

Bisher haben wir der Einfachheit halber Arrays von Zahlen (int oder double) behandelt. In der Praxis hingegen hat man es meist mit Arrays komplexerer Daten, also Arrays von Objekten zu tun: Listen von Produkten, Mitarbeitern, Bankkonten, Benutzeraccounts, Textnachrichten usw.

Wir betrachten hier als Beispiel wieder unsere 2D-Spiele:
Auch Listen von Sprites kann man in Form von Arrays verwalten.

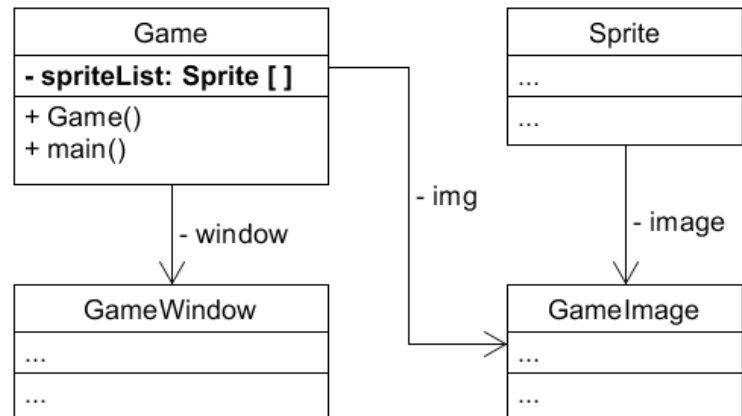
Arrays von Objekten im Klassendiagramm

Arrays werden im Klassendiagramm mit den eckigen Klammern dargestellt, ähnlich wie Arrays von einfachen Datentypen.

In diesem Beispiel hat das Hauptobjekt der Klasse Game ein Array von Sprite-Objekten.

Dass hier eine „hat“-Beziehung zwischen den Klasse Game und Sprite besteht, sieht man leider nicht durch Pfeile.

Man muss also genau hinschauen!



Deklaration und Erzeugen eines Arrays von Objekten

```

public class Game
{
    private Sprite[] spListe;
    private GameImage img;

    public Game()
    {
        int i;

        image = new GameImage(
            "images/bubble.jpg");

        spListe = new Sprite[10];

        for (i = 0; i < 10; i++)
        {
            spListe[i] =
                new Sprite(img);
        }
    }
}
  
```

Bei der Deklaration gibt man durch die **eckigen Klammern** an, dass es sich um ein Array von Sprite-Objekten handelt.

(Außerdem wird ein GameImage deklariert, da die Sprites zur Erzeugung ein Bild benötigen.)

Das **Array** wird wie gewohnt mit **new** erzeugt. Dabei wird allerdings nur eine Liste von 10 Sprite-Referenzen erzeugt, nicht die Sprite-Objekte selbst – diese Referenzen zeigen nach der Erzeugung zunächst alle auf null.

Die **10 Sprite-Objekte** kann man dann mit einer **for-Schleife** erzeugen – vorausgesetzt, dass sie alle das gleiche Bild verwenden.

Objekte in einem Array bearbeiten

Meist wird man Objekte in einer Liste nicht einzeln ansprechen, sondern alle auf die gleiche Weise bearbeiten.

Das folgende Beispiel gibt allen Sprites die Geschwindigkeit 5 und eine zufällige Position:

```
public void main()
{
    int i;
    double x, y;
    for (i = 0; i < spListe.length; i++)
    {
        spListe[i].setSpeed(5.0);
        x = Math.random() * 800;
        y = Math.random() * 600;
        spListe[i].setPosition(x, y);
    }
    ...
}
```

`spListe [i]` steht für das „i-te“ Objekt im Array der Sprites.

