## Dokumentation der Klasse Fenster

|  |  |
| --- | --- |
| Klassenname | Fenster |
| *Oberklasse* |  |
| *Beschreibung* | Ein Fenster ist ein Fenster auf dem Bildschirm, in das mit Objekten der Klasse Grafik gezeichnet werden kann. Zu diesem Zweck ist das Fenster mit einem Koordinatensystem versehen, dessen Ursprung sich in der oberen linken Ecke der Zeichenebene befindet, und dessen Achsen horizontal nach rechts und vertikal nach unten gerichtet sind (wie bei Computeranwendungen üblich). Das Fenster hat eine Größe von 700 Einheiten x 700 Einheiten. |
| *Bezugsklassen* | Maus, Grafik |
| *Protokoll* | Konstruktor **Fenster()**  Außer der Erzeugung des Fensters geschieht nichts. |
|  | Auftrag **setTitle(String str):void**  Das Fenster bekommt den Text str als Überschrift. Er steht dann in die Fensterleiste, in der sich auch die Buttons zum Schließen und Verkleinern des Fensters befinden. |
|  | Auftrag **setLocation(int x, int y):void**  setzt die Fensterposition auf dem Bildschirm. x und y sind die Bildschirmkoordinaten der linken oberen Fensterecke. |
|  | Auftrag **setSize (int width, int height):void**  Das Fenster erhält die angegebene Größe. |
|  | Anfrage **getWidth() : int**  Diese Anfrage liefert die Breite des Fensters. Die Breite entspricht der Breite des Fensterobjekts einschließlich des Fensterrahmens. |
|  | Anfrage **getHeight()** **: int**  Diese Anfrage liefert die Höhe des Fensters. Die Höhe entspricht der Höhe des Fensterobjekts einschließlich des Fensterrahmens. |
|  | Auftrag **setBackground(Color farbe):void**  Die Hintergrundfarbe wird auf farbe gesetzt. Zulässige Farben sind die üblichen Farben (Color.green, Color.blue, Color.red, Color.yellow, Color.black, Color.white) oder eine mit Hilfe des Befehls Color neueFarbe=new Color(zahl1,zahl2, zahl3) mit den Zahlen zahl1, zahl2 und zahl3, die den rot-, grün- und blau-Anteile angeben, selbst gemischte Farbe.  Um Farben verwenden zu können, muss zusätzlich zu hiddenClasses.\* java.awt.\* importiert werden. |
|  | Anfrage **getBackground():Color**  liefert die Hintergrundfarbe. Diese kann in einer Variablen vom Typ Color gespeichert und z.B. an Methoden übergeben werden, die ein Farbobjekt als Parameter benötigen. Um Farben verwenden zu können, muss zusätzlich zu hiddenClasses.\* java.awt.\* importiert werden |
|  | Auftrag **clearScreen():void**  Setzt das Fenster auf die Hintergrundfarbe zurück. Die Malfläche des Fensterobjekts wird vollständig mit der Hintergrundfarbe überzeichnet. |
|  | Auftrag **setResizable(boolean b):void**  legt fest, ob das Fenster in seiner Größe veränderbar ist. Das Fenster kann mit der Maus vergrößert werden, wenn für b der Wert true übergeben wird. Wird der Wert false für b übergeben, kann man das Fenster nicht durch Ziehen mit der Maus vergrößern oder verkleinern. |
|  | Anfrage **isResizable():boolean**  gibt zurück, ob das Fenster in seiner Größe veränderbar ist. Die Fenstergröße ist veränderbar, wenn der Wert true zurückgegeben wird. Andernfalls wird der Wert false zurückgegeben. |
|  | Anfrage **getTitle() : String**  Die Anfrage liefert den Titel des Fensters. Dieses ist der Text, der in der Titelleiste des Fensters steht. |

