Aufgabe 1. Klasse Balloon

In der Vorlesung wurde bruchstückhaft eine Klasse Balloon eingeführt. Programmieren Sie eine entsprechende Klasse mit BlueJ bzw. fügen Sie nun die "Bruchstücke" zusammen. Kommentieren Sie Ihren Code à la Lehrbuch. Erzeugen Sie anschliessend Ballon-Objekte und interagieren Sie mit diesen. Gut Flug!

```
/**
 * Write a description of class Balloon here.
 *
 * @author Felix Rohrer
 * @version 06.10.2011
 */
public class Balloon
{
    // instance variables - replace the example below with your own
    private int position; // m
    private int altitude; // m
    private int radius;   // cm
    private String color; //

    /**
     * Constructor for objects of class Balloon
     *
     * @param  baloonColor   Farbe des Balloon
     * @return void
     */
    public Balloon(String baloonColor)
    {
        // initialise instance variables
        color = baloonColor;
        position = 0;
        altitude = 0;
        radius   = 5;
    }

    /**
     * Position festlegen
     *
     * @param  pos   neue Position
     *         alti  neue Höhe
     * @return void
     */
    public void setLocation(int pos, int alti)
    {
        // Location anhand der Parameter setzen
        position = pos;
        altitude = alti;
    }

     /**
     * Position abfragen
     *
     * @param  -
     * @return int  position
     */
    public int getPosition()
    {
        // Position zurück geben
        return position;
    }

    /**
     * Farbe abfragen
     *
     * @param  -
     * @return String  Farbe
     */
    public String getColor()
    {
        // aktuelle Farbe zurück geben
        return color;
    }

    /**
     * Ballon aufblasen
     *
     * @param  -
     * @return -
     */
    public void blowUp()
    {
```

```
            // Radius um 5 erhöhen
            radius = radius + 5;
        }

        /**
         * Ballon zerstören
         *
         * @param  -
         * @return -
         */
        public void explode()
        {
            // Radius und Höhe auf 0 setzen
            radius   = 0;
            altitude = 0;
        }

        /**
         * Volumen näherungsweise ausrechnen und zurück geben
         *
         * @param  -
         * @return int  Volumen in cm^3
         */
        public int getVolume()
        {
            // Volumen näherungsweise ausrechnen (PI = 4)
            int r3;
            r3 = radius * radius * radius;
            return (4 * r3);
        }
    }
```

Aufgabe 2: ... zusätzliche Instanzvariable

Sie möchten Ihre Ballone noch detaillierter modellieren. Überlegen Sie sich weitere Eigenschaften (z.B. Nummer, Startort, ...) und implementieren eine zusätzliche Instanzvariable. Wahrscheinlich gibt's noch weitere Modifikationen. Testen Sie die neuen Ballone aus.

```
    public class Balloon
    {
        private int nummer;   // (Lizenz-)Nummer
    [...]

        public Balloon(String baloonColor, int newNumber)
        {
            nummer   = newNumber;
    [...]
        }

         /**
         * Lizenz-Nummer abfragen
         *
         * @param  -
         * @return int  nummer
         */
        public int getNummer()
        {
            // Nummer zurück geben
            return nummer;
        }
```

Aufgabe 3: ... zwei zusätzliche Konstruktoren

Implementieren Sie zwei zusätzliche Konstruktoren, und zwar je einen mit und ohne Parameter. Ballone lassen sich jetzt unterschiedlich erzeugen.

```
/**
 * Constructor 2 for objects of class Balloon
 *
 * @param  -
 * @return -
 */
public Balloon()
{
    // initialise instance variables
    color    = "green";
    position = 0;
    altitude = 0;
    radius   = 5;
    nummer   = 1;
}

/**
 * Constructor 3 for objects of class Balloon
 *
 * @param  baloonColor   Farbe des Balloon
 * @return void
 */
public Balloon(String baloonColor)
{
    // initialise instance variables
    color = baloonColor;
    position = 0;
    altitude = 0;
    radius   = 5;
    nummer   = 1;
}
```

Aufgabe 4: … weitere Methoden

Implementieren Sie weitere Methoden, und zwar solche ohne und mit Rückgabewert. Gelingt es Ihnen, zwei Methoden mit ein- und demselben Namen zu programmieren?

