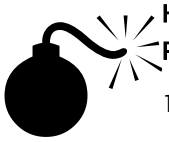


Halten Sie beim Programmieren rigoros ALLE aufgestellten Programmierregeln ein. Ihre Programme werden in Hinsicht auf die Einhaltung dieser Regeln kontrolliert.

1. Die Klasse `Messwert` verfügt über folgende Membervariablen. Definieren Sie die dazugehörigen *Getter- und Settermethoden*. Sollten die übergebenen Werte nicht den Bedingungen entsprechen, dann sollen die Werte von den Methoden nicht gesetzt werden:

```
private int messwert = 0;          // Wert muss zwischen -60 und 50 liegen
private String messdatum = null;   // Datum muss exakt achtstellig sein
private boolean innen = true;      // legt fest ob außen oder innen gemessen
```

2. *Deklarieren* Sie dann die Variable `messwert`, und legen Sie über diese ein Messwert-Objekt an:

```
_____________ messwert = ____________________________________________;
```

3. Setzen Sie im eben angelegten Objekt `messwert` folgende *Werte*: `5`, `"10.12.08"`, `false`

```
messwert_____________________________________;
messwert_____________________________________;
messwert_____________________________________;
```

4. Erzeugen Sie dann ein `Messwert`-*Array*, in dem Sie *1024* Messwerte ablegen können. Instanziieren Sie nur das Array, das den Namen `messwerte` haben soll:

```
_____________ messwerte = ___________________________________________;
```

5. Betrachten Sie folgendes Programmfragment. Wo liegen die *Fehler*? Markieren und beschreiben Sie diese…

```
public static void main(String[] args) {
   Messwert m1 = new Messwert();
   m1.setMesswert(3.5);
   m1.innen = false;
   setMessdatum("01.08.09");
   m2 = m1.clone();
   Messwert m3 = null;
   m3.setMessdatum("02.08.09");
   if (m1.equals(m2) == 0)
      System.out.println(m1.toString());
   Messwert[] m = new Messwert[5];
   for (int i = 0; i <= 5; i = i + 1)
      m[i] = new Messwert();
   m[m.length - 1] = null;
   for (int i = 0; i < m.length; i = i + 1)
      System.out.println(m[i].toString());
}
```

6. Zeichnen Sie den *Inhalt des Speichers* auf, so wie er gefüllt ist, nachdem die folgende Methode durchgeführt wurde:

```
public static void main(String[] args) {
   Messwert m1 = new Messwert();
   Messwert[] m = new Messwert[5];
   for (int i = 0; i < m.length; i = i + 1)
      m[i] = m1;
   Messwert m2 = m1.clone();
   for (int i = 1; i < m.length; i = i + 2)
      m[i] = m2;
   m[m.length / 2] = null;
   Messwert[] m3 = new Messwert[500];
}
```

7. Schreiben Sie dann die Methode `messwertDurchschnitt`, welche von einem ihr übergebenen `Messwert`-*Array* den *Durchschnitt aller Messwerte* ermittelt und zurück liefert. Beachten Sie, dass im Array auch `Messwert`-Elemente *nicht gesetzt* sein können und dass der gelieferte Durchschnitt eine *Kommazahl* sein wird. Deklarieren Sie zuerst den *Methodenkopf* bevor Sie an die Programmierung der Methode herangehen.

8. Programmieren Sie die Klasse `Kreis` indem Sie die *bereitgestellte Dokumentation* dieser Klasse konsultieren und die geforderten *Methoden programmieren*.
   In der Dokumentation wird Ihnen auch ein Beispiel der Verwendung dieser Klasse mit dem zu erzielenden Ergebnis angezeigt. Erstellen Sie aus diesen Programmzeilen die Hauptprogrammklasse `KreisProgramm` und testen Sie dadurch Ihre Klasse.

9. Erstellen Sie die Klasse `Quadrat`, welche eine *einzige interne Objektvariable* `seiteA` vom Typ `double` hat. Die Klasse soll die *Eigenschaften SeiteA*, *SeiteB*, *Umfang* und *Fläche* über die entsprechenden *Getter- und Settermethoden* (`getSeiteA`, `setSeiteA`, `getSeiteB`, `setSeiteB`, `getUmfang`, `setUmfang`, `getFlaeche`, `setFlaeche`) zur Verfügung stellen. Dabei müssen alle vier Eigenschaften auf einen Wert *größer oder gleich 0* gesetzt werden. Da die Klasse ein Quadrat darstellen soll, muss die Klasse dafür sorgen, dass die beiden Seiten-Eigenschaften immer auf denselben Wert gesetzt sind.