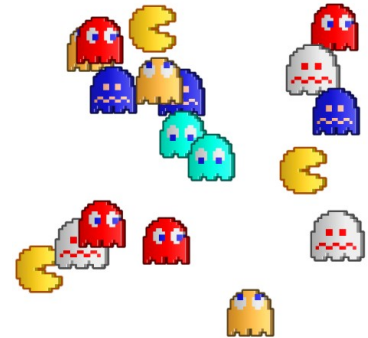
Auf dieses „i-te“ Objekt kann man dann mit dem Punkt-Operator die Methoden der Klasse Sprite anwenden:

`spListe [i] . methode () ;`

Aufgabe 1

Verwende die ausgeteilte BlueJ-Vorlage. Bearbeite die Klasse **Abprallen**. Darin bewegt sich einige Sprites in einer zufälligen Richtung und prallen vom Rand des Fensters ab.

Verändere das Programm so, dass ein Array von 30 Sprites verwendet wird, die sich alle nach dem gleichen Muster bewegen: zufällige Startposition und Geschwindigkeit, Abprallen vom Rand. Der Einfachheit halber ordne den ersten sechs Sprites das erste Bild, den nächsten sechs das zweite Bild, usw. zu.



Zusatz:

Erweitere das Programm so, dass ein Array von GameImages verwendet wird. Im images-Ordner findest du sieben Grafiken für Pacman und die Geister. Die GameImage-Objekte kannst du nicht mit einer Schleife erzeugen, da für jedes ein anderer Dateiname verwendet wird. Sie müssen einzeln erzeugt werden.

Ordne dann jedem der 30 Sprites beim Erzeugen **zufällig** eins der sieben GameImages zu.



Aufgabe 2

Implementiere die Klasse **Drehen**. Die Klasse erzeugt ein 2D-Array von 7 mal 7 Sprites (entscheide selbst, ob du jedem das gleiche Bild oder ein zufälliges, oder die Bilder nach einer bestimmten Reihenfolge zuweist). Die Bilder kannst du z.B. auf 40% verkleinern, damit sie alle im Fenster Platz haben.

Die Sprites sollen in regelmäßigen Abständen schachbrettartig platziert werden, so dass sie das Fenster ausfüllen. Dann bewegt sich jeder Sprite auf einer kleinen Kreisbahn – alle Sprites parallel zueinander.

Aufgabe 3

Implementiere die Klasse **Schlange**. Es wird ein Array von 100 Sprites erzeugt, die alle das Bild mit der Seifenblase benutzen.

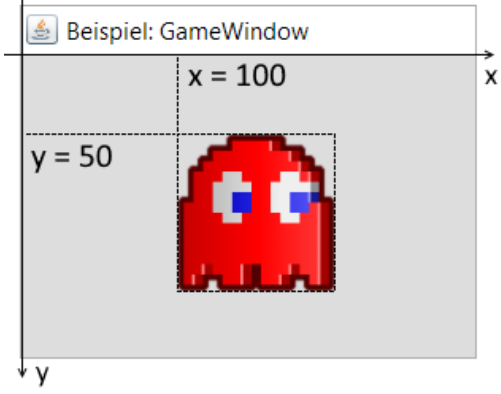
Die Sprites bilden eine „Schlange“ aus 100 kreisförmigen Segmenten. Alle beginnen in der Mitte des Fensters. Der Kopf (also der erste Sprite) folgt mit einer konstanten Geschwindigkeit (z.B. 5 Pixel pro Wiederholung) dem Mauszeiger.

Jeder weitere Sprite folgt seinem Vorgänger. Dabei sollen alle einen gleichmäßigen Abstand von z.B. 8 Pixeln einhalten (damit es wie eine Schlange aussieht und nicht wie 100 Kreise, die einander folgen). Das kannst du erreichen, indem du jedem Sprite in jedem Schritt der Wiederholungsschleife genau die Geschwindigkeit gibst, die er braucht, um bis auf 8 Pixel an seinen Vorgänger heranzukommen. Falls der Sprite schon weniger als 8 Pixel Abstand zum Vorgänger hat (z.B. zu Beginn des Spiels), bleibt er einfach stehen. Zur Berechnung des Abstands stellt die Klasse Sprite die Methode `getAbstand()` zur Verfügung (siehe Dokumentation).