## Dokumentation der Klasse Grafik

|  |  |
| --- | --- |
| Klassenname | Grafik |
| *Oberklasse* |  |
| *Beschreibung* | Mit einem Objekt der Klasse Grafik kann man in einem Objekt der Klasse Fenster zeichnen. Das Fensterobjekt muss vor dem Grafikobjekt erzeugt werden. |
| *Bezugsklassen* | Fenster |
| *Protokoll* | Konstruktor **Grafik()**  Außer der Erzeugung des Grafikobjekts geschieht nichts. Wenn noch kein Fensterobjekt erzeugt worden ist, kommt es zu einer NullPointerException. Das Programm wird dann beendet. |
|  | Auftrag **drawLine(int xa, int ya, int xe, int ye):void**  Es wird eine Linie von der Position (xa,ya) zur Position (xe,ye) gezeichnet. |
|  | Auftrag **drawOval(int x, int y, int width, int height):void**  Es wird eine Ellipse gezeichnet. Dazu denkt man sich ein Rechteck, dass die Ellipse umschreibt. Die linke obere Ecke des umschreibenden Rechtecks liegt an der Position (x,y), width und height geben die Höhe und die Breite an. |
|  | Auftrag **drawRect(int x, int y, int width, int height):void**  Es wird ein Rechteck gezeichnet. Die linke obere Ecke liegt an der Position (x,y), width und height geben die Höhe und die Breite an. |
|  | Auftrag **drawRoundRect(int x, int y, int w, int h):void**  Es wird ein Rechteck gezeichnet, dessen Ecken abgerundet sind, so dass es strenggenommen kein Rechteck mehr ist. Die linke obere Ecke des entsprechenden (richtigen) Rechtecks liegt im Punkt (x,y). w und h geben die Breite und die Höhe an. |
|  | Auftrag **drawArc(int x, int y, int width, int height, int startAngle, int arcAngle):void**  Es wird ein Kreis- oder Ellipsenbogen gezeichnet.  Die ersten vier Parameter (x,y, width, height) sind identisch mit denen der drawOval()-Methode.  startAngle ist der Anfangswinkel , von einer gedachten horizontalen Mittellinie aus gesehen, ab dem der Bogen gezeichnet werden soll. Der Parameter arcAngel legt den Mittelpunktswinkel des Bogens fest. (Gezeichnet wird gegen den Uhrzeigersinn.) |
|  | Auftrag **drawPolygon(int[] xPoints, int[] yPoints, int nPoints):void**  Es wird ein Polygon gezeichnet. Die x-Koordinaten der Eckpunkte des Polygons sind in der Menge (bzw. einem Feld) xPoints (z.B. {1,5,3}) angegeben, die y-Koordinaten in der Menge (bzw. dem Feld) yPoints. nPoints ist die Anzahl der Ecken des Polygons. Das Polygon wird automatisch geschlossen, indem eine Linie vom letzten Punkt zum ersten Punkt gezeichnet wird, falls diese beiden Punkte verschieden sind. |
|  | Auftrag **drawPolyline(int[] xPoints, int[] yPoints, int nPoints):void**  Es werden nPoints Punkte miteinander verbunden. gezeichnet. Die x-Koordinaten der Punkte sind in der Menge (bzw. einem Feld) xPoints (z.B. {1,5,3}) angegeben, die y-Koordinaten in der Menge (bzw. dem Feld) yPoints. nPoints ist die Anzahl der Punkte. Hierbei werden Anfangs- und Endpunkt nicht automatisch miteinander verbunden. |
|  | Auftrag **fillArc(int x, int y, int width, int height, int startAngle, int arcAngle):void**  Es wird ein gefüllter Kreis- oder Ellipsenbogen gezeichnet.  Die ersten vier Parameter (x,y, width, height) sind identisch mit denen der drawOval()-Methode.  startAngle ist der Anfangswinkel , von einer gedachten horizontalen Mittellinie aus gesehen, ab dem der Bogen gezeichnet werden soll. Der Parameter arcAngel legt den Mittelpunktswinkel des Bogens fest. (Gezeichnet wird gegen den Uhrzeigersinn.) |
|  | Auftrag **fillOval(int x, int y, int width, int height):void**  Es wird eine gefüllte Ellipse gezeichnet. Dazu denkt man sich ein Rechteck, dass die Ellipse umschreibt. Die linke obere Ecke des umschreibenden Rechtecks liegt an der Position (x,y), width und height geben die Höhe und die Breite an. |