Ja, auch eine Methode kann überladen werden.

```
/**
 * Lizenz-Nummer abfragen
 *
 * @param  -
 * @return int  nummer
 */
public int getNummer()
{
    // Nummer zurück geben
    return nummer;
}

/**
 * Luft Ablassen
 *
 * @param  -
 * @return -
 */
public void blowOff()
{
    // radius verkleinern
    if (radius >= 5)
    {
        radius = radius - 5;
    }
    else
    {
        radius = 0;
    }
}

public void setAltitude(int newAlti)
{
    // neue Höhe setzen (falls >= 0)
    if (newAlti >= 0)
    {
        altitude = newAlti;
    }
}

    public void setAltitude()
{
    // Höhe +5 setzen
    altitude = altitude + 5;
}
```

Optionale Zusatzaufgabe 5: Klasse Account

Überlegen Sie sich Eigenschaften und Verhalten für Bank-Konten. Implementieren Sie eine entsprechende Klasse Account. Damit's keine Rundungsprobleme gibt, arbeitet Ihre Account-Klasse nur mit ganzen Rappen. Testen Sie Ihre Konten-Objekte aus. Zahlen Sie sich was aus!

```
/**
 * Einfaches Bank-Konto
 *
 * @author Felix Rohrer
 * @version 06.10.2011
 */
public class Account
{
    // instance variables
    private int kontoRappen;
    private int pinNummer;
    private String owner;

    /**
     * Constructor for objects of class Account
     *
     * @param  owner  Konto Besitzer
     *         pin    Geheimnummer
     * @return -
     */
    public Account(String newOwner, int newPin)
    {
        // Kontostand = 0, Owner und PIN setzen
        kontoRappen  = 0;
        owner        = newOwner;
        pinNummer    = newPin;
    }

    /**
     * Rappen in Franken und Rappen umrechnen
     *
     * @param  Rappen
     * @return String Franken.Rappen
     */
    private String convertRappen(int Rappen)
    {
        int tmpFranken;
        int tmpRappen;

        tmpFranken = Rappen / 100;
        tmpRappen  = Rappen % 100;

        return tmpFranken + "." + tmpRappen;
    }

    /**
     * Kontostand anzeigen
     *
     * @param  -
     * @return string Kontostand
     */
    public String getKontoStand()
    {
        return "Kontostand: " + convertRappen(kontoRappen);
    }

    /**
     * Betrag einzahlen
     *
     * @param  franken  Franken Betrag welcher einbezahlt wird
     *         rappen   Rappen Betrag welcher einbezahlt wird
     * @return -
     */
    public void einzahlen(int addFranken, int addRappen)
    {
        // addRappen muss > 0 sein
        int tmpRappen;
        tmpRappen = addRappen + (addFranken * 100);

        if (tmpRappen > 0) {
            // neuer Rappen Betrag speichern
            kontoRappen = kontoRappen + tmpRappen;
        }
    }

    /**
```

```
         * Betrag auszahlen
         *
         * @param  pin      PIN vom Konto
         *         franken  Rappen Betrag welcher ausbezahlt werden soll
         *         rappen   Rappen Betrag welcher ausbezahlt werden soll
         * @return String   Betrag welcher ausbezahlt wird
         */
        public String auszahlen(int pin, int getFranken, int getRappen)
        {
            int tmpRappen;
            String tmpReturn;

            // PIN muss gültig sein
            if (pinNummer == pin) {
                tmpRappen =  getRappen + getFranken * 100;
                if (kontoRappen >= tmpRappen) {
                    //genug Geld auf dem Konto
                    tmpReturn   = "Auszahlen: " + convertRappen(tmpRappen);
                    kontoRappen = kontoRappen - tmpRappen;
                } else {
                    //zuwenig Geld auf dem Konto
                    tmpReturn   = "Auszahlen: " + convertRappen(kontoRappen) + " (mehr war nicht auf dem
Konto...)";
                    kontoRappen = 0;
                }

            } else {
                //PIN ungültig
                tmpReturn = "Ungültiger PIN!";
            }

            // Ausgabe
            return tmpReturn;
        }
    }
```