

Das GameWindow-Package bietet eine einfache Grundlage für zweidimensionale Spiele:
Es können Bilder geladen, gezeichnet und bewegt werden, und das Programm kann auf Benutzereingaben wie Tasten oder Mausbewegungen reagieren.

Klasse GameImage

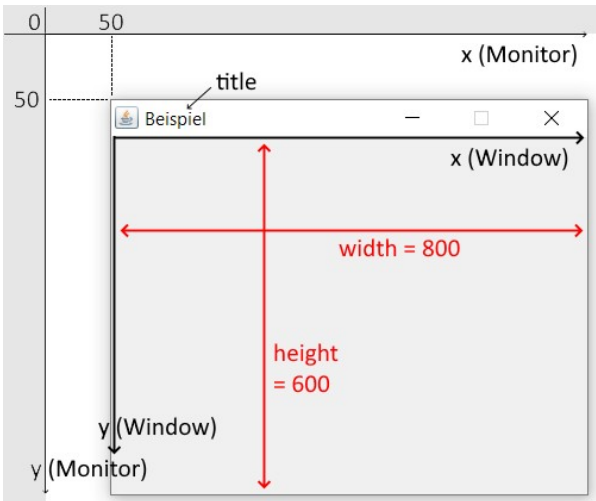
Ein GameImage ist ein Bild, das aus einer Bilddatei (z.B. JPEG oder PNG) geladen wird.
Das JPEG-Format unterstützt keine Transparenzen, daher ist das PNG-Format für Spiele i.d.R. sinnvoller.

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>int getWidth()</code>	Gibt die Breite bzw. Höhe des Bildes zurück.
<code>int getHeight()</code>	Die Größe des Bildes ergibt sich aus der Bilddatei.

Weitere Methoden der Klasse GameImage sind auf Seite 3 und 4 erläutert.

Klasse GameWindow

Ein GameWindow ist ein Fenster mit einer freien Fläche, auf der Bilder gezeichnet und in einem Spielablauf bewegt werden können. Es stellt außerdem Methoden für die Interaktion mit dem Benutzer zur Verfügung.

<pre>import pm.gamewindow.*; import java.awt.*; import java.awt.event.*;</pre> <p>Konstruktor (Beispiel): <code>window = new GameWindow(50, 50, 800, 600, "Beispiel");</code> Erzeugt ein Fenster („Window“) mit Breite 800 und Höhe 600 mit dem Text „Beispiel“ in der Titelleiste. Die linke obere Ecke des GameWindows liegt an Position $x=50$ $y=50$ auf dem Monitor. Der Koordinatenursprung des Monitors ist in der linken, oberen Ecke, und seine y-Achse wird nach unten positiv. Ebenso liegt der Koordinatenursprung des GameWindows in seiner linken, oberen Ecke und seine y-Achse wird nach unten positiv.</p>	<p>GameWindow-Package Package für Font, Color usw. Package für Tastatur-Codes</p> 
<code>int getWidth()</code> <code>int getHeight()</code>	Geben Breite / Höhe des Zeichenbereichs zurück.
<code>int getFrameWidth()</code> <code>int getFrameHeight()</code>	Geben Breite / Höhe des GameWindows zurück (mit Titelleiste etwas größer als der Zeichenbereich)
<code>void setSize(int pwidth, int pheight)</code>	Verändert die Größe des Zeichenbereichs.
<code>void setLocation(int px, int py)</code>	Verschiebt das GameWindow auf dem Bildschirm.
<code>String getTitle()</code>	Gibt den Titel des GameWindows zurück
<code>void setTitle(String pTitle)</code>	Ändert den Titel des GameWindows

GameWindow: Bewegen von Bildern

Die **Bewegung** eines Bildes geschieht in folgendem Ablauf:

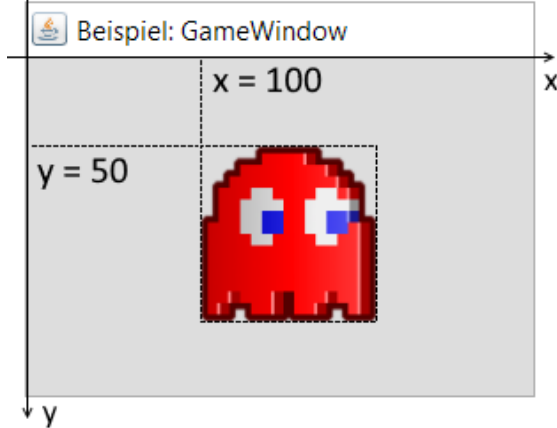
Das Bild wird gezeichnet; dann gelöscht; dann ein paar Pixel verschoben erneut gezeichnet, gelöscht, usw.