Dokumentation der benötigten Klassen

Klasse GameWindow

<code>import pm.gamewindow.*;</code>	GameWindow-Package
Konstruktor (Beispiel): <code>window = new GameWindow(50, 50, 800, 600, "Beispiel");</code> Erzeugt ein GameWindow an der Bildschirm-Position x=50 y=50 mit Breite 800 und Höhe 600.	
<code>void clear()</code>	Übermalt das Fenster in weiß.
<code>int getWidth()</code> <code>int getHeight()</code>	Geben Breite / Höhe des Zeichenbereichs zurück.
<code>void drawImage(GameImage pimg, int px, int py)</code> Zeichnet eine Grafik auf das GameWindow (erscheint nach Aufruf von <code>paintFrame()</code>) <ul style="list-style-type: none"> • Der Koordinatenursprung ist die linke obere Ecke des Zeichenbereichs. • <code>px</code> und <code>py</code> geben linke obere Ecke des Bildes an. • Das Bild kann außerhalb des Zeichenbereichs liegen (es wird nur der sichtbare Teil gezeichnet) 	
<code>void paintFrame()</code>	Überträgt das gezeichnete an den Bildschirm.
<code>int getMouseX()</code> <code>int getMouseY()</code>	Gibt die x- bzw. y-Position des Mauszeigers zurück (Koordinaten beziehen sich auf den Zeichenbereich).

Klasse GameImage

Konstruktor (Beispiel): <code>img = new GameImage("../images/bild.png");</code> Erzeugt ein Bild. Der Parameter gibt den Pfad zur Bilddatei an (relativ zum Programmordner).	
<code>void setScale(double pScale)</code>	$0 < pScale < 1$ verkleinert das Bild, $1 < pScale$ vergrößert das Bild Das Originalbild bleibt erhalten (d.h. jeder Aufruf von <code>setScale</code> bezieht sich auf das Original)
<code>int getWidth()</code> / <code>int getHeight()</code>	Geben Breite / Höhe des skalierten Bildes zurück
<code>int getOriginalWidth()</code> <code>int getOriginalHeight()</code>	Geben Breite bzw. Höhe des Originalbildes (ohne Skalierung) zurück

Klasse Sprite

Konstruktor (Beispiel): <code>sp = new Sprite(pImg);</code> Erzeugt einen Sprite mit dem GameImage pImg. Das GameImage muss vorher erzeugt werden.	
<code>void setPosition(double px, double py)</code>	Setzt die Position des Sprites im GameWindow (bezogen auf die linke obere Ecke des Sprite-Bildes)
<code>void setSpeed(double ps)</code>	Setzt die Geschwindigkeit für bewege() (in Pixeln)
<code>void setRichtung(double pr)</code>	Setzt die Richtung für bewege() in Grad (0° entspricht der x-Achse)
<code>void dreheZu(Sprite psp)</code>	Dreht die Richtung zur Position des Sprites psp
<code>void bewege()</code>	Ändert die Position entspr. speed und richtung.
<code>void pralleAbHorizontal()</code> <code>void pralleAbVertikal()</code>	Ändert die Richtung wie beim Abprallen einer Kugel von einer horizontalen bzw. vertikalen Kante
<code>void draw(GameWindow pwin)</code>	Zeichnet das Bild des Sprites an der aktuellen Position x / y auf das GameWindow pwin
<code>double getX() / double getY()</code>	Geben die x- bzw. y-Koordinate des Sprites zurück
<code>int getWidth() / int getHeight()</code>	Geben die Breite bzw. Höhe des dem Sprite zugeordneten Bildes zurück
<code>boolean berührtRechts(GameWindow pwin),</code> <code>boolean berührtLinks (GameWindow pwin), usw...</code> Gibt true zurück, falls der Sprite den rechten (linken, ...) Rand des Fensters pwin berührt	
<code>boolean berührt(Sprite psp)</code> Gibt true zurück, falls dieser Sprite den Sprite psp berührt	
<code>double getAbstand(Sprite psp)</code>	Gibt den Abstand der Position dieses Sprites zur Position von psp zurück