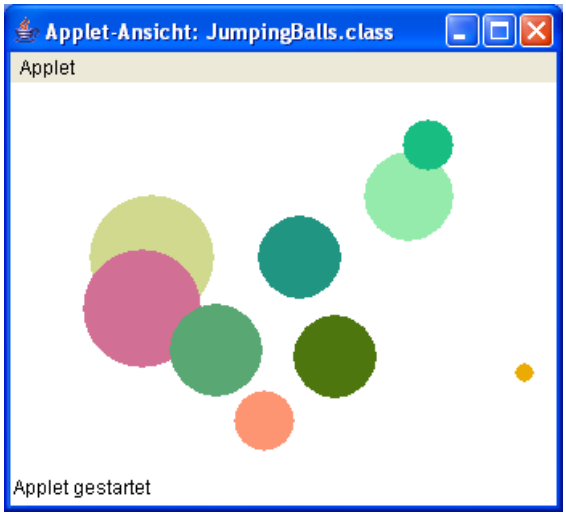
   Die Klasse soll auch die Methoden `clone`, `equals` und `compareTo` mit den Ergebniswerten `-1`, `0` und `1` bereit stellen. Die Methode `toString` gibt für ein Quadrat mit einer Seitenlänge von `3.5` den String `a = 3.5, b = 3.5, U = 14.0, F = 12.25` aus. Die Klasse und ihre Methoden müssen einwandfrei dokumentiert werden.

   Erstellen Sie dann für Ihre Klasse die *HTML-Dokumentation*.

10. Schreiben Sie ein kurzes Testprogramm mit dem Namen `QuadratProgramm`, welches in einem *Array fünfzig* `Quadrat`-Objekte ablegt, diese mit *unterschiedlichen Seitenlängen* versieht und sie dann ausgibt. Dabei sollen die einzelnen Seitenlängen mit einem *zufälligen Wert* zwischen 0 und 10 versehen werden.

    Das Testprogramm soll im Array auch jenes Quadrat ermitteln, welches die *größte Seitenlänge* hat und dieses Quadrat mit `toString` ausgeben.

11. Es soll ein Programm geschrieben werden, durch welches eine Anzahl von Bällen über einen Zeichenbereich so bewegt werden, dass diese an den Rändern abprallen. Definieren Sie sich dazu zuerst die *Klasse* `Ball`, durch welche die Eigenschaften eines Balles abgelegt und der Ball bewegt werden kann.



    Die von Ihnen zu entwickelnde Klasse `Ball` soll folgende *interne Membervariablen* und die dazugehörigen *Getter- und Settermethoden* haben (**HINWEIS:** Nachdem Sie die Membervariablen deklariert haben, können Sie sich die Methodenköpfe der Getter- und Settermethoden von der *Entwicklungsumgebung Eclipse* automatisch generieren lassen):

    - **`private int`** `radius = 0;`
      der *Radius* des Balles der nicht negativ sein darf. Der Standardwert wird auf 0 gesetzt.
    - **`private int`** `xposition = 60;`
      die *X-Position* des Balls vom *Mittelpunkt* des Balls aus gemessen. Diese wird auf den Standardwert 60 gesetzt und darf nicht negativ sind.
    - **`private int`** `yposition = 80;`
      die *Y-Position* des Balls vom Mittelpunkt des Balls aus gemessen. Diese wird auf den Standardwert 80 gesetzt und darf nicht negativ sein.
    - **`private int`** `xrichtung = 0;`
      die *X-Richtung* in der sich der Ball bewegt. Diese wird auf 0 gesetzt. Die Richtungswerte können auch negativ sein.
    - **`private int`** `yrichtung = 0;`
      Die *Y-Richtung* in der sich der Ball bewegt. Diese wird auf 0 gesetzt und kann auch negativ sein.