Um Bilder vom GameWindow zu **löschen** kann man den Zeichenbereich komplett löschen mit `clear()`, oder man übermalt den ganzen Zeichenbereich Fenster mit einem Hintergrundbild mit `drawImage()`.

Der Wechsel „Hintergrund zeichnen – Bilder zeichnen“ geschieht so schnell, dass für das menschliche Auge die Illusion einer flüssigen Bewegung entsteht (ähnlich wie im Kino oder Fernsehen).

Die Zeiteinheit, in der einmal der Hintergrund und die darüber liegenden Bilder gezeichnet werden, wird **Frame** genannt. Ein Frame dauert in der Regel 25 Millisekunden, das ergibt 40 Frames pro Sekunde.

Damit das schnelle Zeichnen mehrerer Bilder bzw. das Übermalen durch den Hintergrund ohne Flackern abläuft, gibt es die Methode **paintFrame()**. Diese Methode wartet auf den richtigen Moment und zeichnet dann Hintergrund und Bilder gleichzeitig auf den Bildschirm. Sie sorgt dafür, dass die Bewegung immer gleich schnell abläuft, unabhängig von der Geschwindigkeit des Computers und der Anzahl der Bilder.

<pre>void clear() void clear(Color pcolor)</pre>	Füllt den Zeichenbereich komplett mit weiß, bzw. mit der angegebenen Farbe
<pre>void drawImage(GameImage pimg, int px, int py);</pre> <p>Zeichnet eine Grafik auf das GameWindow</p> <p>Das Bild erscheint allerdings erst, nachdem die Methode <code>paintFrame()</code> aufgerufen wurde.</p> <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) 	
<pre>void paintFrame()</pre>	Zeichnet dann das komplette Fenster und wartet, bis die Zeit für diesen Frame abgelaufen ist
<pre>void setFrameTime(int pms)</pre>	Setzt die Dauer eines Frames in Millisekunden. Der Standardwert ist 25 ms (man braucht diesen Wert nur in Ausnahmefällen zu verändern)
<pre>void wait(int pms)</pre>	Stoppt den Ablauf für eine gegebene Zeit (in ms).
<p>Beispiel für einen Ablauf (bewegt ein Bild langsam von links nach rechts)</p> <pre>int x = 50, y = 50; while (true) { window.clear(); window.drawImage(img, x, y); x = x + 5; window.paintFrame(); }</pre>	

GameWindow: Tastatur- und Mausereignisse

Spieler steuern das Spiel, am PC meist mit Maus und Tastatur. Dazu kann man in jedem Frame abfragen, ob z.B. eine bestimmte Taste gedrückt ist, oder wo sich der Mauszeiger befindet.

Hinweis: die Event-Bibliothek muss über `import java.awt.event.*;` eingebunden werden.

<code>boolean isKeyDown(int pKeyCode)</code>	Gibt true zurück, falls die Taste mit dem gegebenen Code gedrückt ist. Für die Tastaturcodes stellt Java die Klasse KeyEvent zur Verfügung (siehe unten).
<code>boolean mouseButton1()</code> <code>boolean mouseButton2()</code>	Gibt true zurück, falls die linke bzw. rechte Maustaste gedrückt ist.
<code>int getMouseX()</code> <code>int getMouseY()</code>	Gibt die x- bzw. y-Position des Mauszeigers zurück (Koordinaten beziehen sich auf den Zeichenbereich).
<code>int mouseWheel()</code>	Gibt zurück, wie viele „notches“ sich das Mousrad seit der letzten Abfrage bewegt hat. 0 bedeutet keine Bewegung. Ein Abwärtsdrehen des Mousrads gibt einen negativen Wert, aufwärts einen positiven.
Beispiel für die Abfrage von Tastatur- / Mausereignissen <pre> if (window.isKeyDown(KeyEvent.VK_LEFT)) { x = x - 3; } if (window.mouseButton1()) { x = window.getMouseX(); y = window.getMouseY(); } </pre>	

Tastaturcodes der Klasse KeyEvent (Auszug)

Diese Codes werden in der Methode `isKeyDown()` eingesetzt z.B. als `KeyEvent.VK_ENTER`

Mit den Codes sind die Tasten gemeint, daher wird nicht unterschieden zwischen Klein- und Großbuchstaben. Für einen Großbuchstaben müsste man abfragen, ob z.B. die Taste `VK_A` und die Taste `VK_SHIFT` gleichzeitig gedrückt sind (für eine Spielsteuerung ist das aber meist egal).

