sie folgende Fragen und begründen Sie Ihre Antworten schriftlich:

- Darf eine Referenz der Klasse `Figur2D` auf eine Instanz der Klasse `Rechteck` verweisen?
- Darf eine Referenz der Klasse `Kreis` auf eine Instanz der Klasse `Figur2D` verweisen?
- Darf eine Referenz der Klasse `Kreis` auf eine Instanz der Klasse `Rechteck` verweisen?

### Musterlösung vom 07.01.2004:

```

/***** Aufg10_5.txt *****/
/**
 * Beschreibung:
 *
 * @author Ralf Kunze (rkunze@informatik.uni-osnabrueck.de)
 * @author Olaf Mueller (olaf@informatik.uni-osnabrueck.de)
 *
 * @version 15.12.2003
 */

```

- Ja, denn `Rechteck` ist von `Figur2D` abgeleitet.

```
Figur2D f = new Rechteck(new String("gelb"), 0, 0, 2, 3);
```

Alle Methoden und Attribute, nach denen man eine Referenz der Klasse Figur2D fragen kann, sind in der Klasse Kreis vorhanden.

f.x hat den Wert 0.

Darueberhinaus kann man jede Methode der Subklasse aufrufen. Allerdings muss man casten:

```
((Kreis)f).getBreite();
```

Bei ueberschriebenen Methoden wird die Version der Subklasse ausgefuehrt (dynamic method lookup).

f.flaeche() liefert: 6

b)

Im Code (und damit zur Compilezeit): Ja, mit explizitem cast kann man einer Referenz der Klasse Kreis eine Instanz der Klasse Figur2D zuweisen. Der Compiler ist dann zufrieden, weil er nicht pruefen kann, was sich hinter der Referenz zur Laufzeit verbergen wird:

```
Kreis k = (Kreis) new Figur2D();
```

Aber zur Laufzeit entdeckt die VM den hier vorliegenden Fehler und steigt mit einer ClassCastException aus.

Der Compiler meckert nicht weil ja auch folgende Situation vorliegen koennte:

```
Figur2D f = new Kreis(new String("blau"), 1, 2, 3);  
Kreis k = (Kreis) f;
```

Denn dann ist zur Laufzeit alles gut und man kann z.B.

```
k.getRadius()
```

aufrufen.

Allerdings verweist keine Instanz der Klasse Kreis jemals zur Laufzeit auf eine Instanz der Klasse Figur2D!

Und da zur Compilezeit die Instanzen nicht existieren, lautet die Antwort: Nein.

c)

Im Code (und damit zur Compilezeit): Ja, mit explizitem cast kann man die folgende Zuweisung durchfuehren.

```
Kreis k = (Kreis)((Figur2D)new Rechteck());
```

Der Compiler ist dann zufrieden,  
weil er nicht prüfen kann, was sich hinter der Referenz zur Laufzeit  
verbergen wird.

Allerdings ist es auch hier so, dass zur Laufzeit niemals eine Referenz  
des Typs Kreis auf eine Instanz der Klasse Rechteck verweisen kann!

Also lautet die Antwort auch hier: Nein.

*Frohe Weihnachten und ein erfolgreiches neues Jahr!*