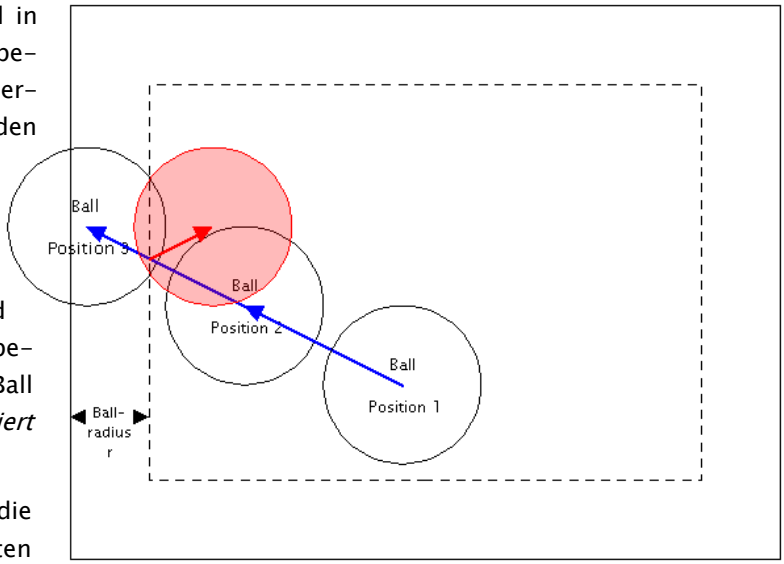
- **private** java.awt.Color farbe = java.awt.Color.BLACK;
  Die *Farbe* des Balls vom Typ java.awt.Color. Die Standardfarbe wird *Schwarz* (java.awt.Color.BLACK) gesetzt.

Definieren Sie dann noch zusätzlich folgende *Methoden* in der Klasse Ball:

- Methode toString welche die Balleigenschaften in der Form "r = 10, xposition = 20, yposition = 70, xrichtung = 10, yrichtung = -5" zurück liefert.
- Methode setZufaellig welche folgende Eigenschaften des Balles mit *zufälligen Werten* im angegebenen Bereich füllt:

Membervariable	Zufällige Werte
radius	im Bereich von 2 bis 40
xrichtung und yrichtung	im Bereich -10 bis 10. Der Wert 0 darf nicht vergeben werden
farbe	zufällige Farbe im *RGB-Farbbereich*. Eine RGB-Farbe wird durch drei Werte die jeweils zwischen 0 und 255 liegen, definiert. Der Konstruktor farbe = **new** Color(255,255,255) definiert beispielsweise die Farbe *Weiß*

- Methode bewege welche den Ball in der eingestellten Richtung fortbewegt, d. h. seine neue Position errechnet und an dieser Position den Ball ausgibt. Dabei muss der Methode der *Zeichnungsbereich* vom Typ java.awt.Graphics auf dem gezeichnet werden soll sowie die *Breite* und *Höhe* des sichtbaren Zeichnungsbereiches übergeben werden. Der Ball soll an allen vier Rändern *reflektiert* werden.



  Normalerweise errechnet sich die neue Ballposition, indem zur alten Position der Bewegungsvektor (xrichtung, yrichtung) addiert wird. Sollte der Ball mit seiner Oberfläche auf den Rand treffen, so wird er reflektiert (gespiegelt). Der Ballmittelpunkt ist zu diesem Zeitpunkt noch um den Ballradius radius innerhalb des Randes (Mittelpunktsrand strichliert eingezeichnet). Anstatt also die Balloberfläche am Rand zu reflektieren, kann man den Ballmittelpunkt am Mittelpunktsrand reflektieren. Sobald die neue Ballposition außerhalb des Mittelpunkts-Randes liegen würde, muss reflektiert werden. Dabei können xposition und yposition getrennt betrachtet werden.

  Beachten Sie, dass nicht nur eine Koordinate der neuen Ballposition am Mittelpunktsrand gespiegelt werden muss, sondern der Ball jetzt auch seine Flugrichtung ändert. Ändern Sie daher auch das passende Vorzeichen im Bewegungsvektor.