<code>VK_0 ... VK_9</code>	Nummern-Tasten 1 bis 9 und 0 (nicht die auf dem Nummernblock rechts)
<code>VK_A ... VK_Z</code>	Buchstaben-Tasten
<code>VK_LEFT, VK_RIGHT, VK_UP, VK_DOWN</code>	Pfeiltasten
<code>VK_SPACE, VK_ENTER</code>	Leertaste, Enter
<code>VK_SHIFT, VK_CAPS_LOCK</code>	Shift und Feststelltaste
<code>VK_ALT, VK_ALT_GRAPH</code>	Alt bzw. Alt-Gr
<code>VK_DELETE, VK_BACK_SPACE</code>	Löschen und Backspace
<code>VK_ESCAPE</code>	Esc-Taste
<code>VK_F1 ... VK_F12</code>	F1 bis F12-Tasten
<code>VK_NUMPAD0 ... VK_NUMPAD9</code>	Nummerntasten (im Nummernblock rechts)

GameWindow: Zeichnen und Textausgabe

drawLine (int px1, int py1, int px2, int py2, Color pColor) Zeichnet eine Linie von px1 / py1 nach px2 / py2 in der gegebenen Farbe.	
drawRectangle (int px, int py, int pWidth, int pHeight, Color pColor) Zeichnet den Umriss eines Rechtecks in der gegebenen Breite, Höhe und Farbe (px / py: linke obere Ecke)	
fillRectangle (int px, int py, int pWidth, int pHeight, Color pColor) Zeichnet ein ausgefülltes Rechteck in der gegebenen Breite, Höhe und Farbe (px / py: linke obere Ecke)	
void drawString (String ps, int px, int py)	Zeichnet einen Text px / py geben die linke obere Ecke des Textes an.
void setFont (Font pfont)	Setzt die Schriftart für drawString().
void setFontColor (Color pcolor)	Setzt die Schriftfarbe für drawString()
Beispiel: Setzt die Schriftart Arial, fett, Schriftgröße 16, orange (Rot = 255, Grün = 128, Blau = 0) <pre>Font f = new Font("Arial", Font.BOLD, 16); Color c = new Color(255, 128, 0); window.setFont(f); window.setFontColor(c);</pre>	

GameImage: Skalierung

GameImage-Objekte können skaliert (d.h. vergrößert bzw. verkleinert) werden.

Für eine Skalierung erzeugt das GameImage intern eine Kopie des Bildes im gewünschten Maßstab.

So ist sichergestellt, dass durch wiederholtes Skalieren nicht die Bildqualität leidet.

<pre>GameImage img = new GameImage(gImage);</pre> <p>Dieser zusätzliche Konstruktor erzeugt eine Kopie eines anderen GameImage-Objekts. Beide benutzen das gleiche Originalbild, aber unterschiedliche Kopien für Skalierung (und Drehung). So kann man den Platz für das Originalbild „sparen“, wenn mehrere Sprites das gleiche Bild verwenden. Falls Drehung erlaubt sein soll, sollte das GameImage vor Erstellen der Kopie drehbar gemacht werden.</p>	
<pre>int getWidth() int getHeight()</pre>	Geben Breite bzw. Höhe des skalierten Bildes zurück (gleich der Größe von gedrehten Bildern)
<pre>int getOriginalWidth() int getOriginalHeight()</pre>	Geben Breite bzw. Höhe des unskalierten Bildes zurück.
<pre>void setScale(double pScale)</pre>	$0 < pScale < 1$ verkleinert, $pScale > 1$ vergrößert Beispiel: Eine Vergrößerung um $pScale = 2$ verdoppelt die Fläche des Bildes (die Seiten verlängern sich dabei um den Faktor $\sqrt{2}$).
<pre>double getScale()</pre>	Gibt den Skalierungsfaktor der Fläche des skalierten Bildes im Vergleich zur Originalfläche zurück.
<pre>void setScale(double pScaleX, double pScaleY)</pre>	Skaliert Breite und Höhe des Bildes unabhängig voneinander um die gegebenen Faktoren (so kann man das Bild strecken). Für drehbare Bilder ist diese Möglichkeit deaktiviert.
<pre>void setScaleSize(int pWidth, int pHeight)</pre>	Skaliert Breite und Höhe auf die Parameter (d.h. das skalierte Bild hat die Ausmaße $pWidth \times pHeight$) Für drehbare Bilder ist diese Möglichkeit deaktiviert.
<pre>double getScaleX() double getScaleY()</pre>	Geben den Skalierungsfaktor der Breite bzw. der Höhe des Bildes zurück. Falls die Skalierung mit <code>setScale(pScale)</code> gesetzt wurde, entsprechen beide \sqrt{pScale} .