|  | Auftrag **fillPolygon(int[] xPoints, int[] yPoints, int nPoints):void**  Es wird ein gefülltes Polygon gezeichnet. Die x-Koordinaten der Eckpunkte des Polygons sind in der Menge (bzw. einem Feld) xPoints (z.B. {1,5,3} angegeben, die y-Koordinaten in der Menge (bzw. dem Feld) yPoints. nPoints ist die Anzahl der Ecken des Polygons. Automatisch werden dabei der Anfangs- und der Endpunkt miteinander verbunden. |
|  | Auftrag **fillRect(int x, int y, int width, int height):void**  Es wird ein gefülltes Rechteck gezeichnet. Die linke obere Ecke liegt an der Position (x,y), width und height geben die Höhe und die Breite an. |
|  | Auftrag **fillRoundRect(int x, int y, int w, int h):void**  Es wird ein gefülltes Rechteck gezeichnet, dessen Ecken abgerundet sind, so dass es strenggenommen kein Rechteck mehr ist. Die linke obere Ecke des entsprechenden (richtigen) Rechtecks liegt im Punkt (x,y). w und h geben die Breite und die Höhe an. |
|  | Anfrage **getColor() : Color**  getColor() liefert die zur Zeit verwendete Farbe. Diese kann in einer Variablen vom Typ Color gespeichert werden oder an Methoden übergeben werden, die ein Farbobjekt als Parameter benötigen. |
|  | Auftrag **setColor(Color c):void**  Von nun an wird in der Farbe c gezeichnet. Zulässige Farben sind die üblichen Farben (Color.green, Color.blue, Color.red, Color.yellow, Color.black, Color.white) oder eine mit Hilfe des Befehls Color neueFarbe=new Color(zahl1,zahl2, zahl3) mit den Zahlen zahl1, zahl2 und zahl3, die den rot-, grün- und blau-Anteile angeben, selbst gemischte Farbe.  Um Farben verwenden zu können, muss zusätzlich zu hiddenClasses.\* java.awt.\* importiert werden. |
|  | Auftrag **setPaintMode():void**  Setzt den Zeichenmodus auf Normal (zurück). Beim Aufruf von Zeichne-Methoden, werden alle betroffenen Pixel in der zur Zeit gültigen Zeichenfarbe übermalt. |
|  | Auftrag **setXORMode(Color c): void**  Beim Aufruf von Zeichnemethoden erhalten betroffene Pixel der momentanen Zeichenfarbe die Farbe c und Pixel der Farbe c die momentane Zeichenfarbe.  Pixel anderer Farben können sich dabei ebenfalls (unvorhergesehen)ändern.  Zulässige Farben sind die üblichen Farben (Color.green, Color.blue, Color.red, Color.yellow, Color.black, Color.white) oder eine mit Hilfe des Befehls Color neueFarbe=new Color(zahl1,zahl2, zahl3) mit den Zahlen zahl1, zahl2 und zahl3, die den rot-, grün- und blau-Anteile angeben, selbst gemischte Farbe.  Um Farben verwenden zu können, muss zusätzlich zu hiddenClasses.\* java.awt.\* importiert werden. |
|  | Auftrag **translate(int x,int y):void**  Der Ursprung des Koordinatensystems der Zeichenebene wird in den Punkt (x,y) gesetzt. Von nun an wird bei weiteren Methodenaufrufen von Methoden dieses Grafikobjekts von diesem Koordinatensystem ausgegangen. |
|  | Auftrag **setFont(Fontobject fo): void**  Setzt den Font, mit dem eine Zeichenkette geschrieben werden kann. Dazu muss ein Fontobject erzeugt sein, das die gewünschte Parameterwerte (Schriftname,Style,Size) erhalten hat.  Zur Verwendung von Fonts muss die java.awt.\*-Bibliothek importiert werden. |
|  | Anfrage **getFont(): Font**  Liefert den aktuell gültigen Schriftfont. Standardvorgabe ist die Schriftart „Dialog“, Schriftstil „PLAIN“ und Schriftgröße 12 Punkte.  Zur Verwendung von Fonts muss die java.awt.\*-Bibliothek importiert werden. |
|  | Auftrag **drawString(String str, int x, int y):void**  Die Zeichenkette str (z.B. ein Text) wird an der Position (x,y) auf der Zeichenebene ausgegeben. Die (gedachte) Grundlinie, auf der die Zeichenkette ausgegeben wird, geht in horizontale Richtung durch den Punkt (x,y). Das erste Zeichen der Zeichenkette beginnt an diesem Punkt. Die Zeichenkette wird mit dem aktuellen Font ausgegeben. |