Es wird Ihnen das Hauptprogramm mit dem Namen JumpingBalls bereits zur Verfügung gestellt, welches die Klasse Ball verwendet und 10 Bälle in einem Applet bewegt. Sie brauchen sich also um das Erstellen des Applets und um das wiederholte Darstellen der einzelnen Bälle im Applet keine Gedanken machen, diese Aufgabe besorgt die Klasse JumpingBalls. Ihre Aufgabe ist lediglich die, die Klasse Ball exakt laut Vorgaben zu programmieren.

12. Programmieren Sie die Klasse Song durch welche ein Musikstück mit Titel (titel), Interpret (interpret), Album (album) und Erscheinungsjahr (erscheinungsjahr vom Typ int) abgelegt werden

kann. Programmieren Sie die entsprechenden *Getter-* und *Settermethoden*. Achten Sie dabei darauf, dass das Erscheinungsjahr *nicht negativ* sein darf.

Für die Klasse sollen Sie die folgenden Methoden programmieren:

- Die Methode `equals` kontrolliert, ob der Song mit dem übergebenen Song übereinstimmt. Dabei werden Titel, Interpret, Album und Erscheinungsjahr einzeln verglichen (**HINWEIS:** Vergleichen Sie dabei Strings ihrerseits wiederum mit equals).
- Die Methode `compareTo` vergleicht zwei Songs auf ihren Interpret, Album und Titel. Dabei gelten folgende Regeln:

Song Bob Dylan Album1 Song1	ist gleich	Song Bob Dylan Album1 Song1
Song Bob Dylan Album1 Song2	ist größer	Song Bob Dylan Album1 Song1
Song Bob Dylan Album2 Song1	ist größer	Song Bob Dylan Album1 Song1
Song Supertramp Album1 Song1	ist größer	Song Bob Dylan Album1 Song1

  Dabei werden die Eigenschaften Interpret, Album und Titel des einen Songs mit denselben Eigenschaften des anderen Songs zeichenweise verglichen.
- `clone` legt ein weiteres Song-Objekt an, füllt es mit den Werten und liefert das neue Objekt zurück
- `toString` liefert ein Song-Objekt in der Form

  ```
  A Hard Rain's A-Gonna Fall;The Best Of Volume 2;Bob Dylan;2000
  ```

  zurück. Dabei werden die einzelnen Eigenschaften durch Strichpunkte (;) voneinander getrennt.
- Die Methode `setSong` zerlegt den ihr übergebenen *String* in *Titel*, *Album*, *Interpret* und *Erscheinungsjahr* und fügt die Teile in das Song-Objekt ein. So wird beispielsweise der String

  ```
  "A Hard Rain's A-Gonna Fall;The Best Of Bob Dylan Volume 2;Bob Dylan;2000"
  ```

  folgendermaßen aufgeteilt:

Titel:	`A Hard Rain's A-Gonna Fall`
Album:	`The Best Of Bob Dylan Volume 2`
Interpret:	`Bob Dylan`
Erscheinungsjahr:	2000

  Das Trennzeichen ist der Strichpunkt (;). Er sagt wo die einzelnen Komponenten voneinander getrennt werden. Sie können davon ausgehen, dass im übergebenen String genau drei Strichpunkte enthalten sind und dass zwischen den Strichpunkten Zeichen vorhanden sind. Nach dem letzten Strichpunkt ist eine Zahl zu finden.

13. Sie sollen dann die Klasse Song verwenden und daraus das Programm `Songliste` erstellen, welches die mitgelieferte Textdatei `tracklist.csv`[1] *einliest*, *sortiert*, und die sortierten Songs in die Datei `sortlist.csv` *zurückschreibt*. Das Programm selbst soll keinerlei Bildschirmausgaben durchführen.

    HINWEIS: Wie aus *Textdateien zeilenweise gelesen* und in diese *geschrieben* werden kann, soll Ihnen das ebenfalls mitgelieferte Programm `DateiLeseSchreibTest.java` zeigen.