GameImage: Drehung

Um ein Bild zu drehen, braucht es einen Rand, damit durch die Drehung nicht die Ecken des Bildes abgeschnitten werden. Möchte man ein Bild drehbar machen, wird daher zunächst automatisch ein Rand zum Originalbild hinzugefügt, so dass bei jedem Drehwinkel das ganze Bild erhalten bleibt. Dadurch ändern sich Breite und Höhe des Bildes, und die Form wird quadratisch (auch wenn das Bild vorher ein Rechteck war).

Diese Vergrößerung bewirkt jedoch, dass Kollisionschecks anhand der Bildgröße nicht mehr richtig funktionieren (die Bilder „berühren“ sich dann schon mit ihrem unsichtbaren Rand).

Für drehbare Bilder gibt es daher zusätzlich eine „**bounding box**“, die den Rand des Bildes für die Berechnung von Kollisionen definiert. Diese wird automatisch berechnet.



```
void makeRotatable()
```

Macht das Bild drehbar, indem ein durchsichtiger Rand zum Bild hinzugefügt wird.

Das Bild ist anschließend ein Quadrat, die Seitenlänge entspricht der Diagonale des Originals.

Eine Skalierung des Bildes bleibt erhalten, jedoch nur, wenn das Bild nicht in Breite oder Höhe gestreckt wurde (bei Streckung wird der Skalierungsfaktor auf 1.0 zurückgesetzt).

Berechnet außerdem die Bounding Box. *Kann nicht rückgängig gemacht werden.*

```
boolean isRotatable()
```

Gibt true zurück, falls das Bild mit makeRotatable() drehbar gemacht wurde.

```
void rotate(double pDegrees)
```

Dreht das Bild um den Parameter im Gradmaß nach rechts (negative Werte: nach links)

Die aktuelle Skalierung des Bildes bleibt erhalten. Die Bildgröße ändert sich nicht.

Falls das Bild nicht drehbar ist, geschieht nichts.

```
double getRotation()
```

Gibt die aktuelle Drehung im Gradmaß zurück.

```
double getBoundX()
```

```
double getBoundY()
```

```
double getBoundWidth()
```

```
double getBoundHeight()
```

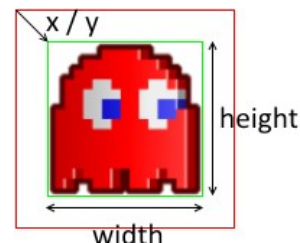
Geben die Maße der Bounding Box zurück.

Diese wird automatisch berechnet unter der Annahme, dass das Originalbild ein Quadrat war.

Falls das Original ein Rechteck war, liegt die Seitenlänge zwischen Breite und Höhe des Originals.

Die Box wird bei Skalierung neu berechnet.

(Für nicht-drehbare Bilder entspricht die Bounding Box den Maßen des skalierten Bildes.)



```
void setBoundingBox(double x, double y, double w, double h)
```

Definiert eine eigene Bounding Box statt der automatisch berechneten.

Achtung: durch anschließendes Skalieren gehen diese Werte verloren.

Autor: Christian Pothmann – cpothmann.de
Freigegeben unter [CC BY-NC-SA 4.0](https://creativecommons.org/licenses/by-nc-sa/4.0/), März 2021



Quellen: Pacman-Grafiken: strategywiki.org, Freigegeben unter [CC BY-SA 3.0](https://creativecommons.org/licenses/by-sa/3.0/)