## Dokumentation der Klasse Maus

|  |  |
| --- | --- |
| Klassenname | Maus |
| *Oberklasse* |  |
| *Beschreibung* | Eine Maus realisiert die Mauseingabe des verwendeten Computers. Diese ist gekennzeichnet durch die aktuelle Position der Maus auf dem Bildschirm, der Bewegung der Maus und die Betätigung der Maustaste. |
| *Bezugsklassen* | Fenster |
| *Protokoll* | Konstruktor **Maus()**  Außer der Erzeugung der Maus geschieht nichts. Wenn noch kein Fensterobjekt erzeugt worden ist, kommt es zu einer NullPointerException. Das Programm wird dann beendet. |
|  | Anfrage **geklickt()** : boolean  liefert den Wert true, wenn die Maus geklickt worden ist und den Wert false, wenn die Maus nicht geklickt worden ist, noch gedrückt wird oder das Fenster verlassen hat. |
|  | Anfrage **gedrueckt()**: boolean  liefert den Wert true, wenn die Maus gedrückt ist und den Wert false, wenn die Maus im Moment nicht gedrückt wird oder aus das Fenster verlassen hat. |
|  | Anfrage **losgelassen()** : boolean  liefert den Wert true, wenn die Maus losgelassen worden ist und den Wert false, wenn die Maus noch gedrückt wird oder das Fenster verlassen hat. |
|  | Anfrage **gezogen()** : boolean  liefert den Wert true, wenn die Maus gedrückt wird und im Fenster bewegt wird und den Wert false, wenn die Maus nicht gedrückt wird oder das Fenster verlassen hat. |
|  | Anfrage **bewegt()** : boolean  liefert den Wert true, im Fenster bewegt wird und nicht gedrueckt ist, den Wert false, wenn die Maus gedrückt ist oder das Fenster verlassen hat oder nicht bewegt wird. |
|  | Anfrage **getX()** : int  Diese Anfrage liefert die gegenwärtige horizontale Koordinate der Position der Maus auf dem Bildschirm als ganzzahligen Wert, unabhängig davon, ob die Maus gedrückt wurde. |
|  | Anfrage **getY()** : int  Diese Anfrage liefert die gegenwärtige vertikale Koordinate der Position der Maus auf dem Bildschirm als ganzzahligen Wert, unabhängig davon, ob die Maus gedrückt wurde. |
|  | Anfrage **im Fenster()**:boolean  liefert den Wert true, wenn die Maus im Fenster ist und den Wert false, wenn die Maus das Fenster verlassen hat. |