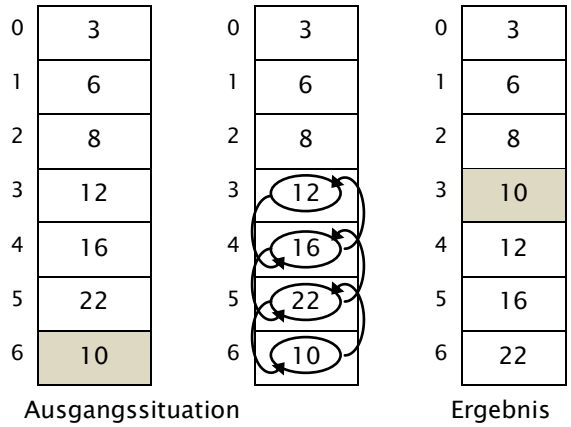
    Das Sortieren der Songs soll in einem Array erfolgen. Lassen Sie sich dazu zuerst vom Programm die *Zeilen in der Datei zählen*. Legen Sie sich dann ein Array an, welches genauso viele Songs aufnehmen kann wie die Datei `tracklist.csv` Songs enthält.

    Der zum Sortieren verwendete Algorithmus soll das *„Sortieren durch Einordnen"* sein. Dieser funktioniert folgendermaßen:

    Zuerst wird der erste Song an der ersten Stelle im Array eingetragen. Das Array ist dadurch sortiert. Dann wird der zweite Song an der zweiten Stelle im Array platziert. Dieser Song wird solang nach vorne –

[1] Quelle `http://www.discographien.de`

gegen Index 0 hin – geschoben, bis der Song vor ihm kleiner ist (verwenden Sie zum Vergleichen der Songs die Methode `compareTo`). Das Array ist dadurch wieder sortiert. Dann wird der dritte Song an der dritten Position eingetragen und solange mit seinem Vorgängersong vertauscht bis der Vorgängersong wiederum kleiner ist. In der nebenstehenden Abbildung wird Ihnen die Funktionsweise des Sortierens durch Einordnen anhand von Zahlen erklärt.



Schreiben Sie die Songs zeilenweise in die Datei `sortlist.csv` der Form *Titel; Album; Interpret; Erscheinungsjahr* so wie sie die Methode `toString` liefert.

14. (**HINWEIS:** Diese Klasse wird in einem der nächsten Kapitel benötigt) Erstellen Sie eine Klasse mit dem Namen `Stoppuhr`, welche sich wie eine Stoppuhr verhält und Zeiten misst, welche zwischen dem Starten und Stoppen der Stoppuhr verflossen sind. Zeiten können durch ein Objekt der Klasse `GregorianCalendar` folgendermaßen gemessen werden:

```
long millis = new java.util.GregorianCalendar().getTimeInMillis();
```

Ein Objekt vom Typ `GregorianCalendar` enthält die aktuelle Uhrzeit. Mit der Methode `getTimeInMillis` wird die Zeit in Millisekunden zurück geliefert und in die Ganzzahlenvariable `millis` vom Typ `long` geschrieben. Dabei ist der primitive Datentyp `long` ähnlich wie der Datentyp `int` ein Typ der ganzzahlige Werte zwischen -9.223.372.036.854.775.808 und 9.223.372.036.854.775.807 abspeichern lässt.

Die Stoppuhr muss gestartet und gestoppt werden können. Nachdem die Stoppuhr gestoppt wurde, kann die gemessene Stoppzeit in Millisekunden abgefragt werden.

Die Stoppuhr muss auch die Möglichkeit vorsehen, maximal 1000 gemessene Stoppzeiten der Reihe nach abzuspeichern. Konkret bedeutet dies, dass nach dem Stoppen der Stoppuhr die gemessene Stoppzeit auch in der Stoppuhr in einem *Array* gespeichert werden soll. Die erste gemessene Stoppzeit soll an der Stelle 0, die Zweite an der Stelle 1, die Dritte an Stelle 2 usw. abgelegt werden.

Die Stoppuhr soll über die Möglichkeit verfügen, alle gemessenen Stoppzeiten in eine *csv-Datei abzuspeichern*. Dabei wird jede gestoppte Zeit in einer *eigenen Zeile* abgelegt.

Die Stoppuhr muss mit dem mitgelieferten Programm `StoppuhrTest` arbeiten können. Betrachten Sie dieses Programm und leiten Sie davon die Methoden und das Verhalten der Klasse `Stoppuhr` ab. Die Klasse `Stoppuhr` muss so programmiert werden, dass das Programm `StoppuhrTest` nicht abgeändert werden braucht.