## Dokumentation der Klasse Tastatur

|  |  |
| --- | --- |
| Klassenname | Tastatur |
| *Oberklasse* |  |
| *Beschreibung* | Eine Tastatur realisiert die Tastatureingabe des verwendeten Computers.Es kann ermittelt werden ob irgendeine Taste gedrückt wurde oder speziell ob eine Cursortaste (Pfeiltaste) gedrückt wurde. Von einigen gedrückten Tasten kann auch das zugehörige Zeichen ermittelt werden, das beim Drücken dieser Taste in das Schreibfeld eines Editors geschrieben wird. |
| *Bezugsklassen* | Fenster |
| *Protokoll* | Konstruktor **Tastatur()**  Außer der Erzeugung der Tastatur geschieht nichts. |
|  | Anfrage: **gedrueckt() : boolean**  liefert den Wert true, wenn eine beliebige Taste gedrückt wird und den Wert false, wenn keine Taste gedrückt wird. |
|  | Anfrage: **zeichen() : char**  liefert nur die druckbaren Zeichen aus dem Haupttastaturfeld, die ohne Shift-Taste, Strg-Taste oder Alt-Taste erzeugt werden. Andere Tasten werden hierbei ignoriert. |
|  | Anfrage **raufPfeilGedrueckt():boolean**  liefert den Wert true, wenn die Cursortaste mit dem Pfeil nach oben gedrückt wird und den Wert false, wenn sie nicht gedrückt wird. |
|  | Anfrage **runterPfeilGedrueckt():boolean**  liefert den Wert true, wenn die Cursortaste mit dem Pfeil nach unten gedrückt wird und den Wert false, wenn sie nicht gedrückt wird. |
|  | Anfrage **rechtsPfeilGedrueckt():boolean**  liefert den Wert true, wenn die Cursortaste mit dem Pfeil nach rechtsgedrückt wird und den Wert false, wenn sie nicht gedrückt wird. |
|  | Anfrage **linksPfeilGedrueckt():boolean**  liefert den Wert true, wenn die Cursortaste mit dem Pfeil nach linksgedrückt wird und den Wert false, wenn sie nicht gedrückt wird. |

## Dokumentation der Klasse Zeitmanager

|  |  |
| --- | --- |
| Klassenname | Zeitmanager |
| *Oberklasse* |  |
| *Beschreibung* | Ein Objekt der Klasse Zeitmanager dient zur Steuerung von zeitlichen Vorgängen und zur Zeitmessung. |
| *Bezugsklassen* |  |
| *Protokoll* | Konstruktor **Zeitmanager()**  Außer der Erzeugung des Zeitmanagers geschieht nichts. |
|  | Auftrag **starteUhr():void**  startet eine Zeitmessung. Die Messung wird mit dem Aufruf einer der Methoden gibSekunden() oder gibMillisekunden() abgeschlossen. |
|  | Anfrage: **gibMillisekunden() : long**  liefert die gemessene Zeit in Millisekunden. Es wird die Zeit angegeben, die seit dem letzten Aufruf der Methode starteUhr() vergangen ist. Wurde diese Methode vorher nicht aufgerufen, wird die Zeit angegeben, die seit dem 1.1.1970 0 Uhr UTC (Weltzeit) vergangen ist. |
|  | Anfrage: **gibSekunden() : long**  liefert die gemessene Zeit in Sekunden. Es ist auf ganze Sekunden abgerundet. Es wird die Zeit angegeben, die seit dem letzten Aufruf der Methode starteUhr()vergangen ist. Wurde diese Methode vorher nicht aufgerufen, wird die Zeit angegeben, die seit dem 1.1.1970 0 Uhr UTC (Weltzeit) vergangen ist. |
|  | Auftrag **warte(long millis):void**  unterbricht ein laufendes Programm. millis gibt die Dauer der Unterbrechung in Millisekunden übergeben. Das Programm wird nach der Unterbrechung an der gleichen Stelle unverändert fortgesetzt, an der es ohne die Unterbrechung weitergegangen wäre. |